ABSTRACT

This paper discusses some of the more advanced macro techniques including:

• Conditionally executing macro code (%IF -- %THEN)
• Repeatedly executing macro code (%DO -- %TO)
• Using Macro Functions
• Interacting with the SAS Data Step
• Indirectly Referencing Macro Variables

After discussing these techniques, an advanced example will be presented that uses these techniques. The example will use macro language and proc freq to automatically generate a macro call for each level of a demographic variable. For example, using one parameter, ‘country’, the macro will produce over 500 macro calls to a second macro.

INTRODUCTION

Starting with simple examples to demonstrate each topic, we will progress to a more difficult example using these topics. It is assumed the audience is familiar with the creation of macros and using macro parameters. It is also assumed that the audience is familiar with the macro commands %LET and %PUT.

The first set of examples will use the following four macros. The first macro is called GETDATA and is used to read in data. The second and third macros, PPLOT and STATS, generate a plot and summary statistics, respectively. The fourth macro, ANALYZE, calls and executes the other three macros.

<table>
<thead>
<tr>
<th>*FIRST MACRO;</th>
<th>*SECOND MACRO;</th>
</tr>
</thead>
<tbody>
<tr>
<td>%Macro GETDATA;</td>
<td>%Macro PPLOT (xvar,yvar);</td>
</tr>
<tr>
<td>data one;</td>
<td>Proc plot data=one;</td>
</tr>
<tr>
<td>run;</td>
<td>Plot &amp;yvar * &amp;xvar;</td>
</tr>
<tr>
<td>%Mend;</td>
<td>Run; %Mend;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>*THIRD MACRO;</th>
<th>*FOURTH MACRO;</th>
</tr>
</thead>
<tbody>
<tr>
<td>%Macro Stats (main,byvar);</td>
<td>%Macro ANALYZE (var1, var2);</td>
</tr>
<tr>
<td>proc freq data=one;</td>
<td>%getdata</td>
</tr>
<tr>
<td>tables &amp;main * &amp;byvar / list;</td>
<td>%plot (&amp;var1,&amp;var2)</td>
</tr>
<tr>
<td>run;</td>
<td>%stats(&amp;var1,&amp;var2)</td>
</tr>
<tr>
<td>proc sort data=one; by &amp;byvar;run;</td>
<td>%Mend;</td>
</tr>
<tr>
<td>proc means data=one;</td>
<td></td>
</tr>
<tr>
<td>var &amp;main;</td>
<td></td>
</tr>
<tr>
<td>by &amp;byvar;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
<tr>
<td>%Mend;</td>
<td></td>
</tr>
</tbody>
</table>

CONDITIONALLY EXECUTING CODE (USING %IF -- %THEN -- %ELSE)

Macros can control when or even if regular SAS code is executed by using the %IF, %THEN and the %ELSE statements. These statements are ONLY valid within a macro and cannot be used in open code.
The syntax is as follows:

\%IF \textit{expression}-- \%THEN \textit{action}--;
\%ELSE \textit{action}--;  
\textbf{\textit{NOTE: the \%ELSE statement is optional;}}

\textit{Expression} can be a comparison or any other regular type of statement that you would put into a data step IF statement. However, using data step variables in a \%IF expression is NOT allowed, only macro variables or constant text can be used.

\textit{Action} is a little different for a macro statement. It can be used to send text to the input stack, to conditionally execute multiple proc statements, or other ways that are impossible to do with typical data step IF statement.

Suppose we wanted to change the macro, ANALYZE, to include a conditional execution of the \%GETDATA macro. We would add \%IF and \%THEN statements, and pass a macro parameter (e.g. RUNDATA) to the ANALYZE macro that tells SAS whether or not we want to run the GETDATA macro.

\begin{verbatim}
This macro code: 
\%Macro ANALYZE (var1,var2,rundata);
   \%IF \&RUNDATA\ eq \&Yes\n   \%THEN \%GETDATA;
   \%pplot (\&var1,\&var2)
   \%stats (\&var1,\&var2)
\%Mend;
\%ANALYZE (revenue,monyy,Yes)
\end{verbatim}

Will send these statements to the input stack:

\begin{verbatim}
 data one;
 set in.yr2004 in.yr2003;
 run;
 proc plot data=one;
   plot monyy*revenue;
 run;
 proc freq data=one;
   tables revenue * monyy / list;
 run;
 proc sort data=one;  by monyy;
 proc means data=one;
   var revenue;
   by monyy;
 run;
\end{verbatim}

