Paper SM06

Effortless Table Generation: A SAS Macro for Streamlining Your Workflow

Zhen (Laura) Li, AstraZeneca, Gaithersburg, USA

ABSTRACT

This paper presents a SAS® macro tool designed to streamline table generation, a critical task in clinical data analysis. It simplifies the process, allowing users to create tables with diverse templates featuring descriptive statistics, counts, and percentages. Practical examples showcase its capabilities. The tool is user-friendly, requiring users to populate an Excel file and execute the macro. Its versatility stands out as it can handle raw data, SDTM data, ADaM data, or any combination thereof. For junior programmers, this macro is a game-changer, enabling quick, high-quality output generation and validation. The paper also covers design principles and programming challenges in the macro's development, offering insights into coding intricacies and best programming practices. In conclusion, it explores potential enhancements and future applications for this SAS macro, emphasizing its role in optimizing data analysis workflows in clinical research and promoting efficiency.

INTRODUCTION

Programmers often encounter requests that are either repetitively delivered or necessitate the generation of outputs using similar templates. Creating individual programs for each output can be a time-consuming task, especially when frequent deliveries are required. To address this challenge, the adoption of a robust and widely utilized program or tool becomes crucial to alleviate the workload.

Macro programs play a vital role in the automation process for SAS programming, and automation stands out as a key strategy to boost efficiency and quality in programming deliverables. Moreover, examining automation from a process perspective sheds light on additional considerations.

This paper introduces a SAS macro program crafted by the author to efficiently address these recurring requests. The tool not only streamlines the process but also holds the potential for further enhancements to accommodate diverse situations. Consequently, this paper will delve into discussions about potential improvements to the SAS macro program. Towards the end of this paper, reflections and thoughts on the broader implications of automation in programming processes will be discussed.

THE TABLE GENERATION MACRO

This macro streamlines the process of generating tables by requiring users to input essential information into an Excel file, which comprises four sheets: Statistics, Title_footnote, Header, and Body. Subsequently, users execute the macro, named "template1", to generate the table. Figure 1 provides an overview of the general execution process.

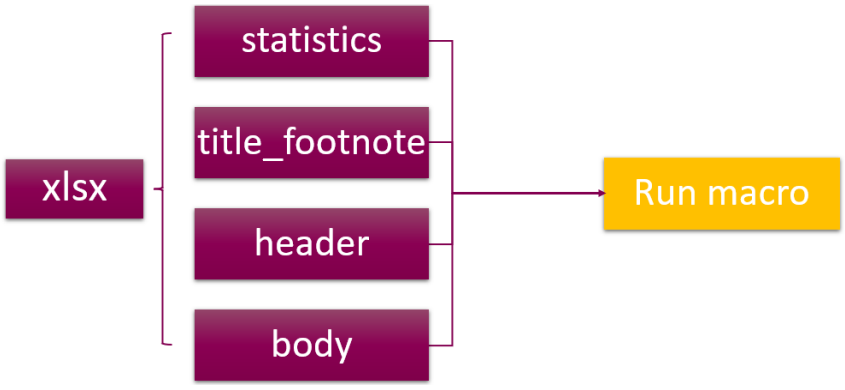

Figure 1: General Execution Process

The subsequent portion of this section will introduce the entire process with examples, covering the utilization of the four sheets within the Excel document and the execution of the macro for an individual table. Figure 2 presents a portion of a template and the framework for generating each section through associated sheets. Users populate information for each section in the Excel. This section will use the portion of the template in Figure 2 as an example to

demonstrate this macro. It's important to note that the versatility of this macro extends beyond the specific template provided in the example below and the data used in this paper is dummy data.

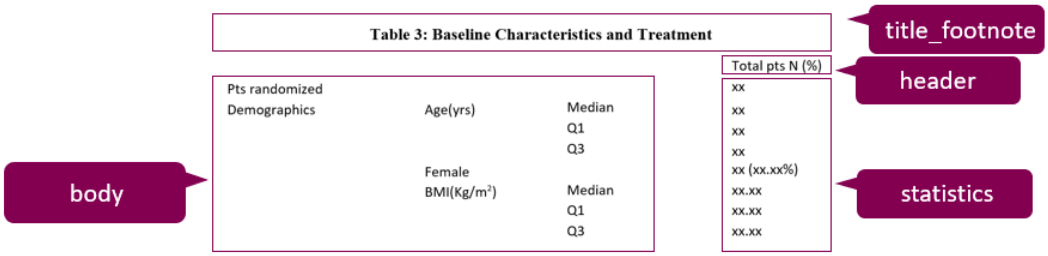

Figure 2: A Portion of An Example Template

THE STATISTICS SHEET

Users populate three columns: Statistics_name, Statistics_order, and Statistics, each equipped with dropdown lists for convenient selection. Statistics_name designates user-defined descriptive statistic groups (e.g., 'stat1, stat2'). Statistics_order establishes the display order for each statistic, while Statistics outlines the desired statistics (e.g., 'n, Mean, SD, Median, (Min, Max), Q1, Q3'). Importantly, these three columns are not table-specific, allowing for reuse across different tables. Figure 3 presents the Statistics sheet with two examples of statistics groups, stat1 and stat3.

