Creating a basic data structure (BDS) Finding ADaM

Lun-Hsien Chang

2024-12-15

Programming workflow

Introduction

This article describes creating a BDS finding ADaM. Examples are currently presented and tested in the context of ADVS. However, the examples could be applied to other BDS Finding ADaMs such as ADEG, ADLB, etc. where a single result is captured in an SDTM Finding domain on a single date and/or time.

Note: All examples assume CDISC SDTM and/or ADaM format as input unless otherwise specified.

Read in Data

To start, all data frames needed for the creation of `ADVS` should be read into the environment. This will be a company specific process. Some of the data frames needed may be `VS` and `ADSL`.

For example purpose, the CDISC Pilot SDTM and ADaM datasets—which are included in `{pharmaversesdtm}`—are used.

```
Warning: package 'pharmaversesdtm' was built under R version 4.4.2
```

```
Warning: package 'lubridate' was built under R version 4.4.2
```

```

Attaching package: 'lubridate'
```

```
The following objects are masked from 'package:base':

    date, intersect, setdiff, union
```

At this step, it may be useful to join `ADSL` to your `VS` domain. Only the `ADSL` variables used for derivations are selected at this step. The rest of the relevant `ADSL` variables would be added later.

```
adsl_vars <- exprs(TRTSDT, TRTEDT, TRT01A, TRT01P)

advs <- derive_vars_merged(
  dataset=vs
  ,dataset_add = adsl
  ,new_vars = adsl_vars
  ,by_vars = exprs(STUDYID, USUBJID)
) # dim(advs) 29643 28

advs %>% select(USUBJID,VSTESTCD,VSDTC,VISIT,TRTSDT,TRTEDT,TRT01A,  TRT01P) %>% head(n=10)
```

```
# A tibble: 10 x 8
   USUBJID     VSTESTCD VSDTC      VISIT     TRTSDT     TRTEDT     TRT01A TRT01P
   <chr>       <chr>    <chr>      <chr>     <date>     <date>     <chr>  <chr>
 1 01-701-1015 DIABP    2013-12-26 SCREENIN~ 2014-01-02 2014-07-02 Place~ Place~
 2 01-701-1015 DIABP    2013-12-26 SCREENIN~ 2014-01-02 2014-07-02 Place~ Place~
 3 01-701-1015 DIABP    2013-12-26 SCREENIN~ 2014-01-02 2014-07-02 Place~ Place~
 4 01-701-1015 DIABP    2013-12-31 SCREENIN~ 2014-01-02 2014-07-02 Place~ Place~
 5 01-701-1015 DIABP    2013-12-31 SCREENIN~ 2014-01-02 2014-07-02 Place~ Place~
 6 01-701-1015 DIABP    2013-12-31 SCREENIN~ 2014-01-02 2014-07-02 Place~ Place~
 7 01-701-1015 DIABP    2014-01-02 BASELINE  2014-01-02 2014-07-02 Place~ Place~
 8 01-701-1015 DIABP    2014-01-02 BASELINE  2014-01-02 2014-07-02 Place~ Place~
 9 01-701-1015 DIABP    2014-01-02 BASELINE  2014-01-02 2014-07-02 Place~ Place~
10 01-701-1015 DIABP    2014-01-14 AMBUL EC~ 2014-01-02 2014-07-02 Place~ Place~
```

Derive/Impute Numeric Date/Time and Analysis Day

ADT

- "Analysis Date" for the vital signs measurement
- The date associated with AVAL and/or AVALC in numeric format.

ADTF

- Analysis Date Imputation Flag
- The level of imputation of analysis date. If ADT (or the date part of ADTM) was imputed, ADTF must be populated and is required. See Section 3.1.3, Date and Time Imputation Flag Variables.

ADTM

- Analysis Datetime
- The datetime associated with AVAL and/or AVALC in numeric format.

ATMF

- Analysis Time Imputation Flag
- The level of imputation of analysis time. If ATM (or the time part of ADTM) was imputed, ATMF must be populated and is required. See Section 3.1.3, Date and Time Imputation Flag Variables.

ADY

- Analysis Relative Day
- The relative day of AVAL and/or AVALC. The number of days from an anchor date (not necessarily DM.RFSTDTC) to ADT. See Section 3.1.2, Timing Variable Conventions. If a dataset contains more than one record per parameter per subject, then an SDTM or ADaM relative timing variable must be present (ADY would meet this requirement).
- ADY = ADT - TRTSDT (Treatment start date)+1

The function `derive_vars_dt()` can be used to derive `ADT`. This function allows the user to impute the date as well.

```
# ADT derived from VSDTC
advs <- derive_vars_dt(advs, new_vars_prefix = "A", dtc = VSDTC,
                       # Partial date imputed to the first of the month
  #highest_imputation = "M"
  ) # dim(advs) 29643 29

advs %>% select(USUBJID,VISIT,VSDTC,ADT) %>% head(n=10)
```

```
# A tibble: 10 x 4
   USUBJID     VISIT                VSDTC      ADT
   <chr>       <chr>                <chr>      <date>
 1 01-701-1015 SCREENING 1          2013-12-26 2013-12-26
 2 01-701-1015 SCREENING 1          2013-12-26 2013-12-26
 3 01-701-1015 SCREENING 1          2013-12-26 2013-12-26
 4 01-701-1015 SCREENING 2          2013-12-31 2013-12-31
 5 01-701-1015 SCREENING 2          2013-12-31 2013-12-31
 6 01-701-1015 SCREENING 2          2013-12-31 2013-12-31
 7 01-701-1015 BASELINE             2014-01-02 2014-01-02
 8 01-701-1015 BASELINE             2014-01-02 2014-01-02
 9 01-701-1015 BASELINE             2014-01-02 2014-01-02
10 01-701-1015 AMBUL ECG PLACEMENT 2014-01-14 2014-01-14
```

By default, the variable `ADTF` for `derive_vars_dt()` or `ADTF` and `ATMF` for `derive_vars_dtm()` will be created and populated with the controlled terminology outlined in the ADaM IG for date imputations.

See also Date and Time Imputation.

Once `ADT` is derived, the function `derive_vars_dy()` can be used to derive `ADY`. This example assumes both `ADT` and `TRTSDT` exist on the data frame.

```
advs <- derive_vars_dy(advs, reference_date = TRTSDT, source_vars = exprs(ADT)) # dim(advs) [1

advs %>% select(USUBJID, VISIT, ADT, ADY,TRTSDT) %>% head(n=10)
```

```
# A tibble: 10 x 5
   USUBJID     VISIT               ADT          ADY TRTSDT
   <chr>       <chr>               <date>     <dbl> <date>
 1 01-701-1015 SCREENING 1         2013-12-26    -7 2014-01-02
 2 01-701-1015 SCREENING 1         2013-12-26    -7 2014-01-02
 3 01-701-1015 SCREENING 1         2013-12-26    -7 2014-01-02
 4 01-701-1015 SCREENING 2         2013-12-31    -2 2014-01-02
 5 01-701-1015 SCREENING 2         2013-12-31    -2 2014-01-02
 6 01-701-1015 SCREENING 2         2013-12-31    -2 2014-01-02
 7 01-701-1015 BASELINE            2014-01-02     1 2014-01-02
 8 01-701-1015 BASELINE            2014-01-02     1 2014-01-02
 9 01-701-1015 BASELINE            2014-01-02     1 2014-01-02
10 01-701-1015 AMBUL ECG PLACEMENT 2014-01-14    13 2014-01-02
```

Assign parameter level values

PARAM

- Parameter name
- The description of the analysis parameter. PARAM must include all descriptive and qualifying information relevant to the analysis purpose of the parameter. Some examples are: "Supine Systolic Blood Pressure (mm Hg)", "Log10 (Weight (kg))", "Time to First Hypertension Event (Days)", and "Estimated Tumor Growth Rate". PARAM should be sufficient to describe unambiguously the contents of AVAL and/or AVALC. Examples of qualifying information that might be relevant to analysis, and are therefore candidates for inclusion in PARAM, are units, specimen type, location, position, machine type, and transformation function. There is no need to include qualifiers that are not relevant to the analysis of PARAM. In contrast to SDTM –TEST, no additional variable is needed to further qualify PARAM. PARAM is restricted to a maximum of 200 characters. If the value of PARAM will be used as a variable label in a transposed dataset, then the producer may wish to limit the value of PARAM to 40 characters. Such limitation to 40 characters should not compromise the integrity of the description. PARAM is often directly usable in Clinical Study Report displays. Note that in the ADaMIG, "parameter" is a synonym of "analysis parameter." PARAM must be present and populated on every record in a BDS dataset.

PARAMCD

- Parameter Code
- The short name of the analysis parameter in PARAM. The values of PARAMCD must be no more than 8 characters in length, start with a letter (not underscore), and be comprised only of letters (A-Z), underscore (_), and numerals (0-9). These constraints will allow for a BDS dataset to be transposed in such a way that the values of PARAMCD can be used as valid ADaM variable names per Section 3.1.1, General Variable Conventions. There must be a one-to-one relationship between PARAM and PARAMCD within a dataset. \n PARAMCD must be present and populated on every record in a BDS dataset.

PARAMN

- Parameter Number
- Numeric representation of PARAM. Useful for ordering and programmatic manipulation. There must be a one-to-one relationship between PARAM and PARAMN within a dataset for all parameters where PARAMN is populated. \n if PARAMN is populated on any record for a PARAM, it must be populated on every record for that PARAM.

