Paper 89

Averaging Numerous Repeated Measures in SAS® Using DO LOOPS and MACROS: A Demonstration Using Dietary Recall Data

Kendra Jones, St. Jude Children's Research Hospital; Kyla Shelton, St. Jude Children's Research Hospital

ABSTRACT

Dietary recall data consist of numerous measures recorded across multiple days for a particular individual. The raw measures must be collapsed and averaged before meaningful analysis can begin. Basic SAS programming can calculate the averages, but cannot handle the nearly countless number of variables without knowledge of each variable name and extensive syntax to process each one. The objective of this paper is to demonstrate a widely applicable process for averaging these or similar types of measures. The process uses intermediate programming techniques to eliminate specific variable name programming and reduce syntax. The complete process yields a single dataset of numerous calculated values suitable for continued application in further analyses.

INTRODUCTION

Averaging a series of repeated measures in SAS requires relatively basic programming techniques such as PROC MEANS with by group processing. However, such techniques require knowledge of each variable name, the number of variables, and syntax to process each one. They are well-suited to handling a limited number of variables as in the following example.

```
proc means data=dataset mean;
    var a b c d;
    by group;
run;
```

What if you have hundreds of variables? What if the variable names are complex or unfamiliar? What if you would like to use the same technique with a different dataset? Basic syntax, like that above, could quickly become cumbersome and difficult to adapt. The purpose of this paper is to describe an efficient, adaptable process for averaging numerous repeated measures using variable processing techniques, SQL, %DO loops, and MACROS. We demonstrate these techniques using data from dietary recall assessments.

In our experience, dietary recall data are comprised of numerous food and beverage consumption measures collected by interview and representing multiple 24-hour periods. Prior to study, each period measurement must be collapsed and averaged at an individual level in order to obtain typical dietary patterns. The collected and calculated measures or variables are numerous and complex in name.

PREPARING THE DATA

Naturally, raw data must first be imported, manipulated, and formatted before real processing or analysis can begin. In our scenario, we download dietary recall data from a web-based server as numerous individual text files. We use a MACRO program to simplify the import process, but you can do whatever is most logical for your data structure.

```
%MACRO DIET (filename);

PROC IMPORT OUT= WORK.&filename.
            DATAFILE="Path\&filename..txt"
            DBMS=TAB REPLACE;
     GETNAMES=YES;
     DATAROW=2;
RUN;

%MEND;

%DIET (file01);
%DIET (file02);
%DIET (file03);
%DIET (file04);
%DIET (file05);
%DIET (file06);
%DIET (file07);
```

The DATA STEP with a SET statement concatenates each imported data set. PROC SORT then sorts the concatenated data set by an individual identifier to allow for later by group processing at a person level.

```
data file;
    set file01 file02 file03 file04 file05 file06 file07;
run;

proc sort data=file; by participant_id; run;
```

CAPTURING THE VARIABLES

The next step, variable processing, begins with identifying each variable within our data set. Remember, we are not interested in hard coding each and every variable of interest into an analysis program. Instead, we use the PROC CONTENTS procedure with the OUT= option to prepare a new SAS data set containing only numeric variables i.e. those suitable for averaging. The ultimate goal is to automatically prepare a variable list that can later be processed through one by one.

The default LISTING output of a PROC CONTENTS procedure provides a description of a SAS data set, including variable names. However, it does nothing to process or incorporate them into subsequent programmatic steps. The addition of the OUT= option allows users to capture variable information from the default output as a new SAS data set. For the purposes of this paper, additional syntax is included to suppress the LISTING output (*noprint*), select only numeric variables (*type=1*), and keep just three columns: variable name (*name*), variable type (*type*), and variable number (*varnum*). The complete syntax and a partial view of the new SAS data set are included below. Note: if your data contain numeric variables not suitable for averaging (ex. 1=male, 2=female), you may wish to take additional measures to drop them from the new variable list dataset.

```
proc contents data=file noprint
    out=varlist (where=(type=1) keep=name type varnum);
run;
```

	Variable Name	Variable Type	Variable Number
1	Acesulfame_Potassium__mg_	1	185
2	Added_Sugars__by_Available_Carbo	1	184
3	Added_Sugars__by_Total_Sugars___	1	210
4	Alanine__g_	1	106
5	Alcohol__g_	1	28
6	Alpha_Carotene__provitamin_A_car	1	146
7	Animal_Protein__g_	1	26
8	Arginine__g_	1	104
9	Ash__g_	1	119
10	Aspartame__mg_	1	112

