PharmaSUG 2018 - Paper AD-13

An Automated Macro to Compare Data Transfers

Michael Stackhouse, Franklin E. Menius Jr., Covance, Inc.

ABSTRACT

Routine data transfers throughout the course of an ongoing study are a necessity for any analysis, but for statistical programmers, updates and modifications to the database can plague programs and break code. Sponsors can also utilize multiple vendors for one study, which can make ensuring clear communication a problem between data management and statistical programming. Problems and changes in the database might not be found until a program breaks somewhere in SDTM, ADaM, tables, listings, or figures. Therefore, it is imperative to review your data transfers and compare back to previous cuts of data. This paper will explore a macro tool used to compare two cuts of data. The tool can identify added or dropped datasets, significant changes in variable metadata, and examines datasets, looking for potentially dropped records. This macro additionally minimizes user input, and only requires the two directory paths containing your cuts of data.

INTRODUCTION

As we all know, it is better to identify problems proactively rather than finding issues the hard way. Refreshing a delivery on new data is an occasion where this is even more detrimental. A great deal of time can be spent executing programs, watching things fail, and tracing issues back to their origins. These problems seem to be amplified when time is tight, and deadlines are looming. A common nightmare for programmers is when a table issue traces back to SDTM; this brings them right back to square one.

BUT, there are even scarier problems! What if during an ongoing study, an issue was found during the production of the fourth Data Monitoring Committee (DMC) meeting package? The delivery is set to wrap up and suddenly, during a statistician's review, it is discovered that the Adverse Event (AE) counts for a subject went **down** compared to last year's delivery. Where does one begin? What should one do? It is a rare occurrence, but it is plausible and potentially serious, leading to questions about data integrity.

PROC COMPARE in SAS is a great and commonly used tool; however, it does not have the functionality needed to approach the problem knowing the data will grow and change over time. The described macro offers a proactive approach to find these types of issues proactively and trace issues back to the source efficiently. To approach this, the separate extractions are reviewed on three different levels:

1. The datasets available *within* the package
2. The metadata *within* each dataset
3. The count of records per subject *within* each dataset

Numbers 1 and 2 above are likely obvious targets. At a high level, the clearest potential issue to check is whether all the source data still exist in the new cut, and whether the structure of those data have changed. The third level of review is added as a safety measure to look for data *loss.* This will be discussed in greater detail later.

A key feature of this macro is the ease of use. Again, it is better to identify problems sooner rather than later. Keeping this goal in mind, our aim was to minimize time spent on user setup and user input. For this macro to work, the user only *needs* to identify two parameters (with a third left as optional):

- The directory containing the old data
- The directory containing the new data
- Additional subject identifiers in your data *(optional)*

The third parameter will be discussed in more detail looking at the third level of review. As for the first two parameters, to keep the code dynamic and portable, the minimum parameters required for input are just the location of your data; the macro handles everything else for you. To begin, we will discuss level one.

DATASET LEVEL REVIEW

The first check is whether any datasets have been removed or added from one extraction to another. It is desirable to know if there are new datasets present, if previous datasets are missing, or if the previous datasets have been received under a new name in the present cut. The latter can happen as the result of, for example, external vendor processes, changes of vendors, or studies shifting companies.

To this end, the program first finds the directories for each extraction and builds a dataset containing all "*sas7bdat"* files.

The program then completes a basic merge, outputting a list of any datasets in the old data extraction but not in the new and vice versa. While not specifically labeling it, this step will also alert the programmer to those datasets submitted with new names, as they also fall into these categories.

Finally, the program produces a PDF report of the results of this first step. See Display 1 for an example of the report.

These datasets are in the OLD cut of data and not in the NEW.

name
ex

These datasets are in the NEW cut of data and not in the OLD.

name
exrename
vs

Display 1. Sample Report from Step 1

METADATA LEVEL REVIEW

Once the program has determined which datasets are in both the new and the old data extractions, it then starts iterating over each dataset and compares it to against its old version. The program looks at several levels that are of prime interest to programmers. Those levels are as follows:

- Variables added
- Variables dropped
- Variable type
- Variable length

While other attributes, including formats, can be added for the program to compare, in terms of programming type and length have the largest implications for programmers. Process changes in a study could lead to these types of data issues, for example, process changes involving dictionary coding may not change structure, but individual variable attributes unexpectedly. This program alerts the programmer to these types of updates.