PARCATy

- Parameter Category y
- A categorization of PARAM within a dataset. For example, values of PARCAT1 might group the parameters having to do with a particular questionnaire, lab specimen type, or area of investigation. Note that PARCATy is not a qualifier for PARAM. PARAM to PARCATy is a many-to-one mapping; any given PARAM may be associated with at most one level of PARCATy (e.g., one level of PARCAT1 and one level of PARCAT2).

To assign parameter level values such as `PARAMCD`, `PARAM`, `PARAMN`, `PARCAT1`, etc., a lookup can be created to join to the source data. For example, when creating `ADVS`, a lookup based on the SDTM `--TESTCD` value may be created:

```r
# Create the param_lookup data.frame
param_lookup <- data.frame(
  VSTESTCD = c("HEIGHT", "WEIGHT", "DIABP", "MAP", "PULSE", "SYSBP", "TEMP"),
  PARAMCD = c("HEIGHT", "WEIGHT", "DIABP", "MAP", "PULSE", "SYSBP", "TEMP"),
  PARAM = c("Height (cm)", "Weight (kg)", "Diastolic Blood Pressure (mmHg)",
            "Mean Arterial Pressure", "Pulse Rate (beats/min)",
            "Systolic Blood Pressure (mmHg)", "Temperature (C)"),
  PARAMN = c(1, 2, 3, 4, 5, 6, 7),
  PARCAT1 = c("Subject Characteristic", "Subject Characteristic", "Vital Sign",
              "Vital Sign", "Vital Sign", "Vital Sign", "Vital Sign"),
  PARCAT1N = c(1, 1, 2, 2, 2, 2, 2),
  stringsAsFactors = FALSE
) # dim(param_lookup) 7 6
```

This lookup may now be joined to the source data:

At this stage, only `PARAMCD` is required to perform the derivations. Additional derived parameters may be added, so only `PARAMCD` is joined to the datasets at this point. All other variables related to `PARAMCD` (e.g. `PARAM`, `PARAMCAT1`, …) will be added when all `PARAMCD` are derived.

```r
advs <- derive_vars_merged_lookup(
  advs,
  dataset_add = param_lookup,
  new_vars = exprs(PARAMCD),
  by_vars = exprs(VSTESTCD)
)
```

```
All `VSTESTCD` are mapped.
```

```
# All `VSTESTCD` are mapped.
# dim(advs) [1] 29643    31

advs %>% select(USUBJID,VSTESTCD,PARAMCD) %>% head(n=10)
```

```
# A tibble: 10 x 3
   USUBJID     VSTESTCD PARAMCD
   <chr>       <chr>    <chr>  
 1 01-701-1015 DIABP    DIABP  
 2 01-701-1015 DIABP    DIABP  
 3 01-701-1015 DIABP    DIABP  
 4 01-701-1015 DIABP    DIABP  
 5 01-701-1015 DIABP    DIABP  
 6 01-701-1015 DIABP    DIABP  
 7 01-701-1015 DIABP    DIABP  
 8 01-701-1015 DIABP    DIABP  
 9 01-701-1015 DIABP    DIABP  
10 01-701-1015 DIABP    DIABP  
```

Please note, it may be necessary to include other variables in the join. For example, perhaps the `PARAMCD` is based on `VSTESTCD` and `VSPOS`, it may be necessary to expand this lookup or create a separate look up for `PARAMCD`.

If more than one lookup table, e.g., company parameter mappings and project parameter mappings, are available, `consolidate_metadata()` can be used to consolidate these into a single lookup table.

Derive Results

AVAL

- Analysis Value
- Numeric analysis value described by PARAM. On a given record, it is permissible for AVAL, AVALC, or both to be null. AVAL is required if AVALC is not present, since either AVAL or AVALC must be present in the dataset.

AVALC

- Analysis Value (C)

- Character analysis value described by PARAM. AVALC can be a character string mapping to AVAL, but if so there must be a one-to-one relationship between AVAL and AVALC within a given PARAM. AVALC should not be used to categorize the values of AVAL. Within a given parameter, if there exists a row on which both AVALC and AVAL are populated, then there must be a one-to-one relationship between AVALC and AVAL on all rows on which both variables are populated. (In other words, there is no requirement that records with a null value in either AVAL or AVALC be included when determining whether the one-to-one relationship requirement is satisfied.) On a given record, it is permissible for AVAL, AVALC, or both to be null. \n AVALC is required if AVAL is not present, since either AVAL or AVALC must be present in the dataset.

The mapping of `AVAL` and `AVALC` is left to the ADaM programmer. An example mapping may be:

```r
advs <- mutate(
  advs,
  AVAL = VSSTRESN
) # dim(advs) [1] 29643    32

advs %>% select(VSTESTCD,PARAMCD,VSSTRESN,VSSTRESC,AVAL) %>% head(n=10)
```

```
# A tibble: 10 x 5
   VSTESTCD PARAMCD VSSTRESN VSSTRESC  AVAL
   <chr>    <chr>      <dbl> <chr>    <dbl>
 1 DIABP    DIABP         64 64          64
 2 DIABP    DIABP         83 83          83
 3 DIABP    DIABP         57 57          57
 4 DIABP    DIABP         68 68          68
 5 DIABP    DIABP         59 59          59
 6 DIABP    DIABP         71 71          71
 7 DIABP    DIABP         56 56          56
 8 DIABP    DIABP         51 51          51
 9 DIABP    DIABP         61 61          61
10 DIABP    DIABP         67 67          67
```

Derive Additional Parameters

Optionally derive new parameters creating `PARAMCD` and `AVAL`. Note that only variables specified in the `by_vars` argument will be populated in the newly created records. This is relevant to the functions `derive_param_map`, `derive_param_bsa`, `derive_param_bmi`, and `derive_param_qtc`.

Below is an example of creating `Mean Arterial Pressure` for `ADVS`, see also Example 3 in section below Derive New Rows for alternative way of creating new parameters.

AVAL when PARAMCD="MAP"

- Mean Arterial Pressure

- MAP= Diastolic BP+ (Systolic BP−Diastolic BP)/3

```
# Mean Arterial pressure (MAP) created as AVAL
advs <- derive_param_map(
  advs,
  by_vars = exprs(STUDYID, USUBJID, !!!adsl_vars, VISIT, VISITNUM, ADT, ADY, VSTPT, VSTPTNUM),
  set_values_to = exprs(PARAMCD = "MAP"),
  get_unit_expr = VSSTRESU,
  filter = VSSTAT != "NOT DONE" | is.na(VSSTAT)
) # dim(advs) [1] 37848    32

advs %>% select(VSTESTCD,PARAMCD,VISIT,VSTPT,AVAL) %>% head(n=10)
```

```
# A tibble: 10 x 5
   VSTESTCD PARAMCD VISIT                VSTPT                          AVAL
   <chr>    <chr>   <chr>                <chr>                         <dbl>
 1 DIABP    DIABP   SCREENING 1          AFTER LYING DOWN FOR 5 MINUTES    64
 2 DIABP    DIABP   SCREENING 1          AFTER STANDING FOR 1 MINUTE       83
 3 DIABP    DIABP   SCREENING 1          AFTER STANDING FOR 3 MINUTES      57
 4 DIABP    DIABP   SCREENING 2          AFTER LYING DOWN FOR 5 MINUTES    68
 5 DIABP    DIABP   SCREENING 2          AFTER STANDING FOR 1 MINUTE       59
 6 DIABP    DIABP   SCREENING 2          AFTER STANDING FOR 3 MINUTES      71
 7 DIABP    DIABP   BASELINE             AFTER LYING DOWN FOR 5 MINUTES    56
 8 DIABP    DIABP   BASELINE             AFTER STANDING FOR 1 MINUTE       51
 9 DIABP    DIABP   BASELINE             AFTER STANDING FOR 3 MINUTES      61
10 DIABP    DIABP   AMBUL ECG PLACEMENT AFTER LYING DOWN FOR 5 MINUTES    67
```

