Ruchita Trivedi

ZX-4954

Abstract

A practical usage of SAS Macros in day to day office work. The one s that I would always need to have in my macros library .

UTILITY BASED MACRO EXAMPLES

***Contents***

[***Example 1 : Macro for Data Dependent Titles 2***](#_Toc459814131)

[***Example 2: Macro for Triggering a PROC based on a data step value 4***](#_Toc459814132)

[***Example 3 : Macro for Splitting a data set. 6***](#_Toc459814133)

[***Example 4: Macro for Getting Variable Names from Dataset 10***](#_Toc459814134)

[***Example 5: Macro for Counting Observations 11***](#_Toc459814135)

[***Example 6: Macro to Test Normality 12***](#_Toc459814136)

[***Example 7: Macro for Renaming Variables Dynamically 15***](#_Toc459814137)

***ABSTRACT***

Numerous utilities have been created using the SAS Macro Language. The examples of macro utilities that I have presented here include the use of:

* **SASHELP views**
* **%DO/ %END loops**
* **Advanced SYMPUT/ SYMGET**
* **DATA step and SCL functions using %SYSFUNC**
* **the AUTOCALL library**
* **system supplied AUTOCALL macros**

## 

## **Example 1 : Data Dependent Titles**

/\* ------------------------------------------------

Create a data dependent title.

-------------------------------------------------\*/

**proc** **means** data=SASUSERS.DEMOG noprint mean;

var age;

output out=average mean=avage;

**run**;

**data** \_null\_;

set average;

call symput('meanage',left(put(avage,**2.**)));

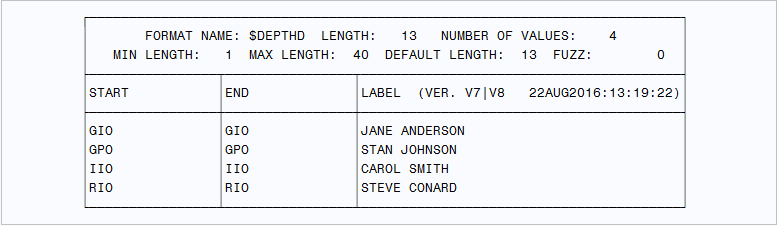
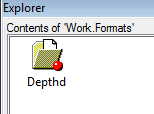
**run**;

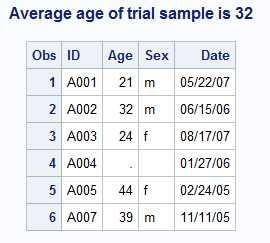
**proc** **print** data =SASUSERS.DEMOG;

title "Average age of trial sample is &meanage";

**RUN**;

Explanation: The MEANS procedure produces a one-variable (avage), one-observation data set (average). The DATA \_NULL\_ step produces no output SAS data set, but is simply a vehicle for getting an independently executable step that will transfer the value of avage into a macro variable meanage. This, of course, will go into the global symbol table. Once the DATA \_NULL\_ step has completed, the macro variable is available for use, and is simply used in a usual TITLE statement. The output produced is:





## **Example 2: Triggering a PROC based on a data step value**

/\* ------------------------------------------------------

Triggering a PROC based on a data step value

------------------------------------------------------\*/

options mprint;

**%macro** ***archive***;

%if &append ^=**0** %then %do;

proc append base=arch data=oldrecs;

run;

%end;

%else %put No archiving to be done;

**%mend** archive;

**data** new;

input @**1** date date7. reading1 reading2;

cards;

01sep91 102 150

19aug91 98 143

05MAY90 98 142

07MAY90 90 140

21aug91 88 135

11MAY90 84 134

**run**;

**data** oldrecs;

set new;

if today() -date > **30** then output oldrecs;

**run**;

**data** \_null\_;

if **0** then set oldrecs nobs=numobs;

call symput('append',left(numobs));

stop;

**run**;

%***archive***

Explanation: MPRINT has been turned on to show the statement generated in the Log. The idea is to archive observations more than 30 days old. This example could be adapted to any dynamic file (say one under FSEDIT control) where it was important to get rid of old observations into some archive or backup file. The logic behind this routine is:

a. Create a data set of those observations more than 30 days old (oldrecs)

b. Determine the number of observations in oldrecs and pass the value to the macro variable append.

c. In the macro execution test the value of append. Only if non-zero run the Append Procedure.