Statistics_name	Statistics_order	Statistics
stat1	1	n
stat1	2	Mean
stat1	3	SD
stat1	4	Median
stat1	5	(Min, Max)
stat3	1	Median
stat3	2	
stat3	3	

n
Mean
SD
Median
(Min. Max)
Q1
Q3

Figure 3: Statistics Sheet Display

THE TITLE_FOOTNOTE SHEET

Users input title(s) and footnote(s) information along with the locations for titles. Dropdown lists facilitate the selection of locations for Title1 and Title2 ('L, C, R'). Users can add footnotes to columns Footnote1 through Footnote3. If more footnotes are needed, multiple footnotes can be added in a cell using '^' as a separator. Figure 4 illustrates the Title_footnote sheet with an example.

Output_name	Title1	Location_Title1	Title2	Location_Title2	Footnote1	Footnote2	Footnote3	Output_Generate
Table3	Table 3: Baseline Characteristics and Treatment	C						
Table3	XXXXXX	L	port Date: &dt	L				

L
C
R

Figure 4: Title_footnote Sheet Display

The Output_Generate column specifies whether to generate the table in .rtf format, with the default being to generate it. Users can suppress it by specifying 'N'. This column is introduced to accommodate situations where only the output dataset is needed, not the .rtf table itself, and an example of this will be provided in Figure 10.

THE HEADER SHEET

This sheet comprises of six columns: Output_name (populated from the Title_footnote sheet), Column_number, Column_label, Indata, Key_variable, and Condition. Users define the orders (‘Column_number’) and labels (‘Column_label’) of columns, along with the source data (‘Indata’), source variables (‘Key_variable’), and programming conditions (‘Condition’) for each table. Figure 5 displays the Header sheet with an example.

Output_name	Column_number	Column_label	Indata	Key_variable	Condition
Table3	1	Total pts N (%)	in.elig2	subject	not missing(iepatien)

Table3
Table3
Table6
Table6

Figure 5: Header Sheet Display

THE BODY SHEET

The Body sheet guides users in specifying information for each row of the table, encompassing columns such as Output_name, Row_number, Label0, Label1, Indata, Id_variable, Summary_method, Variable1, Variable2, and Condition. Similar to the Header sheet, users select the table name from the dropdown list in the Output_name column.

In the Row_number column, users designate the order of each row, with Label0 representing the label of the group/category column, and Label1 indicating the label of the subgroup/subcategory (left blank if nonexistent). Indata contains the source data, and Id_variable is the key variable used to merge with the datasets specified in the Header sheet.

The Summary_method column defines summary statistics, allowing users to choose between predefined descriptive statistic groups from the Statistics_name column in the Statistics sheet or macro-defined statistics like counts, percentages using column totals as denominators, and percentages using the total of all categories as denominators ("cnt", "perct", and "perctcat," respectively).

Variable1 and Variable2 specify the source variables for the row. Variable1 is for the group/category, while Variable2 is for the subgroup/subcategory (left blank if unnecessary). Users set programming conditions in the Condition column.

If the variable defined in Id_variable is also the source variable ('Variable1') for the row, users can use "dummy" as the source variable; a built-in dummy variable is available for this purpose. Figure 6 displays the Body sheet with example rows. Figure 7 demonstrates the associated rows for the example template.

Output_name	Row_number	Label0	Label1	Indata	Id_variable	Summary_method	Variable1	Variable2	Condition
Table3	1	Pts randomized		in.elig2	subject	cnt	dummy		not missing(iepatien)
Table3	2	Demographics	Age(yrs)	in.demog	subject	stat3	age		not missing(age)
Table3	3	Demographics	Female	in.demog	subject	perct	dummy		sex="Female"
Table3	4	Demographics	BMI(Kg/m2)	in.vs_temp	subject	stat3	zbmi		not missing(zbmi)

Figure 6: Body Sheet Display

Table 3: Baseline Characteristics and Treatment

			Total pts N (%)
① Pts randomized			xx
② Demographics	Age(yrs)	Median	xx
		Q1	xx
		Q3	xx
③	Female		xx (xx.xx%)
④	BMI(Kg/m^2)	Median	xx.xx
		Q1	xx.xx
		Q3	xx.xx

Figure 7: Example Template and Associated Rows

RUN THE MACRO

The last step involves executing the macro "template1". After completing the data input on all four excel sheets, users run the macro program while specifying the table name, which should align with the corresponding table name in the Excel sheets. This action initiates the generation of individual tables. Figure 8 depicts a portion of a table generated by this macro.