Display 1. Partial *Varlist* Dataset View from the PROC CONTENTS Procedure with the OUT= Option

By default, the variable name (*name*) variable in the outputted data set is sorted in alphabetical order, see Display 1. However, we want to keep the variables in their original order and so we add a PROC SORT procedure to sort by the variable number (*varnum*) variable instead. This PROC SORT step is merely a preference for future processing order and is not required for a successful program. The complete syntax and a partial view of the sorted SAS data set are included below.

```
proc sort data=varlist; by varnum; run;
```

	Variable Name	Variable Type	Variable Number
15	Total_Grams	1	21
16	Energy__kcal_	1	22
17	Total_Fat__g_	1	23
18	Total_Carbohydrate__g_	1	24
19	Total_Protein__g_	1	25
20	Animal_Protein__g_	1	26
21	Vegetable_Protein__g_	1	27
22	Alcohol__g_	1	28
23	Cholesterol__mg_	1	29
24	Total_Saturated_Fatty_Acids__SFA	1	30
25	Total_Monounsaturated_Fatty_Acid	1	31

Display 2. Partial *Varlist* Dataset View after the PROC SORT Procedure

As you can see in Displays 1 and 2, dietary recall assessment data contains hundreds of variables with complex naming conventions. It was these data features that led us to compile the processing and analysis program described in this paper. The steps covered thus far yield a new SAS data set that contains all of our analysis variables. The next step is to create a macro variable that will serve as our variable list.

