Comparing 2 SAS® Data Sets: An Alternative to Using PROC COMPARE

Kevin King, Wyeth-Ayerst/Genetics Institute, Cambridge, MA

ABSTRACT

While PROC COMPARE is a useful tool, it can also be intimidating. There are dozens of options to be aware of and the output has several sections. If you were interested in doing a simple comparison, wouldn't it be great if there was a simple tool to do it? Well there is! This paper presents a set of two macros that will get the job done. Using a simple merge statement and the ability of the macro facility to access both base SAS and macro functions, the algorithm compares two SAS data sets and presents the differences in an organized fashion. This can be especially useful when the same data set is changing over time, such as cumulative transfers of data occurring several times during the life of a project.

INTRODUCTION

To compare data sets there are two main steps --- first, determining what variables the data sets have in common; second, comparing the actual values of the variables. Two original macros have been written by the author to accomplish this.

The first part of the paper describes the %comp_var macro, which determines the common variables. The basic principle is to build macro variables containing strings of the data set variable names. The %comp_var macro uses macro code exclusively, it does not contain any data step code.

The second step, the %comp_obs macro, is described in part two. Once the list of common variables has been established, we compare the values using merge statements and create a "problem" data set containing those observations that differ. The call of the macro simply requires inputting the names of the two data sets to compare --- %comp_obs(dsn1,dsn2).

A brief section on limitations of the algorithm is included.

COMPARING THE VARIABLES: %COMP_VAR

The first thing we need to do is check what variables the data sets have in common. Presumably, if you're comparing 2 data sets, you already know they contain the same variables, but this is a good double check. The macro will produce 5 macro variables:

LIST1:	A string of all variables in DSN1
LIST2:	A string of all variables in DSN2
BAD1:	A string of all variables in DSN1, but not in DSN2
BAD2:	A string of all variables in DSN2, but not in DSN1
COMMON:	A string of all variables common to both data sets

First, we generate the list of variables present in DSN1 by using the %sysfunc macro function. This allows you to use SAS functions in macro code. The following section of code opens DSN1 for use, determines the number of variables present using the `attrn` function, and loops through one-by-one to create a string of all DSN1 variables:

```
%let dsid=%sysfunc(open(&dsn1,i));
%let nvar1=%sysfunc(attrn(&dsid,nvars));

%do i=1 %to &nvar1;
 %let name1=%sysfunc(varname(&dsid,&i));
 %let list1=%str( &list1 &name1 );
%end;
```

Each time through the loop the name of the current variable is added to the list. Note the use of the `%str` function to put spaces before and after each variable name. This will be useful later. We then create a second macro variable BAD1, a copy of LIST1, and close the data set:

```
%let bad1=&list1;
%let rc=%sysfunc(close(&dsid));
```

Now, we move on to DSN2. Again, we open the data set and determine the number of variables. In addition, we initialize a new macro variable called COMMON:

```
%let dsid=%sysfunc(open(&dsn2,i));
%let nvar2=%sysfunc(attrn(&dsid,nvars));
%let common=;
```

We are again going to loop through each variable one at a time to create a string of variable names:

```
%do i=1 %to &nvar2;
 %let name2=%sysfunc(varname(&dsid,&i));
 %let list2=%str( &list2 &name2 );
```

There is additional code inside the loop this time, however. The next line searches BAD1, the list of DSN1 variable names, for any occurrence of the current DSN2 variable being processed (preceded and followed by a space):

```
%let pos_a=
  %eval(%index(&bad1,%str( &name2 )));
```

POS_A contains the start position of the variable name in the BAD1 string. If it is not present, the `%index` function will return a zero. Keeping spaces on both sides prevents us from incorrectly finding a variable name that is a part of another variable name (e.g. DRUG and DRUGDAY).

We need to calculate the position of the end of the variable name in case it was found:

```
%let len_var=%length(&name2);
%let pos_b=%eval(&pos_a + &len_var + 1);
```

If the variable name is not in BAD1, we have found a variable in DSN2 that is not in DSN1. We want to add this to the BAD2 string:

```
 %if (&pos_a = 0) %then
  %let bad2=%str( &bad2 &name2 );
```

If we do find the name, we want to remove this variable name from BAD1, since we know it is present in both data sets. We do this using the `%substr` function. We also add the variable name to the list of common variables, which will be stored in COMMON. (Note: There are no spaces between the `%str` and `%substr` functions in the code below. One line has wrapped onto three lines here due to space limitations):

```
 %else %do;
   %let bad1=%str( )
%substr(&bad1,1,&pos_a)%substr(&bad1,&pos_b)
%str( );
   %let common=%str( &common &name2 );
  %end;
```

The removal from BAD1 is actually done by recreating BAD1. We concatenate the section before the start of the current variable name (Position 1 to Position A) with the section after the end of the current variable name (Position B to end). The use of the `%str` function allows spaces to remain on both sides of each variable name. This section of code essentially does what the `%sysfunc(tranwrd())` function could do, but that seemed to have limitations at strings of 256 characters.

