PharmaSUG 2022 - Paper AP-171

Self-Modifying Macro to Automatically Repeat a SAS Macro Function

Chary Akmyradov Ph.D., Arkansas Children's Research Institute;

ABSTRACT

SAS Macro language makes the workflow easier. Writing macro functions to shorten a code and to obtain the analysis is a time saver. However, if there are hundreds of variables to plug in a macro function or if the same macro function is repeated by changing one of its arguments, then it is a time consuming process. In this paper, I demonstrate how to develop a SAS macro function using %do and %while loops with %scan and %eval functions to run any macro function automatically over a list of values.

INTRODUCTION

In healthcare data, a task can be to create a table with an outcome variable versus the covariates. This can be demographics table representing the overall picture of the patients in the study, or this can be an odds ratio table with p-values for the risk factors of an outcome. The number of risk factors can be 10, 50, or even more than 100. Let us assume we have a SAS Macro function, which creates a row of your report, such as Gender frequency distribution over the outcomes with a Chi-square or Fisher's exact test p-value. Of course, one daunting task will be running that macro function for a hundreds of variables. Using the following steps, I demonstrate how to run a macro function over a list of variables.

- Obtain the list of categorical and continuous variables
- Create a shell dataset to insert the results from each macro function run
- Develop a macro function and test it
- Run your macro function using the list of variables iteratively
- Export the report as a CSV file.

For this paper, I will use a publicly available data set, `SASHelp.Heart` to demonstrate the steps. `SASHelp.Heart` dataset provides results from the Framingham Heart Study with 5209 observations and 17 variables.

OBTAINING YOUR MACRO LIST

The simplest way to write your macro list is, typing it in the following way:

```
%let variable_list=  first_variable second_variable   …  last_variable ;
```

This can be also useful if I want to test my macro over a short list.

Another way of getting the list of variable names saved into a macro list is by exporting the variable names using proc contents and then using the into clause of proc sql:

```
proc contents   data = sashelp.heart(drop = status)
                out  = varinfo(keep = name type) noprint;
run;
```

Since `status` is an outcome variable, I dropped it and will use it later in the macro function. Above `proc contents` saves the table of variables and types as a temporary dataset `varinfo`.

I will save the categorical variables (`type=2`), and numeric variables (`type=1`) into macro lists using `proc sql`.

```
proc sql noprint;
    select name into : cat_list separated by ' '
          from varinfo
                where type = 2;
    select name into : con_list separated by ' '
          from varinfo
                where type = 1;
quit;
```

Another method would be using `call symputx` and `call execute` routines:

```
%let cat_list=;
%let con_list=;
data one;
    set varinfo;
if type=2 then
    do;
          call symputx('list2',strip(name));
          call execute('%let cat_list=&cat_list &list2');
    end;
if type=1 then
    do;
          call symputx('list1',strip(name));
          call execute('%let con_list=&con_list &list1');
    end;
run;
```

SHELL TABLE

A shell table is a zero observation table with specific column names, variables' length and formats. In the following proc sql I created an empty table named as bivariate_compare_data.

```
proc sql;
   create table bivariate_compare_data(
             Variable    char(50), /*Variable names for each row*/
             Subcategory char(50), /* Variable levels such as Yes/No */
             Dead        char(19), /*Summary such as Mean(Std) or N(percent)*/
             Alive       char(19), /*Summary such as Mean(Std) or N(percent)*/
             Total       char(19), /*Total observations N*/
             P_value     num   label='P-value'  /*P-value from chi-sq etc.*/
             );
quit;
```

DEVELOPING YOUR MACRO

Since I have the list of both continuous and categorical variables I need a macro functions to summarize those variables in a table. I have developed a macro `%chi_freq` to summarize categorical variables with variable name, its levels, number and percent of dead and alive, total numbers and Chi-square p-values. To be able to use this macro within the iterator macro I needed to assign the default values to some of the parameters. This macro is available in the appendix.

```
%chi_freq (dsn=sashelp.heart      /*dsn is dataset name*/
           ,var=                  /*categorical variable*/
           ,comparevar=status     /*an class variable*/
                          );
```

AUTOMATION STEP

Now I have a macro function, a macro list, and I am ready to iterate through the variables. The automating macro, which I call it as iterator takes three parameters. First, one is the `varlist,` which is a macro variable containing list of values. The second parameter is our macro function name without the percent sign. To avoid errors, I define the macro functions with default values other than the parameter that I want to iterate through the list. The third one takes the parameter name from the macro function to iterate through.

I start defining local macro variables `&varidx` and `&v`. Local macro variables will reset at each iteration. `&varidx` is an index macro which I use in the `%scan` function to assign the macro list members to `&v`, which is passed into the macro funtion. The `%do %while` loop will keep running while the &`varidx` index matches with the &varlist member's position. At each iteration`&varidx` is incremented by 1, so we can save the next member of the list to `&v`.

```
%macro iterator( /*runs macrofn macro function over varlist*/
                varlist     /*list to operate through*/
                ,macrofn    /*macro function name, omit the percent sign */
                ,variable  /*macro function's parameter to plug in*/
                );
%local varidx v;
    %let varidx=1;
    %do %while (%scan(&varlist,&varidx) ne);
          %let v=%scan(&varlist,&varidx);
                          %&macrofn.(&variable=&v);
          %let varidx=%eval(&varidx+1);
    %end;
%mend iterator;
```

APPLICATION

Now we have the variable list, macro function to iterate, and the iterator are prepared and ready to use, let's put all together.

