PharmaSUG 2012 - Paper CC04

Using PROC COMPARE to identify new and updated records in a subject listing

Mark McLaughlin, Biogen Idec, Cambridge, MA

ABSTRACT

Subject listings are a vital means for investigators to review clinical trial study data, although their usefulness often wanes as the output gets longer and more unwieldy. Similarly, as the length of the output increases, investigators are often concerned with reviewing only the most recent data points, information that gets increasingly difficult to find as the listings span multiple pages.

This paper examines a method to both identify new records added since the prior listing was run, and flag those records that are no longer new but have since been updated or altered. It uses both DATA STEP programming and the output from PROC COMPARE, with a little PROC SQL thrown in. The result is a data listing that includes the variables of interest, and an additional column which indicates what has been altered since the last iteration of the data listing. If the observation is a new one (one that didn't exist last time around) then the column is flagged with the letter 'N' (for new record). If the observation is not a new record but rather an already existing record where some variable was altered, then the column is flagged with the variable name (or names) that contain the altered information.

This method allows investigators to more efficiently read through potentially lengthy data listings and focus on the records they are most often interested in, those that are new or those that were altered/corrected.

INTRODUCTION

As clinical trial data begins to be collected, one of the most useful tools for displaying and reviewing that data is the subject listing. It allows the programmer, the statistician, and the study management and clinical teams to view the collected case report form (CRF) data by subject and ensure that the information being collected matches what was supposed to be collected. On the flip side, it can often shine a light on potential and/or obvious problems with the data collection process.

However, as the trial progresses and the data accumulate, subject listings, while still useful, can begin to span many pages and take much longer to review. A subject listing that spans hundreds of pages, as many do, has probably outlived its usefulness as a tool for reviewers. At this point, the information can often be better represented by a summary table. As their name suggests, though, summary tables only summarize the data without highlighting precisely which subjects have data that would be considered suspect. If only there was a way to have the detailed perspective of the subject listing while at the same time focusing the reviewer on only the parts of the listing that they would most likely be interested in – that is, new and/or updated records.

This paper hopes to achieve just that. As it turns out, highlighting only new and updated records can be achieved mainly using PROC COMPARE, with some help from PROC SQL, and some simple DATA STEP programming. Used and implemented correctly, this can be used on any SAS dataset, provided that the structure of the dataset (the number/names of the variables) stays the same. The data itself can of course change – that is, after all, the whole point of implementing this in the first place!

The only caveats for this method to work are the following:

1. As data is updated (daily, weekly, monthly), at least the most recent version of the old data must be kept and stored someplace (preferably in a location different than the most current data)
2. In each dataset, key variables must be assigned which make each record unique. Demographic data, which typically collected at the subject level (one record per subject), only has one key variable (subject number). Adverse event data, the example I will use below, can have multiple key variables as there can be multiple records per subject, and even multiple records per subject and adverse event. In this example, the key variables are Subject ID (ID), Adverse event (AETERM), Start date (AESTDT) and Stop date (AEENDT). The only way this method can work effectively is if key variables are identified correctly!!

EXAMPLE

We'll use a very simplified version of some data that one might expect to collect in a clinical trial – adverse event data. This is safety data that is collected in all studies, and is represented by one record per event. Therefore, in this structure, a subject can be represented multiple times if there were multiple adverse events.

SAMPLE DATA

Let's say this was the data collected as of last week:

Subject ID	Adverse event	Start date	Stop date	Severity	Relationship
1001	Headache	11/16/2005	11/18/2005	Mild	Not related
1002	Flu	11/25/2005	11/30/2005	Moderate	Possibly related
1002	Rash	11/09/2005	11/15/2005	Mild	Related
1003	Rash	11/30/2005	12/01/2005	Mild	Related

Figure 1. "Old" data

Here is the data that was collected as of this week:

