Paper 192-2019

Bespoke outputs, just like everyone else's: using the SAS® Macro Language to create template programs for standard presentations of your client's clinical trial data

Sean-Paul A. Claypool, MPH, Synteract, Inc., Carlsbad, CA, 92010

ABSTRACT

While the structure and content of the tables, listings, and figures (TLF) used to present clinical trial data are generally established, myriad trial designs and data types collected across studies present resource challenges to contract research organizations (CRO) to "re-invent the wheel" creating standardized outputs for each client's specific data. Leveraging the SAS® Macro Language, CROs can mitigate constrained budgets by developing template programs to produce common output types. Using adverse event and laboratory result summary tables as examples, this paper shall illustrate how such template programs may be designed with SAS® Macro variables and simple Macro scripts to automate dataset manipulation and output report generation. With a robust library of flexible template programs, CROs can both maximize their resource utilization and deliver quality data products for their clients.

INTRODUCTION

The International Conference on Harmonization's (ICH) guidance for the Structure and Content of Clinical Study Reports (E3) describes the standardized components of clinical study reports for submission to the regulatory bodies of the European Union, Japan, and the United States, including the data tabulations required for review. While the exact structure of each tabulation will be distinct to the study, the data, and the sponsor, the overall display will most likely be a presentation of descriptive statistics, grouped by treatment received. As the number of clinical trials conducted each year continues to increase, early career statistical programmers at a CRO today must create a growing number of unique outputs for their clients – most of which look roughly the same. To deliver to their clients as efficiently as possible, programmers should develop a library of standardized template programs for common output types. Using SAS® Macro variables to substitute variable name and values, and simple macro scripts to automate repetitive coding statements, template programs may be written that can be quickly adapted to unique datasets in order to produce the required output. This concept is illustrated below using the SAS® Macro language to produce an adverse events table, and to automate code execution for a laboratory results table. Note that this paper assumes the programmer is familiar with the SAS® Macro language elements such as %LET, %DO/%TO/%END, and %MACRO/%MEND statements, how to reference a macro variable, as well as how to create tables and assign macro variables using PROC SQL.

MANUALLY ASSIGNING TEMPLATE MACRO VARIABLES

The simple adverse event (AE) table to be produced shall display number of events (m), number of subjects (n), and percentage of subjects (%), by adverse event system organ class (SOC) and preferred term (PT), grouped by treatment arm (TRT). Presented in Table 1 is a mock-up of said table.

	Treatment 1		**Treatment 2**		**Total**	
System Organ Class	**Events**	**Subjects**	**Events**	**Subjects**	**Events**	**Subjects**
Preferred Term	**m**	**n (%)**	**m**	**n (%)**	**m**	**n (%)**
System Organ Class 1	m	n (%)	m	n (%)	m	n (%)
Preferred Term1	m	n (%)	m	n (%)	m	n (%)
Preferred Term2	m	n (%)	m	n (%)	m	n (%)
System Organ Class 2	m	n (%)	m	n (%)	m	n (%)
Preferred Term1	m	n (%)	m	n (%)	m	n (%)
Preferred Term2	m	n (%)	m	n (%)	m	n (%)
etc.						

Table 1: Adverse Events Table Mock

REVIEWING THE MOCK AND FINDING THE MACRO VARIABLES

The table mock is the programmer's reference guide for how to produce the output. The mock should display the expected structure of the table, the source variables required, and the statistics to be calculated. Reviewing Table 1, the programmer finds that the following four variables are required in the source dataset: adverse event system organ class, adverse event preferred term, treatment group, and subject ID.