Obviously, added or dropped variables can have a huge implication as well. Dropped variables easily cause program errors, while added variables may go missed leaving important data out of the submission. The goal of this step is to prevent errors as well as the omission of data.

To accomplish these goals, the program first assembles the subset of metadata needed from the SASHELP.VCOLUMN dataset pulling in data only where LIBNAME in(“NEW”, “OLD”). SASHELP.VCOLUMN is used here, however DICTIONARY.COLUMNS contains the same information. The main difference is that SASHELP.VCOLUMN is available outside of PROC SQL.

The program then iterates over every dataset that contained a match (i.e., was in both OLD and NEW) and compares the metadata. It creates two separate checks:

- Variables added and dropped (NAME was in OLD and not in NEW or NAME was in NEW and not in OLD)
- TYPE or LENGTH did not match

As it did after step one, the program now produces a PDF report of the results of this step. See Display 2 for an example of that report.

Differences in metadata between OLD.DE and NEW.DE. Variables in NEW not in OLD.

Column Name
DEORRES

Differences in metadata between OLD.DS and NEW.DS. Differences in variables.

Column Name	Column Type	Column Length	Column Type	Column Length
DSSTDTC	char	8	num	8

Display 2. Sample Report from Step 2

SUBJECT COUNTS PER DATASET

After comparing the data extractions for matching datasets followed by a check for consistent metadata, next the program compares the number of subject counts per dataset to identify data loss. While this issue is much rarer than the previous items discussed it is more serious if unanticipated. Some examples from studies where this type of issue has arisen include:

- A data extraction where an issue resulted in one subject inadvertently being removed from all the CRF datasets.
- A data extraction between Q1 and Q3 DMC where 14 AEs were removed for one subject without obvious explanation as to why.

In many ways, this program was designed with these two incidents in mind. The aim to improve moving forward and share experiences is an integral piece of this SUG community.

Why stop there? In addition, the program can be enhanced if the database in question has unique record identifiers. Medidata RAVE databases, for instance, have identifiers embedded in the data management columns that create these unique keys for future checks. This is where a close relationship with the data manger can be useful.

To get started with this additional check the program first tries to determine the subject identifier using a set of potential candidates: PT, SUBJECT_NUMBER, SUBNUM, SUBJECT, USUBJID, SUBJID. If a subject identifier needs to be added to this list, the user can update the %let adsubj=; parameter at the beginning of the code. For example:

```
%let addsubj= CHECK1 CHECK2 CHECK3;
```

If the program does not find a matching identifier it will generate a warning in the log stating:

```
WARNING: SUBJECT IDENTIFIER NOT COMPENSATED FOR - COULD NOT FIND
IDENTIFIER WITHIN THE LIST <list of variables> IN THE DATASET <dataset
being reviewed>.
```

The program will also place a similar message in the report.

If the program can establish a subject identifier, it runs frequencies using that identifier and merges the datasets. If the *old* count (those existing in the previous data extraction) is LESS THAN the *new* count (those existing in the new data extraction), then the record is flagged and output.

Having flagged the subjects with fewer records in the new data extraction, the program now produces a PDF report of the results (Display 3).

These subjects in NEW.AE have less records than in OLD.AE.

Subject Identifier for the Study	Frequency Count	Frequency Count
7802	4	.

Display 3. Sample report from step 3

POTENTIAL LIMITATIONS

It is important to address some limitations of the above steps. The first is the potential for false positives. As data are cleaned some records may be determined appropriate to delete. As it difficult to programmatically assess intent, those records will be reported as missing in this check. It would still be wise to confirm with data management that these records should indeed have been removed.

The second limitation is that in the interests of being dynamic and portable, the program doesn't establish key variables for every dataset. It relies on only the frequency of a subject's unique identifier in the dataset, meaning it is possible for issues to slip by. For example, if the previous version of a dataset had a subject with 10 records, but in the new version 2 of those records were removed, but also 6 records were added then the program will not report a possible data loss because there was a net gain in records. To limit this situation and make the results more reliable, it is advised to run the program at regular intervals.

