Paper 14-2018

Creating BDS DERIVED Parameters for a Subject-level Frequency Summary Table? Then this macro can be useful

Masaki Mihaila, Pfizer Inc

ABSTRACT

A BDS (Basic Data Structure)-derived subject level frequency table is a core task for SAS programmers. New DERIVED parameters for the event definitions are needed within BDS ADaM datasets. The consistency of the table structure suggests the use of SAS macro functionality to reduce errors and streamline the process. This paper demonstrates a macro to use original parameters from a BDS ADaM database to generate a new Y/N analysis variable to determine whether a subject experiences a defined event during the clinical trial. This paper will use a “liver function test” mock-up as an example. It does not review basic BDS concepts; readers should have basic knowledge of BDS.

INTRODUCTION

Data manipulation is an essential yet often time-consuming process for SAS programmers. Therefore, SAS programmers should take advantage of every opportunity to automate such tasks. In this case, the purpose is to produce subject-level, BDS-derived event summary tables. These tables require only one new analysis value per subject for each defined event. Events are summarized as a binary event. Each qualifying subject must have all the records for an accurate percentage calculation.

The example (see Table 1) uses a liver function test summary table mock-up and Lab Analysis Dataset (ADLB) as BDS input database. Obviously it is not finalized ADLB. The analysis baseline flag (ABLFL) must be defined before this macro is called. Table 1 displays the sample mock-up shell. The macro is written in PROC SQL which utilizes the basic aggregate SUM function inside CASE/WHEN syntax with a GROUP BY clause.

This paper explains two different methods (macros) to derive criteria ① and ②, highlighted in two colors (the details are noted in the Table 1 caption).

METHODOLOGY

The two different methods (macros) to derive ① and ② criteria are as follows:

① BASELINE, ALT or AST ≥ 3xULN: At baseline, did a subject experience AST (Aspartate Aminotransferase) and/or ALT (Alanine Aminotransferase) chemistry panels larger than or equal to three times its upper limit?

Examining the PROC SQL code shows the criterion ① fits inside the SUM function within the CASE/WHEN syntax (see the bolded lines). If the SUM function result is not 0, the subject experienced the event at least once, and AVALC=Y. With GROUP BY option along with UNIQUE option, the output dataset will contain a new created parameter (ALSGE3BL) record for each qualified subject. Users must make sure users' INPUT DATA includes all the subjects from the chosen population group in ADSL (Subject-Level Analysis Dataset).

```
Proc sql;
        create table [NAME OF YOUR OUTPUT DATA] as
        select unique trtp
                ,usubjid
                ,’DERIVED’ as paramtyp
                ,’ALSGE3BL’ as paramcd
                ,’Elevated ALT or AST at baseline’ as param
                ,111 as paramn
```

```
                        ,case
                                when sum (paramcd in ('AST' 'ALT') and ablfl='Y' and
                                        . < aval >= 3*input (anrhi, best.)) > 0 then 'Y'
                                else 'N' end as avalc
                        ,ifn (calculated avalc='Y', 1, 0) as aval
                from [NAME OF INPUT DATA] (where= (dtype is NULL))
                group by usubjid;
        Quit;
```

② <u>POSTBASELINE, ALT or AST ≥ 3xULN and TBL > 2xULN (any visit date):</u> At any post baseline visit (i.e., they don't need to be the same visits), did a subject experience an AST and/or ALT chemistry panel larger than or equal to three times its upper limit AND was the TBL (Total Bilirubin) chemistry panel larger than or equal to two times its upper limit?

The PROC SQL code looks similar to example ① except two SUM functions are used in the WHEN clause to include both AST/ALT and TBL conditions (see the bolded lines).

```
        Proc sql;
                create table [NAME OF YOUR OUTPUT DATA] as
                select unique trtp
                        ,usubjid
                        ,'DERIVED' as paramtyp
                        ,'AT3BL2PB' as paramcd
                        ,'Elevated ALT or AST and BILI at pbl records' as param
                        ,112 as paramn
                        ,case
                                when sum (paramcd in ('AST' 'ALT') and trtelbfl='Y' and
                                        . < aval >= 3*input (anrhi, best.)) > 0
                                        and sum (paramcd='BILI' and trtelbfl='Y' and
                                        . < aval > 2*input (anrhi, best.)) > 0
                                  then 'Y'
                                else 'N' end as avalc
                        ,ifn (calculated avalc='Y', 1, 0) as aval
                from [NAME OF INPUT DATA] (where= (dtype is NULL))
                group by usubjid;
        Quit;
```

