Paper 977-2017

Deriving Rows in CDISC ADaM BDS Datasets Using SAS® Data Step Programming

Sandra Minjoe, Accenture Life Sciences

ABSTRACT

The ADaM Basic Data Structure (BDS) can be used for many analysis needs. We all know the SAS Data Step is a very flexible and powerful tool for data processing. In fact, the Data Step is very useful in the creation of a non-trivial BDS dataset.

This paper walks through a series of examples showing use of the SAS DATA STEP when deriving rows in BDS. These examples include creating new parameters, new timepoints, and changes from multiple baselines.

INTRODUCTION

The CDISC ADaM Implementation Guide version 1.1 (ADaMIG v1.1) describes the Basic Data Structure (BDS) as “one or more records per subject, per analysis parameter, per analysis timepoint” and that “analysis timepoint is conditionally required, depending on the analysis”. Additionally, a BDS record can “represent an observed, derived, or imputed value required for analysis”.

The simplest BDS analysis datasets are those that represent rows from exactly one SDTM domain dataset. Adding rows to a dataset, following the rules from ADaMIG v1.1 section 4.2, is more complex.

This paper will walk through increasingly more complicated laboratory analysis dataset needs. The code shown in these examples could be more sophisticated, but I’ve left it simple to make it readable for novice SAS programmers.

SIMPLE CASE: BDS FROM A SINGLE SDTM DOMAIN DATASET

When deriving a laboratory analysis dataset, often much of the information we need is directly available in SDTM LB variables. For example, we could concatenate the LBTEST and LBSTRESU variables together to get PARAM. However, how do we handle parameters that don’t have units? Also, for each parameter, we need to know whether we will analyze the data as numeric, in which case we’ll populate AVAL, or as character, in which case we’ll populate AVALC. Additionally, we don’t usually need to analyze all the tests that are collected in SDTM.

One way to help address these needs is to use a look-up table that compiles information from the Statistical Analysis Plan (SAP) and/or analysis table mockups, such as the following:

Table 1: Example Look-up Table

PARAMN	PARAM	PARAMCD	ATYPE	LBCAT	LBTESTCD
1	Potassium (mmol/L)	K	num	CHEMISTRY	K
2	Albumin (g/L)	ALB	num	CHEMISTRY	ALB
3	Creatinine (umol/L)	CREAT	num	CHEMISTRY	CREAT
4	Basophils (10^9/L)	BASO	num	HEMATOLOGY	BASO
5	Prothrombin Time (sec)	PT	num	HEMATOLOGY	PT
6	Color	COLOR	char	URINALYSIS	COLOR
7	pH	PH	num	URINALYSIS	PH
8	Protein	PROT	char	URINALYSIS	PROT

This type of look-up table includes all the parameters that will be analyzed (in BDS standard variables PARAMN, PARAM, PARAMCD), a variable to tell can whether to analyze as numeric or character (here it is called ATYPE, but this is not is not a BDS standard), and SDTM variables (LBCAT, LBTESTCD) that are needed for merging against the LB dataset.

USING A LOOK-UP TABLE TO DERIVE PARAMETER INFORMATION

Sorting this look-up table (here called LBLOOKUP) and merging with SDTM LB is a first step in generating an analysis lab dataset:

```
proc sort data=misc.lblookup out=lblookup;
   by LBCAT LBTESTCD;
run;

proc sort data=sdtm.lb out=lb;
   by LBCAT LBTESTCD;
run;

data lbparams;
   merge lb (in=inlb) lblookup (in=inlookup);
      by LBCAT LBTESTCD;
   if inlb and inlookup;
run;
```

Resulting dataset LBPARAMS here includes all the SDTM LB variables but only for the parameters needed for analysis. In addition, it includes from the look-up table the ADaM standard parameter variables (PARAMN, PARAM, PARAMCD) plus ATYPE for use later when creating variables such as AVAL and AVALC.

DERIVING ANALYSIS TIMING VARIABLES

Timing variables in ADaM are often created from related SDTM and some ADSL variables. In order to derive timing variables, first re-sort LBPARAMS and then merge with ADSL:

```
proc sort data=lbparams;
   by STUDYID USUBJID PARAMCD ADT;
run;

data adlb;
   merge lbparams  (in=inlb   keep=STUDYID USUBJID LBSEQ PARAMN PARAMCD PARAM
                              ATYPE LBSTRESC LBSTRESN LBSTRESU VISIT LBDTC)
         adam.adsl (in=inadsl keep=STUDYID USUBJID COUNTRY AGE AGEGR1 SEX
                              RACE ETHNIC SAFFL TRT01A TRT01AN TRTSDT);
      by STUDYID USUBJID;
```

Deriving Analysis Date and Analysis Study Day

Typically, with laboratory data, there are no partial dates. The code below simply grabs the date parts of the SDTM character datetime (LBDTC) and puts them into a SAS date variable (ADT):

```
   format ADT date9.;
   if length(LBDTC) >= 10 then
      ADT = mdy(substr(LBDTC,6,2), substr(LBDTC,9,2), substr(LBDTC,1,4));
```

If the data contained partial dates and the SAP specified an algorithm for imputing, the above code would be more complex and include date flag variable ADTF.