CREATING A VARIABLE LIST

We use the INTO: host-variable in PROC SQL to create a macro variable that is used to manipulate data within subsequent %DO loop processing. Keep in mind that macro variables created with INTO: can be used in any DATA or PROC step. The annotated syntax below creates our macro variable, vlist. The select statement (①) within PROC SQL selects for the variable name variable (*name*) from the previously created *varlist* data set (②). The SEPERATED BY qualifier (③) inserts a space between each selected variable name observation. The space delimiter is important for later processing as it defines the boundaries of each individual variable within the larger macro variable. The %PUT statement (⑤) writes the new vlist macro variable to the SAS log for the purpose of validation. The resulting SAS log output is included in Display 3.

```
proc sql noprint; ①select name ④into: vlist ③separated by ' ' ②from varlist; quit;

⑤%put &vlist.;
```

```
4730  %put &vlist.;
Total_Grams Energy__kcal_ Total_Fat__g_ Total_Carbohydrate__g_ Total_Protein__g_ Animal_Protein__g_ Vegetable_Protein__g_
Alcohol__g_ Cholesterol__mg_ Total_Saturated_Fatty_Acids__SFA Total_Monounsaturated_Fatty_Acid Total_Polyunsaturated_Fatty_Acid
Fructose__g_ Galactose__g_ Glucose__g_ Lactose__g_ Maltose__g_ Sucrose__g_ Starch__g_ Total_Dietary_Fiber__g_
Soluble_Dietary_Fiber__g_ Insoluble_Dietary_Fiber__g_ Pectins__g_ Total_Vitamin_A_Activity__Intern
Beta_Carotene_Equivalents__deriv Retinol__mcg_ Vitamin_D__calciferol___mcg_ Total_Alpha_Tocopherol_Equivalen
Vitamin_E__Total_Alpha_Tocophero Beta_Tocopherol__mg_ Gamma_Tocopherol__mg_ Delta_Tocopherol__mg_
Vitamin_K__phylloquinone___mcg_ Vitamin_C__ascorbic_acid___mg_ Thiamin__vitamin_B1___mg_ Riboflavin__vitamin_B2___mg_
Niacin__vitamin_B3___mg_ Pantothenic_Acid__mg_ Vitamin_B_6__pyridoxine__pyrido Total_Folate__mcg_
Vitamin_B_12__cobalamin___mcg_ Calcium__mg_ Phosphorus__mg_ Magnesium__mg_ Iron__mg_ Zinc__mg_ Copper__mg_ Selenium__mcg_
Sodium__mg_ Potassium__mg_ SFA_4_0__butyric_acid___g_ SFA_6_0__caproic_acid___g_ SFA_8_0__caprylic_acid___g_
SFA_10_0__capric_acid___g_ SFA_12_0__lauric_acid___g_ SFA_14_0__myristic_acid___g_ SFA_16_0__palmitic_acid___g_
SFA_17_0__margaric_acid___g_ SFA_18_0__stearic_acid___g_ SFA_20_0__arachidic_acid___g_ SFA_22_0__behenic_acid___g_
MUFA_14_1__myristoleic_acid___g_ MUFA_16_1__palmitoleic_acid___g_ MUFA_18_1__oleic_acid___g_ MUFA_20_1__gadoleic_acid___g_
MUFA_22_1__erucic_acid___g_ PUFA_18_2__linoleic_acid___g_ PUFA_18_3__linolenic_acid___g_ PUFA_18_4__parinaric_acid___g_
PUFA_20_4__arachidonic_acid___g_ PUFA_20_5__eicosapentaenoic_acid PUFA_22_5__docosapentaenoic_acid
PUFA_22_6__docosahexaenoic_acid Tryptophan__g_ Threonine__g_ Isoleucine__g_ Leucine__g_ Lysine__g_ Methionine__g_ Cystine__g_
Phenylalanine__g_ Tyrosine__g_ Valine__g_ Arginine__g_ Histidine__g_ Alanine__g_ Aspartic_Acid__g_ Glutamic_Acid__g_
Glycine__g_ Proline__g_ Serine__g_ Aspartame__mg_ Saccharin__mg_ Caffeine__mg_ Phytic_Acid__mg_ Oxalic_Acid__mg_
__Methylhistidine__mg_ Sucrose_Polyester__g_ Ash__g_ Water__g_ __Calories_from_Fat __Calories_from_Carbohydrate
__Calories_from_Protein __Calories_from_Alcohol __Calories_from_SFA __Calories_from_MUFA __Calories_from_PUFA
Polyunsaturated_to_Saturated_Fat Cholesterol_to_Saturated_Fatty_A Total_Vitamin_A_Activity__Retino
TRANS_18_1__trans_octadecenoic_a TRANS_18_2__trans_octadecadieno TRANS_16_1__trans_hexadecenoic_a
Total_Trans_Fatty_Acids__TRANS_ Beta_Carotene__provitamin_A_caro Alpha_Carotene__provitamin_A_car
Beta_Cryptoxanthin__provitamin_A Lutein___Zeaxanthin__mcg_ Lycopene__mcg_ Dietary_Folate_Equivalents__mcg_
Natural_Folate__food_folate___mc Synthetic_Folate__folic_acid___m Energy__kj_ Niacin_Equivalents__mg_ Total_Sugars__g_
Omega_3_Fatty_Acids__g_ Manganese__mg_ Vitamin_E__International_Units_ Natural_Alpha_Tocopherol__RRR_al
Synthetic_Alpha_Tocopherol__all Daidzein__mg_ Genistein__mg_ Glycitein__mg_ Coumestrol__mg_ Biochanin_A__mg_ Formononetin__mg_
Added_Sugars__by_Available_Carbo Acesulfame_Potassium__mg_ Sucralose__mg_ Available_Carbohydrate__g_
Glycemic_Index__glucose_referenc Glycemic_Index__bread_reference_ Glycemic_Load__glucose_reference
Glycemic_Load__bread_reference_ Choline__mg_ Betaine__mg_ Erythritol__g_ Inositol__g_ Isomalt__g_ Lactitol__g_ Maltitol__g_
Mannitol__g_ Pinitol__g_ Sorbitol__g_ Xylitol__g_ Nitrogen__g_ Total_Conjugated_Linoleic_Acid__ CLA_cis_9__trans_11__g_
CLA_trans_10__cis_12__g_ Tagatose__mg_ Vitamin_D2__ergocalciferol___mcg Vitamin_D3__cholecalciferol___mc
Added_Sugars__by_Total_Sugars___
```

Display 3. SAS Log Output from the %PUT &VLIST Statement

DETERMINING THE NUMBER OF VARIABLES

The final preparation step determines the number of individual variable names stored within our vlist macro variable. This value will inform the number of times an analysis step should be performed, ultimately once for each variable. We use the CALL SYMPUT ROUTINE to create a macro variable that represents the number of records in our variable name dataset as shown in the annotated syntax below. First, the _NULL_ argument of the DATA step (①) specifies that SAS need not create a data set upon execution. We want data about the *varlist* data set i.e. the number of observations, but we do not need any of the actual observations. The “if 0 then” portion (②) provides additional efficiency as the number of observations can be obtained without performing the SET statement (Moore, 2001). The NOBS= option (③) creates a temporary variable, “x”, whose value is equal to the total number of observations in the input data set. The CALL SYMPUT routine (④) takes a value from a DATA step and assigns it to a macro variable (Delwiche & Slaughter, 2008). In our case, the value taken is “x” or the number of observations and the macro variable is created as *reccount*. Finally, the %PUT statement (⑤) writes the new reccount macro variable to the SAS log for the purpose of validation. The resulting SAS log output is included in Display 4.

```
①data _null_;
    ②if 0 then set varlist ③nobs=x;
    ④call symput('reccount',x);
    stop;
run;

⑤%put &reccount;
```

```
9107  %PUT &RECCOUNT;
200
```

Display 4. SAS Log Output from the %PUT &RECCOUNT Statement

CALCULATING AND STORING THE AVERAGES

Upon completion of the data preparation steps, we can now utilize our global macro variables within a short SAS program to calculate the numerous food and beverage consumption measures to obtain typical dietary patterns at an individual level. This section details each segment of SAS code we use to accomplish this task.