First, I need to initialize the shell table, and to make things easier, and obtain an empty dataset named as bivariate_compare_data . This name is important, because it should match with the name of the table at the end of the macro function %chi_freq, because the iterator will be repeatedly running the %chi_freq which will write the results to bivariate_compare_data at each step. Another caution is the column names Dead and Alive should be changed according to your class variable level. Assuming we already have the list of categorical variables saved into &cat_list we can run the following line to obtain the final bivariate_compare_data table:

```
%iterator(&cat_list, chi_freq, var);
```

Finally, we can use proc export to export the resulting table as a .csv file.

CONCLUSION

The goal of this paper is to demonstrate how a macro function can be applied through a long list of variables. The codes explained in this paper is straightforward and can be modified to adapt in every situation. I use positional parameters, but it can be changed into keyword parameters and two different iterator can be used within each other for multi-step iterations. One can also put another macro variable similar to `v` within the `%iterator` macro to scan through another macro list parallel to the original list.

REFERENCES

Performing Iterative Processes with the Macro Facility, Katie Joseph, Taylor Lewis NESUG2007 (https://www.lexjansen.com/nesug/nesug07/cc/cc21.pdf)

List Processing Basics: Creating and Using Lists of Macro Variables, Ronald J. Fehd, Art Carpenter (https://www.mwsug.org/proceedings/2009/how/MWSUG-2009-H01.pdf)

Burlew, Michele M., SAS® Macro Programming Made Easy, Cary, NC: SAS Institute Inc., 2001.

SAS Institute Inc. Base SAS® 9.4 Procedures Guide. Cary, NC: SAS Institute Inc.

SAS Institute Inc. SAS/STAT® 9.4 User's Guide. Cary, NC: SAS Institute Inc.

CONTACT INFORMATION

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

Chary Akmyradov, Ph.D.
Arkansas Children's Research Institute
13 Children's Way,
Slot 842,
Little Rock, Arkansas, 72202
(501)364-2107
AkmyradovC@archildrens.org
web: akmyradovc.com

TRADEMARK

APPENDIX

This macro creates frequency table with Chi-square p-values.

```
%macro chi_freq(dsn=sashelp.heart /*dsn is dataset name*/
                      ,var=             /*categorical variable*/
                      ,comparevar=status/*a class variable*/
                            );
ods output
          chisq=pval_(keep =statistic
                            prob
                            warning );
proc freq data = &dsn.;
    table &var.*&comparevar. / norow chisq;
run;
ods output
    crosstabfreqs=freqs(keep =&var
                            &comparevar.
                            frequency
                            colpercent
                            percent
                            _type_);
proc freq data = &dsn.;
    table &var.*&comparevar. /missing norow;
run;
ods output close;

data freqs;
    retain &var.;
    length &var. $50;
    set freqs;
run;
proc sql;
          create table stats as
          select
                case
                      when strip(&var.)=' ' then 'Missing'
                      else &var.
                end as &var.,
                case
                      when _type_='11' then &comparevar.
                      when _type_='10' then 'T'
                      else ''
                end as &comparevar.,
case
when _type_='11' then put(frequency,8.)||' ( '||put(colpercent,F6.2)||'%)'
when _type_='10' then put(frequency,8.)||' ( '||put(Percent,F6.2)||'%)'
                  else ''
                end as &comparevar._
                      from freqs
                            where _type_ in ('10' '11')
```

```
                                    order by &var.;
quit;
proc transpose data = stats out = stats2 (drop = _name_ );
    id &comparevar.;
    var &comparevar._ ;
    by &var;
run;
proc sql noprint;
    select distinct &comparevar into: comparevarlist separated by ' '
          from freqs
                where &comparevar ne ' '
                      order by &comparevar;
quit;
proc sql;
    create table stats_2(drop=ordermiss) as
          select "&var." as Variable,
                      &var. as Subcategory ,
                      %scan(&comparevarlist,1),
                      %scan(&comparevarlist,2),
                      T as Total,
                      case
                            when strip(Subcategory)='Missing'   then 1
                                                                else 0
                      end as ordermiss
                from stats2
                      order by ordermiss;
quit;
proc sql;
          create table pvals as
                select Prob as p_value format=pvalue6.4
                      from Pval_
                            where statistic='Chi-Square';
quit;
data final_;
    merge stats_2 pvals;
    if _N_ gt 1 then do; Variable=''; end;
run;
data bivariate_compare_data;
    set bivariate_compare_data final_;
run;
proc datasets lib = work memtype = data nodetails;
    save bivariate_compare_data;
quit;
%mend;
```