Paper 409-2011

Safely Merging Many Datasets

Taylor Young, The Broad Institute of Harvard and MIT, Cambridge, MA, USA

ABSTRACT

When SAS® merges datasets that have variable names in common, only one instance of the variable name is preserved and the last occurrence of a shared variable name overwrites all previously encountered data. As a result, a SAS user working with unfamiliar datasets may unknowingly lose valuable data particularly if the merge involves a large number of datasets and/or variables. Expanding on previously published methods using PROC SQL and the DICTIONARY.COLUMNS table to dynamically rename common variables before merging two datasets, this paper will present methodology to safely merge all datasets in a library by dynamically renaming and relabeling any variables with the same name.

INTRODUCTION

It is a well-documented behavior that when SAS merges two datasets that share a common variable name, the values of the later variable overwrite those of the former. Consequently, the RENAME statement is a point of focus for new SAS users but even the most seasoned SAS user might accidentally overwrite a variable when merging unfamiliar or numerous datasets. Addressing this issue, Chistopher J. Bost previously presented a method that uses PROC SQL and the DICTIONARY.COLUMNS automatic table to automatically rename any variables that are common to the datasets being merged (NESUG 2007). Expanding on Bost's method, which is limited to two datasets, this paper will outline a method to safely merge every dataset in a library by dynamically renaming and relabeling any variables with the same name.

SORTING THINGS OUT

The core of this method is a DO loop that steps through each dataset in a library, sorts the current dataset by the BY variable, and then merges it with the cumulative merged dataset. This will necessitate a special use case for the first dataset encountered but before we get too far along, let's begin by assigning a LIBREF and a macro variable to store the variable name that will be used as the BY variable in the MERGE statement:

```
libname tomerge 'path/to/datasets';

%let mergeby=variable_to_merge_by;
```

Next, we'll use a PROC SQL statement to query the DICTIONARY.COLUMNS automatic table for the distinct memnames in the assigned library, count their number, and use the SELECT INTO statement to store these items as macro variables:

```
proc sql noprint;
    select distinct memname, count(distinct memname) into
    :ds1 - :ds999, :dsnum
    from dictionary.columns where libname=upcase('tomerge');
    quit;
```

Now that the memname of each dataset in the assigned library is stored in a sequential macro variable, each dataset can be sorted individually by referencing the corresponding macro variable from within a DO loop. Here, we want to sort each dataset by the &MERGEBY macro variable:

```
%do i=1 %to &dsnum;
proc sort data=&&ds&i;
    by &mergeby;
    run;
%end;
```

IDENTIFYING SHARED VARIABLE NAMES

With the DO loop framework in place, we can build a PROC SQL statement that queries the DICTIONARY.COLUMNS automatic table for any variable name in the current dataset that also appears in another dataset within the assigned library. Bost originally described much of this method (NESUG 2007) but let's review. First, we'll select the names and labels of all variables in the current dataset:

```
select distinct name, label from dictionary.columns
    where libname=upcase("tomerge") and memname=upcase("&&ds&i")
    and upcase(name) ne "&mergeby";
```

Next, we'll select the names of any variables in the current dataset that appear in any other dataset in the library by nesting an additional select statement in the previously defined statement:

```
added to the end of the previous select statement
and upcase(name) in
    (select distinct name from dictionary.columns
    where libname=upcase("tomerge") and memname ne upcase("&&ds&i") and
    upcase(name) ne "&mergeby");
```

Finally, the overlapping variable names are concatenated with a suffix of the &i counting variable and the variable labels are prefixed with the current memname so that their origin can be easily traced once all datasets are merged. Notice that we are also concatenating the variable name with an "=" sign so that these values, when stored as sequential macro variables, can be executed as RENAME and LABEL statements:

```
select
    trim(name)||'='||left(trim(name))||"_&i",
    trim(name)||'='||"'"||left(trim("&&ds&i"))||":"||
           left(trim(compress(label,"'")))||"'"
    into :rename1 - :rename99999, :label - :label99999 from (previous statement);

%let renamenum = &sqlobs;
```

Here, the final statement stores the number of SQL observations from the previous query as a macro variable that will guide the next phase of the process. The automatic variable SQLOBS is the number of observations the query selected and thus represents the number of variables in the current dataset that share names with other variables in the assigned library.