As we proceed through each iteration of the loop, we eventually check to see if every variable in DSN2 is in DSN1. At the end, if all the variables are in both data sets, BAD1 and BAD2 will be empty.

Do not forget to end the loop and close the data set. We have now created all five macro variables we need to proceed.

COMPARING THE CONTENTS: %COMP_OBS

The %comp_obs macro actually contains the call to %comp_var inside it. This allows us to localize the macro variables to this block of code. After the %comp_var macro is called and has created our five macro variables, we find ourselves in one of three possible situations:

1) The data sets have no common variables
2) The data sets have some common variables
3) All the variables between the data sets are the same

It does not make much sense to carry on if we have case #1. If so, it is probably best to issue a warning message, list those that are specific to each data set, and stop execution at that point. If either #2 or #3, we can continue. If #2 is the case, we only want to compare the common variables (otherwise every observation will be kicked out as a difference).

The first step is to sort the data sets by every common data set variable. We do this using the macro variable common:

```
   proc sort data=&dsn1(keep=&common);
    by &common;
   run;
```

Sort DSN2 the same way and we are ready to perform the merge that will create the data set containing all the observations that are different. To do this, merge by all common variables and keep every observation that is in one data set but not the other by using the IN= data set option:

```
   data _prob;
     merge &dsn1(in=indsn1) &dsn2(in=indsn2);
     by &common;
     length inds $ 40;
     if indsn1 and not indsn2 then do;
      inds="&dsn1";
      output;
     end;
     if not indsn1 and indsn2 then do;
      inds="&dsn2";
      output;
     end;
    run;
```

If the value of any variable is different from one data set to the other, we will have a non-match and it will be output. The inds variable is created to flag which data set the observation comes from.

If the values are exactly the same between the two data sets, we do get a match when merging and the data set is flagged by the IN= operators as having come from both data sets. We do not output these observations because there are no differences.

Once you have the data set, there are various ways to list the differences. Our code simply uses a proc print. If an observation has been modified, a pair of records is usually listed in the printout directly after, or close to, one another (as long as the changes are not too extensive). The inds variable identifies which data set each observation comes from. Obviously, a completely new observation does not have a matching observation.

We also have some code that checks the _prob data set before the proc print. If _prob contains no observations (i.e. no differences), then a message is issued to the log or output file stating this.

To help organize the output, we have added an optional 3rd parameter to the %comp_obs macro. This allows the user to specify key variables to sort and print by when listing the output. When the output is organized into sections, it is easier to compare the differences.

LIMITATIONS

Some limitations are that the algorithm does not flag duplicates (e.g. 1 record in 1 data set, 10 of the same record in another). Also, varying lengths between the two data sets could potentially result in truncation of some values during the comparison.

Regarding system performance, we have done some minimal testing of the algorithm with SAS version 6.12 on both UNIX and Windows 95. The algorithm seems to handle up to 10,000 observations and 500 variables quite well. Execution time seems to depend more on the number of variables than the number of observations. The sorts and merges are not tremendously time-consuming. Surprisingly, the real hang up seems to be with the %comp_var macro and all its macro code (looping through the list of variables and creating the strings of variable names).

CONCLUSION

While the algorithm could be expanded, the basic idea is established. The major strength is user-friendliness. Invoking the macro is simple and the output of a data set allows the user freedom to display the differences in a variety of ways.

REFERENCES

Macro Facility Enhancements for Release 6.09E and Release 6.11, SAS Institute (Cary, NC), 1996

ACKNOWLEDGMENTS

I want to thank my friends in the SAS programming group at Genetics Institute for their encouragement to prepare this paper, and their help reviewing it.

CONTACT INFORMATION

Kevin King, Wyeth-Ayerst/Genetics Institute
35 Cambridge Park Dr.
Cambridge, MA 02140
Phone: (617) 665-8482, E-mail: kking@genetics.com