CONCLUSION

New data extractions are common during a clinical trial, and it is important to discover changes to the existence, structure, or attributes of the source data prior to processing programs. The above macro can assist the programmer in making these comparisons. While it has limitations, running it frequently and at regular intervals can help detect changes in the presence of datasets, changes in variable metadata and structure, and find potentially missing data. Though the macro program is tailored to be dynamic and portable, small changes can be made to address specific issues relevant to the programmer or the study. This program has the potential to detect major issues early on and save the programmer and company valuable time.

CONTACT INFORMATION

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

Michael Stackhouse
Covance, Inc.
Michael.Stackhouse@Chiltern.com

Frank Menius
Covance, Inc.
Frank.Menius@Chiltern.com

APPENDIX FULL TEXT OF PROGRAM

```
************************************************************************;
*  Covance, Inc.                                                       *;
*                                                                      *;
*  Client:                                                             *;
*                                                                      *;
*  Program:     w_datacutcomp                                          *;
*  Date:        16MAR2017                                              *;
*  Description: Compares all datasets between 2 directories and        *;
*               details the following:                                 *;
*               -Datasets in Old library not in New, vice versa        *;
*               -Variables in old dataset not in New, vice versa       *;
*               -Difference in number of subjects in old not in new.   *;
*                This is flagged only if there were more records for   *;
*                a subject in the old directory than the new one.       *;
*                                                                      *;
*             This program is intended to be used on an old extract    *;
*             of data compared to a new one.                           *;
*                                                                      *;
*  Author:      Michael Stackhouse                                     *;
*  Revisions:   03/01/2018 Frank Menius                                *;
*  Comments:                                                           *;
************************************************************************;
options varlenchk=nowarn;

*Set directories for old and new cut of data;
%let old=/sasdata/un0101/projects/test/bench/ms9497/dev/rawdata/old/;
%let new=/sasdata/un0101/projects/test/bench/ms9497/dev/rawdata/new/;

*Set directory for report destinations;
%let sasexport=/sasdata/un0101/projects/test/bench/ms9497/dev/data/;

*Add subject variables to check - Delimit with spaces;
*Example: %let addsubj= CHECK1 CHECK2 CHECK3;
%let addsubj=;

*****************************;
```

```
**** STARTING MACRO CODE ****;
*****************************;

*Clean out work directory;
proc datasets lib=work nolist kill;
run;

%macro datacut_compare(old_lib=,new_lib=);

  /*************/
  /* Prep-work */
  /*************/
  libname old "&old_lib";
  libname new "&new_lib";

  /************************************/
  /* Step 1: Compare datasets available */
  /************************************/

  *Grab old dataset list from folder to compare;
  data old_cut (keep=name);
    length name $50;

    *set file name for folder to open;
    rc=filename("mydir","&old_lib");

    *Open folder;
    did=dopen("mydir");

    *Pull out variables and select them into separate datasets;
    do i=1 to dnum(did);
      if strip(upcase(scan(dread(did,i),2,'.')))='SAS7BDAT' then do;
        name=scan(dread(did,i),1,'.');
        output;
        call missing(name);
      end;
    end;

    *Close folder;
    rc=dclose(did);
  run;

  proc sort data=old_cut;
    by name;
  run;

  *Grab new dataset list from folder to compare;
  data new_cut (keep=name);
    length name $50;