\subsection*{USING MACRO FUNCTIONS}

A note of caution: SAS macro code is case sensitive. Therefore, if the parameter fed to the macro ANALYZE is \textit{yes} (note lower case), SAS will return a false for the \%IF comparison. This means the \%GETDATA macro would not be executed. We wish to make our code as user friendly as possible, and do not want to restrict the user to only one way of entering the word \textit{Yes}. So we will handle this in our macro code itself by using a Macro Function called \%UPCASE. This function will uppercase all the letters in the parameter, similar to what the UPCASE function does in regular data step code. Here is our revised code:

\begin{verbatim}
This macro code: 
\%Macro ANALYZE (var1,var2,rundata);
   \%IF UPCASE( \&rundata\) \ eq \&YES\n   \%THEN \%GETDATA;
   \%pplot (\&var1,\&var2)
   \%stats (\&var1,\&var2)
\%Mend;
\%ANALYZE (revenue,monyy,Yes)
\end{verbatim}

Will send these statements to the input stack:

\begin{verbatim}
 Same result as the last example.
\end{verbatim}
Other macro functions that parallel similar functions in data step code include:
%INDEX returns the position of the first place it locates a string of characters in a macro variable
%LENGTH returns the length of a macro variable.
%SUBSTR will subset a string of characters from a macro variable

A few functions only exist as Macro functions. Two very useful macro functions are %EVAL and %SYSEVALF. These functions allow numeric calculations on macro variables (which by definition are always character).
%EVAL performs integer arithmetic
%SYSEVALF performs floating point arithmetic.

For example, using %PUT to print the contents of the macro variable we obtain.

<table>
<thead>
<tr>
<th>Code</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>%let numrecs = 5;</td>
<td>5</td>
</tr>
<tr>
<td>%let count = &amp;numrecs * 6</td>
<td>5 * 6</td>
</tr>
<tr>
<td>%let count = %EVAL (&amp;numrecs * 6)</td>
<td>30</td>
</tr>
</tbody>
</table>

Notice the macro function that does not use %eval does not store the result of the computation, but rather the characters string of the formula.

There are many data step functions that do not exist as macro functions. If you wish to use a data step function in a macro, but you cannot find the corresponding macro function, you can still use the dataset function by using the macro function %SYSFUNC. %SYSFUNC allows you to take any data step function and use it as a macro function.

For example, using %PUT to print the contents of the macro variable we obtain.

<table>
<thead>
<tr>
<th>Code</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>%let dt = %SYSFUNC (date(),mmddyy8.);</td>
<td>11/12/03</td>
</tr>
</tbody>
</table>

Here here here

**CONDITIONALLY EXECUTING BLOCKS OF CODE (USING %DO)**

To conditionally execute multiple statements, use a %DO. This prevents the semicolons from ending the %IF statement. Caution: Similar to the %IF statement, the %DO statement is only valid inside a macro and will generate an error if used in open code. A %DO section is closed with a %END statement.

For example, let’s change our third macro, STATS, to run either the Freq or the Means procedure based on a new macro variable called RUNWHAT. First we need to add a new macro parameter to the definition of the macro. Then two %IF statements need to be added. However, now the ‘action’ for the %IF statement will be a %DO. A %DO statement works very much like a data step DO statement. It marks the beginning of code to be executed. The section is ended with a %END statement. We follow it with a %ELSE statement and another %IF - %THEN %DO statement. The last %ELSE statement will print an error statement if the RUNWHAT parameter is anything other than a ‘F’ or ‘M’.

<table>
<thead>
<tr>
<th>This macro code:</th>
<th>Will send these statements to the input stack:</th>
</tr>
</thead>
<tbody>
<tr>
<td>*THIRD MACRO;</td>
<td>proc freq data=one;</td>
</tr>
<tr>
<td>%Macro Stats (main,byvar,RUNWHAT);</td>
<td>tables revenue * monyy / list;</td>
</tr>
<tr>
<td>%IF %upcase(&quot;&amp;RUNWHAT&quot;) EQ &quot;F&quot;</td>
<td>run;</td>
</tr>
<tr>
<td>%THEN %DO;</td>
<td></td>
</tr>
<tr>
<td>proc freq data=one;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tables revenue * monyy / list;</td>
</tr>
<tr>
<td></td>
<td>run;</td>
</tr>
</tbody>
</table>
tables &main * &byvar / list;
run;
%END;
%ELSE %IF %upcase("&RUNWHAT") EQ "M" %THEN %DO;
  proc sort data=one; by &byvar;
  proc means data=one;
  var &main;
  by &byvar;
  run;
%END;
%ELSE %PUT "Parameter RunWhat is "
  "not an 'F' or 'M';"
%mend;
%STATS (revenue, monyy, f)

This macro call: Will send these statements to the input stack:

%STATS (revenue, monyy, m)

proc sort data=one; by monyy;
proc means data=one;
  var revenue;
by monyy;
run;

ITERATIVELY EXECUTING CODE (USING %DO -- %TO)

To repeatedly execute regular or macro SAS code, the %DO, %TO, and the %END statements may be used.