XXXXXX
Report Date: 11MAY2022

Table 3: Baseline Characteristics and Treatment

			Total pts N (%)
Pts randomized			228
Demographics	Age(yrs)	Median	60
		Q1	54
		Q3	67
	Female		38 (16.67%)
	BMI(Kg/m2)	Median	27.78
		Q1	25.37
		Q3	30.97

Figure 8: A Portion of An Example Table

This tool enables users to employ different types of source data, a topic that will be further explored later in this paper. If users opt for raw or uncleaned data as the source data, as exemplified by the table in Figure 8, a preliminary program may be required prior to the macro execution in certain scenarios:

1. Data Checking and Cleaning: Ensure that the data issue is processed before triggering the macro.

2. Variable Derivation: If necessary, derive a variable by using values from the same dataset or multiple datasets.

3. Inclusion of a Macro Variable for Table Generation: In cases where a macro variable is essential for a table, such as the report date which updates with each delivery, generate a macro variable representing the dynamic date.

An illustrative example walks through these scenarios, demonstrating its application to uncleaned data that requires initial cleaning. While data checking macros can aid in identifying issues, it's important to note that the checking process remains external to the macro to preserve its streamlined simplicity. When variables require source variables from multiple datasets, users are advised to derive these variables before invoking the macro, ensuring seamless execution. Figure 9 provides an example for these two situations.

```
/*BMI(Kg/m2), remove duplicates from VS*/
proc sql;
 create table in.vs_temp as
 select distinct *
 from in.vs group by subject having vsdat=max(vsdat) ;
quit;

/*Time from Qualifying Event to Randomization, add a variable in elig2*/
proc sql;
 create table in.elig2_temp as
 select distinct a.*, EVSTDAT, intck('day', datepart(EVSTDAT), datepart(IERNDDAT))   as gerndur
 from in.elig2 a inner join (select distinct subject, EVSTDAT from in.procca group by subject having EVSTDAT=min(EVSTDAT) ) b
 on a.subject=b.subject ;
quit;

/*prepare data for hsCRP and NT proBNP*/
proc sql;
 create table in.bsaf_temp as
 select distinct subjid, battrnam, visit, case when battrnam = 'NTPROBNP' and index(cnvresc, "<")>0 then input(strip(tranwrd(cnvresc,"<","")),
 best.)-1 else  cnvresn end as cnvresn_temp, cnvu, SIRESC, case when  battrnam = 'HS-CRP' and index(SIRESC, "<")>0 then
 input(strip(tranwrd(SIRESC,"<","")), best.)-0.1 else   SIRESN end as SIRESN_temp, SIRESn, cnvresc, cnvresn
 from in.bsaf where (battrnam ='NTPROBNP' and not missing(cnvresc)) or (battrnam = "HS-CRP" and not missing(SIRESC)) and visitnum="SCRN1"
 group by subjid, battrnam
 having lbdtm =max(lbdtm)
 order by subjid;
quit;

%template1(Table3 );
```

clean data

derive a variable

clean data and derive variables

run the macro

Figure 9: Examples of Scenarios Requiring Preliminary Programs

ADVANTAGES

This macro exhibits flexibility across various source data, accommodating raw data, SDTM data, ADaM data, or a combination thereof. This versatility allows programmers to choose the most suitable approach based on specific circumstances. For instance, when facing time constraints or when SDTM or ADaM datasets are unavailable, utilizing raw data becomes a viable option. Alternatively, for analyses requiring variables with complex algorithms already developed in SDTM or ADaM datasets, programmers can enhance efficiency and quality by executing the associated SDTM or ADaM datasets alongside raw datasets for table generation. The macro seamlessly handles challenges related to varied source data.

In addition to Figure 8, Figures 10 and 11 demonstrate alternate template tables generated by this macro. Users have the flexibility to specify whether the .rtf output should be generated in the Title_footnote sheet, aligning with template requirements. In Figure 10, two datasets are generated by default for two different data transfers. Users can merge datasets and subsequently generate a table. Furthermore, the macro supports table generation with or without subgroup/subcategory. For example, users can create tables without subgroup/subcategory, as illustrated in Figure 8, or opt for tables with subcategories by defining the source variable in the Variable2 column in the Body sheet. This functionality proves particularly beneficial for tables like an Adverse Events (AE) table, presenting System Organ Class by Preferred Term terms, as illustrated in Figure 11.

