Paper DM02

A Flexible, User-Friendly Methodology for Data Set Comparison

Michael Thompson, Clinimetrics Research Associates, Mississauga, Ontario

ABSTRACT

A common activity performed by data management personnel within a clinical trial is the tracking of changes in study data. On a periodic basis the database is downloaded into SAS® data sets and the current version is compared to a previous version. One option for doing this is using the COMPARE procedure built into SAS. However, this can result in large amount of output and is not designed to do comparisons on multiple pairs of data sets.

This paper proposes an alternate method for performing data set comparisons of this type. In this approach an initialization file is created which contains the names and primary keys for each of the data sets to be compared. The file is read into memory and using a macro loop statement the data sets are processed iteratively. For each data set a number of comparisons are made and the results are written in a format convenient for data management to analyze.

INTRODUCTION

The purpose of this paper is to introduce a methodology for comparing different versions of a data set, similar in some respects to PROC COMPARE, but with two distinguishing features. First, it allows for the comparison of a large number of pairs of data sets. The user specifies two folders locations, each containing a set of identically named data sets. The user then creates an initialization file containing the names and primary keys for each of the data sets. Second, it provides output in a concise format that allows for the user to quickly identify the differences between the two versions of data sets. In each of the output files the results are categorized by the type of difference. These range from differences in the data set structure, or schema (i.e. the variable names and data types) to differences between records that are present in both versions.

The remainder of this paper outlines the steps of the methodology along with SAS code that was used by the author to implement it.

Step 1: Create an Initialization File

This initialization file contains the name of the data sets to be compared and their primary keys. A primary key consists of one or more variables that can used to uniquely identify a row in a data set. For instance, in a data set containing patient demographic data each patient will have only a single row. Because of this each row would have its own unique patient identifier variable value and this variable could serve as the data set's primary key. Another example is a data set containing concomitant medications. In addition to a patient identifier variable, one or more variables identifying the name of the medication as well as one containing the start date would be needed to for it's primary key.

In practice, a good file format for this purpose is either a comma delimited file or, if available, an Microsoft Excel worksheet. An example initialization file is as follows. The file contains two columns, one for the data set name and a second for the primary key. The variables making up the primary key are delimited by commas.

Data set	***Key***
CMCM	PT, ATC01T, PN, STRTDTF
DEMO	PT
CHEMCHEM	PT, LPARM, CPEVENT
PKPK	PT, LPARM, CPEVENT, TIMEPT

Step 2: Load the Initialization File

The initialization file is read by SAS which stores the data set and primary key variable names into macro variables. The macro variable names are created incorporating the order the variable appears in the initialization file so that they can be processed in a looping fashion.

In the SAS® code below the initialization file is imported into SAS using the IMPORT procedure. The DATA step that follows reads the values into the macro variables described below.

- *ds_cnt*: contains the number of data sets in the initialization file.
- *ds1*, *ds2*, ..: contains the data set names (in the example above, ds1 would be equal to 'CMCM', ds2 would be equal to 'DEMO', etc.).
- *key1*, *key2*,.. : contains the space delimited data set keys for each of the data sets (in the example above, key1 would be equal to 'PT ATC01T PN STRTDTF', key2 would be equal to 'PT').

```
%let init_path = C:\Clinical Programming\Program\Comparison\Init File.xls;

proc import datafile="&init_path)
            out=w1 (keep=dataset_name key_variables where=(not
missing(dataset_name)))
            replace;
run;

data _null_;
  set w1 end=last;

  call symput('ds' || trim(left(put_N_, 3.))), trim(left(dataset_name)));
  call symput('key' || trim(left(put_N_, 3.))),
              trim(left(translate(key_variables, '  ',',',''))));
  if last then call symput(ds_cnt, trim(left(put_N_, 3.))));
run;
```

Step 3: Create the Control Macro

A short macro is created to loop through each of the data sets in the initialization file from Step 1. This is done using the macro variables created in the previous step. In each iteration of the loop, the two versions of the data set are compared multiple times using comparison macros. Each macro, with some exceptions, generally operates the same way. The macro takes as input two data sets, one each for the previous and current versions, examines them for differences of a certain type, and returns as output three data sets. One detailing the differences between the two versions and the remaining two are the input data sets, minus the differences caught at that step. For example, the first macro checks for schema differences. It would return the two input data sets after they've been stripped of variables which are not common to both data sets. The output data sets would have the same schema. This will be necessary for the macros following it to work properly.