Assuming the source dataset source dataset adheres to the Analysis Data Model (ADaM) Data Structure for Adverse Event Analysis (ADAE), these variables would be AEBODSYS, AEDECOD, ARM, and USUBJID, respectively. However, this may not always be the case. Some datasets may not be fully ADaM-compliant, but rather ADaM-like, and may not contain all the variables assumed by the model. Or perhaps the programmer will be required to create another table of the same structure, but using a different treatment group variable (i.e. a treatment group assigned later as part of the study design) and does not wish to find-and-replace each instance of "ARM" within their code. This is the utility of SAS® Macro variables: coding a template program using placeholder macro variables allows for re-use of the same code with minimal updates, obviating the need to write brand new code for each unique output. In this way, it does not matter whether the source dataset is fully ADaM-compliant, or which treatment group is to be used, the same code can produce different output. For this table template program, the programmer will use four macro variables, one for each variable identified above.

Further reviewing the AE mock, the programmer finds that counts of events, counts of subjects, and percentages of subjects, within SOC and PT and grouped by treatment group, will need to be calculated. With this information, the programmer can begin to build their template program.

BUILDING THE TEMPLATE

After reviewing the WORK.ADAE dataset (**Error! Reference source not found.**), the programmer can use the SAS® Macro function %LET to assign the required source dataset variables to macro variables in the template program:

```
%let socvar=aebodsys;
%let ptvar=aedecod;
%let trtvar=arm;
%let subjvar=usubjid;
```

ARM	USUBJID	AEBODSYS	AEDECOD
A	A1	SOC 3	PT 3
A	A1	SOC 2	PT 2
A	A2	SOC 1	PT 1
A	A2	SOC 2	PT 2
A	A2	SOC 3	PT 1
B	B1	SOC 3	PT 2
B	B1	SOC 1	PT 2
B	B2	SOC 3	PT 3
B	B2	SOC 1	PT 3
B	B2	SOC 2	PT 1
TOTAL	A1	SOC 3	PT 3
TOTAL	A1	SOC 2	PT 2
TOTAL	A2	SOC 1	PT 1
TOTAL	A2	SOC 2	PT 2
TOTAL	A2	SOC 3	PT 1
TOTAL	B1	SOC 3	PT 2
TOTAL	B1	SOC 1	PT 2
TOTAL	B2	SOC 3	PT 3
TOTAL	B2	SOC 1	PT 3

Table 2: Generic WORK.ADAE dataset (Note: Arm TOTAL is a combination of Arm A and Arm B)

Using the SQL procedure, event and subject counts for SOC and PT are calculated from the WORK.ADAE source dataset by referencing the macro variables instead of the direct variable name:

```
proc sql noprint;
    /*** NUMBER OF SUBJECTS WITH AE BY SOC ***/
    create table work.soc as
          select &trtvar, &socvar,
                count(&subjvar) as m,
                count(distinct(&subjvar)) as n
          from work.adae
          group by &trtvar, &socvar;

    /*** NUMBER OF SUBJECTS WITH AE BY SOC AND PT ***/
    create table work.pt as
          select &trtvar, &socvar, &ptvar,
                count(&subjvar) as m,
                count(distinct(&subjvar)) as n
          from work.adae
          group by &trtvar, &socvar, &ptvar;
quit;
```

From the WORK.ADAE dataset, this code produces two datasets, WORK.SOC (Table 3) and WORK.PT (Table 4).

ARM	AEBODSYS	M	N
A	SOC 1	1	1
A	SOC 2	2	2
A	SOC 3	2	2
B	SOC 1	2	2
B	SOC 2	1	1
B	SOC 3	2	2
TOTAL	SOC 1	3	3
TOTAL	SOC 2	3	3
TOTAL	SOC 3	4	4

Table 3: WORK.SOC dataset containing adverse event counts by system organ class

ARM	AEBODSYS	AEDECOD	M	N
A	SOC 1	PT 1	1	1
A	SOC 2	PT 2	2	2
A	SOC 3	PT 1	1	1
A	SOC 3	PT 3	1	1
B	SOC 1	PT 2	1	1
B	SOC 1	PT 3	1	1
B	SOC 2	PT 1	1	1
B	SOC 3	PT 2	1	1
B	SOC 3	PT 3	1	1
TOTAL	SOC 1	PT 1	1	1
TOTAL	SOC 1	PT 2	1	1
TOTAL	SOC 1	PT 3	1	1
TOTAL	SOC 2	PT 1	1	1
TOTAL	SOC 2	PT 2	2	2
TOTAL	SOC 3	PT 1	1	1
TOTAL	SOC 3	PT 2	1	1
TOTAL	SOC 3	PT 3	2	2