First, we need a base data set to collect and manage each calculated value for each unique person. We chose to use simple PROC SQL syntax with the UNIQUE statement. If preferred, you may also use PROC SORT with the NODUPKEY and OUT= options. The complete SQL syntax is included below.

```
proc sql;
    create table Averages as select unique participant_id from file
    order by participant_id;
quit;
```

The remaining SAS code segments are encompassed within a single MACRO program. It is this program that brings together each component of this process for meaningful, actionable output. The entire MACRO program and each individual segment are discussed next.

THE MACRO PROGRAM

Macro-level programming is used to, among other things, generate code iteratively or repetitively. In this demonstration, our iterations are centered on numerous dietary recall assessment measures. Our macro program uses a %DO loop to iteratively process a multi-faceted program, a %LET statement and SCAN function to select each analysis variable, the previously created macro variables *vlist* and *reccount* to guide iterations, PROC SQL with a GROUP BY clause to calculate the mean, a %PUT statement to monitor progress, a DATA step program with a MERGE statement to collect results, and a PROC DATASETS segment to keep things orderly. The annotated syntax is included below.

```
①%macro loop;

    ②%do I=1 %to &RECCOUNT;

        ③%let varnow = %scan(&vlist., &I., " ");

        ④proc sql;
           create table tmp&I. as select participant_id, mean(&varnow.) as &varnow.
           format=6.1 from file group by participant_id;
         quit;

        ⑤%PUT I= &I. VARNOW=&VARNOW.;

        ⑥data Averages;
           merge Averages tmp&I.;
           by participant_id;
         run;

        ⑦proc datasets;
           delete tmp&I.;
         run;

    ⑧%end;

 %mend loop;

⑨%loop;
```

① First, we define our macro program with the macro-name *Loop*. The %MACRO statement indicates the start of the macro program. The %MEND statement concludes the program. All text in between is considered the macro-text.

② Next, we implement dynamic programming with the iterative %DO loop. While a DO statement is confined to the DATA step, the iterative %DO can be used anywhere within a macro. We use the %DO to define and increment the index variable *I*. We start with the integer '1' and stop with the macro expression *&reccount* which generates an integer equal to the number of measures to be processed.

③ The third component of the macro program serves to select a diet recall variable for processing. The %LET statement creates the macro variable *varnow* and assigns it a value based on the results of a %SCAN function. The %SCAN function processes our macro variable *vlist* as a text string and selects the appropriate variable name based

on its position in the string. The macro expression *&I* defines the position integer as equal to the iteration integer. Therefore, variable one is processed during the first iteration, variable two during the second iteration, and so on. More specifically, recall that *&vlist* resolves to the text string shown in Display 3. When *&I* resolves to '2', *&varnow* resolves to 'Energy__kcal_'. Finally, we use the " " expression to inform the %SCAN function that words in the *vlist* text string are separated by a space.

④ Next we calculate the mean of the selected diet recall variable for each participant. SAS provides various procedures to calculate means. We chose the SQL procedure since it creates a dataset, calculates the mean, and sorts the results all within one procedure. First, specify the name of the dataset that will store the mean values after the CREATE TABLE statement. We suggest placing the *&I* extension at the end of the dataset so that a new dataset is generated for each diet recall variable. The *&I* resolves to the integer of the current iteration of the %DO loop. Second, select the variables for the dataset. In addition to selecting the variable *participant_id,* we tell SAS to create a new variable containing averages of the current diet recall variable in use by stating MEAN(&VARNOW.) as &VARNOW. The following FORMAT statement is optional. Finally, we specify the input dataset name, *file*, and utilize the GROUP BY statement to average the diet recall measure across multiple records per participant. Using iteration number two as an example, a dataset named *tmp2* is created and contains two variables, participant ID and the mean value for the 2nd diet recall variable 'Energy__kcal_'. A partial view of the dataset is included in Display 5.

