SESUG Paper 185-2019

Data-driven Programming Techniques using SAS® Macros to Semi-automate Generation of Descriptive Tables in Healthcare Research

Katie Mercaldi, Evidera

ABSTRACT

Nearly all healthcare studies include one or more tables with descriptive statistics summarizing characteristics of the sample population. When the relevant variables in the SAS analytic data set are properly formatted and labeled, the process of producing publication-quality descriptive tables can be streamlined so that little more is needed than the names of the variables to use as table columns and rows. This process is implemented with four SAS macros -- %COLCNTL, %ROWCNTL, %MKTABLE, and %MKREPORT. The SAS macros %COLCNTL and %ROWCNTL create "control files" in the form of SAS data sets that act as sets of instructions for designing the table's columns and rows respectively. The macros extract relevant metadata, including variable attributes such as labels and formats, about the column and row variables found in the specified analysis data set. Using this metadata, these macros then determine what type of summary statistics to produce for each row variable, such as means with standard deviations for continuous variables and counts with percentages for categorical measures. The SAS macros %MKTABLE and %MKREPORT generate the final output table. The %MKTABLE macro creates a version of the table as a SAS data set by converting the metadata to lists of macro variables using the INTO clause in PROC SQL or, alternately, by constructing macro calls within a data step with CALL EXECUTE. Finally, the %MKREPORT macro generates PROC REPORT syntax that creates the final deliverable-quality table in an Excel workbook.

INTRODUCTION

Whether based on healthcare data from clinical trials or real-world sources such as administrative claims, one of the first tables you typically see in published manuscripts contains descriptive statistics of demographic and/or clinical characteristics of the sample population. You likely already have macros that you use to calculate means, standard deviations, counts, percentages and other statistics that you report in these tables. Perhaps you have noticed that the variables you use in your analyses usually have labels that closely resemble the text printed in the table for that variable. Maybe you have thought about how variable formats can provide clues to what type of statistics you plan to calculate. (Or maybe you have been looking for the necessary motivation to consistently label and format all of your variables!) Here, we look at techniques to extract this information about the data, often described as "metadata," and using it to quickly and easily create deliverable-quality study tables.

To maintain some degree of flexibility in the appearance of the summary table, this process has not fully automated. Instead, we use "control files" to bridge the gap from the lists of variable names to include as columns and rows to the final table. The control files act as a set of instructions to create the table based on the variable attributes which you can review and tweak if necessary. Since variables used as columns and those used as rows play a slightly different role in creating the table, you need to construct separate control files for column and row variables. The macros %COLCNTL and %ROWCNTL create these two files with minimal effort. Even better, if you have several tables that use the same column structure but with different rows variables, the same column control file can be used with more than one row control file (and vice versa when running subgroup or sensitivity analyses). The control files provide the data that drive the table creation macros.

Once you have finalized the column and row control files, the %MKTABLE macro then reads the control files to produce the study table in SAS format. I provide two versions of the %MKTABLE macro. The first version of %MKTABLE constructs macro calls using our group's standard validated macros for calculating descriptive statistics and runs them using CALL EXECUTE. If you would like to use this method, you will almost certainly need to edit the %MKTABLE code to fit the macros you use for calculating these measures. The second version of %MKTABLE uses the TABULATE procedure to calculate all of the table

statistics within a single procedure run. This alternate method has the benefit of executing much more quickly than calculating each table cell with a separate macro call but lacks the flexibility to easily adapt the macro to report measures not directly calculated by PROC TABULATE, such as P-values or standardized differences.

With either method that you choose, you can use the %MKREPORT macro as long as you preserve the same table data structure as presented here. Just place the macro call within an ODS EXCEL definition to generate the REPORT procedure syntax needed to print the tables in an Excel workbook. The macro automatically determines labels, headers and column widths so that any post-processing needed is minimal-to-none.

I use the SASHELP.HEART data set to show how to use the macros throughout this paper so you can experiment with the code on your own. The full demonstration code is in Appendix G: Example Program. You can also access the complete set of programs at github.com/kmercaldi/SESUG-2019-Paper-185.