Report Date: 11MAY2022 *Table 6: Events through Follow up*

	Last report date: 07APR2022	*This report date: 11MAY2022*
Pts randomized	205	228
Deaths	2 (0.98%)	2 (0.88%)
CV death	0	0
MI (STEMI or NSTEMI)	1 (0.49%)	1 (0.44%)
Unstable Angina	0	1 (0.44%)
Stroke (any)	0	0
Ischemic Stroke	0	0
Hemorrhagic Stroke	0	0
Hospitalization for heart failure	1 (0.49%)	1 (0.44%)

Figure 10: Example Output with Two Different Data Transfers

XXXXXX *Table 7: AE by System Organ Class and Preferred Term*
Report Date: 03JAN2024 *Safety Population*

		Placebo N=185	*IP N=528*
Subjects with at least one adverse event		4 (2.16%)	21 (3.98%)
Blood and lymphatic system disorders		1 (0.54%)	0
	Lymphadenopathy	1 (0.54%)	0
Cardiac disorders		1 (0.54%)	4 (0.76%)
	Angina pectoris	0	1 (0.19%)
	Atrial fibrillation	0	1 (0.19%)
	Cardiac failure	1 (0.54%)	0
	Coronary artery disease	0	1 (0.19%)
	Sinus bradycardia	0	1 (0.19%)
	Tachycardia	0	1 (0.19%)

Figure 11: Example Output with Subcategories

This macro significantly enhances the efficiency of discrepancy investigations. Programmers can utilize the Excel file as a versatile document for checking and updating programming conditions, source variables, or any other necessary information. Being a stand-alone macro, its implementation is straightforward - users can effortlessly integrate it by invoking this single macro. Moreover, it doesn't rely on platform-specific tools, ensuring compatibility across various platform environments.

The macro’s simplicity is a notable feature, requiring users to fill out the Excel sheets. This user-friendly approach makes it accessible even to junior programmers or statisticians less familiar with SAS, empowering them to independently generate tables within a short timeframe.

POTENTIAL ENHANCEMENTS & APPLICATIONS

This macro was initially created to primarily address ad-hoc requests, resulting in templates that deviate from company standards. This presents significant potential for enhanced integration by incorporating macros to generate company-standard table formats, ensuring alignment with established formatting norms.

In its current state, the macro emphasizes the presentation of counts, percentages, and descriptive statistics, excluding statistical tests such as the chi-squared test, t-test, or statistical models. To enhance its versatility, potential improvements include integrating statistical tests and models, facilitating format adjustments through an additional sheet, and introducing sorting order options for categorical values of sourced variables in the Body sheet. Adding the option to generate a missing category for a categorical variable with missing values in the Body sheet proves beneficial for requests involving uncleaned data. In addition to individual executions, an impactful upgrade could involve enabling batch runs for all tables.

Beyond its current application for table generation, this macro holds potential for validation purposes. The opportunity to automate the validation process presents itself in the macro's output dataset generation step. If adjustments are made in this step to standardize output datasets from either the primary program (when utilizing this macro for validation) or from this macro (when developing new validation programs), it could further streamline the validation process, leading to higher efficiency and reliability.

CODE STRUCTURE DESIGN & CHALLENGES

The macro is systematically organized into the following components, and a portion of the detailed code is available in the attached appendix:

- Read all sheets in the Excel file.
- Generate a macro variable for each column of each sheet.
- Utilize PROC MEANS to create descriptive statistics based on the list of the descriptive statistics groups defined in the Statistics sheet.
- Employ PROC SQL and the COUNT function to generate counts and percentages with two different denominators: one based on the column total and another on the total counts of all subcategories.
- Utilize %DO loops to execute the above programs for each row and column individually.
- Concatenate all datasets with derived statistical results and then generate the output dataset and .rtf table.

This macro adeptly resolves patient ID variable name discrepancies across different source data, enabling users to effectively utilize various datasets. To tackle this challenge, the code checks whether the Key_variable in the Header sheet matches the Id_variable in the Body sheet; if not, it renames the Id_variable in the Body sheet to align with the Header sheet. An illustrative example in Figure 12 showcases the utilization of different Key_variable and Id_variable from distinct source data (one from EDC data and another from TPV data).

