PharmaSUG 2013 – Poster # P018

Compare Without PROC COMPARE

Pavan Vemuri, PPD, Morrisville, NC

ABSTRACT

In clinical trial programming, the quality of data is of paramount importance. Hence parallel programming is performed where the programmer and the validator program independently followed by comparison of the two results. When the comparison involves data sets, PROC COMPARE is widely used. Validation of data set is a twofold process where a) the attributes of variables and number of observations in both the data sets is matched and then b) the values of observations themselves are compared. Often times when deriving analyses data sets like Exposure, Labs etc. or programming tables that involve a lot of conditions, trying to match the number of observations can become tricky. Although using PROC COMPARE with ID statement helps in identifying mismatches, the need to sort the data and in some cases identifying the unique ID variables themselves can become time consuming.
This paper proposes the macro **%Counts** that helps in reducing the time taken to resolve the discrepancies on number of observations. The macro does not require the user to sort the data and shows mismatches as counts of observations grouped by a user identified variable. The idea is to give an overall pattern on the nature of mismatches. Once the number of observations is equal, comparing the data becomes easy and PROC COMPARE can be used to complete the validation.

INTRODUCTION

Validation in clinical trial programming is a critical step to ensure the quality of data. Depending on the complexity involved in the derivation, validation of a data set can often be challenging. In these situations PROC COMPARE with an ID statement needs to be used to yield useful information. The short comings are more than one key variable might need to be identified that will make data unique at observation level, the data needs to be sorted and the result from the comparison is at observation level and hence the validator is presented with vast amount of information. The proposed macro **%Counts** aims to address these points and reduce the time required to resolve the discrepancies.

MACRO %COUNTS

The macro proposed in this paper saves time as the data need not be sorted by the user, only one key variable (e.g. USUBJID) needs to be selected and the compare output is built around the key variable to help understand the pattern of mismatches. The rest of the paragraph explains the macro and its functionality starting with the code shown below.

Compare Without PROC COMPARE cont.

```
/********************Program Header*****************************

*Name of the program  -  counts.sas

*Author- Pavan Vemuri

*Purpose- To get information on mismatches at subject level

*Input Parameters

*ID - Subject identifier variable. Usually USUBJID if ADaM data, PT if raw
*     data.

* S - Name of the source/programmer's data set

* V - Name of the validator's data set

* Output - .lst file

********************End Program Header*******************************/

/*****Program Begins****/

%macro counts (ID=, S=, V=);

%* Parameter checks removed for brevity;

Proc Sql noprint;

 /**Get the total number of observations from the programmer's and validator's data
set given in the input macro variables S and V respectively. These will be displayed
in column headers**/

 Select count(&ID) into:S1 from &S;

 Select count(&ID) into:V1 from  &V;

 /****************Create data sets grouping the counts of observations by ID
variables***********************************/

 Create table count1 as select &ID,count(&ID)as CNTS from &S  group by &ID order by
&ID,CNTS;

 Create table count2 as select &ID,count(&ID) as CNTV from &V group by &ID order by
&ID,CNTV;

Quit;

/****Merge the two data sets to compare the counts  ******/

Data check;

Retain ID CNTS CNTV;

ATTRIB FLAG length =$40

              CNTS  Label = "CNT_S(&S1.)"

             CNTV  Label = "CNT_V(&V1.)";

 Merge count1(in=a) count2(in=b);
```

Compare Without PROC COMPARE cont.

```
      by &ID;
/****Create FLAG variable to capture the nature of mismatch****/
      if a and ^b then FLAG= 'Only in source data set';
      else if b and ^a then FLAG = 'Only in validation data set';
      if FLAG ='' and CNTS^=CNTV then FLAG='Count mismatch';
      else;
Run;
/*****Print out the findings***********/
Proc print data = check
    label;
Run;
%mend counts;
```

The macro can be saved at a central location and can be included as part of the SASAUTOS path to access with AUTOCALL or can be called into the program using the **%include** statement. Looking at the program above, it can be seen that the ID variable (USUBJID for ADaM data sets), the programmer's data set name and validator's data set name form the input parameters to the macro. Two level data set names are acceptable. In subsequent steps, the counts of observations are grouped by the ID variable and then merged together into a final data set. Depending on the nature of mismatch a new variable called "FLAG" is populated by either 'Only in source data set' or 'Only in validation data set' or 'Count mismatch'. The print procedure prints out the result on to output window. Shown below is the sample of **%Counts** output.

ID VARIABLE VALUE	CNTS_S(5)	CNTS_V(7)	FLAG
100	2		Only Present in source Data set
101		3	Only Present in Validation Data set
102	3	4	Count mismatch

Table 1. %Counts sample output.

The "ID VARIABLE VALUE" column has the values of the ID variable. CNTS_S (xx) and CNTS_V (xx) columns indicate the total number of observations in the header and counts of observations grouped by the ID variable in the body of the table. The FLAG column indicates the nature of mismatch. It is important to notice that the information in the FLAG column is not at observation level unlike PROC COMPARE but is presented around the values of ID variable. In table1, the FLAG column shows that there are 2 records for subject 100 in programmer's data set while none present in the validator's data set. The second row indicates that the validator's data set has 3 records for Subject 101 while there are none in the programmer's data. The final row indicates that even though both the data sets have Subject 102, the validator has one more observation compared to the programmer.