PREPARING THE DATA

Before you can dive into creating the tables, you must make sure that your analytic data set is set up for success. That means labelling and formatting any variable that you would like to include in your table. You most likely want to use the exact text that you want to appear in the table for row variable labels to keep later revisions to a minimum. For example, if you want to include the variable CHOLESTEROL from SASHELP.HEART as a row in your table, you might assign the following label:

```
label cholesterol = "Cholesterol, mg/dL";
```

I always use numeric data types for any variable that I plan to include in a table, so I have structured the macros in this paper to only work with numeric variables. Obviously, the numeric data type makes sense for continuous variables like age, but why use a numeric rather than character variable to represent categorical variables? To see, try running the FREQ procedure on the CHOL_STATUS variable:

```
proc freq data=sashelp.heart;
     tables chol_status / missing;
     run;
```

Output 1 shows the results:

The FREQ Procedure

Cholesterol Status

Chol_ Status	Frequency	Percent	Cumulative Frequency	Cumulative Percent
	152	2.92	152	2.92
Borderline	1861	35.73	2013	38.64
Desirable	1405	26.97	3418	65.62
High	1791	34.38	5209	100.00

Output 1. Frequency of Cholesterol Status as a Character Variable in SASHELP.HEART

You can see from the output that the frequencies are listed in alphabetical order, but you would probably prefer to see the results in order from “low” to “high” with “Desirable” listed first, followed by “Borderline,” “High,” and finally a category for patients with missing data. This ordering can be achieved by converting CHOL_STATUS to a numeric variable and applying a format containing the desired text for the table in the correct order:

```
proc format library = work;
     value chol 1 = "Desirable"
                2 = "Borderline"
                3 = "High"
                9 = "Unknown"
                ;
```

```
*** Snippet of code from the DATA step defining HEART ***;
* Use the same label as CHOL_STATUS variable with %GETLABEL ;
label nchol_status = "%getlabel(var=chol_status, data=sashelp.heart)";
format nchol_status chol.;

* The "=" operator expressions resolve to 0 or 1, allowing for the
  conversion of the character variable CHOL_STATUS to the numeric
  variable NCHOL_STATUS in one statement for efficiency ;
nchol_status = sum(1*(chol_status = "Desirable"),
                   2*(chol_status = "Borderline"),
                   3*(chol_status = "High"),
                   9*(chol_status = "")
                   );
```

Now the frequency results have a more logical order in Output 2:

The FREQ Procedure

Cholesterol Status

nchol_ status	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Desirable	1405	26.97	1405	26.97
Borderline	1861	35.73	3266	62.70
High	1791	34.38	5057	97.08
Unknown	152	2.92	5209	100.00

Output 2. Frequency of Cholesterol Status as a Formatted Numeric Variable

For column stratification variables, the formatted values provide the text for the individual table column headers. Use this to your advantage by assigning formats that matches the order and text that you want to appear in the table. The column variable labels group the individual columns under a second header printed directly above the individual column headers. In the SASHELP.HEART example, we use patient status reformatted as a numeric variable NSTATUS (values 0 = “Dead” and 1 = “Alive”) as columns, as well as a column summarizing all patients called ALLPT. These variables are coded, labeled and formatted in a similar manner as for CHOL_STATUS above:

```
proc format library = work;
     value allpt  1 = "All Patients"
                  ;
     value status 0 = "Dead"
                  1 = "Alive"
                  9 = "Unknown"
                  ;
     run;

**** Snippet of code from the DATA step defining HEART ***;
* Column variable definitions ;
label  allpt   = "All Patients"
       nstatus = "Patient Status"
       ;
format allpt allpt.
       nstatus status.
       ;
allpt = 1;
nstatus = sum(0*(status = "Dead"),
              1*(status = "Alive"),
              9*(status = ""),
              );
```

Once you have finished labelling and formatting all of the column and row variables in your analytic data set, you are ready to start making your tables.

CONSTRUCTING THE CONTROL FILES: %COLCNTL AND %ROWCNTL

You may have noticed the %GETLABEL macro in the LABEL statement for the variable NCHOL_STATUS in the section above. This handy little macro returns the label of a variable in an existing data set and plays an important role in the macros that create the control files. While you can access variable metadata in a number of ways, such as using the CONTENTS procedure or SASHELP.VCOLUMN (or SQL's equivalent DICTIONARY.COLUMNS), my favorite way is using the SAS variable information functions as macro functions to return the variable attribute. The benefit of using this method is that the %GETLABEL macro does not use any DATA steps or procedures and therefore can be used anywhere in a SAS program. Here is the code for the %GETLABEL macro:

```
%macro getlabel(var=, data=);

  %let dsid = %sysfunc(open(&data,i));
  %let varnum = %sysfunc(varnum(&dsid,&var));
  %if &varnum > 0 %then %qsysfunc(varlabel(&dsid,&varnum));
  %let rc = %sysfunc(close(&dsid));

%mend getlabel;
```

The %GETFORMAT macro is nearly identical but uses the VARFMT function in place of the VARLABEL function above.