The row in the Header sheet using “subject” as the Key_variable:

Output_name	Column_number	Column_label	Indata	Key_variable	Condition
Table3	1	Total pts N (%)	in.elig2	subject	not missing(iepatien)

The rows in the Body sheet using “subjid” as the Id_variable:

Output_name	Row_number	Label0	Label1	Indata	Id_variable	Summary_method	Variable1	Variable2	Condition
Table3	26	Inclusion Criteria M	hsCRP mg/L	in.bsaf_temp	subjid	stat3	cnvresn		battrnam = 'HS-CRP'
Table3	27	Inclusion Criteria M	NT proBNP pg/	in.bsaf_temp	subjid	stat3	cnvresn		battrnam = 'NTPROBNP'

The output of rows 26 and 27 from the above sheets:

			Total pts N (%)
	Percutaneous coronary intervention (other than qualifying event)		33 (14.47%)
	Coronary artery bypass (other than qualifying event)		0
	Heart failure		24 (10.53%)
	Stroke		4 (1.75%)
	Transient Ischaemic Attack		3 (1.32%)
	Diabetes Mellitus		41 (17.98%)
Inclusion Criteria Metrics	26hsCRP mg/L	Median	0.26
		Q1	0.16
		Q3	0.42
	27NT proBNP pg/ml	Median	361
		Q1	140
		Q3	607
Medication	Use of any statin at randomization		228 (100%)
	Use of high intensity statin regimen at randomization		180 (78.95%)

Figure 12: Example of Different Merging Variables

Introducing this macro to the workflow presents challenges beyond the realm of programming, particularly in seamlessly incorporating information from an Excel file into a formal deliverable. The primary concern revolves around potential traceability gaps and heightened risks associated with the information not being directly embedded within the code. This poses considerations regarding version control and the transparency of the analytical process.

PROGRAMMING PRACTICE

When faced with a programming request, the initial and pivotal step involves accurately comprehending the request and translating it into precise programming logic. This programming logic is of paramount importance, as it fundamentally shapes the efficiency and quality of the resulting program. For developing the programming logic, the author advocates for designing code in a modular manner to derive maximum benefits. While the programming logic stands as a cornerstone, the selection of appropriate code is equally vital. Considerations in this realm fall into subcategories aimed at improving computer processing resource efficiency, streamlining debug/update/review processes, and enhancing overall robustness. Embracing several key principles constitutes good programming practice, and various papers have extensively discussed programming techniques in this context. For detailed insights, readers are encouraged to explore the referenced materials listed in the “REFERENCES” section. The following examples provide a glimpse into suggested techniques:

IMPROVING COMPUTER PROCESSING RESOURCE EFFICIENCY:

- Optimize by selectively reading in and retaining only the necessary datasets, variables, and observations, minimizing CPU and I/O operations. Utilize conditional logic (e.g., WHERE), and control variables using KEEP and DROP statements.
- Manage space by storing data as character when applicable.
- In most cases, joining by PROC SQL is more efficient than the merging operation in the data steps.

STREAMLINING DEBUG/UPDATE/REVIEW PROCESSES:

- Improve code clarity by incorporating comments, documentation, and maintaining well-formatted code. Documentation elements, including headers and change history, within each program are instrumental in providing other programmers with a clear understanding of the code. Structured comments within modules contribute to maintaining an organized comprehension of the entire program.
- Assign meaningful names and labels to datasets and variables to aid reviewers, code updaters, and validation programmers.
- Employ DO loops or macro programs to streamline repetitive tasks and minimize the risk of updating programs for new data.
- Consider the utilization of macro variables when assigning values that may undergo changes.

ENHANCING OVERALL ROBUSTNESS:

- Incorporate data checks to identify issues within uncleaned data, such as duplicates, missing values, or outliers. Generate user-defined logs, errors, notes, or employ PUT/%PUT statements for effective debugging.
- Maintain the length of numeric data to prevent inaccuracies in storage.
- Read in source data only once in the entire program to reduce the risk of programming mistakes or accidental modifications to source data.
- Utilize PROC DATASETS at the conclusion of the program to delete unnecessary datasets from the work library.

THOUGHTS ABOUT AUTOMATION BEYOND THIS MACRO TOOL

The entire statistical output development process involves setting up programs for the primary and validation sides, running these programs to generate outputs or create comparison results, and delivering the analysis, as depicted in Figure 13. Macro tools can be devised to assist in any of these three steps. The macro tool introduced in this paper serves as a valuable resource, enabling users to automate table generation for certain types of templates and significantly reduce the programming workload associated with individual program setup. The development of tools, such as macro programming, represent a substantial advancement in automating programming tasks. However, an additional avenue for enhancing automation exists within the broader process.

Within the author's company, the adoption of platforms like entimICE® exemplifies a process-oriented approach to automation. Such platforms empower programmers with the ability to schedule automatic runs at specified frequencies—whether daily, weekly, or monthly. Moreover, the IT department can configure automatic output transfers from the platform to a shared folder. This comprehensive approach to automation, covering both the running programs and the delivery of outputs, becomes exceptionally potent when complemented by the implementation of macro tools, such as the table generation tool presented here. Through integration at the process level, organizations can achieve heightened efficiency, reliability, and timeliness in programming workflows.