Next the numeric date is used to derive the analysis study day:

```
   format ADY 8.;
   if ADT > . and TRTSDT > . then do;
      if ADT < TRTSDT then ADY = ADT - TRTSDT;
      else                 ADY = ADT - TRTSDT + 1;
   end;
```

The treatment start date from ADSL was used here as the reference for analysis study day. Treatment start date is a common choice for study day, but refer to the study SAP to determine which reference date to use when deriving study day. (Note: this reference date is typically found in ADSL.)

Determining Baseline

The next step is to determine which record per subject and parameter is considered to be the baseline. The definition of baseline will be found in the study SAP. Here baseline is set to the value at analysis study day 1:

```
   format ABLFL $1.;
   if ADY = 1 then ABLFL = 'Y';
   else            ABLFL = '';
```

A baseline of the last non-missing value prior to first dose would require more complex code. In fact, code to derive this baseline would probably take place in a separate dataset and then be merged in.

Deriving Analysis Visits

Analysis visit (AVISIT) is often different than the SDTM collected visit (VISIT). The visits used for analysis and directions on how to derive them will be found in the SAP/table mockup. Below is the application of a simple windowing scheme defining Baseline; Pre-Treatment (other than Baseline); Weeks 4, 8, and 12; and Post-Week 12:

```
   format AVISITN 8.;
   format AVISIT  $15.;

   if   ABLFL='Y'      then do; AVISITN = 1;  AVISIT='Baseline';      end;
   else if  .< ADY<= 1 then do; AVISITN = 0;  AVISIT='Pre-Treatment'; end;
   else if 21<=ADY<=35 then do; AVISITN = 4;  AVISIT='Week 4';        end;
   else if 49<=ADY<=63 then do; AVISITN = 8;  AVISIT='Week 8';        end;
   else if 77<=ADY<=91 then do; AVISITN = 12; AVISIT='Week 12';       end;
   else if 92<=ADY     then do; AVISITN = 99; AVISIT='Post-Week 12';  end;
run;
```

Note that with the above code:

- A subject and parameter can have multiple rows with the same analysis visit.
- Study days 2-20, 36-48, and 64-76 will not have an assigned analysis visit.

Issues such as these, with multiple or no rows per visit, are common with derived analysis visits in BDS data. Later examples show how to flag exactly one record per visit, and also how non-flagged data can be used when creating new rows, even when not used directly for analysis.

DERIVING ANALYSIS VALUES

In BDS, the analysis value from the record flagged as baseline is often copied onto all the subsequent rows within the same subject and parameter. The code below sorts and then sets the dataset by not only study, subject, and parameter, but also the numeric version of the new analysis visit, in order to start that process:

```
proc sort data=adlb;
   by STUDYID USUBJID PARAMCD AVISITN;
run;

data adlb;
   set adlb;
     by STUDYID USUBJID PARAMCD AVISITN;
```

Next, the variables used for numeric analysis (AVAL, BASE and CHG) or character analysis (AVALC, BASEC, SHIFT1) are created, and baseline variables (BASE and BASEC) are designated as retained:

```
format AVAL BASE CHG best12.;
format AVALC BASEC   $15.;
format SHIFT1        $35.;
retain BASE BASEC;
```

For those unfamiliar with the RETAIN statement, this allows the value of a variable to be kept (retained) across rows as the dataset is developed. It's a very useful way of copying the baseline values to all later rows within the subject and parameter.

For retain to properly work, values are initialized at the start of a new subject and parameter, to prevent the baseline from a previous subject or parameter from accidentally being carried forward to a new subject/parameter. An IF statement is used to initialize the first record for the parameter within subject.

```
if first.PARAMCD then do;
   BASE  = .;
   BASEC = '';
end;
```

Now the ATYPE variable, brought in from the look-up table and which tells us whether the parameter will be analyzed as a character or numeric, is put to use. For parameters that will be analyzed as numeric, such as in a change from baseline table, variables AVAL, BASE, and CHG are created:

```
if ATYPE = 'num' then do;
   AVAL = LBSTRESN;
   if ABLFL = 'Y'           then BASE = AVAL;
   if BASE > . and AVAL > . then CHG  = AVAL - BASE;
end;
```

For parameters that will be analyzed as character, such as in a shift table, variables AVALC, BASEC, and SHIFT1 are created:

```
else if ATYPE = 'char' then do;
   AVALC = LBSTRESC;
   if ABLFL = 'Y' then BASEC = AVALC;
   if BASEC > '' and AVALC > '' then
      SHIFT1 = strip(BASEC) || ' to ' || strip(AVALC);
end;
```

CREATING THE ANALYSIS FLAG

Finally, a single record per subject, parameter, and analysis visit is selected for use on the analysis table. The SAP or table mockup will specify how to choose the appropriate record when there are multiple options within an analysis visit. Here, variable ANL01FL is used to flag the first record within an analysis visit with a non-missing value. A retained variable called FLAGGED (not an ADaM standard variable) is set to "N" at the start of each analysis visit, and updated to "Y" when this record has been found.