MERGING MANY DATASETS

The final process will execute the RENAME and LABEL statements on each dataset encountered in the DO loop and merge it with a cumulative, merged dataset. This will require a special case to create the cumulative dataset on the first merge and it also necessitates that the merging begin on the second iteration of the DO loop. First, we'll make a copy of the current dataset and use another DO loop within the data statement to rename and re-label the previously identified variables:

```
data copy_&i;
    set tomerge.&&ds&i;
    %if &renamenum > 0 %then %do;
    %do q=1 %to &renamenum;
           rename &&rename&q;
       label &&relabel&q;
    %end;
    %end;
    run;
```

Next, we'll use an IF/THEN/DO statement to create a specific use case that merges the second dataset in the DO loop with the first:

```
%if &i~=1 and &i=2 %then %do;
data tomerge.merged_data;
    retain &mergeby;
    merge copy_1 copy_&i;
    by &mergeby;
    run;
%end;
```

Finally, an ELSE/DO statement will merge all subsequent datasets with the cumulative dataset:

```
%else %if &i~=1 %then %do;
data tomerge.merged_data;
    retain &mergeby;
    merge tomerge.merged_data copy_&i;
    by &mergeby;
    run;
%end;
```

CONCLUSION

The appendix contains a functional macro of the code outlined in this paper that can be used to safely merge all datasets in a library. Additional consideration should be given to datasets with multiple observations per BY variable value as this may unnecessarily duplicate observations and can also greatly inflate the number of observations in the final merged dataset. It would not be too much trouble to modify this code to include a validation that checks for duplicate BY variable values and to incorporate a list of hazardous datasets that should not be processed by this macro. Finally, if a large number of datasets are to be merged, it would be a good idea to delete the copies of each dataset created in the WORK library either after each iteration of the DO loop or after the DO loop has finished all iterations.

REFERENCES

Bost, Christopher J. 2007. “Automatically Renaming Common Variables Before Merging.” *Proceedings of the North East SAS Users Group 2007 Conference*. Baltimore, MD: NESUG. Available at http://www.nesug.org/proceedings/nesug07/cc/cc06.pdf

SAS Institute Inc. 2007. “Sample 1582: Dynamically rename multiple variables in a SAS data set.” http://support.sas.com/ctx/samples/index.jsp?sid=1582&tab=code

RECOMMENDED READING

- Base SAS® Procedures Guide

CONTACT INFORMATION

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

Name: Taylor Young
Enterprise: The Broad Institute of Harvard and MIT
Address: 7 Cambridge Center
City, State ZIP: Cambridge, MA 02142
E-mail: tyoung@broadinstitute.org
Web: www.broadinstitute.org

```
/*****************************************************

A macro to safely merge all datasets in a library
by renaming and relabeling variables that share
names.

*****************************************************/

/* Open the macro */
%macro merge_library;

/* Assign the libref */
*libname tomerge 'path/to/datasets';

/* Assign the merge by variable to mergeby macro variable */
%let mergeby=var_name;

/* Get all memnames as macro variables */
proc sql noprint;
    select distinct memname, count(distinct memname) into
    :ds1 - :ds999, :dsnum
    from dictionary.columns where libname='TOMERGE' and memname~='MERGED_DATA';
    quit;

/* DO loop for each dataset */
%do i=1 %to &dsnum;

    /* Sort by mergeby variable */
    proc sort data=tomerge.&&ds&i;
           by &mergeby;
           run;

    /* Select duplicate variable names in the current dataset */
    proc sql noprint;
    select trim(name)||'='||left(trim(name))||"_&i",
           trim(name)||'='||"'"||left(trim("&&ds&i"))||":
"||left(trim(compress(label,"'")))||"'"
    into :rename1 - :rename99999, :relabel1 - :relabel99999 from
           (select distinct name, memname, label
           from dictionary.columns
           where libname=upcase("tomerge") and memname=upcase("&&dS&i")
           and not indexw(upcase("&mergeby"),upcase(name))
           and upcase(name) in
                  (select upcase(name)
                  from dictionary.columns
                  where libname=upcase("tomerge") and memname ne upcase("&&ds&i")
                  )
           );
           quit;

           /* Store number of duplicate variables as a macro variable */
           %let renamenum = &sqlobs;

    /* Make a copy and execute rename/label statements */
    data copy_&i;
           set tomerge.&&ds&i;
           %if &renamenum > 0 %then %do;
           %do q=1 %to &renamenum;
                  rename &&rename&q;
                  label &&relabel&q;
           %end;
           %end;
```

```
            run;

        /* Special case for the first merge */
        %if &i~=1 and &i=2 %then %do;

        data tomerge.merged_data;
               retain &mergeby;
               merge copy_1 copy_&i;
               by &mergeby;
               run;

        %end;
        /* Regular case for all subsequent merges */
        %else %if &i~=1 %then %do;

        data tomerge.merged_data;
               retain &mergeby;
               merge tomerge.merged_data copy_&i;
               by &mergeby;
               run;

        %end;

        %put &i &&ds&i;

/* End DO loop */
%end;

/* Close the Macro */
%mend merge_library;

/* Call the macro */
%merge_library;
```