Likewise, function call below, to create parameter `Body Surface Area` (BSA) and `Body Mass Index` (BMI) for `ADVS` domain. Note that if height is collected only once use `constant_by_vars` to specify the subject-level variable to merge on. Otherwise BSA and BMI are only calculated for visits where both are collected.

```
# Add AVAL when PARAMCD=BSA
advs <- derive_param_bsa(
  advs,
  by_vars = exprs(STUDYID, USUBJID, !!!adsl_vars, VISIT, VISITNUM, ADT, ADY, VSTPT, VSTPTNUM),
  method = "Mosteller",
  set_values_to = exprs(PARAMCD = "BSA"),
  get_unit_expr = VSSTRESU,
  filter = VSSTAT != "NOT DONE" | is.na(VSSTAT),
  constant_by_vars = exprs(USUBJID)
) # dim(advs) [1] 39898    32

advs %>% select(VSTESTCD,PARAMCD,VISIT,VSTPT,AVAL) %>% tail(n=10)
```

```
# A tibble: 10 x 5
```

```
   VSTESTCD PARAMCD VISIT       VSTPT  AVAL
   <chr>    <chr>   <chr>       <chr> <dbl>
 1 <NA>     BSA     WEEK 4      <NA>   1.70
 2 <NA>     BSA     WEEK 6      <NA>   1.68
 3 <NA>     BSA     WEEK 8      <NA>   1.69
 4 <NA>     BSA     WEEK 12     <NA>   1.68
 5 <NA>     BSA     SCREENING 1 <NA>   1.49
 6 <NA>     BSA     BASELINE    <NA>   1.51
 7 <NA>     BSA     WEEK 2      <NA>   1.49
 8 <NA>     BSA     WEEK 4      <NA>   1.50
 9 <NA>     BSA     WEEK 6      <NA>   1.50
10 <NA>     BSA     WEEK 8      <NA>   1.49
```

```
# Add AVAL when PARAMCD=BMI
advs <- derive_param_bmi(
  advs,
  by_vars = exprs(STUDYID, USUBJID, !!!adsl_vars, VISIT, VISITNUM, ADT, ADY, VSTPT, VSTPTNUM),
  set_values_to = exprs(PARAMCD = "BMI"),
  get_unit_expr = VSSTRESU,
  filter = VSSTAT != "NOT DONE" | is.na(VSSTAT),
  constant_by_vars = exprs(USUBJID)
) # dim(advs) [1] 41948    32

advs %>% select(VSTESTCD,PARAMCD,VISIT,VSTPT,AVAL) %>% tail(n=10)
```

```
# A tibble: 10 x 5
   VSTESTCD PARAMCD VISIT       VSTPT  AVAL
   <chr>    <chr>   <chr>       <chr> <dbl>
 1 <NA>     BMI     WEEK 4      <NA>   28.0
 2 <NA>     BMI     WEEK 6      <NA>   27.4
 3 <NA>     BMI     WEEK 8      <NA>   27.8
 4 <NA>     BMI     WEEK 12     <NA>   27.2
 5 <NA>     BMI     SCREENING 1 <NA>   20.0
 6 <NA>     BMI     BASELINE    <NA>   20.5
 7 <NA>     BMI     WEEK 2      <NA>   20.0
 8 <NA>     BMI     WEEK 4      <NA>   20.2
 9 <NA>     BMI     WEEK 6      <NA>   20.1
10 <NA>     BMI     WEEK 8      <NA>   20.1
```

Corrected QT intervals are often abbreviated as follows, depending on the correction formula used:

- QTCBF: QT interval corrected using Bazett's Formula, with the F indicating the formula applied.
- QTCBS: QT interval corrected using Bazett's Formula, with the S indicating a study-specific threshold for QTc prolongation (e.g., thresholds defined in the protocol for safety monitoring).
- QTCL: QT interval corrected using a Linear Correction Method, typically study-specific or defined as per protocol. Sagie's (Framingham) formula might sometimes be denoted here if explicitly stated.

Similarly, for ADEG, the parameters QTCBF QTCBS and QTCL can be created with a function call. See example below for PARAMCD = QTCF.

```r
adeg <- tibble::tribble(
  ~USUBJID, ~EGSTRESU, ~PARAMCD, ~AVAL, ~VISIT,
  "P01", "msec", "QT", 350, "CYCLE 1 DAY 1",
  "P01", "msec", "QT", 370, "CYCLE 2 DAY 1",
  "P01", "msec", "RR", 842, "CYCLE 1 DAY 1",
  "P01", "msec", "RR", 710, "CYCLE 2 DAY 1"
)

adeg <- derive_param_qtc(
  adeg,
  by_vars = exprs(USUBJID, VISIT),
  method = "Fridericia",
  set_values_to = exprs(PARAMCD = "QTCFR"),
  get_unit_expr = EGSTRESU
) # dim(adeg) 6 5

adeg
```

```
# A tibble: 6 x 5
  USUBJID EGSTRESU PARAMCD  AVAL VISIT
  <chr>   <chr>    <chr>   <dbl> <chr>
1 P01     msec     QT       350  CYCLE 1 DAY 1
2 P01     msec     QT       370  CYCLE 2 DAY 1
3 P01     msec     RR       842  CYCLE 1 DAY 1
4 P01     msec     RR       710  CYCLE 2 DAY 1
5 P01     <NA>     QTCFR    371. CYCLE 1 DAY 1
6 P01     <NA>     QTCFR    415. CYCLE 2 DAY 1
```

Similarly, for ADLB, the function derive_param_wbc_abs() can be used to create new parameter for lab differentials converted to absolute values. See example below:

```r
adlb <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~AVAL, ~PARAM, ~VISIT,
  "P01", "WBC", 33, "Leukocyte Count (10^9/L)", "CYCLE 1 DAY 1",
  "P01", "WBC", 38, "Leukocyte Count (10^9/L)", "CYCLE 2 DAY 1",
  "P01", "LYMLE", 0.90, "Lymphocytes (fraction of 1)", "CYCLE 1 DAY 1",
  "P01", "LYMLE", 0.70, "Lymphocytes (fraction of 1)", "CYCLE 2 DAY 1"
)

derive_param_wbc_abs(
  dataset = adlb,
  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"
)
```

```
# A tibble: 6 x 6
  USUBJID PARAMCD  AVAL PARAM                      VISIT         DTYPE
  <chr>   <chr>   <dbl> <chr>                      <chr>         <chr>
1 P01     WBC      33   Leukocyte Count (10^9/L)   CYCLE 1 DAY 1 <NA>
2 P01     WBC      38   Leukocyte Count (10^9/L)   CYCLE 2 DAY 1 <NA>
3 P01     LYMLE     0.9 Lymphocytes (fraction of 1) CYCLE 1 DAY 1 <NA>
4 P01     LYMLE     0.7 Lymphocytes (fraction of 1) CYCLE 2 DAY 1 <NA>
5 P01     LYMPH    29.7 Lymphocytes Abs (10^9/L)   CYCLE 1 DAY 1 CALCULATION
6 P01     LYMPH    26.6 Lymphocytes Abs (10^9/L)   CYCLE 2 DAY 1 CALCULATION
```

When all PARAMCD have been derived and added to the dataset, the other information from the look-up table (PARAM, PARAMCAT1,…) should be added.

```
# Derive PARAM and PARAMN
advs <- derive_vars_merged(
  advs,
  dataset_add = select(param_lookup, -VSTESTCD),
  by_vars = exprs(PARAMCD)
) # dim(advs) [1] 41948    36

advs %>% select(VSTESTCD, PARAMCD,PARAM,PARAMN,PARCAT1,PARCAT1N) %>% head(n=10)
```

```
# A tibble: 10 x 6
   VSTESTCD PARAMCD PARAM                              PARAMN PARCAT1    PARCAT1N
   <chr>    <chr>   <chr>                               <dbl> <chr>         <dbl>
 1 DIABP    DIABP   Diastolic Blood Pressure (mmHg)         3 Vital Sign        2
 2 DIABP    DIABP   Diastolic Blood Pressure (mmHg)         3 Vital Sign        2
 3 DIABP    DIABP   Diastolic Blood Pressure (mmHg)         3 Vital Sign        2
 4 DIABP    DIABP   Diastolic Blood Pressure (mmHg)         3 Vital Sign        2
 5 DIABP    DIABP   Diastolic Blood Pressure (mmHg)         3 Vital Sign        2
 6 DIABP    DIABP   Diastolic Blood Pressure (mmHg)         3 Vital Sign        2
 7 DIABP    DIABP   Diastolic Blood Pressure (mmHg)         3 Vital Sign        2
 8 DIABP    DIABP   Diastolic Blood Pressure (mmHg)         3 Vital Sign        2
 9 DIABP    DIABP   Diastolic Blood Pressure (mmHg)         3 Vital Sign        2
10 DIABP    DIABP   Diastolic Blood Pressure (mmHg)         3 Vital Sign        2
```

Derive Timing Variables

APHASE

- Analysis Phase
- APHASE is a categorization of timing within a study, for example a higher-level categorization of APERIOD or an analysis epoch. For example, APHASE could describe spans of time for SCREENING, ON TREATMENT, and FOLLOW-UP. APHASE may be used alone or in addition to APERIOD. APHASE is independent of TRTxxP within ADSL. APHASE may be populated for spans of time where a subject is not on treatment. The value of APHASE (if populated) must be one of the values found in the ADSL APHASEw variables.

AVISIT

- Analysis Visit
- The analysis visit description; required if an analysis is done by nominal, assigned or analysis visit. AVISIT may contain the visit names as observed (i.e., from SDTM VISIT), derived visit names, time window names, conceptual descriptions (such as Average, Endpoint, etc.), or a combination of any of these. AVISIT is a derived field and does not have to map to VISIT from the SDTM. AVISIT represents the analysis visit of the record, but it does not mean that the record was analyzed. There are often multiple records for the same subject and parameter that have the same value of AVISIT. ANLzzFL and other variables may be needed to identify the records selected for any given analysis. See Section 3.3.8, Indicator Variables for BDS Datasets, for information about flag variables. AVISIT should be unique for a given analysis visit window. In the event that a record does not fall within any predefined analysis timepoint window, AVISIT can be populated in any way that the producer chooses to indicate this fact (e.g., blank or "Not Windowed"). The way that AVISIT is calculated, including the variables used in its derivation, should be indicated in the variable metadata for AVISIT. The values and the rules for deriving AVISIT may be different for different parameters within the same dataset. Values of AVISIT are producer-defined, and are often directly usable in Clinical Study Report displays. If a dataset contains more than one record per parameter per subject, then an SDTM or ADaM relative timing variable must be present (AVISIT could meet this requirement).