This is the code to create ANL01FL, making use of retained variable FLAGGED:

```
   format ANL01FL FLAGGED $1.;
   retain FLAGGED;
   if first.AVISITN then FLAGGED = 'N';
   if FLAGGED = 'N' and (AVAL > . or AVALC > '') and AVISITN in (1 4 8 12)
      then do;
      ANL01FL = 'Y';
      FLAGGED = 'Y';
   end;
run;
```

The table mockup will specify which analysis visits to include. Here ANL01FL was assigned only for analysis visits of Baseline, Week 4, Week 8, and Week 12, since other visits will not be used on the summary tables.

EXAMPLE DATASET

This is an example of a few variables within one parameter for one subject:

Table 2: Example BDS Dataset with No Derived Rows

Obs	LBSEQ	PARAMN	PARAMCD	AVISIT	LBSTRESC	AVAL	BASE	CHG	ABLFL	ANL01FL
13	36	3.0	CREAT	Pre-Treatment	83.880	83.88	.	.		
14	100	3.0	CREAT	Baseline	40.410	40.41	40.41	0.00	Y	Y
15	122	3.0	CREAT	Week 4	99.130	99.13	40.41	58.72		Y
16	177	3.0	CREAT	Week 8	83.880	83.88	40.41	43.47		Y
17	232	3.0	CREAT	Week 12	53.380	53.38	40.41	12.97		Y
18	287	3.0	CREAT	Post-Week 12	68.630	68.63	40.41	28.22		

Not all variables are shown.

FINALIZING THE BDS DATASET

This dataset is close to final. The remaining tasks, not shown, are to:

- Drop any unneeded variables.
- Rearrange variables into a useful for order for anyone doing a manual review.
- Add labels for all derived variables and for the dataset itself.

One set of variables to consider dropping are SDTM LB domain variables used to create ADaM variables but no longer needed, such as LBDTC and LBSTRESN. (Note: variable LBSTRESC is often not dropped because it can be useful for listings.) Variables that were used just to help us create ADaM variables, but not needed for analysis, such as ATYPE and FLAGGED, would also be dropped.

BDS ADDING ROWS

Section 4.2 in ADaMIG v1.1 lists many situations that require creation of rows not found in SDTM. This paper includes examples for creating a new parameter, creating a new visit, and the need for more than one baseline.

Here are a few reasons a new parameter might be needed in a lab dataset:

- A lab test needs to be reported in multiple different units. For example, both Conventional and System International (SI) units.
- The new parameter is derived from other parameters. For example, cholesterol ratio is derived from the components.
- A transposition, such as logarithm, of an existing parameter needs to be analyzed.

In each case, rows would be created for the new parameter so that it will have the appropriate baseline and other analysis values.

There are also several different reasons for creating a new row within a parameter in a lab analysis dataset, such as:

- The Average across all records within a timepoint is needed for analysis at that timepoint.
- When there is no value at a timepoint, and it needs to be imputed with logic such as using “last observation carried forward” (LOCF) or “worst observation carried forward” (WOCF).

- A new timepoint, such as last observed value, needs to be analyzed.

The examples that follow describe how to add rows when deriving a new parameter and when deriving new rows within a parameter.

COMPLEX EXAMPLE 1: CREATING A NEW PARAMETER

Here the lab dataset will be expanded from the above example to not only include Chemistry test Creatinine in the SI units of umol/L, but also the conventional units of mg/dL. The mathematics behind this conversion is to divide the results in umol/L by the conversion factor 88.4. For example, if the value is 100 umol/L, that is equivalent to 100/88.4=1.131 mg/dL.

The following new DATA STEP, added after the code above, will generate this new parameter:

```
data adlb2;
  set adlb;

  if PARAMCD = 'CREAT' then do;

     *OUTPUT RECORD IN ORIGINAL UNITS;
     output;

       * UPDATE VARIABLES AND OUTPUT NEW RECORD;
       PARAMCD = 'CREATCV';
       PARAM   = 'Creatinine (mg/dL)';
       PARAMN  = 14;
       if AVAL NE . then do;
          AVAL = AVAL / 88.4;
          if BASE NE . then do;
             BASE = BASE / 88.4;
             CHG  = AVAL - BASE;
          end;
       end;
       output;
  end;

  * OUTPUT ALL OTHER PARAMETERS;
  else output;
run;
```

A few things to notice:

- The values of PARAM, PARAMCD, and PARAMN must all be different for the new Creatinine (in units of mg/dL) than the values of PARAM, PARAMCD, and PARAMN, respectively, for the original Creatinine (in units of umol/L). The new PARAM value includes the units “mg/dL”, which naturally makes it different from the original (in units of umol/L). A new PARAMCD was created just to allow for a 1:1 correspondence, here by adding letters “CV” (an abbreviation for “Conventional”) to the end of the original PARAMCD value “CREAT”. PARAMN is for sorting the parameters, such as on an output table, so this number is determined by the parameter placement in the table mockup.
- Output statements have been added, since now the number of rows output will be potentially larger than the number of rows read in. In the code above there are three output statements:
    - The first output statement is for the original Creatinine (in units of umol/L).
    - The second output statement is for the new Creatinine (in units of mg/dL).
    - The third output statement is for all the other (non-Creatinine) parameters.