In the SAS code below, the LIBNAME statements are defined for the folders containing the two versions of the data sets. Following that is a %do statement that loops through each of macro data set variables creating in the previous step. Six macro calls are made, 5 performing the comparisons and final one writing the results to an output file. Steps 4 and 5 detail these macro calls.

```
%include 'C:\Clinical Programming\Program\Comparison\studinit.sas';

libname prev 'C:\Clinical Programming\Data\RawData\archive\20090108';
libname curr 'C:\Clinical Programming\Data\RawData';

%macro compare;

  %do i = 1 %to &ds_cnt;

    %let ds = &&ds&i;
    %let key = &&key&I;

    proc sort data=prev.&ds
              out=prev_ds;
      by &key;
    run;

    proc sort data=curr.&ds
             out=curr_ds;
      by &key;
    run;

    %check_for_structural_differences(inprev=prev_ds,
                                      incurr=curr_ds,
                                      outprev=prev_ds2,
                                      outcurr=curr_ds2,
                                      output=structural);

    %check_for_duplicate_rows(inprev=prev_ds2,
                              incurr=curr_ds2,
                              outprev=prev_ds3,
                              outcurr=curr_ds3,
                              output=duplicates);
```

```
    %check_for_deleted_rows(inprev=prev_ds3,
                            incurr=curr_ds3,
                            outprev=prev_ds4,
                            output=deletions);

    %check_for_added_rows(inprev=prev_ds4,
                          incurr=curr_ds3,
                          outcurr=curr_ds4,
                          output=additions);

    %check_for_modified_rows(inprev=prev_ds4,
                             incurr=curr_ds,
                             output=modifications);

    %write_output(struct=structural,
                  dup=duplications,
                  del=deletions,
                  add=additions,
                  mod=modifications);

%mend compare;

%compare;
```

Step 4: Perform the comparisons

A: *Structural differences*
The two data set versions are checked for differences in schema, i.e:

- Does one of the versions have a variable that is not present in the other?
- Does one of the variables common to both versions differ in type (string vs number)?

In the SAS code below, PROC CONTENTS is used to create data sets specifying the variable names and data types for the two versions of the data set. The DATA step that follows creates the *output* data set detailing the schema differences as well as the 'var' data set. This file is used in the next two DATA steps to create macro variables containing the data set variables that differ between the two versions. The last two DATA steps create the modified *outprev* and *outcurr* which are *inprev* and *incurr* with the schema differences dropped.

In the example *output* data set below: (a) the BSFS variable is present in the previous version, but missing from the current, (b) the BTGS variable is present in the current version, but missing from the previous, and (c) the VCRFD variable is present in both data sets, but the data types differ.

name	***prev_var***	***curr_var***
BSFS		Missing
BTGS	Missing	
VCRFD	Numeric	String

```
%macro check_for_structural_differences(inprev=,incurr=,outprev=,outcurr=,output=);

  proc format;
    value dtype
      1 = 'Numeric'
      2 = 'String';
                run;

  proc contents data=&inprev
                out=_prev_ds (keep=name type) noprint;
  run;

  proc contents data=&incurr
                out=_curr_ds (keep=name type) noprint;
  run;

  data &output (keep=name prev_var curr_var)
```

```
     var (keep=var src);
  merge _curr_ds (in=in_curr rename=(type=curr_type))
        _prev_ds (in=in_prev rename=(type=prev_type));
    by name;

  length prev_var curr_var $9;
  retain prev_cnt curr_cnt 0;
  label name='Variable Name'
        prev_var='Previous Version'
        curr_var='Current Version';

  if in_prev and not in_curr then
    do;
      curr_var = 'Missing';
   prev_var = '';
      output &output;

       src = 'Previous';
       var = name;
       output var;

    end;
  else if in_curr and not in_prev then
    do;
      curr_var = '';
      prev_var = 'Missing';
      output &output;

       src = 'Current';
       var = name;
       output var;
    end;
  else if curr_type NE prev_type then
    do;
      prev_var = put(prev_type, dtype.);
      curr_var = put(curr_type, dtype.);
      output &output;

      src = 'Current';
      var = name;
      output var;

      src = 'Previous';
      var = name;
      output var;
    end;
run;

  data _null_;
  set var (where=(src='Previous')) end=last;

  call symput('svold' || trim(left(put(_n_, 3.))), var);
  if last then call symput('prev_cnt', put(max(_n_, 0), 3.));
run;

%if not %symexist(prev_cnt) %then
  %let prev_cnt = 0;

  data _null_;
  set var (where=(src='Current')) end=last;

  call symput('svnew' || trim(left(put(_n_, 3.))), var);
  if last then call symput('curr_cnt', put(max(_n_, 0), 3.));
run;

%if not %symexist(curr_cnt) %then
  %let new_cnt = 0;
```

```
  data &outprev;
    set &inprev (drop= %do i = 1 %to &prev_cnt; &&svold&i %end; );
  run;

  data &outcurr;
    set &incurr (drop= %do i = 1 %to &curr_cnt; &&svnew&i %end; );
  run;

%mend check_for_structural_differences;
```