Subject ID	Adverse event	Start date	Stop date	Severity	Relationship
1001	Headache	11/16/2005	11/18/2005	Mild	Possibly related
1002	Flu	11/25/2005	11/30/2005	Mild	Not related
1002	Rash	11/09/2005	11/15/2005	Mild	Related
1003	Rash	11/30/2005	12/01/2005	Mild	Related
1003	Back pain	12/15/2005	12/20/2005	Severe	Not related

Figure 2. "New" data

With a quick scan of the data above, one can see that there is one additional record (Subject 1003's 'Back pain') and some of the existing data was altered (Subject 1001's 'Headache' went from 'Not related' to 'Possibly related' and Subject 1002's 'Flu' went from 'Moderate' to 'Mild' and from 'Possibly related' to 'Not related').

The subject listing based on this week's data would report all 5 adverse events, but what wouldn't be explicitly reported is what specifically had been altered since last week's report. If we truly wanted to see not only the most recent data but additional information regarding the changes to the data, then we need another column which identifies exactly what has been altered. We do this by comparing this week's data to last week's data two ways: once using DATA STEP programming to identify "new" records, and once using PROC COMPARE on the existing records to identify "altered" records. We then use the records we identified as "new" along with the output dataset generated by the PROC COMPARE to derive a new variable which highlights what is new and/or altered.

Step 1

Once the key variables in the dataset have been identified (remember that this only works best when the variables which make each record unique are identified), then the old dataset (OLDAE) is merged with the new dataset (NEWAE). Both datasets have already been sorted on the key variables.

```
data new check (keep= id aeterm aestdt aeendt);
    merge oldae (in=a) newae (in=b);
    by id aeterm aestdt aeendt;
    if b and not a then output new;
    else if a and b then output check;
run;
```

The dataset NEW will contain the records that exist in the current data that did not exist in the older dataset. This can be set aside for now while we focus on the dataset CHECK.

Step 2

The dataset CHECK represents all the records that exist in both the old and new datasets. Since CHECK contains only the key variables, merge them back into the original old and new datasets and use the IN= option to output all variables but only for records common to the dataset CHECK.

```
data oldae2;
    merge check (in=a) oldae;
    by id aeterm aestdt aeendt;
    if a;
run;

data newae2;
    merge check (in=a) newae;
    by id aeterm aestdt aeendt;
    if a;
run;
```

Step 3

Use PROC COMPARE to compare the contents of the selected records from the old data to the same selected records from the new data (using the key variables in the ID statement). Additionally, use the OUTNOEQUAL options so that only those observations SAS considers 'unequal' are output, as those are the records of interest.

```
proc compare noprint base=oldae2 compare=newae2 outnoequal out=compout;
    id id aeterm aestdt aeendt;
run;
```

The output generated from the PROC COMPARE, while not exactly intuitive, provides all the information necessary to identify which variables have been altered.

```
Obs  _TYPE_  _OBS_   id   aeterm      aestdt      aeendt           aesev                 aerel

 1    DIF      1    1001  HEADACHE  11/16/2005  11/18/2005  ....................  X.XXXX.XXXXXXXXX....
 2    DIF      2    1002  FLU       11/25/2005  11/30/2005  .XXXXXXX............  X.XXXX.XXXXXXXXX....
```

Output 1. Output from the PROC COMPARE

Comparing the output generated by the PROC COMPARE with what we saw on the last page when we looked at the datasets, we see Subject 1001's 'Headache' and Subject 1002's 'Flu'. Looking back at the data, we know that for Subject 1001 the 'AE relationship' variable had been altered. For Subject 1002, we know that both the 'AE severity' and 'AE relationship' variables had been altered. Looking at the PROC COMPARE output above, we can see that when SAS finds a difference between values of two variables, the difference is represented by an 'X'. The number and position of the 'X' characters doesn't matter. For our purposes, all that matters is the existence of the 'X' anywhere in the PROC COMPARE output for each variable being compared. An 'X' anywhere means the values are different and, thus, the data was altered.