Here is an example of a few variables for one subject's Creatinine rows now in two parameters:

Table 3: Example of a New Parameter in a BDS Dataset

Obs	LBSEQ	PARAMN	PARAMCD	PARAM	AVISIT	AVAL	BASE	CHG	LBSTRESC	LBSTRESU
13	36	3.0	CREAT	Creatinine (umol/L)	Pre-Treatment	83.88	.	.	83.880	umol/L
14	100	3.0	CREAT	Creatinine (umol/L)	Baseline	40.41	40.41	0.00	40.410	umol/L
15	122	3.0	CREAT	Creatinine (umol/L)	Week 4	99.13	40.41	58.72	99.130	umol/L
16	177	3.0	CREAT	Creatinine (umol/L)	Week 8	83.88	40.41	43.47	83.880	umol/L
17	232	3.0	CREAT	Creatinine (umol/L)	Week 12	53.38	40.41	12.97	53.380	umol/L
18	287	3.0	CREAT	Creatinine (umol/L)	Post-Week 12	68.63	40.41	28.22	68.630	umol/L
49	36	14.0	CREATCV	Creatinine (mg/dL)	Pre-Treatment	0.95	.	.	83.880	umol/L
50	100	14.0	CREATCV	Creatinine (mg/dL)	Baseline	0.46	0.46	0.00	40.410	umol/L
51	122	14.0	CREATCV	Creatinine (mg/dL)	Week 4	1.12	0.46	0.66	99.130	umol/L
52	177	14.0	CREATCV	Creatinine (mg/dL)	Week 8	0.95	0.46	0.49	83.880	umol/L
53	232	14.0	CREATCV	Creatinine (mg/dL)	Week 12	0.60	0.46	0.15	53.380	umol/L
54	287	14.0	CREATCV	Creatinine (mg/dL)	Post-Week 12	0.78	0.46	0.32	68.630	umol/L

Notice that the content of the original SI Creatinine observations (13-18) is very similar to the added Conventional Creatinine observations (49-54). In fact, other than ADaM derived variables PARAM, PARAMCD, PARAMN, AVAL, BASE, and CHG, the content from the original Creatinine rows, including all the original SDTM LB variables, are unchanged. Keeping all this SDTM content, even when the ADaM values are in different units, provides useful information for a reviewer.

As described in the first example, there are a few remaining tasks to finalize this dataset:

- Drop any unneeded variables.
- Rearrange variables into a useful for order for anyone doing a manual review.
- Add labels for all derived variables and for the dataset itself.

COMPLEX EXAMPLE 2: AVERAGE VALUE WITHIN AN ANALYSIS VISIT

Sometimes when there are multiple values in a visit, the SAP might state that the average across all of the values within the visit is used for analysis. Deriving that average can be done in another DATA STEP, after the ones above.

First, the dataset is set with a BY statement. Variables to sum and count are created and retained across the visit:

```
data adlb3;
   set adlb2;
      by STUDYID USUBJID PARAMN AVISITN;

   format ASUM ACOUNT 8.2;
   retain ASUM ACOUNT;
```

Note that variables SUM and ACOUNT are not standard ADaM variables, are used within the dataset only to create the average value, and would not be kept in the final dataset.

For the first record within the visit, the values for ASUM and ACOUNT are initialized to remove any values from a prior visit:

```
if first.AVISITN then do;
   ACOUNT = 1;
   if AVAL NE . then ASUM = AVAL;
   else              ASUM = .;
end;
```

For other rows within the visit, ASUM and ACOUNT are increased when the value of AVAL is non-missing:

```
else do;
   if AVAL NE . then do;
      ASUM   = ASUM + AVAL;
      ACOUNT = ACOUNT + 1;
   end;
end;
```

Before any averages are derived, every row in the input dataset is output as-is:

```
output;
```

For the last row within the visit, the additional row containing the average is created only when:

1. the visit is being analyzed (analysis visit is non-missing),
2. there is more than one record in an analysis visit (the first visit and last visit are not the same row), and
3. there is at least one non-missing value within the visit (ASUM is non-missing)

When these conditions are met, the variables for the new record are derived and the row is output:

```
if last.AVISITN then do;
   if AVISITN NE . and not(first.AVISITN) and ASUM > . then do;
      AVAL = ASUM/ACOUNT;
      if BASE > . then CHG = AVAL - BASE;

      LBSEQ    = .;
      LBSTRESC = '';
      LBSTRESN = .;
      LBSTRESU = '';
      VISIT    = '';
      LBDTC    = '';
      ADT      = .;
      ADY      = .;
      ANL01FL  = '';
      DTYPE    = 'AVERAGE';
      output;
   end;
end;
run;
```

Some things to note with the above code before the OUTPUT statement:

- AVAL is derived from ASUM and ACOUNT.
- BASE is used with the new value of AVAL to derive CHG.