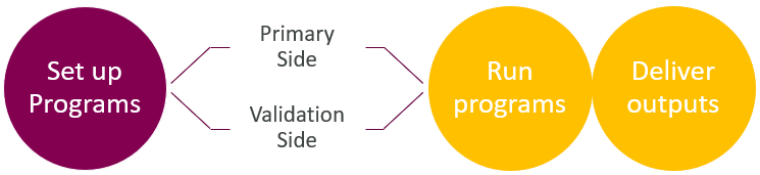

Figure 13: Statistical Output Development Process

CONCLUSION

Establishing programs consistently demands substantial programming efforts and entails varying levels of risk for errors among different programmers. The development of macro tools, as demonstrated by the table generation macro tool, emerges as an effective strategy for enhancing efficiency and overall quality. However, the promotion of such tools poses challenges and necessitates a significant investment of time. Implementing the macro tool in select studies, gradually enhancing its functionalities, and ultimately showcasing its advantages are pivotal steps that require ongoing dedication.

Similar to the development of individual programs, creating macros mandates careful consideration of programming techniques aligned with best practices. Moreover, in macro development, the emphasis on programming logic surpasses that of individual program development, making the adoption of a modular approach to code development highly recommended.

Beyond macro tools, enhancing efficiency in statistical output development from a process perspective holds significant importance. The combination of macro tools with process innovations or simplifications has the potential to streamline the entire table generation process, leading to substantial improvements in efficiency and quality. This holistic approach represents a valuable avenue for advancing programming workflows and achieving significant enhancements in the realm of data analysis and reporting.

APPENDIX

```
proc sql;
 select distinct min(row_number), max(row_number) into: min, :max from body ;
 select distinct min(column_number), max(column_number) into: clmin, :clmax from header;
quit;

%do i=   &min    %to   &max;

data _null_;
 set body;
 where row_number=&i;
 call symput('cal', strip(summary_method));
 call symput('indt', indata);
 call symput('var1', variable1);
-user's code-
```

Comments: In the Body sheet of each table, assign macro variables for the variables of each row. Additionally, in the Header sheet of each table, assign macro variables for the variables of each column.

```
data indt_&i %if %sysfunc(strip(%lowcase(&idv ))) ne %str( )  %then (rename=(&idv=&subj)) ; ;
 set &indt (encoding=any);
 %if &condyn=Y %then where &cond;;
 if &var1 = dummy then dummy=" ";
run;
```

Comments: If the Id_variable in the Body sheet is not equal to the Key_variable in the Header sheet, this code renames the Id_variable to match the Key_variable. A dummy variable is created as a character variable.

```
%put &cal;
%if %str(&stat_nm) = %str(&cal) %then %do;
proc transpose data=statistics out=stat_w&i._&j prefix=ord_;
 where Statistics_name="&cal";
 id statistics_order;
 var statistics;
run;

%if %substr(%sysfunc(strip(%lowcase(&cal))),1,3)=%str(sta) %then %do;
proc sql noprint;
  select distinct min(Statistics_order), max(Statistics_order) into: min_st, :max_st from
statistics  where Statistics_name="&cal";
quit;
data stat_w&i._&j.a;
 set stat_w&i._&j;
 stat_full=compress(catx(',', of ord_%trim(&min_st) - ord_%trim(&max_st)));
run;

proc sql noprint;
 select distinct countc(stat_full, ",")+1 into: max_c from stat_w&i._&j.a ;
quit;
```

```
/*get statistics*/
%do h=1 %to &max_c;

proc sql noprint;
 select distinct scan(stat_full, &h, ",") into: stat_lst from stat_w&i._&j.a ;
quit;

%if &stat_lst=SD %then %let stat_lst=std;

proc sql;
 create table stat_m_&i._&j as
 select &subj, &var1 as var1_, &i as row, &j as col, &h as statn, "&stat_lst" as stat
 from indt_&i where &subj in (&subjlst);
quit;

proc means data=stat_m_&i._&j;
 var var1_;
 by statn stat ;
 id row col ;
 output out=statsr_s_&i._&j._&h  &stat_lst=res ;
run;
-user's code for concatenating all statsr_s_&i._&j._&h datasets and processing for the final
dataset-
```