Table 4: WORK.PT dataset containing adverse event counts by preferred term

Using a few DATA steps to set WORK.SOC and WORK.PT together, merge with denominator data for the treatment group population (for this example, there are 10 subjects each in Arms A and B, for a total of 20 subjects in TOTAL), then calculate the percentages, a tall dataset displaying the desired adverse event counts is produced (Table 5):

```
data work.socpt;
    set work.soc work.pt;
run;

proc sort data=work.socpt;
    by &trtvar;
run;

data work.socpt;
    merge socpt work.total (keep=&trtvar denom);
    by &trtvar;
    length xm npct $20;
    xm=strip(put(m,best.));
    npct=catx(" ", put(n,best.), cats("(", put(100*n/denom,best.) ,"%)"));
    drop m;
    rename xm=m;
run;

proc sort data=work.socpt;
    by &socvar &ptvar &trtvar;
run;

proc transpose data=work.socpt out=work.tm (drop=_:) suffix=_m;
    by &socvar &ptvar;
    id &trtvar;
    var m;
run;

proc transpose data=work.socpt out=work.tnpct (drop=_:) suffix=_npct;
    by &socvar &ptvar;
    id arm;
    var npct;
run;

data work.aetable;
    merge work.tm work.tnpct;
    by &socvar &ptvar;
    length desc $200;

    if missing(&ptvar) then
          desc=&socvar;
    else desc="   "||strip(&ptvar);
run;
```

ARM	AEBODSYS	AEDECOD	M	N	DENOM	NPCT
A	SOC 1		1	1	10	1 (10%)
B	SOC 1		2	2	10	2 (20%)
TOTAL	SOC 1		3	3	20	3 (15%)
A	SOC 1	PT 1	1	1	10	1 (10%)
TOTAL	SOC 1	PT 1	1	1	20	1 (5%)
B	SOC 1	PT 2	1	1	10	1 (10%)
TOTAL	SOC 1	PT 2	1	1	20	1 (5%)
B	SOC 1	PT 3	1	1	10	1 (10%)
TOTAL	SOC 1	PT 3	1	1	20	1 (5%)
A	SOC 2		2	2	10	2 (20%)
B	SOC 2		1	1	10	1 (10%)
TOTAL	SOC 2		3	3	20	3 (15%)
B	SOC 2	PT 1	1	1	10	1 (10%)
TOTAL	SOC 2	PT 1	1	1	20	1 (5%)
A	SOC 2	PT 2	2	2	10	2 (20%)
TOTAL	SOC 2	PT 2	2	2	20	2 (10%)
A	SOC 3		2	2	10	2 (20%)
B	SOC 3		2	2	10	2 (20%)
TOTAL	SOC 3		4	4	20	4 (20%)
A	SOC 3	PT 1	1	1	10	1 (10%)
TOTAL	SOC 3	PT 1	1	1	20	1 (5%)
B	SOC 3	PT 2	1	1	10	1 (10%)
TOTAL	SOC 3	PT 2	1	1	20	1 (5%)
A	SOC 3	PT 3	1	1	10	1 (10%)
B	SOC 3	PT 3	1	1	10	1 (10%)
TOTAL	SOC 3	PT 3	2	2	20	2 (10%)

Table 5: Combined tall WORK.SOCPT adverse event dataset with percentages

Using the TRANPOSE procedure to convert the combined adverse events count dataset from tall to wide, and the DATA step to process text, a wide dataset matching the mock in Table 1 is produced (Table 6).

