Performing Iterative Processes with the Macro Facility

ABSTRACT
The most basic type of iterative process SAS® offers is the BY statement, which nearly all PROCs support, that performs a separate analysis on a subset(s) of previously sorted data. The structure of the data or task at hand, however, may not allow for this built-in form of iterative programming. For example, computer resource limitations or an externally-contingent nature of requirements could lend way to employing the macro facility to aid in the reporting process. Through brief but widely applicable samples of a %DO loop, a %SCAN routine, and the PROC SQL INTO command, users will take with them the added know-how of building simple iterative macros to accomplish a variety of tasks in the SAS system. You need not be an expert in SAS macro programming to make use of these powerful tools.

BACKGROUND
The data set SURVEY will be used for examples in this paper. SURVEY contains individual responses to a survey with 73 questions (variables Q1 – Q73), with each individual belonging to one of 87 agencies (variable AGENCY with possible values agy1 – agy87), as shown in Figure 1. The objective of the analysis of SURVEY is to calculate statistics, such as weighted percentages and standard errors, on each of the 73 questions for the separate 87 agencies.

![Figure 1 – Data Set SURVEY](image)

The BY statement, which most SAS programmers are familiar with, is the most fundamental method of iterative processing. A simple PROC FREQ with AGENCY in the BY statement can calculate weighted percentages for each agency with the following code:

```sas
proc sort data=survey;
  by agency;
run;
proc freq data=survey;
  by agency;
  weight weight;
  table Q1-Q73;
run;
```

THE %DO LOOP
PROC SURVEYMEANS – AN EXAMPLE OF WHEN A BY STATEMENT CANNOT BE USED
In some situations, however, the BY statement is not appropriate. If our analysis of the survey data becomes more complex and we need to compute standard errors for our weighted percentages, PROC SURVEYMEANS must replace PROC FREQ. PROC SURVEYMEANS is the SAS/STAT procedure that estimates sampling errors in addition to population means. Due to finite sample populations, it is essential that we do not subset data using a WHERE or BY statement, as PROC SURVEYMEANS requires all observations in order to compute accurate statistics for subgroups of data. Instead, we must use the DOMAIN statement to specify the variable that defines the subgroups.
The following code is our first attempt at using the DOMAIN statement to get estimates and standard errors for each agency for all 73 questions in the survey:

```plaintext
proc surveymeans data=survey total=frametotals;
   strata STRATA;
   var Q1-Q73;
   weight weight;
   domain agency;
   ods output domain=outstats;
run;
```

In submitting this code, we learn that our DOMAIN variable, with such 87 possible values, is too computationally intense for our average computer and the system runs out of memory. If our DOMAIN variable has only 2 values, it will run more efficiently. We can use the DATA step to create a variable called SPLIT to cut the data set into two parts: SPLIT='Y' if AGENCY is equal to our agency of interest; else SPLIT='N'. We can then modify the PROC SURVEYMEANS code by replacing AGENCY with SPLIT in the DOMAIN statement. The following code will give us estimates and standard errors for agy1 without overburdening the system:

```plaintext
data split;
   set survey;
   length split $1;
   if agency='agy1' then split='Y';
   else split='N';
run;
```

```plaintext
proc surveymeans data=split total=frametotals;
   strata STRATA;
   var Q1-Q73;
   weight weight;
   domain split;
   ods output domain=outstats_agy1 (where=(split='Y'));
run;
```

In order to obtain statistics for agy2 through agy87, we could painstakingly copy and paste this code 86 times, each time changing agy1 to agyX for our agency of interest. However, the numeric nomenclature of AGENCY lends for an ideal circumstance to use a %DO loop in a macro to save us time and potential mistakes.

%DO LOOP
A %DO loop is a useful feature of the macro facility that iteratively executes a block of code sandwiched between a %DO statement and a %END statement. The syntax for a %DO loop, which must be used in a macro, is as follows:

```plaintext
%do iteration_variable=start_value %to stop_value;
   macro code with references to &iteration_variable
%end;
```

The start value and stop value are integers (or other macro variables that resolve to integers). The iteration variable resolves to the start value in the first loop. In each subsequent loop, the iteration variable increases by a default value of 1 until it attains the stop value (to change the default increment, use a %BY increment_value after stop_value). As with all macro variables, an ampersand must precede any reference to the iteration variable.

The following %DO loop executes the DATA step/PROC SURVEYMEANS block of code 87 times, adding 1 to the numeric suffix of the AGENCY variable in each loop until the iteration variable “agynum” resolves to 87:
%macro agydoloop();
%do agynum=1 %to 87;
data split;
  set survey;
  length split $1;
  if agency="agy&agynum" then split='Y';
  else split='N';
run;

proc surveymeans data=split total=frametotals;
  strata STRATA;
  var Q1-Q73;
  weight weight;
  domain split;
  ods output domain=outstats_agy&agynum (where=(split='Y'));
run;
%end;
%mend agydoloop;

%agyandQdoloop();
%do qnum=1 %to 73;
data split;
  set survey;
  length split $1;
  if agency="agy&agynum" then split='Y';
  else split='N';
run;
%do qnum=1 %to 73;
proc surveymeans data=split total=frametotals;
  strata STRATA;
  var Q&qnum;
  weight weight;
  domain split;
  ods output domain=outstats_agy&agynum._Q&qnum (where=(split='Y'));
run;
%end; /* qnum do loop */
%end; /* agynum do loop */
%mend agyandQdoloop;

The highlighted portions of code point to macro variable references of &AGYNUM. Note that the string containing the first reference is encased double quotes. Remember that double quotes are necessary for a macro variable to be resolved in a quoted string.