AVISITN

- Analysis Visit (N)

- Numeric representation of AVISIT. Since study visits are usually defined by certain timepoints, defining AVISITN so that it represents the timepoint associated with the visit can facilitate plotting and interpretation of the values. Alternatively, AVISITN may be a protocol visit number, a cycle number, an analysis visit number, or any other number logically related to AVISIT or useful for sorting that is needed for analysis. \n There must be a one-to-one relationship between AVISITN and AVISIT (i.e., AVISITN has the same value for each distinct AVISIT) within a parameter. A best practice is to extend the one-to-one relationship to within a study, but this is not an ADaM requirement. In the event that a record does not fall within any predefined analysis timepoint window, AVISITN can be populated in any way that the producer chooses to indicate this fact (e.g., may be null). Values of AVISITN are producer-defined. \n AVISITN cannot be present unless AVISIT is also present. On a given record, AVISITN cannot be populated if AVISIT is null. AVISITN can be null when AVISIT is populated, as long as the one-to-one relationship is maintained within a parameter on all rows on which both variables are populated.

APERIOD

- Analysis Period
- APERIOD is a record-level timing variable that represents the analysis period within the study associated with the record for analysis purposes. The value of APERIOD (if populated) must be one of the xx values found in the ADSL TRTxxP variable names. APERIOD is required if ASPER is present. APERIOD must be populated on all records where ASPER is populated. \n

ATPT

- Analysis Timepoint
- The analysis timepoint description; required if an analysis is done by nominal, assigned or analysis timepoint (instead of or in addition to by-visit). Timepoints are relative to ATPTREF. ATPT may contain the timepoint names as observed (i.e., from SDTM –TPT), derived timepoint names, time window names, conceptual descriptions (such as Average, Endpoint, etc.), or a combination of any of these. This variable is often used in conjunction with AVISIT. ATPT represents the analysis timepoint of the record. \n ATPT can be within an analysis visit (e.g., blood pressure assessments at 10 min, 20 min, and 30 min post-dose at AVISIT=Week 1) or can be unrelated to AVISIT (e.g., migraine symptoms 30 min, 60 min, and 120 min post-dose for attack 1). \n The way that ATPT is calculated, including the variables used in its derivation, should be indicated in the variable metadata for ATPT. The values and the rules for deriving ATPT may be different for different parameters within the same dataset. Values of ATPT are producer-defined, and are often directly usable in Clinical Study Report displays. \n If a dataset contains more than one record per parameter per subject, then an SDTM or ADaM relative timing variable must be present (ATPT could meet this requirement).

ATPTN

- Analysis Timepoint (N)
- Numeric representation of ATPT. Defining ATPTN so that its values represent the planned timepoints (e.g., minutes or hours after dosing) is not required but can facilitate plotting and interpretation of the values. There must be a one-to-one relationship between ATPTN and ATPT within a parameter. (Best practice would dictate that the mapping would be one-to-one within a study, but that is not an ADaM requirement.) \n ATPTN cannot be present unless ATPT is also present. When ATPT and ATPTN are present, then on a given record, either both must be populated or both must be null.

Categorical timing variables are protocol and analysis dependent. Below is a simple example.

```r
advs <- advs %>%
  mutate(
    AVISIT = case_when(
      str_detect(VISIT, "SCREEN") ~ NA_character_,
      str_detect(VISIT, "UNSCHED") ~ NA_character_,
      str_detect(VISIT, "RETRIEVAL") ~ NA_character_,
      str_detect(VISIT, "AMBUL") ~ NA_character_,
      !is.na(VISIT) ~ str_to_title(VISIT)
    ),
    AVISITN = as.numeric(case_when(
      VISIT == "BASELINE" ~ "0",
      str_detect(VISIT, "WEEK") ~ str_trim(str_replace(VISIT, "WEEK", ""))
    )),
    ATPT = VSTPT,
    ATPTN = VSTPTNUM
  )

count(advs, VISITNUM, VISIT, AVISITN, AVISIT)
```

```
# A tibble: 16 x 5
   VISITNUM VISIT                AVISITN AVISIT       n
      <dbl> <chr>                  <dbl> <chr>    <int>
 1      1   SCREENING 1               NA <NA>      4313
 2      2   SCREENING 2               NA <NA>      3245
 3      3   BASELINE                   0 Baseline  4048
 4      3.1 UNSCHEDULED 3.1           NA <NA>        13
 5      3.5 AMBUL ECG PLACEMENT       NA <NA>      2678
 6      4   WEEK 2                     2 Week 2    3976
 7      5   WEEK 4                     4 Week 4    3628
 8      6   AMBUL ECG REMOVAL         NA <NA>      2460
 9      7   WEEK 6                     6 Week 6    3336
10      8   WEEK 8                     8 Week 8    3020
11      9   WEEK 12                   12 Week 12   2736
12     10   WEEK 16                   16 Week 16   2351
13     11   WEEK 20                   20 Week 20   2049
14     12   WEEK 24                   24 Week 24   1852
15     13   WEEK 26                   26 Week 26   1775
16    201   RETRIEVAL                 NA <NA>       468
```

```r
count(advs, VSTPTNUM, VSTPT, ATPTN, ATPT)
```

```
# A tibble: 4 x 5
  VSTPTNUM VSTPT                         ATPTN ATPT                              n
     <dbl> <chr>                         <dbl> <chr>                         <int>
1      815 AFTER LYING DOWN FOR 5 MINUTES   815 AFTER LYING DOWN FOR 5 MI~ 10944
2      816 AFTER STANDING FOR 1 MINUTE      816 AFTER STANDING FOR 1 MINU~ 10938
```

```
3      817 AFTER STANDING FOR 3 MINUTES     817 AFTER STANDING FOR 3 MINU~ 10942
4       NA <NA>                              NA <NA>                        9124
```

For assigning visits based on time windows and deriving periods, subperiods, and phase variables see the "Visit and Period Variables" vignette.

Timing Flag Variables

ONTRTFL

- On Treatment Record Flag
- Character indicator of whether the observation occurred while the subject was on treatment. ONTRTFL is producer-defined, and its definition may vary across datasets in a study based on analysis needs.

ONTRTFN

- On Treatment Record Flag (N)
- Numeric representation of ONTRTFL. There must be a one-to-one relationship between ONTRTFN and ONTRTFL within a dataset. \n ONTRTFN cannot be present unless ONTRTFL is also present. When ONTRTFL and ONTRTFN are present, then on a given record, either both must be populated or both must be null.

In some analyses, it may be necessary to flag an observation as on-treatment. The admiral function `derive_var_ontrtfl()` can be used. For example, if on-treatment is defined as any observation between treatment start and treatment end, the flag may be derived as:

```
# Derive on-treatment flag (ONTRTFL) in an ADaM dataset with a single assessment date (e.g ADT
advs <- derive_var_ontrtfl(
  advs,
  start_date = ADT,
  ref_start_date = TRTSDT,
  ref_end_date = TRTEDT
) # dim(advs) [1] 41948    41

advs %>% select(USUBJID,PARAMCD,ADT,TRTSDT,TRTEDT,ONTRTFL) %>% head(n=10)
```

```
# A tibble: 10 x 6
   USUBJID     PARAMCD ADT        TRTSDT     TRTEDT     ONTRTFL
   <chr>       <chr>   <date>     <date>     <date>     <chr>
 1 01-701-1015 DIABP   2013-12-26 2014-01-02 2014-07-02 <NA>
 2 01-701-1015 DIABP   2013-12-26 2014-01-02 2014-07-02 <NA>
 3 01-701-1015 DIABP   2013-12-26 2014-01-02 2014-07-02 <NA>
 4 01-701-1015 DIABP   2013-12-31 2014-01-02 2014-07-02 <NA>
 5 01-701-1015 DIABP   2013-12-31 2014-01-02 2014-07-02 <NA>
 6 01-701-1015 DIABP   2013-12-31 2014-01-02 2014-07-02 <NA>
 7 01-701-1015 DIABP   2014-01-02 2014-01-02 2014-07-02 Y
 8 01-701-1015 DIABP   2014-01-02 2014-01-02 2014-07-02 Y
 9 01-701-1015 DIABP   2014-01-02 2014-01-02 2014-07-02 Y
10 01-701-1015 DIABP   2014-01-14 2014-01-02 2014-07-02 Y
```

This function returns the original data frame with the column `ONTRTFL` added. Additionally, this function does have functionality to handle a window on the `ref_end_date`. For example, if on-treatment is defined as between treatment start and treatment end plus 60 days, the call would be:

```
advs <- derive_var_ontrtfl(
  advs,
  start_date = ADT,
  ref_start_date = TRTSDT,
  ref_end_date = TRTEDT,
  ref_end_window = 60 ) # dim(advs) [1] 41948    41
```

```
Warning: Variable "ONTRTFL" already exists in the dataset.
```

In addition, the function does allow you to filter out pre-treatment observations that occurred on the start date. For example, if observations with `VSTPT == PRE` should not be considered on-treatment when the observation date falls between the treatment start and end date, the user may specify this using the `filter_pre_timepoint` parameter:

```
advs <- derive_var_ontrtfl(
  advs,
  start_date = ADT,
  ref_start_date = TRTSDT,
  ref_end_date = TRTEDT,
  filter_pre_timepoint = ATPT == "AFTER LYING DOWN FOR 5 MINUTES"
) # dim(advs) [1] 41948    41
```

```
Warning: Variable "ONTRTFL" already exists in the dataset.
```

Lastly, the function does allow you to create any on-treatment flag based on the analysis needs. For example, if variable `ONTR01FL` is needed, showing the on-treatment flag during Period 01, you need to set `new var = ONTR01FL`. In addition, for Period 01 Start Date and Period 01 End Date, you need `ref_start_date = AP01SDT` and `ref_end_date = AP01EDT`.

```
# advs <- derive_var_ontrtfl(
#   advs,
#   new_var = ONTR01FL,
#   start_date = ASTDT,
#   end_date = AENDT,
#   ref_start_date = AP01SDT,
#   ref_end_date = AP01EDT,
#   span_period = TRUE
# )
```

Assign Reference Range Indicator

ANRLO

- Analysis Normal Range Lower Limit
- Normal range lower limit for analysis; may be based on SDTM –NRLO or an imputed or assigned value.