The syntax is as follows:
%macro x;
  %DO I = 1 %TO 5;
    statements to be repeatedly executed
  %END;
%mend;

There are several ways to use this.

REPEATING TEXT USING MACRO CODE

The iterative Do-Loop can be used to repeat the same or similar program code multiple times. For example, it can be used to produce the following statement:
DATA DSN1 DSN2 DSN3 DSN4 DSN5;

The following code allows for specification of not only the number of datasets, but what the text prefix should be in the names. Note there is no semicolon after the statement in the %do loop.

This macro code: Will send these statements to the input stack:


%macro names (pre, num);
  %DO I = 1 %TO &num;
    &pre&n
  %END;
%mend;

Data %names(dsn,5);
....Other statements;
Run;

Data dsn1 dsn2 dsn3 dsn4 dsn5;
....Other statements;
Run;

MAKING MULTIPLE OUTPUT FILES

The iterative %DO loop may also be used to create filename statements for many output files. For example, to produce a filename statement for each state. The code for the macro will look like the following. Note the semicolon at the end of the filename statement. This differs from the last example because we need a semicolon at the end of each filename statement and in the last example, we did not want a semicolon after each dataset name.

This macro code:  | Will send these statements to the input stack:
---|---
%macro files (name, number);
  %DO n = 1 %TO &number;
    Filenam in&n "c:\&name.ST&n..txt";
  %END;
%mend;
%files(YR2003,52)
  Filename in1 "c:\yr2003ST1.txt";
  Filename in2 "c:\yr2003ST2.txt";
  Filename in3 "c:\yr2003ST3.txt";
  Etc.

REPEATING BLOCKS OF CODE

To repeat multiple SAS statements using the iterative %DO loop. In this example we would like to run the data step twice and do a proc contents on each.

This macro code:  | Will send these statements to the input stack:
---|---
%macro monthly (number);
  %DO n = 1 %TO &number;
    data month&n;
      infile in&n;
      input verbal quant analyt;
      run;
    proc contents data=month&n;
  %END;
%mend;
%monthly (2)
  data month1;
    infile in1;
    input verbal quant analyt;
    run;
  proc contents data=month1;
  run;
  data month2;
    infile in2;
    input verbal quant analyt;
    run;
  proc contents data=month2;
  run;

Note that among the three examples above, the %DO statements do not change very much. What
changes is what is inside the %DO section. This is what causes such variety in how macro code can be used.

DATA STEP INTERFACES – FROM DATA STEP TO MACRO

There are times where you may need to take a variable from a SAS dataset and store its value in a macro variable. The CALL SYMPUT statement allows you to do this. This statement will create one or a series of macro variables and store values from the SAS dataset in them. There are two parameters that you need to supply to the CALL SYMPUT statement.

CALL Symput (macro variable name, macro variable value);

Both the macro variable name and the value can be assigned using constant text (in quotes) or a variable from your data set. Here are some examples:

- This creates a macro variable called “MyName” and will store the value ‘Wendi’ in it.
  Call Symput (’MyName’,’Wendi’);

- This creates one macro variable called ‘Address’ and will store the value of the data set variable address1 from the last record in the macro variable.
  Call Symput (’address’, address1);

- This creates a series of macro variables, one for each testcode, and stores the value from the dataset variable called meanscore for that observation in that macro variable.
  Call Symput (’TC’ || testcode , meanscore);

Let’s look at this last example closer. Assume we have a dataset as shown below. Assume that the variable Testcode is a character variable of length 3.