B: *Duplicate Rows*

Each of the data set versions is checked for instances of multiple rows sharing the same primary key value. This is a violation of the primary key and the presence of this condition precludes remaining comparisons from being made. For example, in the CHEMCHEM data set from Step 1, this would be the case if two or more rows have the same PT variable value, the same LPARM variable value, and the same CPEVENT variable value.

In the SAS code below, the SORT procedure is used with the DUPOUT option to capture the cases where the primary key value is shared by multiple rows in either of the two data set versions. Following this, the rows sharing these values are deleted to create the *outprev* and *outcurr* data sets. The duplicate primary keys values are then consolidated into the *output* data set.

In the example *output* data set below, the previous version of the data set has multiple rows with the PT, LPARM, and CPEVENT variables equaling '101', 'ALT', and 'SCREENING', respectively and the current version of the data set has multiple rows with the PT, LPARM, and CPEVENT variables equaling '211', 'AST', and 'DAY 3', respectively.

PT	***LPARM***	***CPEVENT***	***Source***
101	ALT	SCREENING	Previous Version
211	AST	DAY 3	Current Version

```
%macro check_for_duplicate_rows(inprev=,incurr=,outprev=,outcurr=,output=);

  proc sort data=&inprev
            out=jnk
            dupout=_prev_dup (keep=&key.) nodupkey;
    by &key.;
  run;

  proc sort data=&inprev
            out=_prev_ds;
    by &key.;
  run;

  data &outprev;
    merge _prev_ds
          _prev_dup (in=in_dup);
      by &key;

    if not in_dup;
  run;

  proc sort data=&incurr
            out=jnk
            dupout=_curr_dup (keep=&key.) nodupkey;
    by &key.;
  run;

  proc sort data=&incurr
           out=_curr_ds;
    by &key.;
  run;

  data &outcurr;
    merge _curr_ds
          _curr_dup (in=in_dup);
      by &key;
```

```
    if not in_dup;
  run;

  data &output;
    set _prev_dup (in=in_prev)
        _curr_dup;
      by &keyvar;

    label source='Source File';
    length source $15;

    if in_prev then
      source = 'Previous Version';
    else
      source = 'Current Version';
  run;

%mend check_for_duplicate_rows;
```

C: *Rows not Present in Both Versions*

Compare the two data set versions by linking them with the primary key variables and identify rows that are present in one, but not both versions. For example, in the DEMO data set from Step 1, this would be the case if a row with the PT variable equaling '106' was present in the current version, but not the previous.

In the SAS code below, two macros are defined, %check_for_deleted_rows and %check_for_added_rows. The first macro identifies rows, using the primary key, that were present in the previous version of the data set, but are not found in the current. The *output* data set contains the resulting key variable values. The macro also returns *outprev*, which is equal to *inprev* with applicable rows deleted. The %check_for_added_rows macro does the same thing, but in reverse.

In the example *output* data set from the %check_for_deleted_row macro below, the previous version of the data set contains a row with PT equaling '106', LPARM equalling 'LDH' and CPEVENT equaling 'DAY 1' that was not present in the current version.

PT	***LPARM***	***CPEVENT***
106	LDH	DAY 1

```
%macro check_for_deleted_rows(inprev=,incurr=,outprev=,output=);

  data &outprev
       &output (keep=&key.);
    merge &inprev
          &incurr (keep=&key in=in_curr);
      by &key;

      if in_curr then
        output &outprev;
      else
        output &output;
  run;

%mend check_for_deleted_rows;


%macro check_for_added_rows(inprev=,incurr=,outcurr=,output=);

  data &outcurr
       &output (keep=&key.);
    merge &incurr
          &inprev (keep=&key. in=in_prev);
      by &key.;

      if in_prev then
        output &outcurr;
      else
```

```
      output &output;
  run;

%mend check_for_added_rows;
```

D: Values in Common Rows Differ

At this point there is a 1-1 correspondence between all of the rows in the two data set versions; using the primary key variables all rows in the two can be matched to each other. Check to see if for any of the corresponding rows one or more variable values differ.