ANRHI

- Analysis Normal Range Upper Limit
- Normal range upper limit for analysis; may be based on SDTM –NRHI or an imputed or assigned value.

ANRIND

- Analysis Reference Range Indicator
- Indicates where AVAL or AVALC falls with respect to the normal reference range for analysis; may be based on SDTM –NRIND or an imputed or assigned value. ANRIND=Low when AVAL < ANRLO; ANRIND=High when AVAL > ANRHI; ANRIND=Normal when ANRLO <=AVAL <= ANRHI

The admiral function `derive_var_anrind()` may be used to derive the reference range indicator `ANRIND`. This function requires the reference range boundaries to exist on the data frame (`ANRLO`, `ANRHI`) and also accommodates the additional boundaries `A1LO` and `A1HI`. The function is called as:

```
#advs <- derive_var_anrind(advs)
#Error in `derive_var_anrind()`:
#! Required variables `ANRLO` and `ANRHI` are missing in `dataset`
```

Derive Baseline

BASETYPE

- Baseline Type

- Producer-defined text describing the definition of baseline relevant to the value of BASE on the current record. Required when there are multiple ways that baseline is defined. If used for any PARAM within a dataset, it must be non-null for all records for that PARAM within that dataset where either BASE or BASEC are also non-null. Refer to Section 4.2.1.6, Rule 6, for an example.

ABLFL

- Baseline Record Flag
- Character indicator to identify the baseline record for each subject, parameter, and baseline type (BASETYPE) combination. See BASETYPE in Table 3.3.4.1.1. ABLFL is required if BASE is present in the dataset. \n A baseline record may be derived (e.g., it may be an average), in which case DTYPE must also be populated. If BASE is populated for a parameter, and BASE is non-null for a subject for that parameter, then there must be a record flagged by ABLFL for that subject and parameter.

ABLFN

- Baseline Record Flag (N)
- Numeric representation of ABLFL. There must be a one-to-one relationship between ABLFN and ABLFL. \n ABLFN cannot be present unless ABLFL is also present. When ABLFL and ABLFN are present, then on a given record, either both must be populated or both must be null.

BASE

- Baseline Value
- The subject's baseline analysis value for a parameter and baseline definition (i.e., BASETYPE) if present. BASE contains the value of AVAL copied from a record within the parameter on which ABLFL = "Y". Required if dataset supports analysis or review of numeric baseline value or functions of numeric baseline value. If BASE is populated for a parameter, and BASE is non-null for a subject for that parameter, then there must be a record flagged by ABLFL for that subject and parameter. Note that a baseline record may be derived (e.g., it may be an average) in which case DTYPE must be populated on the baseline record.
- **Scenario**: If a subject's weight is 70 kg at baseline, `BASE = 70` for all subsequent records for weight.

BASEC

- Baseline Value (C)

- The subject's baseline value of AVALC for a parameter and baseline definition (i.e., BASETYPE) if present. May be needed when AVALC is of interest. BASEC contains the value of AVALC copied from a record within the parameter on which ABLFL = "Y". If both AVAL and AVALC are populated within a parameter, the baseline record for AVALC must be the same record as that for AVAL. \n Within a given parameter, if there exists a row on which both BASEC and BASE are populated, then there must be a one-to-one relationship between BASEC and BASE on all rows on which both variables are populated. (In other words, there is no requirement that records with a null value in either BASE or BASEC be included when determining whether the one-to-one relationship requirement is satisfied.) On a given record, it is permissible for BASE, BASEC, or both to be null.

BNRIND

- Baseline Reference Range Indicator
- ANRIND of the baseline record identified by ABLFL.

The `BASETYPE` should be derived using the function `derive_basetype_records()`. The parameter `basetypes` of this function requires a named list of expression detailing how the `BASETYPE` should be assigned. Note, if a record falls into multiple expressions within the basetypes expression, a row will be produced for each `BASETYPE`.

```r
advs <- derive_basetype_records(
  dataset = advs,
  basetypes = exprs(
    "LAST: AFTER LYING DOWN FOR 5 MINUTES" = ATPTN == 815,
    "LAST: AFTER STANDING FOR 1 MINUTE" = ATPTN == 816,
    "LAST: AFTER STANDING FOR 3 MINUTES" = ATPTN == 817,
    "LAST" = is.na(ATPTN)
  )
) # dim(advs) [1] 41948    42

count(advs, ATPT, ATPTN, BASETYPE)
```

```
# A tibble: 4 x 4
  ATPT                            ATPTN BASETYPE                                 n
  <chr>                           <dbl> <chr>                                <int>
1 AFTER LYING DOWN FOR 5 MINUTES    815 LAST: AFTER LYING DOWN FOR 5 MINUT~  10944
2 AFTER STANDING FOR 1 MINUTE       816 LAST: AFTER STANDING FOR 1 MINUTE    10938
3 AFTER STANDING FOR 3 MINUTES      817 LAST: AFTER STANDING FOR 3 MINUTES   10942
4 <NA>                               NA LAST                                  9124
```

It is important to derive BASETYPE first so that it can be utilized in subsequent derivations. This will be important if the data frame contains multiple values for BASETYPE.

Next, the analysis baseline flag ABLFL can be derived using the {admiral} function derive_var_extreme_flag(). For example, if baseline is defined as the last non-missing AVAL prior or on TRTSDT, the function call for ABLFL would be:

```
advs <- restrict_derivation(
  advs,
  derivation = derive_var_extreme_flag,
  args = params(
    by_vars = exprs(STUDYID, USUBJID, BASETYPE, PARAMCD),
    order = exprs(ADT, ATPTN, VISITNUM),
    new_var = ABLFL,
    mode = "last")
  ,filter = (!is.na(AVAL) & ADT <= TRTSDT & !is.na(BASETYPE))
) # dim(advs) [1] 41948    43

advs %>% select(USUBJID,BASETYPE,PARAMCD,ADT,TRTSDT,ATPTN,ABLFL) %>% head()
```

```
# A tibble: 6 x 7
  USUBJID     BASETYPE PARAMCD ADT        TRTSDT     ATPTN ABLFL
  <chr>       <chr>    <chr>   <date>     <date>     <dbl> <chr>
1 01-701-1015 LAST     BMI     2013-12-26 2014-01-02    NA <NA>
2 01-701-1015 LAST     BMI     2014-01-02 2014-01-02    NA Y
3 01-701-1015 LAST     BSA     2013-12-26 2014-01-02    NA <NA>
4 01-701-1015 LAST     BSA     2014-01-02 2014-01-02    NA Y
5 01-701-1015 LAST     HEIGHT  2013-12-26 2014-01-02    NA Y
6 01-701-1015 LAST     TEMP    2013-12-26 2014-01-02    NA <NA>
```

Note: Additional examples of the `derive_var_extreme_flag()` function can be found above.

Lastly, the `BASE`, and `BNRIND` columns can be derived using the `{admiral}` function `derive_var_base()`. Example calls are:

```
advs <- derive_var_base(
  advs,
  by_vars = exprs(STUDYID, USUBJID, PARAMCD, BASETYPE),
  source_var = AVAL,
  new_var = BASE
) # dim(advs) [1] 41948    44

#advs <- derive_var_base(
#  advs,
#  by_vars = exprs(STUDYID, USUBJID, PARAMCD, BASETYPE),
#  source_var = ANRIND,
#  new_var = BNRIND
# )
# Error in `derive_var_base()`:
#! Required variable `ANRIND` is missing in `dataset`
```

Derive Change from Baseline

CHG

- Change from Baseline
- Change from baseline analysis value. Equal to `AVAL-BASE`. If used for a given PARAM, should be populated for all post-baseline records of that PARAM regardless of whether that record is used for analysis. The decision on how to populate pre-baseline and baseline values of CHG is left to producer choice.

PCHG

- Percent Change from Baseline
- Percent change from baseline analysis value. Equal to `((AVAL-BASE)/BASE)*100`. If used for a given PARAM, should be populated (when calculable) for all post-baseline records of that PARAM regardless of whether that record is used for analysis. The decision on how to populate pre-baseline and baseline values of PCHG is left to producer choice.

Change and percent change from baseline can be derived using the `{admiral}` functions `derive_var_chg()` and `derive_va`
These functions expect `AVAL` and `BASE` to exist in the data frame. The `CHG` is simply `AVAL - BASE` and the `PCHG` is `(AVAL - BASE) / absolute value (BASE) * 100`. Examples calls are:

```
advs <- derive_var_chg(advs) # dim(advs) [1] 41948    45

advs <- derive_var_pchg(advs) # dim(advs) [1] 41948    46

advs %>% select(USUBJID,VISIT,BASE,AVAL,CHG,PCHG) %>% head()
```

```
# A tibble: 6 x 6
  USUBJID     VISIT         BASE   AVAL      CHG   PCHG
  <chr>       <chr>        <dbl>  <dbl>    <dbl>  <dbl>
1 01-701-1015 SCREENING 1  25.1   24.9  -0.207   -0.827
2 01-701-1015 BASELINE     25.1   25.1   0        0
3 01-701-1015 SCREENING 1   1.49   1.49 -0.00618 -0.414
4 01-701-1015 BASELINE      1.49   1.49  0        0
5 01-701-1015 SCREENING 1 147.   147.    0        0
6 01-701-1015 SCREENING 1  36.2   36.1  -0.160   -0.442
```

If the variables should not be derived for all records, e.g., for post-baseline records only, `restrict_derivation()` can be used.