Comments: This code initially filters the Statistics sheet by the statistics group and subsequently utilizes PROC MEANS to generate each statistic from the statistics list. Since the template requires the presentation of SD, the code renames SD to STD before employing PROC MEANS.

```
%if  %sysfunc(strip(%lowcase(&cal))) =%str(cnt) or  %sysfunc(strip(%lowcase(&cal))) =%str(perct)
%then %do;
proc sql;
 %if %sysfunc(strip(%lowcase(&cal)))=perct %then %do;
 select distinct count(distinct &subj) into: deno from indt_&i where &subj in  (&subjlst);
 %end;
create table statsr_n_&i._&j as
 select  distinct &var1 as var1_, %if &nmcnt>0 %then  &var2  as var2_ , ;
 &i as row, &j as col, 1 as statn, "&cal" as stat, count(distinct &subj) as cnt_sub
 from indt_&i where &subj in (&subjlst) group by var1_ %if &nmcnt>0 %then , var2_ ; ;
quit;

data statsr_n_&i._&j (drop=var1_ %if &nmcnt gt 0 %then var2_;) ;
 format var1 %if &nmcnt>0 %then  var2 $200.;;
 set statsr_n_&i._&j;
 %if &tp1 eq 1 %then var1=strip(input(var1_, $200.)) ;;
 %if &tp1 ne 1 %then var1=strip(var1_);;
 %if &nmcnt gt 0 %then %do;
  %if &tp eq 1 %then var2=strip(input(var2_, $200.)) ;;
  %if &tp ne 1 %then var2=strip(var2_);;
 %end;
 %if %sysfunc(strip(%lowcase(&cal)))=cnt %then res_c=strip(input(cnt_sub, $20.)) ;;
 %if %sysfunc(strip(%lowcase(&cal)))=perct %then res_c=strip(input(cnt_sub, $20.))||"
("||strip(input(round(100*cnt_sub/&&denm&j, .01),  best.))||"%)";;
run;
-user's code for concatenating all statsr_n_&i._&j datasets and processing for the final dataset-
```

Comments: This code generates counts and percentages, using the column total as the denominators.

```
%if  %sysfunc(strip(%lowcase(&cal)))=%str(perctcat)  %then %do;
proc sql;
 create table statsr_pcn_&i._&j as
 select  distinct &var1 as var1_, %if &nmcnt>0 %then  &var2  as var2_ , ;
 &i as row, &j as col, 1 as statn, "&cal" as stat, count(distinct &subj) as cnt_sub_n
 from indt_&i where &subj in (&subjlst) and not missing(&var1) %if &nmcnt>0 %then and  not
missing(&var2) ;
 group by var1_ %if &nmcnt>0 %then , var2_ ;
```

```
 order by row, col, var1_ %if &nmcnt>0 %then , var2_ ;;

create table statsr_pcd_&i._&j as
 select  distinct %if &nmcnt>0 %then  &var1 as var1_ , ;
 &i as row, &j as col, count(distinct &subj) as cnt_sub_d
 from indt_&i where &subj in (&subjlst) and not missing(&var1) %if &nmcnt>0 %then and   not
missing(&var2);  %if &nmcnt>0 %then group by var1_ ;
 order by row, col %if &nmcnt>0 %then , var1_ ;;
quit;

data statsr_pci_&i._&j;
 merge statsr_pcn_&i._&j statsr_pcd_&i._&j;
 by row col  %if &nmcnt>0 %then  var1_ ;;
run;

data statsr_pc_&i._&j (drop=var1_ %if &nmcnt gt 0 %then var2_;);
 format res_c var1 %if &nmcnt>0 %then  var2 $200.;;
 set statsr_pci_&i._&j(in=a) statsr_pcd_&i._&j(in=b);
 %if &tp1 eq 1 %then var1=strip(input(var1_, $200.)) ;;
 %if &tp1 ne 1 %then var1=strip(var1_);;
 %if &nmcnt gt 0 %then %do;
  %if &tp eq 1 %then var2=strip(input(var2_, $200.)) ;;
  %if &tp ne 1 %then var2=strip(var2_);;
 %end;
if a then res_c=strip(input(cnt_sub_n, $20.))||" ("||strip(input(round(100*cnt_sub_n/cnt_sub_d,
.01),  best.))||"%)";
if b then res_c=strip(input(cnt_sub_d, $20.));
if a then row_1=1;
else do; row_1=0; stat="n"; end;
run;
-user's code for concatenating all statsr_pc_&i._&j datasets and processing for the final
dataset-
```