- DTYPE is set to ‘AVERAGE’ to describe how the row was created. DTYPE is required when deriving a new row within a parameter, and ‘AVERAGE’ is a valid controlled terminology value.
- All the SDTM LB and the analysis date and study day variables are all set to missing on the new row. Because multiple records were used to derive AVAL, it would be confusing to reference just the last one for this new row that contains the average value.
- ANL01FL is set to missing because the average row is not part of the first analysis described in the earlier section of this paper. Another ANLzzFL would likely be needed for this analysis, but this code was not included here.

Here is an example of some of the variables for the set of rows within one parameter and one subject, sorted by date within subject, parameter, and visit:

Table 4: Example of a New Average Row within a Parameter in a BDS Dataset

Obs		LBSEQ	PARAMN	AVISITN	AVISIT	ADT	AVAL	ABLFL	ANL01FL	DTYPE
1		39	1.0	0	Pre-Treatment	12JAN2015	4.23			
2		103	1.0	1	Baseline	13JAN2015	2.30	Y	Y	
3		125	1.0	4	Week 4	11FEB2015	5.10		Y	
4		180	1.0	8	Week 8	10MAR2015	.			
5		.	1.0	12	Week 12	.	4.90			AVERAGE
6		235	1.0	12	Week 12	07APR2015	4.30		Y	
7		0	1.0	12	Week 12	08APR2015	5.50			
8		290	1.0	99	Post-Week 12	15MAY2015	4.20			

Notice that:

- Observations 5, 6, and 7 are all for analysis visit Week 12.
- Observation 5 has a DTYPE value of ‘AVERAGE’ and contains the average value across the other two rows with analysis visit of Week 12 (observations 6 and 7).
- Observation 6 has ANL01FL = ‘Y’, and is the Week 12 record used in the first analysis summary.

As before, there are a few remaining tasks to finalize the dataset:

- Drop any unneeded variables, such as ACOUNT and ASUM.
- Rearrange variables into a useful for order for anyone doing a manual review.
- Add labels for all derived variables and for the dataset itself.

COMPLEX EXAMPLE 3: IMPUTING A VISIT WITH LOCF

When an analysis visit value is missing, the SAP might state to impute it by copying the last non-missing value from a prior record. One case of this need is when a new analysis visit is created to hold the last non-missing value across the subject. For example, visits for analysis might be Baseline, Week 4, Week 8, Week 12, and Endpoint, where Endpoint is the last non-missing value of the parameter.

Naïve BDS users might want to flag the last non-missing record in the original data rather than creating a new row. However, adding the row makes the table program much simpler, because each analysis visit, including the endpoint, is simply a selection of rows. (Using a flag instead of adding the row would require separate programming in the table program to handle this last visit.)

The first step in generating this new visit is to re-sort the data by date within parameter:

```
proc sort data=adlb3;
   by STUDYID USUBJID PARAMN ADT;
run;
```

The next step is to create a dataset with only the last non-missing (`AVAL NE .`) collected value (`LBSEQ NE .`) within the parameter:

```
data adlblast;
   set adlb3 (where = (AVAL NE . and LBSEQ NE .));
      by STUDYID USUBJID PARAMN ADT;

   if last.PARAMN then do;
      AVISIT  = 'Endpoint';
      AVISITN = 999;
      DTYPE   = 'LOCF';
      ANL01FL = 'Y';
      output;
   end;
run;
```

Some things to note with the above code:

- Because only records with non-missing values are kept, and the only OUTPUT statement is for last row within the parameter, the resulting dataset will have no more than one record per parameter per subject.
- The analysis visit information was updated to give it the new 'Endpoint' name and a high value of AVISITN so that it'll sort to the end of the all the analysis visits.
- DTYPE is set to 'LOCF' (Last Observation Carried Forward) to describe how the row was created. DTYPE is required when deriving a new row within a parameter, and 'LOCF' is a valid controlled terminology value.
- ANL01FL is set to 'Y' so that this new Endpoint row will be part of the table output shown in the first analysis.
- None of the SDTM variables were modified before output. This provides traceability from the new Endpoint row back to the SDTM input data.

This dataset, containing just the last visit per subject and parameter, is then re-sorted and interleaved with the rest of the analysis data:

```
proc sort data=adlb3;
   by STUDYID USUBJID PARAMN AVISITN;
run;

data adlb4;
   set adlb3 adlblast;
      by STUDYID USUBJID PARAMN AVISITN;
run;
```

Here is an example of one subject and one parameter with the new Endpoint row added:

Table 5: Example of a New Endpoint Row within a Parameter in a BDS Dataset

Obs	LBSEQ	PARAM	AVISITN	AVISIT	ADT	AVAL	ABLFL	ANL01FL	DTYPE
46	53	pH	0	Pre-Treatment	12JAN2015	7.40			
47	85	pH	1	Baseline	13JAN2015	5.60	Y	Y	
48	162	pH	4	Week 4	11FEB2015	9.00		Y	
49	217	pH	8	Week 8	10MAR2015	7.40		Y	
50	272	pH	12	Week 12	07APR2015	7.00		Y	
51	327	pH	99	Post-Week 12	15MAY2015	6.60			
52	327	pH	999	Endpoint	15MAY2015	6.60		Y	LOCF

Notice that the Endpoint analysis visit in observation 52 is a copy of the Post-Week 12 analysis visit in observation 51. One difference is that observation 51 has a missing value for ANL01FL, meaning it was not included in the analysis, but observation 52 has a value of 'Y'. This is an example where a value that was not otherwise used on an analysis table is now being used in the LOCF analysis.