Derive Shift

SHIFTy

- Shift y
- A shift in values depending on the defined pairing for group y within a parameter. SHIFTy can only be based on the change in value of any of the following pairs (BASECATy, AVALCATy), (BNRIND, ANRIND), (ByIND, AyIND), (BTOXGR, ATOXGR), (BTOXGRL, ATOXGRL), (BTOXGRH, ATOXGRH), (BASE, AVAL) or (BASEC, AVALC). Useful for shift tables. For example, "NORMAL to HIGH". If used for a given PARAM, should be populated (when calculable) for all post-baseline records of that PARAM regardless of whether that record is used for analysis. The decision on how to populate baseline and pre-baseline values of SHIFTy is left to producer choice.
- Represents the shift or change in a categorical assessment from baseline to a post-baseline timepoint. Often expressed in the format: "Baseline → Post-baseline", such as:
  - "Normal → High"
  - "Low → Normal"
  - "High → Low"

Shift variables can be derived using the `{admiral}` function `derive_var_shift()`. This function derives a character shift variable concatenating shift in values based on a user-defined pairing, e.g., shift from baseline reference range `BNRIND` to analysis reference range `ANRIND`. Examples calls are:

```
# advs <- derive_var_shift(advs,
#   new_var = SHIFT1,
#   from_var = BNRIND,
#   to_var = ANRIND )
#Error in `derive_var_shift()`:
#! Required variables `BNRIND` and `ANRIND` are missing in `dataset`
```

If the variables should not be derived for all records, e.g., for post-baseline records only, `restrict_derivation()` can be used.

Derive Analysis Ratio

R2BASE

- Ratio to Baseline
- Ratio to the baseline value. Equal to `AVAL / BASE`. If used for a given PARAM, should be populated for all post-baseline records of that PARAM regardless of whether that record is used for analysis. The decision on how to populate pre-baseline and baseline values of R2BASE is left to producer choice.
- R2BASE= AVAL / BASE

R01ANRLO

Analysis ratio variables can be derived using the `{admiral}`function `derive_var_analysis_ratio()`. This function derives a ratio variable based on user-specified pair. For example, Ratio to Baseline is calculated by `AVAL / BASE` and the function appends a new variable `R2BASE` to the dataset. Examples calls are:

```r
advs <- derive_var_analysis_ratio(advs,
  numer_var = AVAL,
  denom_var = BASE
) # dim(advs) [1] 41948    47

# advs <- derive_var_analysis_ratio(advs,
#   numer_var = AVAL,
#   denom_var = ANRLO,
#   new_var = R01ANRLO
# )
#Error in `derive_var_analysis_ratio()`:
#! Required variable `ANRLO` is missing in `dataset`

advs %>% select(USUBJID,VISIT,BASE,AVAL,R2BASE) %>% head()
```

```
# A tibble: 6 x 5
  USUBJID     VISIT         BASE   AVAL R2BASE
  <chr>       <chr>        <dbl>  <dbl>  <dbl>
1 01-701-1015 SCREENING 1  25.1   24.9   0.992
2 01-701-1015 BASELINE     25.1   25.1   1
3 01-701-1015 SCREENING 1   1.49   1.49  0.996
4 01-701-1015 BASELINE      1.49   1.49  1
5 01-701-1015 SCREENING 1 147.   147.    1
6 01-701-1015 SCREENING 1  36.2   36.1   0.996
```

If the variables should not be derived for all records, e.g., for post-baseline records only, `restrict_derivation()` can be used.

Derive Analysis Flags

ANLzzFL

- Analysis Flag zz
- ANLzzFL is a conditionally required flag to be used in addition to other selection variables when the other selection variables in combination are insufficient to identify the exact set of records used for one or more analyses. Often one ANLzzFL will serve to support the accurate selection of records for more than one analysis. Note that it is allowable to add additional descriptive text to the label (see Section 3.1.6, Additional Information about Section 3, Item 1). \n When defining the set of records

used in a particular analysis or family of analyses, ANLzzFL is supplemental to, and is intended to be used in conjunction with, other selection variables, such as subject-level, parameter-level and record-level population flags, AVISIT, DTYPE, grouping variables such as SITEGRy, and others. The lower-case letter "zz" in the variable name is an index for the zzth record selection algorithm where "zz" is replaced with a zero-padded two-digit integer [01-99]. Every record selection algorithm "zz" (i.e., every algorithm for populating an ANLzzFL) must be defined in variable metadata. When the set of records that the algorithm "zz" operates on is pre-filtered by application of other criteria, such as a record-level population flag, then the selection algorithm definition in the metadata must so specify. \n Note that the ANLzzFL value of Y indicates that the record fulfilled the requirements of the algorithm, but does not necessarily imply that the record was actually used in one or more analyses, as whether or not a record is used also depends on the other selection variables applied. The ANLzzFL flag is useful in many circumstances; an example is when there is more than one record for an analysis timepoint within a subject and parameter, as it can be used to identify the record chosen to represent the timepoint for an analysis. "zz" is an index for a record selection algorithm, such as "record closest to target relative day for the AVISIT, with ties broken by the latest record, for each AVISIT within <list of AVISITS>." \n Note that it is not required that a specific ANLzzFL variable has the same definition across a project or even across datasets within a study. There is also no requirement that the ANLzzFL variables in a dataset or study be used in numerical order; e.g. ANL02FL might occur in a dataset or study without ANL01FL present in the same dataset or study.

ANLzzFN

- Analysis Flag zz (N)
- Numeric representation of ANLzzFL. There must be a one-to-one relationship between ANLzzFN and ANLzzFL within a dataset. \n ANLzzFN cannot be present unless ANLzzFL is also present. When ANLzzFL and ANLzzFN are present, then on a given record, either both must be populated or both must be null.

WORSTFL

- Identify the record representing the "worst" or most extreme value for a specific parameter within a defined group of records.

In most finding ADaMs, an analysis flag is derived to identify the appropriate observation(s) to use for a particular analysis when a subject has multiple observations within a particular timing period.

In this situation, an analysis flag (e.g. ANLxxFL) may be used to choose the appropriate record for analysis.

This flag may be derived using the {admiral} function derive_var_extreme_flag(). For this example, we will assume we would like to choose the latest and highest value by USUBJID, PARAMCD, AVISIT, and ATPT.

```
advs <- restrict_derivation(
   dataset = advs
  ,derivation = derive_var_extreme_flag
  ,args = params(
    by_vars = exprs(STUDYID, USUBJID, BASETYPE, PARAMCD, AVISIT)
    ,order = exprs(ADT, ATPTN, AVAL)
    ,new_var = ANL01FL
    ,mode = "last"
  ),
  filter = !is.na(AVISITN)
) # dim(advs) [1] 41948    48

advs %>% select(USUBJID,PARAMCD,AVISIT,ATPTN,ADT,AVAL,ANL01FL) %>% tail(n=11)
```

```
# A tibble: 11 x 7
   USUBJID     PARAMCD AVISIT ATPTN ADT         AVAL ANL01FL
   <chr>       <chr>   <chr>  <dbl> <date>     <dbl> <chr>
 1 01-718-1328 TEMP    <NA>      NA 2013-02-13 36.4  <NA>
 2 01-718-1328 TEMP    <NA>      NA 2013-03-09 36.7  <NA>
 3 01-718-1328 TEMP    <NA>      NA 2013-07-24 36.4  <NA>
 4 01-718-1355 TEMP    <NA>      NA 2013-03-15 36.1  <NA>
 5 01-718-1355 TEMP    <NA>      NA 2013-03-30 37.8  <NA>
 6 01-718-1371 TEMP    <NA>      NA 2013-05-07 35.8  <NA>
 7 01-718-1427 TEMP    <NA>      NA 2012-12-30 36.6  <NA>
 8 01-718-1427 TEMP    <NA>      NA 2013-01-19 35.9  <NA>
 9 01-718-1427 TEMP    <NA>      NA 2013-06-03 36.3  <NA>
10 01-701-1047 BSA     <NA>      NA 2013-03-29  1.66 <NA>
11 01-701-1047 BMI     <NA>      NA 2013-03-29 30.4  <NA>
```

Another common example would be flagging the worst value for a subject, parameter, and visit. For this example, we will assume we have 3 PARAMCD values (SYSBP, DIABP, and RESP). We will also assume high is worst for SYSBP and DIABP and low is worst for RESP.

```
advs <- slice_derivation(
  advs,
  derivation = derive_var_extreme_flag,
  args = params(
    by_vars = exprs(STUDYID, USUBJID, BASETYPE, PARAMCD, AVISIT),
    order = exprs(ADT, ATPTN),
    new_var = WORSTFL,
    mode = "first"
  ),
  derivation_slice(
    filter = PARAMCD %in% c("SYSBP", "DIABP") & (!is.na(AVISIT) & !is.na(AVAL))
  ),
  derivation_slice(
    filter = PARAMCD %in% "PULSE" & (!is.na(AVISIT) & !is.na(AVAL)),
    args = params(mode = "last")
  )
) %>%
  arrange(STUDYID, USUBJID, BASETYPE, PARAMCD, AVISIT) # dim(advs) [1] 41948    49
```

```
advs %>% select(USUBJID,PARAMCD,AVISIT,AVAL,ADT,ATPTN,WORSTFL) %>% tail()
```

```
# A tibble: 6 x 7
  USUBJID     PARAMCD AVISIT  AVAL ADT        ATPTN WORSTFL
  <chr>       <chr>   <chr>  <dbl> <date>     <dbl> <chr>
1 01-718-1427 SYSBP   Week 8   147 2013-02-18   817 Y
2 01-718-1427 SYSBP   <NA>     164 2012-12-13   817 <NA>
3 01-718-1427 SYSBP   <NA>     138 2012-12-15   817 <NA>
4 01-718-1427 SYSBP   <NA>     136 2012-12-30   817 <NA>
5 01-718-1427 SYSBP   <NA>     146 2013-01-19   817 <NA>
6 01-718-1427 SYSBP   <NA>     136 2013-06-03   817 <NA>
```