DATA \_NULL\_ step has been used to create the macro variable with CALL SYMPUT. The line:

if 0 then set oldrecs nobs=numobs;

uses the fact that the variable assigned to the nobs option is given its value at data step compile time. Therefore numobs holds the number of observations in the data set oldrecs without having to read an observation from it. Hence the dummy negative condition if 0. The stop is necessary. The normal way of terminating a data step is to reach an end-of-file marker on a raw data file or a SAS data set; if this is not present a STOP stops the data step trying to loop. However, the whole point of the example is to test the number of observations in oldrecs. Only when this is greater than zero is the PROC APPEND step generated.

Part of Log:

MPRINT(ARCHIVE): proc append base=arch data=oldrecs;

MPRINT(ARCHIVE): run;

NOTE: Appending WORK.OLDRECS to WORK.ARCH.

NOTE: BASE data set does not exist. DATA file is being copied to BASE file.

NOTE: There were 6 observations read from the data set WORK.OLDRECS.

NOTE: The data set WORK.ARCH has 6 observations and 3 variables.

## **Example 3 - Splitting a data set.**

Consider the following extract from the data set Chap1.Glow\_11m, concerning the probability of getting fracture. Suppose we wish to take a data set such as work.ds1 which has multiple observations for each age of patients , and construct a data set for each age - to split up the data set by each age of patient for others to do separate analyses for all ages of patients.

To do this in normal open code would require a data step of the form:

Data

age20 age24 age25....;

set chap1.glow\_11m;

if age=20 then output ds20;

else if age=24 then output ds24;

. . run;

The problems here are:

● The number of output data sets can vary

● The age variable is numeric, so a character prefix is required for the output data set name

● The first IF statement is plain, all the others need an ’ELSE’. Below is the macro solution to all these problems.

/\* ----------------------------------------------------

Splitting a data set.

------------------------------------------------------\*/

**%macro** split(inputds,byvar,prefix);

proc freq data=&inputds;

tables &byvar / noprint out=numbys(keep=&byvar);

run;

data \_null\_;

set numbys end=eof;

call symput('mvar'||left(put(\_n\_,**2.**)),

left(put(&byvar,**3.**)));

if eof then call symput('numobs',put(\_n\_,**2.**));

run;

data

%do i = **1** %to &numobs;

&prefix&&mvar&i

%end;

;

set &inputds;

%let else=;

%do i= **1** %to &numobs;

&else if &byvar=&&mvar&i then

output &prefix&&mvar&i;

%let else=else;

%end;

run;

**%mend** split;

%***split***(chap1.glow\_11m,age,ds)

Explanation: Note the parameters are:

● the input data set

● the variable to be used for the split

● the prefix to the output data set names The output from the original PROC FREQ step gives one observation per value of &byvar.

The DATA \_NULL\_ step creates multiple macro variables, one per each observation of the output data set from the PROC FREQ step. The second CALL SYMPUT gives a macro variable containing the number of observations in the data set, so as to give a variable ceiling to the subsequent loops.

Because the number of observations is variable, the number of macro variables generated is variable, and indirect reference to them must be used. Note that the first %DO loop generates a variable DATA statement, the second generates the IF..THEN..ELSE statements, omitting the ELSE from the first one.

The log below shows the datasets generated for each age, the contents of work library has datasets for each age . Also an example dataset for age 63 is shown below.

Part of log (for example 3)

NOTE: There were 238 observations read from the data set CHAP1.GLOW\_11M.

NOTE: The data set WORK.DS56 has 8 observations and 16 variables.

NOTE: The data set WORK.DS57 has 8 observations and 16 variables.

NOTE: The data set WORK.DS58 has 6 observations and 16 variables.

NOTE: The data set WORK.DS59 has 4 observations and 16 variables.

NOTE: The data set WORK.DS60 has 4 observations and 16 variables.

NOTE: The data set WORK.DS61 has 10 observations and 16 variables.

NOTE: The data set WORK.DS62 has 10 observations and 16 variables.

NOTE: The data set WORK.DS63 has 6 observations and 16 variables.

NOTE: The data set WORK.DS64 has 4 observations and 16 variables.