As before, there are a few remaining tasks to finalize the dataset:

- Drop any unneeded variables.
- Rearrange variables into a useful for order for anyone doing a manual review.
- Add labels for all derived variables and for the dataset itself.

MULTIPLE BASELINES

There are often multiple records that occur prior to dosing. In fact, in **Table 4** above there is a Pre-Treatment record and a Baseline record that each occur prior to dosing. In the prior analysis examples, the record at analysis day 1 was used as baseline. However, it is not uncommon for statisticians to use more than one derivation for baseline.

The ADaMIG section 4.2 rule 6 states "When there is more than one definition of baseline, each additional definition of baseline requires the creation of its own set of rows." This is because each row will need to contain analysis variables such as CHG that use the appropriate baseline in the derivation, and only one instance of each of the analysis variables is allowed on the dataset. (In other words, a BDS structure can't have variables such as BASE1, BASE2, CHG1, CHG2, etc.)

There are two different options to handle multiple baselines: creating multiple rows within the same dataset, or creating multiple datasets. In both cases, the examples shown below will create a second baseline defined as the average across all pre-treatment records (records with ADY <=1).

COMPLEX EXAMPLE 4: MULTIPLE BASELINES IN ONE DATASET

One baseline result was already created above, with ABLFL = 'Y' and the value copied to BASE for all rows after that baseline. In order to create a second baseline within the same dataset, this time as the average across all pre-treatment records, the following steps have to happen:

1. Create the second baseline record, derived as average.
2. Flag the second baseline record with ABLFL = 'Y'.
3. Create a new set of post-baseline records that use the second baseline.
4. Add BASETYPE to each record with a BASE value, showing which baseline was used.

Here is that code, step by step.

First, bring in the original data, add the BASETYPE variable, and output the record:

```
data adlb5;
   set adlb4;
      by STUDYID USUBJID PARAMN AVISITN;

   format BASETYPE $8.;

   * OUTPUT ORIGINAL DATA;
   If BASE > . then BASETYPE = 'DAY 1';
   output;
```

Each of these output rows represents the data as it was prior to creating a second baseline. The BASETYPE variable is added to every row with a non-missing baseline value so that it can be distinguished from the new records created below. BASETYPE does not have controlled terminology, and the content "DAY 1" is just a short description of how the baseline was determined.

New variables to help create baseline are then added and initialized:

```
format ASUM ACOUNT NEWBASE 8.2;
retain ASUM ACOUNT NEWBASE;

if first.PARAMN then do;
   ACOUNT = 1;
   if AVAL NE . then ASUM = AVAL;
   else              ASUM = .;
   NEWBASE = .;
end;
```

Notice that:

- Other than variable NEWBASE, the above code is exactly the same as what was included in **Complex Example 2** on pages 7-8.
- The code for NEWBASE is the same as the code for BASE in the **Simple Case Example** on page 3, in the section “Deriving Analysis Values”.

Similar to **Complex Example 2**, ASUM and ACOUNT are then increased when the record is pre-treatment (ADY <=1) and AVAL is non-missing:

```
if ADY <= 1 and AVAL NE . then do;
   ASUM   = ASUM + AVAL;
   ACOUNT = ACOUNT + 1;
end;
```

Next, similar to **Complex Example 2**, the new average baseline record is created:

```
if ADY = 1 and ASUM > . and ACOUNT > . then do;
   AVAL    = ASUM/ACOUNT;
   BASE    = AVAL;
   NEWBASE = BASE;

   LBSEQ    = .;
   LBSTRESC = '';
   LBSTRESN = .;
   LBSTRESU = '';
   VISIT    = '';
   LBDTC    = '';
   ADT      = .;
   ADY      = .;

   ANL01FL  = '';
   ANL02FL  = 'Y';
   ABLFL    = 'Y';
   DTYPE    = 'AVERAGE';
   BASETYPE = 'AVERAGE';
   output;
end;
```

Some things to note with the above code before the OUTPUT statement:

- AVAL is derived from ASUM and ACOUNT.
- BASE is used with the new value of AVAL to derive CHG.
- NEWBASE is assigned so it can be retained for post-baseline rows.

- All the SDTM LB and the analysis date and study day variables are set to missing. Because multiple records were used to derive AVAL, it would be confusing to reference just the last one for this new row that contains the average value.
- ANL01FL is set to missing and ANL02FL is set to 'Y' to denote that these rows are used for the second, not the first, analysis.
- DTYPE is set to 'AVERAGE' to describe how the row was created. DTYPE is required when deriving a new row within a parameter, and 'AVERAGE' is a valid controlled terminology value.
- BASETYPE is set to 'AVERAGE' to describe how the baseline was determined. There is no controlled terminology for BASETYPE, and the content 'AVERAGE' is just a short description of how the baseline was derived.