Comments: This code generates percentages, utilizing the total subcategories as the denominators.

```
data ttl_ft_temp;
format title $200.;
length title $200;
 title="";
 var="n1_C";output; var="n2_C"; output;
 var="n1_L";output; var="n2_L"; output;
 var="n1_R";output; var="n2_R"; output;
run;

proc sql;
 create table ttl_ft as
 select *, compress(ifc(not missing(location_title1), "n1_","")||location_title1 ) as  var1,
compress(ifc(not missing(location_title2), "n2_","")||location_title2 ) as var2
 from title_footnote ;
quit;

data ttl_ft_2(where=(not missing(var)));
 set ttl_ft(keep=Title1 var1 rename=(title1=title var1=var))  ttl_ft(keep=Title2 var2
rename=(title2=title var2=var));
run;

proc sort data=ttl_ft_temp; by var; run;
proc sort data=ttl_ft_2; by var; run;

proc sql;
 create table ttl_ft_3 as
 select distinct b.var, COALESCEC(a.title, b.title) as title_
 from ttl_ft_2 a right join ttl_ft_temp b
 on a.var=b.var;
quit;
```

```
data null;
 set ttl_ft_3;
 call symput(strip(var), title_);
run;

proc sql;
 select count(var1) into: cnt_mvar from final_2&pgm where not missing(var1);
 select count(label0) into: cnt_mlabel0 from final_2&pgm where not missing(label0);
 select count(label1) into: cnt_mlabel1 from final_2&pgm where not missing(label1);
quit;
```

Comments: This code generates macro variables for the variables in the Title_footnote sheet for each table.

```
options nodate number pageno=1 center mprint LINESIZE = 256  orientation=landscape;
ods listing close;
ods escapechar='^';
%if  &output_g ge 1 %then %do;
 ods rtf file=" &root/&state/&drug/&prot/temp/operational_report/&pgm..rtf"
 style= Journal BODYTITLE_AUX;
%end;
title j=left "&n1_L" j=c "&n1_C"  j=right "&n1_R"  bold ;
title2 j=left "&n2_L" j=c "&n2_C"  j=right "&n2_R"  bold ;
footnote J=left "&ft1" J=left;
footnote2 J=left "&ft2"  ;
footnote3 J=left "&ft3"  ;

proc report data=final_2&pgm missing out=outdt.&pgm nowd split="^" nocenter headline headskip
style(report)={width=100%};
column row %if &cnt_mlabel0 >0 %then label0; row_number %if &cnt_mlabel0 >0 %then label1; var1
row_1 %if &check>0 %then var2; statn  &colmin_ -- &colmax_ ;

 define row/ order order=internal noprint ;
 %if &cnt_mlabel0 >0 %then  define label0/order order=internal  style(column)=[cellwidth=10%] " "
style(header)=[textalign=l];;
  define row_number /order order=internal   noprint ;
 %if &cnt_mlabel0 >0 %then define label1/order order=internal style(column)=[cellwidth=12%] " "
style(header)=[textalign=l];;
 define var1/ /*%if &cnt_mvar >0 %then display*/order order=internal
style(column)=[cellwidth=8%] " " style(header)=[textalign=l]; /*%else order order=internal
noprint ; ;*/
 define row_1/order order=internal   noprint ;
 %if &check>0 %then define var2/ %if &cnt_mvar >0 %then order order=internal
style(column)=[cellwidth=8%] " " style(header)=[textalign=l]; ;
 define statn/order order=internal  noprint ;
%do w=&clmin %to &clmax; define col&w/ display style(column)=[cellwidth=10%] %str("&&colbl&w")
style(header)=[textalign=l]; %end;
run;
ods _all_ close;
title; title2;
footnote;footnote2;footnote3;footnote4;
```

Comments: This code generates the final output. If &output_g is greater than or equal to 1, indicating that Output_Generate is not specified as 'N', then the code generates the final .rtf output.

REFERENCES

Gregory S. Nelson, Jay Zhou (2012), “Good Programming Practices in Healthcare Creating Robust Programs”, SAS Global Forum.

John Schmitz (2020), “Best Practices in SAS® Programming to Support Ever Increasing Data Loads”, SAS Global Forum.

Jason Ford (2009), "Good Programming Practices in SAS", NESUG.

PhUSE GPP Steering Board, "Good Programming Practice at a Glance"

Kirk Paul Lafler, Mary Rosenbloom (2017), "Best Practice Programming Techniques for SAS® Users", SAS Global Forum.

Navneet Agnihotri, Sumit Pradhan, Rachel Brown (2022), "SAS Standard Programming Practices – Handy Tips for the Savvy Programmer", PharmaSUG.

Alice M. Cheng (1999), “Robust Programming Techniques in the SAS® System”, NESUG.

Charu Shankar (2019), "Top 10 SAS programming efficiencies", PharmaSUG.

ACKNOWLEDGMENTS

The author would like to acknowledge the AstraZeneca colleagues Grace Lu, Minsi Su and Kelly Fang for their thoughtful reviews of this paper.

CONTACT INFORMATION

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

Zhen (Laura) Li
AstraZeneca
zhen.li@astrazeneca.com