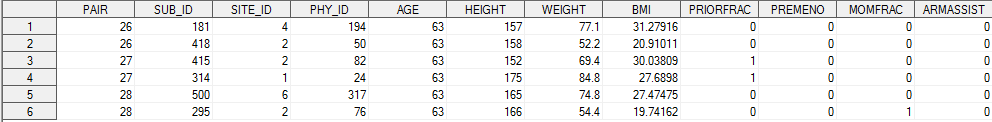
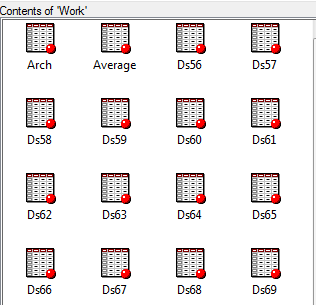
NOTE: The data set WORK.DS65 has 20 observations and 16 variables.

NOTE: The data set WORK.DS66 has 4 observations and 16 variables.

NOTE: The data set WORK.DS67 has 10 observations and 16 variables.

NOTE: The data set WORK.DS68 has 4 observations and 16 variables.

NOTE: The data set WORK.DS69 has 6 observations and 16 variables.



## **Example 4: GET VARIABLE NAMES FROM A DATASET**

Suppose you want to create a macro variable that puts all the variable names from a data set. The following simple Proc Sql program does that. Below i have used ds89 dataset (generated from sub setting macro above) to demonstrate its use . The only thing to note is

a. Make sure library and dataset names in CAPS. Or you can use UPCASE function to make it in caps.

b. Also Libname is the library name and Memname is the member name. To see the variable names, use the %put statement:

\*Example 5: Selecting all the variables;

**proc** **sql** noprint;

select name into : vars separated by " "

from dictionary.columns

where LIBNAME = upcase("work")

and MEMNAME = upcase("ds89");

**quit**;

%put variables = &vars.;

Part of log:

400 %put variables = &vars.;

variables = PAIR SUB\_ID SITE\_ID PHY\_ID AGE HEIGHT WEIGHT BMI PRIORFRAC PREMENO MOMFRAC

ARMASSIST SMOKE RATERISK FRACSCORE FRACTURE

## **Example 5: COUNTING OBSERVATIONS**

In the following example the macro %OBSCNT acts like a macro function in that the macro call resolves to a value that is the number of observations in the stated data set.

**%macro** obscnt(dsn);

%local nobs;

%let nobs=.;

%\* Open the data set of interest;

%let dsnid = %sysfunc(open(&dsn));

%\* If the open was successful get the; A and then calls %***OBSCNT*** to write the

%\* number of observations and CLOSE;

%\* &dsn;

%if &dsnid %then %do;

%let nobs =

%sysfunc(attrn(&dsnid,nobs));

%let rc =%sysfunc(close(&dsnid));

%end;

%else %do;

%put Unable to open &dsn - ;

%put %sysfunc(sysmsg());

%end;

%\* Return the number of observations;

&nobs

**%mend** obscnt;

Explanation:

a. The user passes the name of the data set (&DSN) into the macro.

b. The selected data set is opened and is assigned an identification number which is stored in &DSNID.

c. If the data set was found and opened successfully &DSNID will be greater than 0 and this %IF expression will be true.

d. The ATTRN function is used to determine the number of observations in the data set. The ATTRN function can be used to make a number of queries on the data set once it is opened. These include password and indexing information as well as the number of variables and the status of active WHERE clauses.

e. The data set should be closed after retrieving the desired information.

f. When the open is unsuccessful we may want to write a message to the LOG. The SYSMSG() function returns the reason the OPEN failed.

g. Since this is the last statement in the macro, the resolved value of &NOBS will be effectively 'returned' to the calling program and its value will be a period (.) if the data set was not opened successfully.

The following program creates the data set A and then calls %OBSCNT to write the number of observations to the LOG.

**data** a;

do i = **1** to **10**;

x=i\*\*i;

output;

end;

**run**;

%put number of obs is %obscnt(a);

Log:

NOTE: The data set WORK.A has 10 observations and 2 variables.

NOTE: DATA statement used (Total process time):

real time 0.01 seconds

cpu time 0.01 seconds

Log: 441 %put number of obs is 10

## **Example 6: TEST FOR NORMAL DISTRIBUTION**

\*SAS Macro for Normality;

**%macro** normal(input=, vars=, output=);

ods output TestsForNormality = Normal;

proc univariate data = &input normal;

var &vars;

run;

ods output close;

data &output;

set Normal ( where = (Test = 'Shapiro-Wilk'));

if pValue > **0.05** then Status ="Normal";

else Status = "Non-normal";

drop TestLab Stat pType pSign;

run;

**%mend**;

%*normal*(input=ds80, vars= WEIGHT, output=Normality);

Explanation:

In most of the statistical tests, you need check assumption of normality. There is a test called Shapiro-Wilk W test that can be used to check normal distribution. If the p-value is greater than .05, it means we cannot reject the null hypothesis that a variable is normally distributed.