As can be seen, the information is very specific and at subject level. Hence Looking at the FLAG column in rows one, two of the output and referring back to the source data one can easily identify why there is subject present in one data set while absent in the other and vice-versa.

COMPARISON

Let's look at an example to the compare the work flow of **%Counts** Vs PROC COMPARE. Consider the data set "ADLB" that contains n records of fasting glucose readings. The key attributes of interest in this data set are subject number(USUBJID), site number(SITE), visit(VISIT), value of lab sample(AVAL), sequence number(LBSEQ) and fasting status(LBFAST) respectively as shown below in Table 2.

USUBJID	SITE	VISIT	AVAL	LBSEQ	LBFAST
101	1001	Week1	10	1	Y
101	1001	Week2	11	2	Y
101	1001	Week3	10	3	N
102	1001	Week1	12	1	N
.	.	.	.	.	.
n	.	.		.	.

Table 2. Sample ADLB Data set

For the sake of this example let's assume that the requirement is to produce a table with mean of all values for all subjects who have fasting status as "Y". In order to create discrepancy, let's assume that the programmer did not include the condition LBFAST ='Y' due to which the validator encounters an error during initial PROC COMPARE. The next step if using PROC COMPARE is to identify key variables, sort the two data sets by these variables and then run the procedure using the ID statement to identify mismatches. Alternatively using **%Counts** and its three parameters, the only step involved is to call the macro and run it. The flow chart below illustrates the work flow and differences between the two methods.

Compare Without PROC COMPARE cont.

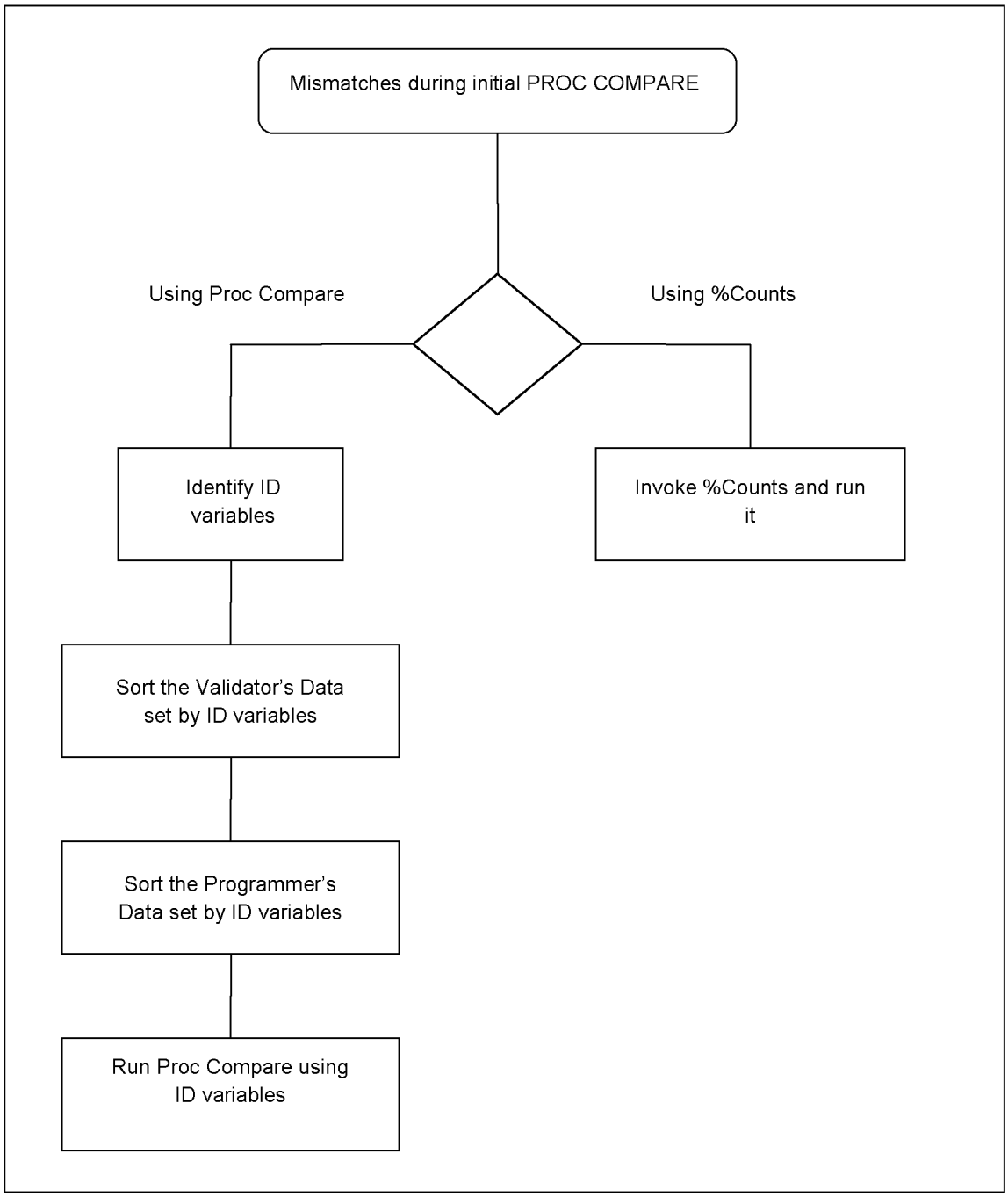

Figure 1. Flow chart using PROC COMPARE Vs. %Counts.