<table>
<thead>
<tr>
<th>Dataset MEANTEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obs</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Running this code:

Data _null_;  
Set meantest;  
Call Symput (’TC’ || testcode , meanscore);  
Run;

Will produce these macro variables:

<table>
<thead>
<tr>
<th>Macro Variable Name</th>
<th>Macro Variable Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC101</td>
<td>55</td>
</tr>
<tr>
<td>TC102</td>
<td>43</td>
</tr>
<tr>
<td>TC103</td>
<td>76</td>
</tr>
</tbody>
</table>

DATA STEP INTERFACES – FROM MACRO VAR TO DATA STEP VAR
The reverse of what we just discussed is also true. Sometimes you may need to take a value from a macro variable and store it into a variable in a SAS data set. This becomes difficult if the macro variable name you want to get depends on the value of a variable in the dataset. You can do this with the SYMGET statement. The SYMGET statement accesses a macro variable and returns the current value of it. There is only one parameter for the SYMGET statement and that is the name of the macro variable you want to access. Here is an example. Assume that the dataset TEST2 has four observations.

<table>
<thead>
<tr>
<th>First assign values to three macro variables.</th>
</tr>
</thead>
<tbody>
<tr>
<td>%let s1=a;</td>
</tr>
<tr>
<td>%let s2=b;</td>
</tr>
<tr>
<td>%let s3=c;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Then execute this code:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data test2;</td>
</tr>
<tr>
<td>Set test2;</td>
</tr>
<tr>
<td>Length score $4;</td>
</tr>
<tr>
<td>Score = SYMGET ('S'</td>
</tr>
<tr>
<td>Run;</td>
</tr>
</tbody>
</table>

Assuming that Test2 has four observations and that the macro variable S4 does not exist, then the following assignments will take place.

| Data step iteration 1: score = &S1 or ‘a ’. |
| Data step iteration 2: score = &S2 or ‘b ’. |
| Data step iteration 3: score = &S3 or ‘c ’. |
| Data step iteration 4: score = &S4 or ‘ ’.  |

The fourth iteration also produces the following note in the log: “Invalid argument to SYMGET”.

**INDIRECTLY REFERENCING MACRO VARIABLES**

We’ve seen one way to access macro variables by using a data step variable. Macro code can also be used to find the value of a macro variable.

Suppose we have 20 macro variables named city1 to city20. The values of these variables could be listed using a %DO iterative loop and a %PUT statement.

| The following macro code: Will produce the following LOG messages 20 times! |
|--------------------------|--------------------------------------------------------------------------------|
| %macro getcity; %do i = 1 %to 20; %put &city\&n; %end; %mend; %getcity     |
| "WARNING: Apparent symbolic reference CITY not resolved"                    |
| "WARNING: Apparent symbolic reference N not resolved"                      |
| "WARNING: Apparent symbolic reference CITY not resolved"                   |
| "WARNING: Apparent symbolic reference N not resolved"                      |
| "WARNING: Apparent symbolic reference CITY not resolved"                   |
| "WARNING: Apparent symbolic reference N not resolved"                      |
The warnings indicate that SAS is looking for two variables, one called CITY and the other called N. This is not what was intended. We want SAS to interpret the N first, and then go back and interpret the macro variables CITY1, CITY2, etc. To tell SAS to do this, use two ampersands. This is interpreted by SAS as: ‘If you find two ampersands, replace them with one ampersand, process the rest of the string, and then come back to the beginning of the macro variable and start over again’. If we modify the program as noted below, it will then process correctly.

<table>
<thead>
<tr>
<th>This code now works correctly.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%macro getcity;</code></td>
</tr>
<tr>
<td><code>    %do i = 1 %to 20;</code></td>
</tr>
<tr>
<td><code>        %put &amp;&amp;city&amp;n;</code></td>
</tr>
<tr>
<td><code>    %end;</code></td>
</tr>
<tr>
<td><code>%mend;</code></td>
</tr>
<tr>
<td><code>%getcity</code></td>
</tr>
</tbody>
</table>

PUTTING THIS ALL TOGETHER

All these individual examples of macro code can now be used together to create a macro that will create macro calls from the output of a proc freq. Here is how it is done.

Imagine you were asked to run a series of statistical analyses for more than 500 countries in the world. The easiest method is to create a macro that runs the statistical analyses for each country and call the macro once for each country. However, writing more than 500 macro calls would be very tedious. And the next time you are asked to run this, the countries in the data may be different and you would need to retype many of the macro calls.

Proc Freq to get the list of the countries and macro language can be used to generate the necessary macro calls. The technique can be generalized to any variable or variables. It is not limited to just country (e.g. income level, educational level, ethnicity, etc.).