Now that we seem to be heading in the right direction, we need to find a way to incorporate our findings with the data.

Step 4

Now we want to compile all the information from the PROC COMPARE output into a new variable indicating which variables had been altered for each record. It helps to use PROC SQL beforehand to create macro variables indicating the number of variables being compared as well as their variable names, but for this exercise we'll use straight SAS code with %let statement.

```
%let allvars= aesev aerel;
data compout2;
    length datflag $200;
    set compout;
    array vars {2} $ &allvars ;
    datflag="";
    do i = 1 to 2;
        if index(vars{i},'X') then do;
            if datflag="" then datflag=strip(upcase(scan("&allvars",i,' ')));
            else if datflag ne "" then
            datflag=strip(datflag)|| ',' || strip(upcase(scan("&allvars",i,' ')));
        end;
    end;
    keep id aeterm aestdt aeendt datflag;
run;
```

In the above code, the output from the PROC COMPARE is being read and for each record the variables being compared are scanned. If an 'X' exists in the values for the first variable being compared, then the variable name is output to a text string called DATFLAG while the do-loop moves to the next variable. If an 'X' exists in the values for the second variable being compared, then that variable name is added to the text string. This would be done for all variables in the comparison (in this example, there are only two variables being compared).

Once the above DATA Step is run and only the variables of interest are kept, the dataset looks like this:

Subject ID	Adverse event	Start date	Stop date	DATFLAG
1001	Headache	11/16/2005	11/18/2005	AESEV
1002	Flu	11/25/2005	11/30/2005	AESEV, AEREL

Figure 3. Dataset containing altered records along with description of change

Step 5

Now it's just a matter of putting the pieces back together. There are three datasets of interest.

1. NEW dataset – This contains the key variables for all the "new" records in the most recent data (that did not exist in the old data)
2. NEWAE dataset – This contains the newest version of the records that exist in both the old and new data (all variables)
3. COMPOUT2 dataset – This contains, for each record where the data was altered, the key variables and DATFLAG which highlights exactly which variables were altered.

Merging them together by the key variables provides the listing that we wanted: all the records in the most current data with a flag variable indicating which records were new and which were altered (and more specifically, which variables had been altered).

```
data ae;
    merge new (keep= id aeterm aestdt aeendt in=a) newae compout2;
    by id aeterm aestdt aeendt;
    if a then datflag='N';
run;
```

The new dataset would look like:

Subject ID	Adverse event	Start date	Stop date	Severity	Relationship	DATFLAG
1001	Headache	11/16/2005	11/18/2005	Mild	Possibly related	AEREL
1002	Flu	11/25/2005	11/30/2005	Mild	Not related	AEREL, AESEV
1002	Rash	11/09/2005	11/15/2005	Mild	Related	
1003	Rash	11/30/2005	12/01/2005	Mild	Related	
1003	Back pain	12/15/2005	12/20/2005	Severe	Not related	N

Figure 4. Final dataset ready for output as subject listing

Again, this was a very simplified example. Most data will certainly have more variables and will accumulate hundreds or thousands more records over time. Additionally, the programming steps shown here were also simplified. In order to fully utilize the power of this method, it would make sense to create a macro out of it that could be run at the beginning of each listing program. Implemented correctly, the only macro variables one would need would be the input dataset and of course the key variables. Everything else could be supplied via DATA Step programming or via PROC SQL.

CONCLUSION

With a basic knowledge of DATA Step programming as well as PROC COMPARE, it is possible to access the vast and detailed information that SAS provides and provide data reviewers with an accurate, up-to-date account of all the important pieces of your data, along with a means of identifying exactly which information they may want to focus on. When you've cut their review workload from thousands of records to just a handful, they'll no doubt want to thank you!

CONTACT INFORMATION

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

Name: Mark McLaughlin
Enterprise: Biogen Idec
Address: 12 Cambridge Center
City, State ZIP: Cambridge, MA 02142
Email: mark.mclaughlin@biogenidec.com