Assign Treatment

TRTA

- Actual Treatment
- TRTA is a record-level identifier that represents the actual treatment attributed to a record for analysis purposes. TRTA indicates how treatment varies by record within a subject and enables analysis of crossover and other multi-period designs. Though there is no requirement that TRTA will correspond to the TRTxxA as defined by the record's value of APERIOD, TRTA must match at least one value of the character actual treatment variables in ADSL (e.g., TRTxxA, TRTSEQA, TRxxAGy). \n As noted previously, at least one treatment variable is required. This requirement is satisfied by any subject-level or record-level treatment variables (e.g., TRTxxP, TRTP, TRTA). Even if not used for analysis, any ADSL treatment variable may be included in the BDS dataset.

TRTAN

- Actual Treatment (N)
- Numeric representation of TRTA. There must be a one-to-one relationship between TRTAN and TRTA within a study. \n TRTAN cannot be present unless TRTA is also present. When TRTA and TRTAN are present, then on a given record, either both must be populated or both must be null.

TRTP

- Planned Treatment
- TRTP is a record-level identifier that represents the planned treatment attributed to a record for analysis purposes. TRTP indicates how treatment varies by record within a subject and enables analysis of crossover and other designs. Though there is no requirement that TRTP will correspond to the TRTxxP as defined by the record's value of APERIOD, if populated, TRTP must match at least one value of the character planned treatment variables in ADSL (e.g., TRTxxP, TRTSEQP, TRxxPGy). \n As noted previously, at least one treatment variable is required even in non-randomized trials. This requirement

is satisfied by any subject-level or record-level treatment variables (e.g., TRTxxP, TRTP, TRTA). Even if not used for analysis, any ADSL treatment variable may be included in the BDS dataset.

TRTPN

- Planned Treatment (N)
- Numeric representation of TRTP. There must be a one-to-one relationship between TRTPN and TRTP within a study. \n TRTPN cannot be present unless TRTP is also present. When TRTP and TRTPN are present, then on a given record, either both must be populated or both must be null.

`TRTA` and `TRTP` must match at least one value of the character treatment variables in ADSL (e.g., `TRTxxA`/`TRTxxP`, `TRTSEQA`/`TRTSEQP`, `TRxxAGy`/`TRxxPGy`). An example of a simple implementation for a study without periods could be:

```
advs <- mutate(advs, TRTP = TRT01P, TRTA = TRT01A) # dim(advs) [1] 41948    51
count(advs, TRTP, TRTA, TRT01P, TRT01A)
```

```
# A tibble: 4 x 5
  TRTP                 TRTA                 TRT01P               TRT01A        n
  <chr>                <chr>                <chr>                <chr>     <int>
1 Placebo              Placebo              Placebo              Placebo   16018
2 Xanomeline High Dose Xanomeline High Dose Xanomeline High Dose Xanomeli~ 12156
3 Xanomeline High Dose Xanomeline Low Dose  Xanomeline High Dose Xanomeli~   747
4 Xanomeline Low Dose  Xanomeline Low Dose  Xanomeline Low Dose  Xanomeli~ 13027
```

For studies with periods see the "Visit and Period Variables" vignette.

Assign `ASEQ`

ASEQ

- Analysis Sequence Number
- Sequence number given to ensure uniqueness of subject records within an ADaM dataset. As long as values are unique within a subject within the dataset, any valid number can be used for ASEQ. ASEQ uniquely indexes records within a subject within an ADaM dataset. \n ASEQ is useful for traceability when the dataset is used as input to another ADaM dataset. To refer to a record in a predecessor ADaM dataset, set SRCDOM to the name of the predecessor dataset, and set SRCSEQ to the value of ASEQ in the predecessor dataset.
- A sequential integer starting from 1 for each subject (`USUBJID`), incremented for every new record for that subject.

The `{admiral}` function `derive_var_obs_number()` can be used to derive `ASEQ`. An example call is:

```
advs <- derive_var_obs_number(
  advs
  ,new_var = ASEQ
  ,by_vars = exprs(STUDYID, USUBJID)
  ,order = exprs(PARAMCD, ADT, AVISITN, VISITNUM, ATPTN)
  ,check_type = "error"
) # dim(advs) [1] 41948    52

advs %>% select(USUBJID,PARAMCD,ADT,AVISITN,ATPTN,VISIT,ASEQ) %>% filter(USUBJID=="01-701-1015
```

```
# A tibble: 216 x 7
   USUBJID     PARAMCD ADT        AVISITN ATPTN VISIT        ASEQ
   <chr>       <chr>   <date>       <dbl> <dbl> <chr>       <int>
 1 01-701-1015 BMI     2013-12-26      NA    NA SCREENING 1     1
 2 01-701-1015 BMI     2014-01-02       0    NA BASELINE        2
 3 01-701-1015 BMI     2014-01-16       2    NA WEEK 2          3
 4 01-701-1015 BMI     2014-01-30       4    NA WEEK 4          4
 5 01-701-1015 BMI     2014-02-12       6    NA WEEK 6          5
 6 01-701-1015 BMI     2014-03-05       8    NA WEEK 8          6
 7 01-701-1015 BMI     2014-03-26      12    NA WEEK 12         7
 8 01-701-1015 BMI     2014-05-07      16    NA WEEK 16         8
 9 01-701-1015 BMI     2014-05-21      20    NA WEEK 20         9
10 01-701-1015 BMI     2014-06-18      24    NA WEEK 24        10
# i 206 more rows
```

Derive Categorization Variables

AVALCATy

- Analysis Value Category y
- A categorization of AVAL or AVALC within a parameter. Not necessarily a one-to-one mapping to AVAL and/or AVALC. For example, if PARAM is "Headache Severity" and AVAL has values 0, 1, 2, or 3, AVALCAT1 can categorize AVAL into "None or Mild" (for AVAL 0 or 1) and "Moderate or Severe" (for AVAL 2 or 3). AVALCATy is parameter variant.
- Categorise the analysis values (`AVAL`) into predefined groups

Admiral does not currently have a generic function to aid in assigning `AVALCATx`/ `AVALCAxN` values. Below is a simple example of how these values may be assigned:

```
avalcat_lookup <- tibble::tribble(
  ~PARAMCD, ~AVALCA1N, ~AVALCAT1,
  "HEIGHT",         1, ">140 cm",
  "HEIGHT",         2, "<= 140 cm")

format_avalcat1n <- function(param, aval) {
  case_when(
    param == "HEIGHT" & aval > 140 ~ 1,
    param == "HEIGHT" & aval <= 140 ~ 2)
}

advs <- advs %>%
  mutate(AVALCA1N = format_avalcat1n(param = PARAMCD, aval = AVAL)) %>%
  derive_vars_merged(
    avalcat_lookup,
    by = exprs(PARAMCD, AVALCA1N)
  ) # dim(advs) [1] 41948    54

advs %>% select(USUBJID,PARAMCD,AVAL,AVALCA1N,AVALCAT1) %>% filter(!is.na(AVALCAT1))
```

```
# A tibble: 254 x 5
   USUBJID     PARAMCD   AVAL AVALCA1N AVALCAT1
   <chr>       <chr>    <dbl>    <dbl> <chr>
 1 01-701-1015 HEIGHT    147.        1 >140 cm
 2 01-701-1023 HEIGHT    163.        1 >140 cm
 3 01-701-1028 HEIGHT    178.        1 >140 cm
 4 01-701-1033 HEIGHT    175.        1 >140 cm
 5 01-701-1034 HEIGHT    155.        1 >140 cm
 6 01-701-1047 HEIGHT    149.        1 >140 cm
 7 01-701-1097 HEIGHT    169.        1 >140 cm
 8 01-701-1111 HEIGHT    158.        1 >140 cm
 9 01-701-1115 HEIGHT    182.        1 >140 cm
10 01-701-1118 HEIGHT    180.        1 >140 cm
# i 244 more rows
```

Add ADSL variables

If needed, the other ADSL variables can now be added. List of ADSL variables already merged held in vector adsl_vars

```
advs <- advs %>%
  derive_vars_merged(
    dataset_add = select(adsl, !!!negate_vars(adsl_vars)),
    by_vars = exprs(STUDYID, USUBJID)
  ) # dim(advs) [1] 41948    98

advs %>% select(USUBJID,RFSTDTC,RFENDTC,DTHDTC,DTHFL,AGE,AGEU) %>% head()
```

```
# A tibble: 6 x 7
  USUBJID     RFSTDTC    RFENDTC    DTHDTC DTHFL   AGE AGEU
  <chr>       <chr>      <chr>      <chr>  <chr> <dbl> <chr>
```

```
1 01-701-1015 2014-01-02 2014-07-02 <NA>   <NA>      63 YEARS
2 01-701-1015 2014-01-02 2014-07-02 <NA>   <NA>      63 YEARS
3 01-701-1015 2014-01-02 2014-07-02 <NA>   <NA>      63 YEARS
4 01-701-1015 2014-01-02 2014-07-02 <NA>   <NA>      63 YEARS
5 01-701-1015 2014-01-02 2014-07-02 <NA>   <NA>      63 YEARS
6 01-701-1015 2014-01-02 2014-07-02 <NA>   <NA>      63 YEARS
```

Derive New Rows

When deriving new rows for a data frame, it is essential the programmer takes time to insert this derivation in the correct location of the code. The location will vary depending on what previous computations should be retained on the new record and what computations must be done with the new records.