FIRST MACRO THAT CALCULATES THE STATISTICS

First let’s take a quick look at the macro that does the analyses. See Appendix A. . The parameters TC (testcode), VAR (demographic variable) and VALUE (which subvalue of the demographic value to process) are provided in the macro call. And, assume there is a pre-defined format for each variable that has the same name as the variable (preceded with a $ sign).

Inside the analysis macro, the first use of macro code is the selection of the sample and the creation of the macro variable that holds the description of the sample (used later in the title of the final output). This code uses the CALL SYMPUT statement.

<table>
<thead>
<tr>
<th>This code:</th>
<th>Does the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td>data temp;</td>
<td>1. Selects the sample based on the values of the macro parameters provided.</td>
</tr>
<tr>
<td>set one;</td>
<td>2. Creates the macro variable SAMP and places the formatted value of the &quot;&amp;var&quot; variable in it.</td>
</tr>
<tr>
<td>if tc eq &quot;&amp;tc&quot; and &amp;var eq &quot;&amp;value&quot; then do;</td>
<td></td>
</tr>
<tr>
<td>call symput (‘samp’, trim(put(&amp;var,$&amp;var..)));</td>
<td></td>
</tr>
<tr>
<td>output;</td>
<td></td>
</tr>
<tr>
<td>end;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
</tbody>
</table>
1. Uses the Call Symput statement to create the macro variable TCNAME and place that observations value of the dataset variable tcname into the macro variable.

2. Note that it is possible to have both a dataset variable and a macro variable with the same name.

The rest of the macro calculates the statistics and creates the output table. The full program is not provided here. Any code that produces analyses can be substituted.

Sample calls to this macro need to look like:
- `%freqtab (0141034, Gender, Female)`
- `%freqtab (0141034, Gender, Male)`
- `%freqtab (0141034, Country, India)`
- `%freqtab (0141034, Country, France)`
- `%freqtab (0141034, Country, Spain)`

**SECOND MACRO THAT PRODUCES THE MACRO CALLS**

Now we get into the good stuff!

You can see that typing in macro calls for more than 500 countries would be VERY tedious. The next macro will produce these macro calls for us! See Appendix B for the complete macro. There are two parameters used in this macro, the testcode (tc), and the demographic variable you want analyzed. Note that both of these parameters will be used when the calls to the analysis macro are constructed.

The first part calculates a frequency distribution of the demographic variable and stores it in a SAS dataset.

```
proc freq data=one;
  tables &byvar / out=a;
run;
```

If the demographic variable is country, this will get us a saved dataset similar to:

<table>
<thead>
<tr>
<th>Obs</th>
<th>Country</th>
<th>Count</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>India</td>
<td>15</td>
<td>.03</td>
</tr>
<tr>
<td>2</td>
<td>France</td>
<td>25</td>
<td>.07</td>
</tr>
<tr>
<td>3</td>
<td>Spain</td>
<td>18</td>
<td>.05</td>
</tr>
</tbody>
</table>

The next portion of the code uses the output dataset from Proc Freq and does two things.

First, a macro variable called 'TOTALN' is created. This macro variable holds the total number of observations in the dataset. Note the use of the SET statement option of NOBS to get the total number of observations. We only need to assign the macro variable once, so this is only done for the first observation.

Second, the code creates a series of macro variables called f1-fxx where xx is the number of observations in the dataset. There will be one macro variable for each observation. The value stored in each of these macro variables is the value of the demographic variable.

```
data a;
  set a;
```
if _n_ eq 1 then call symput ('totaln',left(numobs));
call symput ('f'||left(_n_),&byvar);
run;

Assuming the resulting dataset above produced 540 observations, the macro variables produced and their respective values will be:

<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Macro Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTALN</td>
<td>540</td>
</tr>
<tr>
<td>F1</td>
<td>France</td>
</tr>
<tr>
<td>F2</td>
<td>India</td>
</tr>
<tr>
<td>F3</td>
<td>Spain</td>
</tr>
<tr>
<td>Etc up to F540</td>
<td>The 540th country</td>
</tr>
</tbody>
</table>

The last part of the macro creates and executes the actual macro calls.

First it checks to see if there were any observations in the dataset. If not, no macro calls will be created,

Then it cycles through each macro variable (up to the number of TOTALN) and creates a macro call for each. The first two parameters of the call constructed are part of the parameters passed to this macro. The last parameter is provided by the macro variables created from the step above. Remember that the double ampersand is interpreted as one ampersand, the program will replace &j with the correct loop number, and then the program goes back and resolves the final macro variable &Fj where j is the iteration number.