Below is an example of code used to investigate the distribution of a variable.  In our example, we will use the ds80 dataset( generated from sub setting macro above) and we will investigate the distribution of the continuous variable WEIGHT **and HEIGHT** of patients of age 80 **using Proc Univariate**,. The PROC UNIVARIATE statement is required to invoke the UNIVARIATE procedure. You can use the PROC UNIVARIATE statement by itself to request a variety of statistics for summarizing the data distribution of each analysis variable:

* sample moments
* basic measures of location and variability
* confidence intervals for the mean, standard deviation, and variance
* tests for location
* tests for normality
* trimmed and Winsorized means
* robust estimates of scale
* quantiles and related confidence intervals
* extreme observations and extreme values
* frequency counts for observations
* missing values

Part of log:

NOTE: There were 2 observations read from the data set WORK.NORMAL.

WHERE Test='Shapiro-Wilk';

NOTE: The data set WORK.NORMALITY has 2 observations and 4 variables.

NOTE: DATA statement used (Total process time):

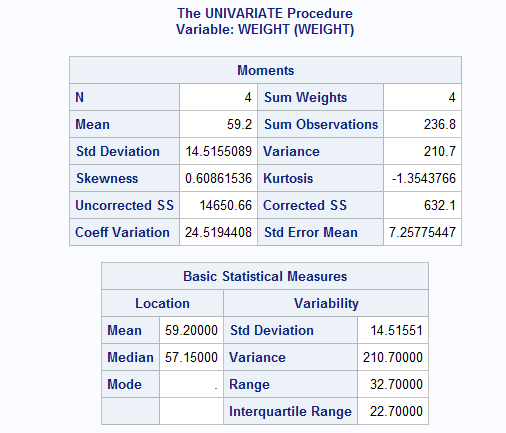
real time 0.02 seconds

cpu time 0.01 seconds

Contents of Normality:



Here the p-values for both the variables is more than .05, which indicates normatily.



## **Example 7: SAS macro programs for renaming variables dynamically**

We have a list of variables and a list for the new names of these variables.  In the example below, we want to rename variablesfaminc1andfaminc2to be aandbfor no particular reason.

**%macro** rename1(oldvarlist, newvarlist);

%let k=1;

%let old = %scan(&oldvarlist, &k);

%let new = %scan(&newvarlist, &k);

%do %while(("&old" NE "") & ("&new" NE ""));

rename &old = &new;

%let k = %eval(&k + 1);

%let old = %scan(&oldvarlist, &k);

%let new = %scan(&newvarlist, &k);

%end;

**%mend**;

**data** faminc;

input famid faminc1-faminc12 ;

cards;

1 3281 3413 3114 2500 2700 3500 3114 -999 3514 1282 2434 2818

2 4042 3084 3108 3150 -999 3100 1531 2914 3819 4124 4274 4471

3 6015 6123 6113 -999 6100 6200 6186 6132 -999 4231 6039 6215

;

**run**;

**data** a ;

set faminc;

%***rename1***(faminc1 faminc2, a b);

**run**;

**proc** **print** data = a heading= h noobs;

**run**;

Explanation: The highlights of this macro are the %do %while loop and scan function.

%DO %WHILE loop take be any macro expression that resolves to a logical value. The macro processor evaluates the expression at the top of each iteration. The expression is true if it is an integer other than zero. The expression is false if it has a value of zero. If the expression resolves to a null value or to a value containing nonnumeric characters, the macro processor issues an error message.

**SCAN**(string,n,delimiters): returns the nth word from the character string string, where words are delimited by the characters in delimiters.    
It is used to extract words from a  character value when the relative order of words is known, but their starting positions are not.

Log:

NOTE: There were 3 observations read from the data set WORK.FAMINC.

NOTE: The data set WORK.A has 3 observations and 13 variables.

NOTE: DATA statement used (Total process time):

real time 0.11 seconds

cpu time 0.01 seconds