DESC	A_M	A_NPCT	B_M	B_NPCT	TOTAL_M	TOTAL_NPCT
SOC 1	1	1 (10%)	2	2 (20%)	3	3 (15%)
PT 1	1	1 (10%)	0	0	1	1 (5%)
PT 2	0	0	1	1 (10%)	1	1 (5%)
PT 3	0	0	1	1 (10%)	1	1 (5%)
SOC 2	2	2 (20%)	1	1 (10%)	3	3 (15%)
PT 1	0	0	1	1 (10%)	1	1 (5%)
PT 2	2	2 (20%)	0	0	2	2 (10%)
SOC 3	2	2 (20%)	2	2 (20%)	4	4 (20%)
PT 1	1	1 (10%)	0	0	1	1 (5%)
PT 2	0	0	1	1 (10%)	1	1 (5%)
PT 3	1	1 (10%)	1	1 (10%)	2	2 (10%)

Table 6: Final WORK.AETABLE adverse event count dataset

Table 6 displays the number of adverse events, the number of subjects, and the percentage of subjects by system organ class and preferred term, grouped by treatment arm, as required by the adverse events table mock. This dataset can be output as required using the REPORT procedure or other ODS function.

DYNAMICALLY ASSIGNING MACRO VARIABLES FOR CODE AUTOMATION

The SAS Macro language can also be used to automate code processing, such as calculating mean laboratory results over different laboratory parameters. Instead of hardcoding a procedure for each laboratory test, which could result in hundreds of lines of repetitive code with only the input variable changing, the statistical programmer can utilize PROC SQL to assign laboratory parameters to macro variables automatically, and loop through them using a simple macro %DO loop. Combined with the techniques demonstrated above, the programmer can develop a template program that can dynamically produce output based on the input data itself.

For this example, it is assumed that the end output should display as shown in the laboratory results table mock (Table 7) and that the source dataset complies with the ADaM Data Structure for Laboratory Analysis (ADLB) (Table 8).

Laboratory Parameter	Time Point	Treatment A	Treatment B	Total
Parameter 1	Visit 1			
	n	n	n	n
	Mean (SD)	xx.x (xx.xx)	xx.x (xx.xx)	xx.x (xx.xx)
	Median	xx.x	xx.x	xx.x
	Min, Max	xx, xx	xx, xx	xx, xx
	Visit 2			
	n	n	n	n
	Mean (SD)	xx.x (xx.xx)	xx.x (xx.xx)	xx.x (xx.xx)
	Median	xx.x	xx.x	xx.x
	Min, Max	xx, xx	xx, xx	xx, xx
Parameter 2	Visit 1			
	n	n	n	n
	Mean (SD)	xx.x (xx.xx)	xx.x (xx.xx)	xx.x (xx.xx)
	Median	xx.x	xx.x	xx.x
	Min, Max	xx, xx	xx, xx	xx, xx
	Visit 2			
	n	n	n	n
	Mean (SD)	xx.x (xx.xx)	xx.x (xx.xx)	xx.x (xx.xx)
	Median	xx.x	xx.x	xx.x
	Min, Max	xx, xx	xx, xx	xx, xx
etc.				

Table 7: Laboratory Results Table Mock

ARM	USUBJID	VISIT	PARAMCD	AVAL
A	A1	VISIT 1	TEST1	66.7
A	A1	VISIT 1	TEST2	13.4
A	A1	VISIT 2	TEST1	3.25
A	A1	VISIT 2	TEST2	87.9
A	A2	VISIT 1	TEST1	18
A	A2	VISIT 1	TEST2	81.8
A	A2	VISIT 2	TEST1	23.6
A	A2	VISIT 2	TEST2	61.2
B	B1	VISIT 1	TEST 1	6.23
B	B1	VISIT 1	TEST 2	69.6
B	B1	VISIT 1	TEST 3	66.7
B	B2	VISIT 1	TEST 1	13.4
B	B2	VISIT 1	TEST 2	3.25
B	B2	VISIT 1	TEST 3	87.9
TOTAL	A1	VISIT 1	TEST1	18
TOTAL	A1	VISIT 1	TEST2	81.8
TOTAL	A1	VISIT 2	TEST1	23.6
TOTAL	A1	VISIT 2	TEST2	61.2
TOTAL	A2	VISIT 1	TEST1	6.23