The basic concept for the PROC SQL statement is identical for both ① and ②. Speaking of generalization, this same structure can be controlled by a macro *%do-loop* to iterate and execute the code until all the desired conditions are all achieved. Inside the loop, the macro parameters such as condlist (for conditions), cdlist (for PARAMCD – the short name of the analysis parameter in PARAM), cdnlist (for PARAMN – ordering numbers with one-to-one mapping with PARAM), and nlist (for PARAM – the description of the analysis parameter) are defined as lists by separating each set with your delimiter. The macro runs each PROC SQL statement in an implicit *%do loop* defined by 1 to the number obtained from counting function, countw in the sysfunc macro function which turns into the macro variable (see † below in the macro code). For each individual iteration a unique set of values are called in from the macro parameters. The values from iterative parameters are distinguished by a delimiter (&delim), which is carefully selected by users so as to avoid conflict with any macro text strings. See the below to examine the macro that the author created.

```
%let delim = /* [YOUR DELIMITER] */;
%macro [NAME OF MACRO] (
         Indta             = /* [NAME OF YOUR INPUT DATA] */
        ,outdta            = /* [NAME OF YOUR OUTPUT DATA] */
        ,condlist         = %str (/* [CONDITION1&delim.CONDITION2&delim....] */)
        ,cdlist           = /* [CODE1&delim.CODE2&delim2....] */
        ,cdnlist= /* [CODE NUMBER1&delim.CODE NUMBER2&delim...] */
        ,nlist            = %str (/* [PARAMETER NAME1&delim.PARAMETER NAME2&delim..] */)
        );

        %do i = 1 %to %sysfunc (countw (&cdlist.)); †
                proc sql;
                        create table [YOUR TEMPORARY DATA] as
                        select unique trtp
                                ,usubjid
                                ,'DERIVED' as paramtyp
                                ,"%scan (&cdlist., &i., &delim.)" as paramcd
                                ,"%scan (&nlist., &i., &delim.)" as param length=150
                                ,%scan (&cdnlist., &i., &delim.) as paramn
                                ,case
                                        when %unquote (%qscan (&condlist., &i., &delim.)) then 'Y'
                                        else 'N' end as avalc
                                ,ifn (calculated avalc = 'Y', 1, 0) as aval
                        from &indta. (where = (dtype is NULL))
                        group by usubjid;
                quit;

                %* Append data to create &outdta. to include all the DERIVED records;
                Proc append base=&outdta. data= [YOUR TEMPORARY DATA]; run; quit;
        %end;
%mend;
```

The author uses countw counting function since PARAMETER CODE (PARAMCD) acts like words so countw function can dynamically count how many loops needs to happen. The index *i* is the iteration number of the loop and this 'do-iterative' will loop proc sql statement taking each *i*-th values in condlist, cdlist, cdnlist, and nlist parameters by using macro scan function (either %scan or %qscan). Scan functions are used to extract substrings from words when the relative order of words is known and words are delimited by the character in delimiter. Notice that %UNQUOTE function is invoked to unmask the masked characters inside the condlist macro expression.

See the below to examine how the two new DERIVED parameters for criteria ① and ②are created with this macro.

```
% [NAME OF THE MACRO] (
         Indta  = [NAME OF YOUR INPUT DATA]
        ,outdta= [NAME OF YOUR OUTPUT DATA]
        ,condlist= %str (sum (paramcd in ('AST' 'ALT') and ablfl='Y' and
                                . <aval>=3*input (anrhi, best.)) > 0
                        &delim.
                         sum (paramcd in ('AST' 'ALT') and trtelbfl='Y' and
                                . <aval>=3*input (anrhi, best.)) > 0
                                and sum (paramcd='BILI' and trtelbfl='Y' and
                                . <aval>2*input (anrhi, best.)) > 0)
        ,cdlist  = ALSGE3BL&delim.AT3BL2PB
        ,cdnlist= 111&delim.112
        ,nlist   = %str (Elevated Total Bilirubin at baseline
                        &delim.Elevated ALT or AST and BILI at pbl records)
)
```