Example 1 (Creating a New Record):

To add a new record based on the selection of a certain criterion (e.g. minimum, maximum) `derive_extreme_records()` car be used. The new records include all variables of the selected records.

Adding a New Record for the Last Value

For each subject and Vital Signs parameter, add a record holding last valid observation before end of treatment. Set `AVISIT` to `"End of Treatment"` and assign a unique `AVISITN` value.

```r
advs_ex1 <- advs %>%
  derive_extreme_records(
    dataset_add = advs,
    by_vars = exprs(STUDYID, USUBJID, PARAMCD),
    order = exprs(ADT, AVISITN, ATPTN, AVAL),
    mode = "last",
    filter_add = (4 < AVISITN & AVISITN <= 12 & ANL01FL == "Y"),
    set_values_to = exprs(
      AVISIT = "End of Treatment",
      AVISITN = 99,
      DTYPE = "LOV"
    )
  ) # dim(advs_ex1) [1] 43617    99

advs_ex1 %>% select(USUBJID,PARAMCD,ADT,AVISITN,AVISIT,ATPTN,AVAL,  DTYPE,ANL01FL) %>% filter(
```

```
# A tibble: 6 x 9
  USUBJID     PARAMCD ADT        AVISITN AVISIT        ATPTN   AVAL DTYPE ANL01FL
  <chr>       <chr>   <date>       <dbl> <chr>         <dbl>  <dbl> <chr> <chr>
1 01-701-1015 BMI     2014-03-26      99 End of Trea~     NA  24.5  LOV   Y
2 01-701-1015 BSA     2014-03-26      99 End of Trea~     NA   1.47 LOV   Y
```

```
3 01-701-1015 DIABP   2014-03-26      99 End of Trea~   817  64    LOV   Y
4 01-701-1015 MAP     2014-03-26      99 End of Trea~   817  88.7  LOV   Y
5 01-701-1015 PULSE   2014-03-26      99 End of Trea~   817  52    LOV   Y
6 01-701-1015 SYSBP   2014-03-26      99 End of Trea~   817 138    LOV   Y
```

Adding a New Record for the Minimum Value

For each subject and Vital Signs parameter, add a record holding the minimum value before end of treatment. If the minimum is attained by multiple observations the first one is selected. Set `AVISIT` to `"Minimum on Treatment"` and assign a unique `AVISITN` value.

```
advs_ex1 <- advs %>%
  derive_extreme_records(
    dataset_add = advs,
    by_vars = exprs(STUDYID, USUBJID, PARAMCD),
    order = exprs(AVAL, ADT, AVISITN, ATPTN),
    mode = "first",
    filter_add = (4 < AVISITN & AVISITN <= 12 & ANL01FL == "Y" & !is.na(AVAL)),
    set_values_to = exprs(
      AVISIT = "Minimum on Treatment",
      AVISITN = 98,
      DTYPE = "MINIMUM"
    )
  ) # dim(advs_ex1) [1] 43617    99

advs_ex1 %>% select(USUBJID,PARAMCD,ADT,AVISITN,AVISIT,ATPTN,AVAL,  DTYPE,ANL01FL) %>% filter(
```

```
# A tibble: 10 x 9
   USUBJID     PARAMCD ADT        AVISITN AVISIT      ATPTN   AVAL DTYPE ANL01FL
   <chr>       <chr>   <date>       <dbl> <chr>       <dbl>  <dbl> <chr> <chr>
 1 01-701-1015 WEIGHT  2014-06-18      24 Week 24        NA  53.1  <NA>  Y
 2 01-701-1015 WEIGHT  2014-07-02      26 Week 26        NA  53.5  <NA>  Y
 3 01-701-1015 BMI     2014-02-12      98 Minimum on~    NA  24.5  MINI~ Y
 4 01-701-1015 BSA     2014-02-12      98 Minimum on~    NA   1.47 MINI~ Y
 5 01-701-1015 DIABP   2014-02-12      98 Minimum on~   815  55    MINI~ Y
 6 01-701-1015 MAP     2014-02-12      98 Minimum on~   815  86    MINI~ Y
 7 01-701-1015 PULSE   2014-03-26      98 Minimum on~   817  52    MINI~ Y
 8 01-701-1015 SYSBP   2014-02-12      98 Minimum on~   816 137    MINI~ Y
 9 01-701-1015 TEMP    2014-02-12      98 Minimum on~    NA  36.3  MINI~ Y
10 01-701-1015 WEIGHT  2014-02-12      98 Minimum on~    NA  53.1  MINI~ Y
```

Example 2 (Deriving a Summary Record)

For adding new records based on aggregating records derive_summary_records() can be used. For the new records only the variables specified by by_vars and set_values_to are populated.

For each subject, Vital Signs parameter, visit, and date add a record holding the average value for observations on that date. Set DTYPE to AVERAGE.

```r
advs_ex2 <- derive_summary_records(
  advs,
  dataset_add = advs,
  by_vars = exprs(STUDYID, USUBJID, PARAMCD, VISITNUM, ADT),
  set_values_to = exprs(
    AVAL = mean(AVAL, na.rm = TRUE),
    DTYPE = "AVERAGE")
) # dim(advs_ex2) [1] 62023    99

advs_ex2 %>% select(USUBJID,PARAMCD,VISITNUM,ADT,AVAL,DTYPE) %>% filter(USUBJID=="01-701-1015"
```

```
# A tibble: 10 x 6
   USUBJID     PARAMCD VISITNUM ADT         AVAL DTYPE
   <chr>       <chr>      <dbl> <date>     <dbl> <chr>
 1 01-701-1015 BMI            1 2013-12-26  24.9 <NA>
 2 01-701-1015 BMI            1 2013-12-26  24.9 AVERAGE
 3 01-701-1015 BMI            3 2014-01-02  25.1 <NA>
 4 01-701-1015 BMI            3 2014-01-02  25.1 AVERAGE
 5 01-701-1015 BMI            4 2014-01-16  24.5 <NA>
 6 01-701-1015 BMI            4 2014-01-16  24.5 AVERAGE
 7 01-701-1015 BMI            5 2014-01-30  24.9 <NA>
 8 01-701-1015 BMI            5 2014-01-30  24.9 AVERAGE
 9 01-701-1015 BMI            7 2014-02-12  24.5 <NA>
10 01-701-1015 BMI            7 2014-02-12  24.5 AVERAGE
```

Example 3 (Deriving a New `PARAMCD`)

Use function `derive_param_computed()` to create a new `PARAMCD`. Note that only variables specified in the `by_vars` argument will be populated in the newly created records.

Below is an example of creating `Mean Arterial Pressure` (`PARAMCD = MAP2`) with an alternative formula.

```r
# Derive (AVAL.SYSBP - AVAL.DIABP) / 3 + AVAL.DIABP as AVAL when PARAMCD="MAP2"
advs_ex3 <- derive_param_computed(
  advs,
  by_vars = exprs(USUBJID, VISIT, ATPT),
  parameters = c("SYSBP", "DIABP"),
  set_values_to = exprs(
    AVAL = (AVAL.SYSBP - AVAL.DIABP) / 3 + AVAL.DIABP,
    PARAMCD = "MAP2",
    PARAM = "Mean Arterial Pressure 2 (mmHg)"
  )
) # dim(advs_ex3) [1] 50153    98
```

```r
advs_ex3 %>% select(USUBJID,PARAMCD,ATPT,AVAL) %>% filter(USUBJID=="01-701-1015" & PARAMCD %in
```

```
# A tibble: 10 x 4
   USUBJID     PARAMCD ATPT                          AVAL
   <chr>       <chr>   <chr>                        <dbl>
 1 01-701-1015 MAP2    AFTER STANDING FOR 3 MINUTES  80
 2 01-701-1015 MAP2    AFTER STANDING FOR 3 MINUTES  79.3
 3 01-701-1015 MAP2    AFTER STANDING FOR 3 MINUTES  76.3
 4 01-701-1015 MAP2    AFTER STANDING FOR 3 MINUTES  86
 5 01-701-1015 MAP2    AFTER STANDING FOR 3 MINUTES  94
 6 01-701-1015 MAP2    AFTER STANDING FOR 3 MINUTES  88.7
 7 01-701-1015 MAP2    AFTER STANDING FOR 3 MINUTES  94.7
 8 01-701-1015 MAP2    AFTER STANDING FOR 3 MINUTES  87.3
 9 01-701-1015 MAP2    AFTER STANDING FOR 3 MINUTES  93
10 01-701-1015 MAP2    AFTER STANDING FOR 3 MINUTES  79.7
```

References

Creating a BDS Finding ADaM

The ADaM Basic Data Structure for Time-to-Event Analyses

ADaM Basic Data Structure (BDS) using PARAMCD

ADaMIG V1.3