Table 8: Generic WORK.ADLB dataset (Note: Arm TOTAL is a combination of Arm A and Arm B)

DYNAMICALLY ASSIGNING MACRO VARIABLES

Macro variables can be assigned by hardcoding a %LET statement as shown previously, but they can also be assigned from dataset values using SELECT and INTO in a PROC SQL statement:

```
proc sql noprint;
    select distinct paramcd
          into :param1-
          from work.adlb
          order by paramcd;
    %let nparam=&sqlobs;
quit;
```

This simple code will process the laboratory parameters in the WORK.ADLB dataset to find the n number of unique values (Note: &SQLOBS is a macro variable automatically generated by the SQL process that holds the number of records found), order them alphabetically, and assign them sequentially to the macro variables &PARAM1, &PARAM2, … &PARAMn. The value of n is assigned to the macro variable &NPARAM. In this example, there are n=3 unique values of the variable PARAMCD and they are assigned to macro variables as follows: &PARAM1=TEST1, &PARAM2=TEST2, and &PARAM3=TEST3. The macro variable &NPARAM is assigned the value of 3.

LOOPING CODE OVER MACRO VARIABLES

The programmer now has the macro variables required to loop the statistical procedure over the WORK.ADLB dataset using a %DO loop in a %MACRO statement:

```
%macro paramloop;
    %do i=1 %to &nparam;

          proc univariate data=work.adlb;
                where paramcd="&&param&i";
                by visit;
                var aval;
                class &trtvar;
                output out=&&param&i n=n mean=mean std=std median=median
                      min=min max=max;
          run;

    %end;
%mend;

%paramloop;
```

The PROC UNIVARIATE procedure will be repeated three times in the %DO loop, each time producing a set of test statistics for the unique parameter values in WORK.ADLB. The code achieves this by iteratively assigning the &I macro variable a value from 1 to &NPARAM with each pass of the %DO loop, and indirectly referencing the laboratory parameter name stored in the &PARAMn macro variable. To fully resolve the macro variable &&PARAM&I it must pass through macro facility twice. On the first pass, &I resolves to 1, resulting in the macro variable &PARAM1. When it passes through the macro facility again, &PARAM1 resolves to TEST1, as assigned in the PROC SQL step above. (For a thorough discussion of &&VAR&I macro variable resolution, please reference Chapter 11, “Writing Dynamic Programs,” of *Carpenter’s Complete Guide to the SAS® Macro Language* by Art Carpenter.) The %PARAMLOOP macro generates three output datasets from WORK.ADLB, one for each laboratory parameter (Table 9, Table 10, Table 11).

VISIT	ARM	N	MEAN	STD	MAX	MEDIAN	MIN
VISIT 1	A	2	34.975	44.865925266	66.7	34.975	3.25
VISIT 1	B	2	39.6	30.547012947	61.2	39.6	18
VISIT 1	TOTAL	4	37.2875	31.450818087	66.7	39.6	3.25
VISIT 2	A	2	51.25	12.940054096	60.4	51.25	42.1
VISIT 2	TOTAL	2	51.25	12.940054096	60.4	51.25	42.1

Table 9: WORK.TEST1 dataset

VISIT	ARM	N	MEAN	STD	MAX	MEDIAN	MIN
VISIT 1	A	2	50.65	52.679455198	87.9	50.65	13.4
VISIT 1	B	2	44.015	53.436059454	81.8	44.015	6.23
VISIT 1	TOTAL	4	47.3325	43.491610973	87.9	47.6	6.23
VISIT 2	A	2	41.75	0.2121320344	41.9	41.75	41.6
VISIT 2	TOTAL	2	41.75	0.2121320344	41.9	41.75	41.6

Table 10: WORK.TEST2 dataset

VISIT	ARM	N	MEAN	STD	MAX	MEDIAN	MIN
VISIT 1	B	2	46.6	32.526911935	69.6	46.6	23.6
VISIT 1	TOTAL	2	46.6	32.526911935	69.6	46.6	23.6

Table 11: WORK.TEST3 dataset

Through a series of DATA steps and TRANSPOSE procedures similar to those demonstrated for adverse events above, the WORK.TESTn datasets can be combined and processed to produce a final WORK.LABTABLE (Table 12).