Finally, the post-baseline rows are created, using the new baseline value:

```
   if (ADY > 1 or AVISITIN > 1) and AVAL > . then do;
      if NEWBASE > . then do;
         BASE = NEWBASE;
         CHG  = AVAL - BASE;
      end;
      BASETYPE = 'AVERAGE';
      if ANL01FL = 'Y' then do;
         ANL01FL = '';
         ANL02FL = 'Y';
      end;
      output;
   end;
run;
```

Notice that:

- BASE is a copy of retained variable NEWBASE and used to create CHG.
- BASETYPE is set to 'AVERAGE' to denote which baseline was used.
- Using the content of ANL01FL to derive ANL02FL allows us to flag the same rows for analysis in this second set of data.

The resulting dataset has quite a few more rows than before:

Table 6: Example of an Analysis Dataset with Two Baselines

Obs	LBSEQ	PARAM	AVISIT	ADY	BASETYPE	AVAL	BASE	CHG	ABLFL	ANL01FL	ANL02FL	DTYPE
75	53	pH	Pre-Treatment	-1		7.40	.	.				
76	85	pH	Baseline	1	DAY 1	5.60	5.60	0.00	Y	Y		
77	162	pH	Week 4	30	DAY 1	9.00	5.60	3.40		Y		
78	217	pH	Week 8	57	DAY 1	7.40	5.60	1.80		Y		
79	272	pH	Week 12	85	DAY 1	7.00	5.60	1.40		Y		
80	327	pH	Post-Week 12	123	DAY 1	6.60	5.60	1.00				
81	327	pH	Endpoint	123	DAY 1	6.60	5.60	1.00		Y		LOCF
82	.	pH	Baseline	.	AVERAGE	6.50	6.50	0.00	Y		Y	AVERAGE
83	162	pH	Week 4	30	AVERAGE	9.00	6.50	2.50			Y	
84	217	pH	Week 8	57	AVERAGE	7.40	6.50	0.90			Y	
85	272	pH	Week 12	85	AVERAGE	7.00	6.50	0.50			Y	
86	327	pH	Post-Week 12	123	AVERAGE	6.60	6.50	0.10				
87	327	pH	Endpoint	123	AVERAGE	6.60	6.50	0.10			Y	LOCF

Some things to note with the above table:

- There are two records with ABLFL = 'Y'

- Each row with a value of BASE also has a BASETYPE value to describe which baseline was used.
- Observations 75-81 use the original baseline and are unchanged from the prior dataset, other than the addition of the new variable BASETYPE = 'DAY 1' on all rows with a non-missing baseline value.
- Observations 82-86 are the new rows:
    - Observation 82 is the new baseline, where AVAL is the average of the values from observations 75 and 76. This value of AVAL, 6.5, is used as the baseline value on all rows following it, within the same subject and parameter.
    - Observations 83-87 are the new post-baseline records. Notice that the LBSEQ values match records 77-81. In fact, all content other than BASE, CHG, BASETYPE, and the analysis flags are exactly the same between the two sets of records (77-81 and 83-87).

As before, there are a few remaining tasks to finalize the dataset:

- Drop any unneeded variables, such as ASUM, ACOUNT, and NEWBASE.
- Rearrange variables into a useful for order for anyone doing a manual review.
- Add labels for all derived variables and for the dataset itself.

COMPLEX EXAMPLE 5: ALTERNATE BASELINES IN DIFFERENT DATASETS

In the prior example, adding new rows for each additional baseline can make the dataset much larger. It can also add confusion when a reviewer is unfamiliar with the use of BASETYPE.

Another option is to create separate datasets, one for each baseline. With this option, the original dataset, such as shown in **Table 5**, would be left as is, and a second dataset would be created with the other (in this case "average") baseline. In this additional dataset, a new analysis baseline needs to be derived as the average of all pre-treatment records, and then applied to the post-baseline records. Because each dataset would contain exactly one definition of baseline, variable BASETYPE is not needed.

Similar to **Complex Example 4** above, a new baseline for the new dataset needs to be derived. Here temporary variables ASUM, ACOUNT, and NEWBASE are created, retained, and initialized:

```
data adlbavg (drop = ASUM ACOUNT);
   set adlb4 (where =(ATYPE = 'num'));
      by STUDYID USUBJID PARAMN;

   format ASUM ACOUNT NEWBASE 8.2;
   retain ASUM ACOUNT NEWBASE;

   if first.PARAMN then do;
      ACOUNT = 1;
      if AVAL NE . then ASUM = AVAL;
      else              ASUM = .;
      NEWBASE = .;
   end;
```

Next, ASUM and ACOUNT are increased when the record is pre-treatment (ADY <=1) and AVAL is non-missing. Also, the analysis baseline flag from the original baseline is removed:

```
   else if ADY <= 1 and AVAL NE . then do;
      ASUM   = ASUM + AVAL;
      ACOUNT = ACOUNT + 1;
      ABLFL  = '';
   end
```