    *set file name for folder to open;
    rc=filename("mydir","&new_lib");

    *Open folder;
    did=dopen("mydir");

    *Pull out variables and select them into separate datasets;
```

```
  do i=1 to dnum(did);
    if strip(upcase(scan(dread(did,i),2,'.')))='SAS7BDAT' then do;
      name=scan(dread(did,i),1,'.');
      output;
      call missing(name);
    end;
  end;

  *Close folder;
  rc=dclose(did);
run;

proc sort data=new_cut;
  by name;
run;

*Merge - separated in old, in new, and in both;
data ds_list in_old in_new;
  merge old_cut (in=a) new_cut (in=b);
  by name;

  if a and not b then output in_old;
  else if b and not a then output in_new;
  else output ds_list;
run;

*print PDF reports of in_old and in_new;
ods _all_ close;
options nodate nonumber orientation=portrait;
ODS PDF File="&sasexport.cutcompare1.pdf";
title1 "These datasets are in the OLD cut of data and not in the NEW.";
proc report data=in_old;
run;
title1;

title1 "These datasets are in the NEW cut of data and not in the OLD.";
proc report data=in_new;
run;
title1;

ods pdf close;

 *Clean;
proc datasets lib=work nolist;
  delete old_cut new_cut in_old in_new;
run;

/*************************************************************/
/* Step 2: Compare metadata of like named datasets available */
/*************************************************************/

*Grab list and counts for loops;
proc sql noprint feedback;
  select name into: dslist separated by " " from ds_list;
  select count(name) into: dscnt separated by "" from ds_list;
  *Compile VCOLUMN dataset so that this only needs to be done once;
  create table meta as
```

```
    select libname, memname, name, type, length
      from sashelp.vcolumn
        where strip(upcase(libname)) in("NEW" "OLD");
quit;

*Macro to compare metadata;
%macro comp_ds(ds=);

  *Grab metadata of old and new datasets;
  proc sql;
    create table old_meta as
      select name, type as type_old, length as length_old
        from meta
          where strip(upcase(libname))="OLD" and
                strip(upcase(memname))=upcase("&ds.")
            order by name;
    create table new_meta as
      select name, type as type_new, length as length_new
        from meta
          where strip(upcase(libname))="NEW" and
                strip(upcase(memname))=upcase("&ds.")
            order by name;
  quit;

  *Compare differences. Separate variables in old or variables in new.;
  data meta_vars_old meta_vars_new meta_diff (drop=flag);
    merge old_meta (in=a) new_meta (in=b);
    by name;
    if a and not b then output meta_vars_old;
    else if b and not a then output meta_vars_new;
    else do;
      if type_old^=type_new then flag="Y";
      if length_old^=length_new then flag="Y";
      if flag="Y" then output meta_diff;
    end;
  run;

  *print to variable differences to PDF;
  ODS PDF File="&sasexport.cutcompare2a.pdf";
  title1 "Differences in metadata between OLD.%upcase(&ds.) and
         NEW.%upcase(&ds.).";
  title2 "Variables in OLD not in NEW.";
  proc report data=meta_vars_old;
    columns name;
      define name / order;
  run;
  title2;

  title2 "Variables in NEW not in OLD.";
  proc report data=meta_vars_new;
    columns name;
      define name / order;
  run;
  title2;

    ods pdf close;
```

```
      ODS PDF File="&sasexport.cutcompare2b.pdf";
    title2 "Differences in variables.";
    proc report data=meta_diff;
    run;
    title2;
    title1;

      ods pdf close;

    proc datasets lib=work nolist;
      delete old_meta new_meta meta_vars_: meta_diff;
    run;

/*****************************************************************/
/* Step 3: Compare number of records per subject -               */
/* flag if new cut has less records                              */
/*****************************************************************/

    %let varlist_new=;
    %let varlist_old=;
    %let subj=;
     ODS PDF File="&sasexport.cutcompare3.pdf";
    *Grab variable names into list so we can scan it for the subject
    identifier that we want;
    proc sql noprint feedback;
      select name into: varlist_new separated by "#"
        from meta where strip(upcase(libname))="NEW" and