PARAMCD	VISIT	STAT	A	B	TOTAL
TEST1	VISIT 1	n	2	2	4
TEST1	VISIT 1	Mean (SD)	35 (44.87)	39.6 (30.55)	37.3 (31.45)
TEST1	VISIT 1	Median	34.975	39.6	39.6
TEST1	VISIT 1	Min, Max	3.25, 66.7	18, 61.2	3.25, 66.7
TEST1	VISIT 2	n	2	0	2
TEST1	VISIT 2	Mean (SD)	51.3 (12.94)	0	51.3 (12.94)
TEST1	VISIT 2	Median	51.25	0	51.25
TEST1	VISIT 2	Min, Max	42.1, 60.4	0	42.1, 60.4
TEST2	VISIT 1	n	2	2	4
TEST2	VISIT 1	Mean (SD)	50.7 (52.68)	44 (53.44)	47.3 (43.49)
TEST2	VISIT 1	Median	50.65	44.015	47.6
TEST2	VISIT 1	Min, Max	13.4, 87.9	6.23, 81.8	6.23, 87.9
TEST2	VISIT 2	n	2	0	2
TEST2	VISIT 2	Mean (SD)	41.8 (0.21)	0	41.8 (0.21)
TEST2	VISIT 2	Median	41.75	0	41.75
TEST2	VISIT 2	Min, Max	41.6, 41.9	0	41.6, 41.9
TEST3	VISIT 1	n	0	2	2
TEST3	VISIT 1	Mean (SD)	0	46.6 (32.53)	46.6 (32.53)
TEST3	VISIT 1	Median	0	46.6	46.6
TEST3	VISIT 1	Min, Max	0	23.6, 69.6	23.6, 69.6

Table 12: Final WORK.LABTABLE laboratory results dataset

The final WORK.LABTABLE dataset displays the statistical results by laboratory parameter and by visit, grouped by treatment arm, as required by the mock, and can be output with PROC REPORT or other ODS function as required. Using macro variables derived from the source dataset, combined with simple %DO loops, the statistical programmer can quickly develop a laboratory results table without hard coding repetitive statistical procedures for each parameter.

CONCLUSION

Using the techniques demonstrated above, the statistical programmer now has the tools necessary to begin developing template programs that can produce a standard adverse event table regardless of the source variable names, and to automate the production of laboratory results tables based on the data provided. Indeed, the general nature of the adverse event template code above can be extended to other tables that display events by groups, such as concomitant medications or medical history, and the laboratory results code can be adapted for any manner of repetitive coding tasks. However, the code presented above is by no means comprehensive, and clinical output tables typically consider many other variables not discussed here (e.g. population flag, severity, treatment emergence). The programmer is encouraged to review the table mocks and source datasets together, consider future programming requirements, and develop flexible template programs utilizing the macro variables and code automations that best meet the needs of the programmer, the CRO, and the sponsor. With a library of flexible template programs, and an understanding of how the SAS® Macro language works, the efficient statistical programmer can quickly adapt a template program to produce a wide variety of clinical outputs, allowing CROs to both maximize their resource utilization and deliver quality data products for their clients.

REFERENCES

Carpenter, Art. 2016. *Carpenter's Complete Guide to the SAS® Macro Language.* 3rd ed. Cary, NC: SAS Institute, Inc.

CONTACT INFORMATION

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

Sean-Paul A. Claypool, MPH
Synteract, Inc.
sean-paul.claypool@synteract.com