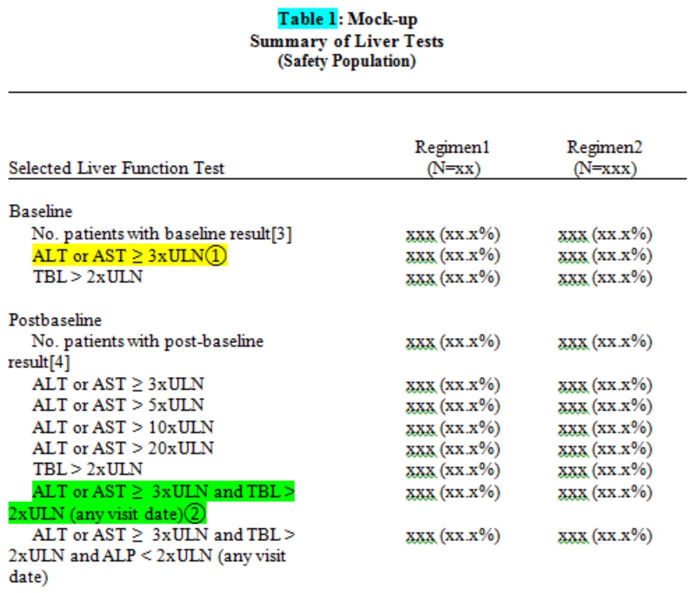

Table 1: Mock-up
Summary of Liver Tests
(Safety Population)

Selected Liver Function Test	Regimen1 (N=xx)	Regimen2 (N=xxx)
Baseline		
No. patients with baseline result[3]	xxx (xx.x%)	xxx (xx.x%)
ALT or AST ≥ 3xULN①	xxx (xx.x%)	xxx (xx.x%)
TBL > 2xULN	xxx (xx.x%)	xxx (xx.x%)
Postbaseline		
No. patients with post-baseline result[4]	xxx (xx.x%)	xxx (xx.x%)
ALT or AST ≥ 3xULN	xxx (xx.x%)	xxx (xx.x%)
ALT or AST > 5xULN	xxx (xx.x%)	xxx (xx.x%)
ALT or AST > 10xULN	xxx (xx.x%)	xxx (xx.x%)
ALT or AST > 20xULN	xxx (xx.x%)	xxx (xx.x%)
TBL > 2xULN	xxx (xx.x%)	xxx (xx.x%)
ALT or AST ≥ 3xULN and TBL > 2xULN (any visit date)②	xxx (xx.x%)	xxx (xx.x%)
ALT or AST ≥ 3xULN and TBL > 2xULN and ALP < 2xULN (any visit date)	xxx (xx.x%)	xxx (xx.x%)

Table 1 : From this example shell, the author selects two criteria for demonstration: ①-highlighted in yellow [ALT or AST is ≥ 3*ULN at baseline] and ②-highlighted in green [at any post-baseline visit ALT or AST ≥ 3*ULN and TBL > 2*ULN. These two conditions do not have to happen at the same visit.]

The appeal of this macro tool is its simplicity and flexibility. If more DERIVED parameters are added, the parameters merely need to be concatenated with the delimiter *&delim*.

CONCLUSION

Every so often, programmers encounter projects where macro tools can streamline the programming to improve efficiency. This paper demonstrates the use of a reusable macro to create new needed parameters derived for subject-level event summary table within BDS ADaM dataset. As more analysis needed, the macro input can be adjusted by concatenating a set of information repeatedly, (i.e., defined requirement (conditions that go into SUM function), paramcd, param, and paramn in the same order). This approach is not limited to only the SUM function, other aggregate functions within the GROUP BY clause such as MEAN, COUNT, MIN and MAX can easily be used. The author is aware of one limitation: since this macro aggregates all row observations, the macro is not applicable in the cases where multiple events of interests need to be compared within the same subject by visits or dates.

CONTACT INFORMATION

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

Masaki Mihaila
Pfizer, Inc.
Masaki.mihaila@pfizer.com