In the SAS code below, the first DATA step puts each of the primary key variable names in a macro variable. It then uses PROC CONTENTS to get all the variable names from the data set and using the primary key macro variables creates a set of macro variables for the non-primary key variables. In the final DATA step the two data set versions are merged together and the non-primary key macro variables are used to rename these variables in one of the data sets so the variable values from the second data set do not overwrite those from the first. A comparison is then made for each of these variables between the two versions and differences are written to the *output* data set.

In the example *output* data set below, the previous version of the data set, for the row defined by the PT, LPARM, and CPEVENT primary key variables, has an LVALUE variable value of 1.54, while the current version has the value 1.90 for this variable.

PT	***LPARM***	***CPEVENT***	***Variable***	***Prev_Val***	***Curr_Val***
106	LDH	DAY 1	LVALUE	1.54	1.90

```
%macro check_for_modified_rows(inprev=,incurr=,output=);

  data _null_;

    length str wrd $256;
    str = "&key";

    last = 0;
    i = 0;

    do until (last);

      i = i + 1;
      wrd = scan(str, i, ' ');

      if not missing(wrd) then
        call symput(compress('key' || put(i, 3.)), compress(wrd));
      else
       do;
         call symput(compress('key_cnt'), put(i - 1, 3.));
         last = 1;
       end;
    end;
  run;

  proc contents data=&inprev
                out=_vname (keep=name type) noprint;
  run;

  data _null_;
    set _vname;

    retain cnt 0;

    %do i=1 %to &key_cnt;
      if lowcase(name) NE lowcase("%cmpres(&&key&i)");
    %end;

    cnt = cnt + 1;
    call symput(compress('tvar' || put(cnt, 3.)), name);
    call symput(compress('tvartyp' || put(cnt, 3.)), put(type, 3.));
    call symput('tvar_cnt', put(cnt, 3.));
```

```
  run;

  %let rename =;

  %do i = 1 %to &tvar_cnt;
    %let vt = &&tvar&i;
    %let nve&i = &vt._n;

    %let rename = &rename %cmpres(&&tvar&i) = &&nve&i;
  %end;

  data &output (keep=&keyvar variable prev_val curr_val);
    merge &inprev
          &incurr (rename=(&rename));
      by &keyvar;

    length variable $32
           prev_val curr_val $256
           suffix $2;
    label variable='Variable Name'
          prev_val='Value in the Previous Version'
          curr_val='Value in the Current Version';

    %do i = 1 %to &tvar_cnt;

      if &&tvar&i NE &&nve&i then do;

        variable = "%cmpres(&&tvar&i)";

        suffix = upcase(substr(trim(left(variable)),
                       length(trim(left(variable))) - 1, 2));

       %if &&tvartyp&i EQ 1 %then
         %do;
           if suffix EQ 'TM' then
             do;
               prev_val = put(&&tvar&i, time5.);
               curr_val = put(&&nve&i, time5.);
             end;
           else if suffix EQ 'DT' then
             do;
               prev_val = put(&&tvar&i, date9.);
               curr_val = put(&&nve&i, date9.);
             end;
           else
             do;
               prev_val = trim(left(put(&&tvar&i, best32.)));
               curr_val = trim(left(put(&&nve&i, best32.)));
             end;

         output;
        %end;
     %else
        %do;
          prev_val = &&tvar&i;
          curr_val = &&nve&i;
          output;
        %end;
      end;
    %end;
  run;

%mend check_for_modified_rows;
```

Step 5: Output the Results

The results from the code in Step 4 can now be combined to form a report that details the differences between the two versions. Options for doing this include using the REPORT procedure with Rich Text Format (RTF) or Portable

Document Format (PDF) Output Delivery System (ODS) destinations or using the Dynamic Data Exchange (DDE) functionality built into SAS to write output directly to an MS Word or Excel document.

```
%macro write_output(struct=structural,
            dup=duplications,
            del=deletions,
            add=additions,
            mod=modifications);

  /* Combine data from input data sets into a report and write to output
     destination of choice */

%mend write_output;
```

CONCLUSION

The methodology presented in this paper provides a useful reference for clinical programmers developing data management tools for the tracking data changes throughout the course of a clinical trial. The initialization file driven approach provides a powerful, flexible model for addressing analysis tasks of this type.

CONTACT INFORMATION

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

Michael Thompson
Clinimetrics Research Associates
2655 North Sheridan Way
Suite 120
Mississauga, Ontario, L5K 2P8
Canada

E-Mail: mike.thompson@clinimetrics.com
Telephone: 905-403-9901 (687)
Fax: 905-403-9083