	Participant_ID	Energy__kcal_
1	[redacted]	3677.2
2	[redacted]	2058.2
3	[redacted]	2336.5

Display 5. Partial *TMP2* Dataset View

⑤ The %PUT statement writes the current value of macro variables *I* and *varnow* to the SAS log. This helps to verify that the *varnum* variable was incremented as expected, see Display 6.

```
i= 2 VARNOW=Energy__kcal_
```

Display 6. SAS Log Output from the %PUT I= &I. VARNOW=&VARNOW. Statement

⑥ Joining the diet recall averages together into one dataset is the last key step within the macro. The MERGE statement joins the current diet recall variable averages to the previously created base dataset titled *averages*. Again using iteration two as an example, prior to merging, the base dataset contains the *participant_ID* plus the averages for the first diet recall variable. Dataset *tmp2* is merged with the base dataset resulting in a dataset now containing the averages for two diet recall variables. A partial view of the dataset after iteration two is included in Display 7.

	Participant_ID	Total_Grams	Energy__kcal_
1	[redacted]	5680.7	3677.2
2	[redacted]	3568.9	2058.2
3	[redacted]	2600.2	2336.5

Display 7. Partial Averages Dataset View after the Second Iteration of the Loop Macro

⑦ In an effort to keep the working directory uncluttered, we deleted the temporary dataset using the DATASETS procedure at the end of each iteration.

⑧ After SAS performs the various statements between the %DO and %END statements, the index variable *I* is incremented by 1 at the bottom of the loop. The process is repeated until the index variable reaches the stop value. In this case, the maximum number of diet recall measures as specified by the macro variable *reccount*.

⑨ Finally, we call the macro with %LOOP which causes the macro to execute. Out of habit, we close with a semicolon though it is not technically required.

CONCLUSION

This method successfully combines multiple intermediate SAS programming components to calculate the average of numerous repeated measures. More importantly, the use of variable list and record count macro variables eliminates the need to program in specific variable names and reduces syntax when the number of variables is large. Therefore, it is a widely applicable process that can be customized to a variety of needs and datasets. In addition to efficiency and flexibility gains, this process can also enhance accuracy by reducing the likelihood of typographical errors. Regardless of the complexity of variables needing to be processed, this process has merit in numerous settings.

REFERENCES

- Delwiche, Lora D., and Slaughter, Susan J. 2008. *The Little SAS® Book: A Primer, Fourth Edition*. 220-221. Cary, NC: SAS Institute Inc.
- Delwiche, Lora D., and Slaughter, Susan J. 2004. "SAS® Macro Programming for Beginners. *Proceedings of the Twenty-Ninth Annual SAS® Users Group International Conference*. Cary, NC: SAS Institute Inc. Available at http://www2.sas.com/proceedings/sugi29/243-29.pdf.
- Moore, Edward. 2001. "Performing Multiple Statements for Each Record in a SAS® Data Set". *Proceedings of the Twenty-Sixth Annual SAS® Users Group International Conference*. Cary, NC: SAS Institute Inc. Available at http://www2.sas.com/proceedings/sugi26/p093-26.pdf.
- Pass, Ray. (2003, March 21). PROC SQL UNIQUE vs. DISTINCT [listserve.uga.edu]. Retrieved from http://listserv.uga.edu/cgi-bin/wa?A2=ind0303c&L=sas-l&P=42049

RECOMMENDED READING

- SAS. "%DO, Iterative Statement". *SAS® 9.2 Documentation*. 8/12/2013. Available at http://support.sas.com/documentation/cdl/en/mcrolref/61885/HTML/default/viewer.htm#a000543755.htm
- SAS. "Set Statement". *SAS® 9.2 Documentation*. 8/12/2013. Available at http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000173782.htm
- SAS. "Data Statement". *SAS® 9.2 Documentation*. 8/12/2013. Available at http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000188132.htm
- SAS. "SQL Procedure". *SAS®* 9.2 SQL Procedure User's Guide. 8/15/2013. Available at http://support.sas.com/publishing/pubcat/chaps/59727.pdf

ACKNOWLEDGMENTS

The authors would like to thank the management team and staff for their support and review of this paper.

CONTACT INFORMATION

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

Kendra Jones
St. Jude Children's Research Hospital
262 Danny Thomas Place, MS 735
Memphis, TN 38105
(901) 595-5957
Kendra.Jones@stjude.org

Kyla Shelton
St. Jude Children's Research Hospital
262 Danny Thomas Place, MS 735
Memphis, TN 38105
(901) 595-5502
Kyla.Shelton@stjude.org