This code:

Will be interpreted by the macro processor as:

<table>
<thead>
<tr>
<th>This code:</th>
<th>Will be interpreted by the macro processor as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>%if &amp;totaln ne 0 %then %do;</td>
<td>• Since &amp;totaln = 540, this IF statement resolves to true and the next statement is processed.</td>
</tr>
<tr>
<td>%do j=1 %to &amp;totaln;</td>
<td>• This do loop will run through 540 times.</td>
</tr>
<tr>
<td>%freqtab (&amp;tc,&amp;byvar,&amp;&amp;F&amp;j)</td>
<td>• Using the %Repeat macro call given at the left, the macro calls created will look like:</td>
</tr>
<tr>
<td>%end;</td>
<td>%freqtab (0131024,Country,France)</td>
</tr>
<tr>
<td>%end;</td>
<td>%freqtab (0131024,Country,India)</td>
</tr>
<tr>
<td>Call to repeat macro is: Repeat (0131024, Country)</td>
<td>%freqtab (0131024,Country,Spain)</td>
</tr>
<tr>
<td></td>
<td>etc.</td>
</tr>
</tbody>
</table>

CONCLUSION

I hope the techniques provided here will prove useful to you in the future. These topics are just a few examples of the power of macro programming. I encourage you to read some of the recommended readings below for additional information. Or, if you have any questions on any of these topics, please feel free to contact me.

To learn more, I would recommend the SAS Macro Language: Reference. The introduction in this book provides a similar overview to macro processing as this paper. Another very good book if you want to learn more about macros is Art Carpenter's book called "Carpenter's Complete Guide to the SAS Macro Language". His book has many examples and includes exercises at the end of most chapters (with solutions at the end of the book).
SAS and SAS/GRAPH are registered trademarks or trademarks of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.

AUTHOR CONTACT

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

Wendi L. Wright
Educational Testing Service, 11-L
Princeton, NJ 08541
Phone: (609) 683-2292
Fax: (609) 683-2130
E-mail: wwright@ets.org
Appendix A

%MACRO FREQTAB (tc, var, value);
   %* SELECT THE SAMPLE AND
   CREATE A MACRO VARIABLE THAT HOLDS THE DEFINITION OF THE
   SAMPLE (TO INCLUDE IN THE TITLE).;
   %* NOTE: You will need a previously defined format for the
   'var' variable. For example, if your variable is named
   'edlevel', then you will need a format name $edlevel.;
   data temp;
      set one;
      if tc eq "&tc" and var eq "&value" then do;
         call symput('samp',trim(put(var,$&var..)));
         output;
      end;
   run;

   %* CREATE A MACRO VARIABLE THAT HOLDS THE TEST NAME
   TO INCLUDE IN THE TITLE;
   data tcname;
      set tcname;
      if tc eq "&tc" then call symput('tcname',trim(tcname));
   run;

   %* CALCULATE THE FREQ DISTRIBUTION, AND THE PERCENT BELOW;
   %... various SAS statements ...

   %* SORT FREQ DIST;
   %... various SAS statements ...

   %* CALCULATE THE N, MEAN, AND THE STANDARD DEVIATION;
   %... various SAS statements ...

   %* PRINT OUT THE FREQUENCY DISTRIBUTION AND THE SUMMARY STATS;
   %... various SAS statements ...

%MEND freqtab;
Appendix B

* This macro will execute a macro multiple times based on the number of subgroups in the byvar. (e.g. if byvar is gender, will execute the other macro once each for male and for female);

%macro repeat (tc,byvar);
  %* CALCULATE A LIST OF ALL LEVELS OF THE DEMOGRAPHIC VARIABLE;
  proc freq data=one;
    tables &byvar / out=a;
  run;

  % *CALCULATE TWO TYPES OF MACRO VARIABLES:
  1. TOTALN (one macro variable)= has the number of observations in the output dataset from freq above (or the number of different values found in the demographic variable.
  2. F# - has one macro variable for each observation in the dataset 'a' - output from freq above. Will hold the value (e.g. male) of the demographic variable that occurs in that observation.

  data a;
    set a;
    if _n_ eq 1 then call symput ('totaln',left(numobs));
    call symput ('f'||left(_n_),&byvar);
  run;

  %* Using the TOTALN and the 'X#' macro variables created above, call the macro ctable once for each level of the by-variable.;
  %if &totaln ne 0 %then %do;
    %do j=1 %to &totaln;
      %freqtab (&tc,&byvar,&&x&j)
    %end;
  %end;
%mend;