                        strip(upcase(memname))=upcase("&ds.");
      select name into: varlist_old separated by "#"
        from meta where strip(upcase(libname))="OLD" and
                        strip(upcase(memname))=upcase("&ds.");
    quit;

    *Scan variable lists for SUBJECT and SUBJID;
    %if &varlist_new.^= and &varlist_old.^= %then %do;
      *Editing # on start and end of varlist for easy indexing;
      %let varlist_new=#&varlist_new.#;
      %let varlist_old=#&varlist_old.#;
      %put;
      %put NEW VARIABLE LIST: &varlist_new.;
      %put OLD VARIABLE LIST: &varlist_old.;
      %put;

      %let check_list = PT SUBJECT_NUMBER SUBNUM SUBJECT USUBJID SUBJID
           &addsubj.;
      %put %str(NOT)E: Subject variable to check: &check_list;

      %local i check;
      %do i=1 %to %sysfunc(countw(&check_list));
         %let check = %scan(&check_list, &i);
         %put %str(NOT)E: CHECKING VARIABLE &check.;
         %if %index(%upcase(&varlist_new.),#&check.#)>0 and
             %index(%upcase(&varlist_old),#&check.#)>0 %then %let
             subj=%upcase(&check.);
      %end;

      %if &subj.= %then %do;
```

```
      *Print message if could not find subject variable;
        data _null_;
        put @1 "SUBJECT IDENTIFIER NOT COMPENSATED FOR - COULD NOT FIND
               IDENTIFIER WITHIN THE LIST &check_list. &addsubj. IN THE
               DATASET %upcase(&ds.)";
      run;
      %put %str(WARN)ING: SUBJECT IDENTIFIER NOT COMPENSATED FOR - COULD
           NOT FIND IDENTIFIER WITHIN THE LIST &check_list. &addsubj. IN
           THE DATASET %upcase(&ds.);
  %end;
  %else %do;
     %put %str(NOT)E: In &DS. SUBJ = &subj.;
  %end;
%end;

%else %do;
  *Print message if could not find subject variable;
  data _null_;
    put @1 "SUBJECT IDENTIFIER NOT COMPENSATED FOR - COULD NOT FIND
           IDENTIFIER WITHIN THE LIST &check_list. &addsubj. IN THE
           DATASET %upcase(&ds.)";
  run;
  %put %str(WARN)ING: SUBJECT IDENTIFIER NOT COMPENSATED FOR - COULD NOT
       FIND IDENTIFIER WITHIN THE LIST &check_list. &addsubj. IN THE
       DATASET %upcase(&ds.);
%end;

*Check if subject identifier as found;
%if &subj. ^= %then %do;

  *Get frequencies of subjects on both sides;
  proc freq data=old.&ds.;
    tables &subj./list noprint out=freq_old (rename=(count=count_old));
  run;

  proc freq data=new.&ds.;
    tables &subj./list noprint out=freq_new (rename=(count=count_new));
  run;

  proc sort data=freq_old (drop=percent);
    by &subj.;
  run;

  proc sort data=freq_new (drop=percent);
    by &subj.;
  run;

  *Check variable type;
  %let dsid=%sysfunc(open(new.&ds.,i));
    %do i=1 %to %sysfunc(attrn(&dsid,nvars));
      %if %upcase(%sysfunc(varname(&dsid,&i)))=%upcase(&subj.) %then %do;
        %let vartype=%sysfunc(vartype(&dsid,&i));
        *Set length value to be used;
        %if &vartype=N %then %let len=8;
        %else %let len=$20;
      %end;
    %end;
```

```
        %let rc=%sysfunc(close(&dsid));

      *Merge together - flag if the old data cut had more records per subject
      than the new;
      data check_counts;
         length &subj. &len.;
        merge freq_old (in=a) freq_new (in=b);
        by &subj;
        *Strip off formats so proc prints display properly;
        format _all_;
        informat _all_;
        if count_old > count_new then diff="Y";
      run;

      *Print differences to PDF;

      title1 "These subjects in NEW.%upcase(&ds.) have less records than in
             OLD.%upcase(&ds.).";
      proc report data=check_counts (where=(diff="Y"));
        columns &subj. count_old count_new;
            define &subj. / display;
            define count_old / display;
            define count_new / display;
      run;
      title1;
      ods pdf close;


      proc datasets library=work nolist;
        delete freq_old freq_new check_counts;
      run;
    %end;

  %mend comp_ds;

  *Macro for looping through list;
  %macro loop;
    %do j=1 %to %eval(&dscnt.);
       %let cds=%scan(&dslist,&j);
        %comp_ds(ds=&cds);
    %end;
  %mend loop;

  *Call loop;
  %loop;

%mend datacut_compare;

options mprint;
%datacut_compare(new_lib=&new.,old_lib=&old.);
```