All pre-baseline records are then output unchanged:

```
   if ADY < 1 then output;
```

The former baseline record is modified to remove the baseline information, then output:

```
   else if ADY = 1 then do;
      if ASUM = . or ACOUNT = . then output;

      else do;
         ABLFL   = '';
         ANL01FL = '';
         BASE    = .;
         CHG     = .;
         output;
```

Similar to **Complex Example 4** above, the new average baseline is derived from ASUM and ACOUNT, retained NEWBASE is assigned, all the SDTM LB and the analysis date and study day variables are set to missing, DTYPE is assigned to 'AVERAGE', and the new record is output:

```
         AVAL     = ASUM/ACOUNT;
         BASE     = AVAL;
         NEWBASE  = BASE;

         LBSEQ    = .;
         LBSTRESC = '';
         LBSTRESN = .;
         LBSTRESU = '';
         VISIT    = '';
         LBDTC    = '';
         ADT      = .;
         ADY      = .;

         ANL01FL  = 'Y';
         ABLFL    = 'Y';
         DTYPE    = 'AVERAGE';
         output;
      end;
   end;
```

Finally, the post-baseline records are modified to replace BASE with NEWBASE, CHG is re-derived using the new baseline, and those records are output:

```
   else if ADY > 1 then do;
      if NEWBASE > . then do;
         BASE = NEWBASE;
         if AVAL > . then CHG = AVAL - BASE;
         output;
      end;
   end;
run;
```

Here is an example of the same subject and parameter as **Table 5**, but using the derived average baseline:

Table 7: Example of a Second Analysis Dataset with an Average Baseline

Obs	LBSEQ	PARAM	AVISIT	ADY	AVAL	BASE	CHG	ABLFL	ANL01FL	DTYPE
45	53	pH	Pre-Treatment	-1	7.40	.	.			
46	85	pH	Baseline	1	5.60	.	.			
47	.	pH	Baseline	.	6.50	6.50	.	Y	Y	AVERAGE
48	162	pH	Week 4	30	9.00	6.50	2.50		Y	
49	217	pH	Week 8	57	7.40	6.50	0.90		Y	
50	272	pH	Week 12	85	7.00	6.50	0.50		Y	
51	327	pH	Post-Week 12	123	6.60	6.50	0.10			
52	327	pH	Endpoint	123	6.60	6.50	0.10		Y	LOCF

Comparing this output to the prior two tables:

- In **Table 5**, observation 46 was the baseline record, with a value of 5.60. In **Table 7**, the new record, observation 47, averages the values of the 2 prior observations to get a derived average baseline of 6.50. This new record is flagged as baseline (ABLFL = 'Y'), and the baseline value of 6.50 is copied to all the post-baseline records and used to derive CHG.
- In **Table 7**, observations 45 and 46 are used only to create observation 47. These records correspond to observations 75 and 76 in **Table 6**: each pair of records references LBSEQ of 53 and 85.
- In **Table 7**, observations 48-52 correspond to observations 83-87 in **Table 6**.

As before, there are a few remaining tasks to finalize the dataset:

- Drop any unneeded variables.
- Rearrange variables into a useful for order for anyone doing a manual review.
- Add labels for all derived variables and for the dataset itself. Note that each of the two datasets described here in **Complex Example 5** would be used for a specific analysis table or set of analysis tables. To clarify which dataset is used for what analysis, each dataset would need different dataset names and dataset labels, such as:

Dataset Name	Dataset Label
ADLB	Laboratory Analysis Dataset Base=Last
ADLBAVG	Laboratory Analysis Dataset Base=Average

METADATA FOR EACH OF THE TWO BASELINE OPTIONS

Anytime more than one baseline is needed, there is potential for confusion. The best way to reduce confusion is to include documentation to explain why and how each baseline was derived and in which analyses each was used. In **Complex Example 4**, where all the records are in one dataset, much of this explanation would be in metadata at the variable-level, specifically for variables BASETYPE and ANLzzFL. In **Complex Example 5**, where each baseline is in its own dataset, it is instead described in dataset-level metadata. In both cases, additional clarity can come from results-level metadata and an explanation in the Analysis Data Reviewers Guide (ADRG).

Which of the two methods to use is the creator's choice. Both are allowed in BDS.

CONCLUSION

Creating BDS datasets, including adding rows, is not difficult using SAS. All examples in this paper used BASE SAS, specifically the DATA STEP and PROC SORT, to create various versions of an ADaM BDS

analysis dataset. Examples included adding a new parameter, a new timepoint within a parameter, and two different ways to derive a second baseline.

REFERENCES

Clinical Data Interchange Standards Consortium. 2017. “Analysis Data Model (ADaM).” Accessed January 30, 2017. https://www.cdisc.org/standards/foundational/adam.

PhUSE wiki. 2017. “Optimizing the Use of Data Standards.” Accessed January 30, 2017. http://www.phusewiki.org/wiki/index.php?title=Optimizing_the_Use_of_Data_Standards.

- This site contains the ADRG package.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Sandra Minjoe
Accenture Life Sciences
sandra.minjoe@Accenture.com