Apart from reducing the number of steps required, another key advantage of using the macro is that the information it is providing is very easy to read and helps in identifying discrepancies faster as seen below in figures 2 and 3.

```
The COMPARE Procedure
                Comparison of WORK.LB_S with WORK.LB_V
                   (Method=EXACT)

                        Data Set Summary

          Data set          Created         Modified  NVar    NObs

          WORK.LB_S  10APR14:16:42:03  10APR14:16:42:03    6      18
          WORK.LB_V  10APR14:16:42:03  10APR14:16:42:03    6      13


                        Variables Summary

                Number of Variables in Common: 6.
                Number of ID Variables: 3.

                 Comparison Results for Observations

        Observation 3 in WORK.LB_S not found in WORK.LB_V: USUBJID=101
         VISIT=Week 3 LBSEQ=3.

        Observation 4 in WORK.LB_S not found in WORK.LB_V: USUBJID=102
         VISIT=Week 1 LBSEQ=1.

        Observation 5 in WORK.LB_S not found in WORK.LB_V: USUBJID=103
         VISIT=Week 1 LBSEQ=1.

        Observation 15 in WORK.LB_S not found in WORK.LB_V: USUBJID=110
         VISIT=Week 2 LBSEQ=2.

        Observation 16 in WORK.LB_S not found in WORK.LB_V: USUBJID=110
         VISIT=Week 3 LBSEQ=3.


                        Observation Summary

      Observation     Base  Compare  ID

      First Obs       1       1  USUBJID=101 VISIT=Week 1 LBSEQ=1
      Last  Obs       18      13  USUBJID=110 VISIT=Week 5 LBSEQ=5

         Number of Observations in Common: 13.
         Number of Observations in WORK.LB_S but not in WORK.LB_V: 5.
         Total Number of Observations Read from WORK.LB_S: 18.
         Total Number of Observations Read from WORK.LB_V: 13.

         Number of Observations with Some Compared Variables Unequal: 0.
         Number of Observations with All Compared Variables Equal: 13.

         NOTE: No unequal values were found. All values compared are exactly equal.
```

Figure 2. PROC COMPARE output.

Obs	USUBJID	CNT_S(18)	CNT_V(13)	FLAG
1	101	3	2	Count mismatch
2	102	1		Only in source data set
3	103	1		Only in source data set
4	104	2	2	
5	105	1	1	
6	106	1	1	
7	107	1	1	
8	108	2	2	
9	109	1	1	
10	110	5	3	Count mismatch

Figure 3. %Counts output

As seen from Figure 3, only relevant information around the values of USUBJID is presented in a user friendly manner. Another advantage is that the common pitfalls [1] of PROC COMPARE can be avoided using this macro. In this particular example, looking at the second and third row and reconciling with source data, the validator can easily recognize that the source for the mismatch is the exclusion of LBFAST Flag. Once the source for mismatch is identified and fixed PROC COMPARE can be used to complete the validation process. The benefit of this approach will be exponential when there are multiple data sets to validate.

CONCLUSION

Validation is a critical process in clinical trial programming to ensure quality of data. When a data set is involved it becomes a twofold process where a) the attributes of variables and number of observations are matched and b) the values of observations themselves are compared. Matching the number of observations can become challenging. PROC COMPARE can be used with an ID statement to resolve the discrepancies but the need to sort the data sets, Identifying Id variable(s) and the vast amount of information in the compare output being at observation level can make it challenging to reconcile the mismatches. This paper proposes %**Counts** with the aim to address these shortcomings and to save time in identifying the mismatches. %**Counts** saves time as it does not require the data to be sorted by the user, only one ID variable is needed (usually USUBJID) and the compare output is tailored around the ID variable in order to give the user relevant information to help identify a pattern of mismatches. Once the mismatches are resolved, PROC COMPARE can be used to finish the validation.

REFERENCES

[1] Don't Get Blindsided by PROC COMPARE
Joshua Horstman, Nested Loop Consulting, Indianapolis, IN Roger Muller, Data-to-Events.com, Carmel, IN

[2] An Easy, Concise Way to Summarize Multiple PROC COMPAREs
Using the SYSINFO Macro Variable Lex Fennell, PPD, Inc, Wilmington, NC

CONTACT INFORMATION

For comments and suggestions

Pavan Vemuri

Sr.Programmer/Analyst

PPD, Morrisiville NC

Pavan.Vemuri@ppdi.com

919-456-4534