MULTIPLE %DO LOOPS
We can also take advantage of the numeric suffixes of the Q1-Q73 variables by inserting another %DO loop within our macro to iterate through question as well. Specifying only one variable as opposed to 73 in the VAR statement will further ease the computational burden of PROC SURVEYMEANS:

%macro agyandQdoloop();
%do agynum=1 %to 87;
data split;
  set survey;
  length split $1;
  if agency="agy&agynum" then split='Y';
  else split='N';
run;

%do qnum=1 %to 73;
proc surveymeans data=split total=frametotals;
  strata STRATA;
  var Q&qnum;
  weight weight;
  domain split;
  ods output domain=outstats_agy&agynum._Q&qnum (where=(split='Y'));
run;
%end; /* qnum do loop */
%end; /* agynum do loop */
%mend agyandQdoloop;

CHECKING PROGRESS WITH THE %PUT STATEMENT
The %PUT statement, which writes text and/or macro variables to the SAS log, is useful for checking the progress of an iterative process. For example, we could add the following %PUT statement between the two %END statements to indicate that PROC SURVEYMEANS has finished looping through all questions Q1-Q73 for an agency:

%put agy&agynum complete;

%SCAN ROUTINE
For instances in which there is no numeric naming convention upon which to declare an iteration, the %DO loop must
be supplemented. Continuing with our survey data example, imagine that agencies are no longer identified as agy1, agy2, ..., or agy87, but rather by a standardized two-digit code, such as ‘ED’ for Education and ‘BO’ for Office of Management and Budget. By assigning a string of such codes to be one parameter in the macro definition, we demonstrate below a %SCAN routine used to run the SURVEYMEANS processes across a given list of agency codes.

```sas
%macro agyscanloop(agylist=);
%let num=1;
%let agy=%scan(&agylist,&num);
%do %while (&agy ne );
   *** PROC SURVEYMEANS Code - uses &agy as the agency code;
   %let num=%eval(&num+1);  *** increase by one the position pointer;
   %let agy=%scan((&agylist,&num);
%end;
%mend agyscanloop;
%agyscanloop(agylist=ED BO CM);
```

The macro %SCAN function is very similar to the SCAN function in BASE SAS; it can parse a macro variable string of characters into two or more substrings based on a given delimiter. The first argument is a macro variable (string of agency codes, in this case) or character constant. The second is a position pointer for which substring to be returned. For instance, %SCAN(&AGYLIST,1) returns the code ED. There is an optional third argument specifying a particular delimiter to use, though SAS will recognize by default a set of commonly used delimiters such as a space, comma, or pipe. Hence, there is no need to explicitly give it in this case.

The first two macro variable assignments are initializations to set the position pointer to the agency code found before the first delimiter. We then embark on a %DO loop routine but use the %WHILE condition as opposed to specifying a start and stop number as we had done earlier. The routine between the %DO and %END statements will be repeated so long as &AGY is not blank. We reason that after the third and final iteration, a position pointer of 4 returned from the %SCAN function will result in a blank value since there are only three substrings possible from delimiting ED BO CM with a space.

PROC SQL INTO CLAUSE

The %SCAN routine works well for processing data for a manageable number of agencies. Recall, however, that we have 87 agencies in total, so keying in all 87 codes is a tedious task prone to error and should be avoided. PROC SQL, the BASE SAS component that invokes SAS’s version of the SQL language, offers a quick way to peek down a column of a dataset and return a macro variable string holding a unique listing of values found. The basic syntax is:

```sql
proc sql;
   select distinct varname into :macro-varname separated by ' ' from datasetname;
quit;
```

VARNAME is the column we want to pull the distinct values from and the macro variable named immediately after the colon is where the string will be stored. To check, we can use a %PUT statement followed by the macro variable name to ensure we achieved the desired result, a sorted list of all values observed. Once confirmed, we can pass this macro variable back through the same form of %SCAN routine demonstrated in the previous section. Rest assured that this technique can be used even with many distinct values since a macro variable string can be a little more than 65,000 characters long in SAS version 9 (32,000 is the cap for versions 6 through 8).

Returning to our example, one reality of a large survey project is that data may be collected in waves. In our case, an updated dataset was submitted once a week which included all previous data plus any new agencies’ data that had since been received. Instead of determining which agencies were new and updating the manually-keyed agency code list, we set up the following routine:
proc sql;
  select distinct agy into :agylist separated by ' ' 
  from survey;
%put &agylist;
quit;

%macro agySQLscanloop();
%let num=1;
%let agy=%scan(&agylist,&num);
%do %while (&agy ne );
  *** PROC SURVEYMEANS Code - uses &agy as the agency code;
  %let num=%eval(&num+1);
  %let agy=%scan(&agylist,&num);
%end;
%mend agySQLscanloop;
%agySQLscanloop();

Notice that the new macro definition is nearly the same except we no longer require a parameter since &AGYLIST is populated via the PROC SQL INTO command.

CONCLUSION
This goal of this paper was to outline a few techniques that can be employed from the macro facility in SAS to automate any kind of data processing. With massive data sets or certain situations where the BY statement is not feasible, we hope that we have demonstrated how programmers can call upon these techniques to make life easier, breaking the task down into smaller chunks and reducing the amount of code necessary to achieve a given result.

REFERENCES

CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the authors at:
  Katie Joseph
  US Office of Personnel Management
  1900 E St, NW, Room 7439
  Washington, DC 20415
  Work Phone: (202) 606-1817
  Fax: (202) 606-1719
  E-mail: Katie.Joseph@opm.gov

  Taylor Lewis
  U.S. Office of Personnel Management (OPM)
  1900 E St., NW, Room 7439
  Washington, DC 20415
  Work Phone: (202) 606-1309
  Fax: (202) 606-1719
  E-mail: Taylor.Lewis@opm.gov

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. © indicates USA registration.
Other brand and product names are trademarks of their respective companies.