THE COLUMN CONTROL FILE: %COLCNTL

The SAS code for the %COLCNTL macro is located in Appendix A: Full SAS Code for %COLCNTL. Below is an example of what the macro call might look like:

```
* Run the macro to create the column control file
 ------------------------------------------------------------------------
  DATA    = Name of the analytic data set containing the column variables
  OUT     = Desired name for the column control data set
  COLVARS = Space-delimited list of column variable names
  PVARS   = Space-delimited list of variables for statistical tests
            (optional)
 ------------------------------------------------------------------------;
%colcntl(data    = heart,
         out     = colcntl,
         colvars = allpt status,
         pvars   = status
         );
```

The resulting output data set COLCNTL in Display 1 looks like this:

	Analysis Data Set	Column Number	Column Variable Number	Column Variable Value	Column Variable	Column Variable Format	Column Label	Column Header
1	heart	1	1	1	allpt	ALLPT.	All Patients~N = 5,209	All Patients
2	heart	2	2	0	nstatus	STATUS.	Dead~N = 1,991	Patient Status
3	heart	3	2	1	nstatus	STATUS.	Alive~N = 3,218	Patient Status
4	heart	4	2	P	nstatus	STATUS.	P-value for Patient Status	Patient Status

Display 1. The Column Control Data Set: COLCNTL

The "Column Number" variable (COLNUM) shows that tables created with this control file will have 4 columns—First a column summarizing all study patients (ALLPT), then 2 columns stratified by whether the patient was alive or deceased at the end of the study (NSTATUS), then the final column with P-values

comparing the NSTATUS groups. The column order is determined first by the order of the variables listed in the COLVARS parameter in the %COLCNTL macro call and then by the unformatted order of the variable values. (Our formatted numeric variables come through once again!). The "Column Label" (COLLBL) values, derived from the formatted values and column sample size, will be printed directly above each column's statistics. "Column Header" (COLHDR) contains the column variable labels, extracted with the %GETLABEL macro described above, and groups all columns derived from the same variable.

Here is where the semi-automated aspect of this method comes into play. It is possible to extract the variable metadata in a "black box" macro that uses the information to make the tables directly without any intermediate step—And hopefully you have labeled and formatted your data so well that direct processing would have worked! But you probably want a little bit more control on the final appearance of your tables and maybe occasionally forget to label and format every variable perfectly for the table. Perhaps upon reviewing the column control file, you find that you want to tweak some of the labelling text or change the order of the columns (carefully). Because the control file is saved as a SAS data set, you can conveniently modify it just like you would any other data set.

For the example table, both the column header and column label for the first column read "All Patients." While ideally you might like to merge these two headings in the Excel version of the table, that is tricky to do with SAS alone. Instead, let's rename the column header to "Overall" for a better appearance. Maybe you prefer having "Alive" coded to one and "Dead" coded to zero for your time-to-event analyses but want to report the surviving patients' descriptive statistics column first. The implementation is simple:

```
* Modify the column control file as desired ;
data t01_colcntl;
     set colcntl;

     * Change column header for "All Patients" column ;
     if colvar = "allpt" then colhdr = "Overall";

     * Swap order for patient status values ;
     if collbl =: "Alive" then colnum = 2;
     else if collbl =: "Dead" then colnum = 3;

     run;

proc sort data=t01_colcntl;
     by colnum;
     run;
```

Display 2 shows the final column control file after applying these changes:

	Analysis Data Set	Column Number	Column Variable Number	Column Variable Value	Column Variable	Column Variable Format	Column Label	Column Header
1	heart	1	1	1	allpt	ALLPT.	All Patients^N = 5,209	Overall
2	heart	2	2	1	nstatus	STATUS.	Alive^N = 3,218	Patient Status
3	heart	3	2	0	nstatus	STATUS.	Dead^N = 1,991	Patient Status
4	heart	4	2	P	nstatus	STATUS.	P-value for Patient Status	Patient Status

Display 2. The Modified Column Control Data Set: T01_COLCNTL

THE ROW CONTROL FILE: %ROWCNTL

Perhaps not surprisingly, the %ROWCNTL macro uses many of the same techniques as %COLCNTL (see Appendix B: Full SAS Code for %ROWCNTL). Here is the macro call to create the row control file for our example table:

```
* Run the macro to create the row control file
 ----------------------------------------------------------
```

```
  DATA    = Name of the analytic data set
  OUT     = Desired name for the column control data set
  ROWVARS = Space-delimited list of row variable names
 ------------------------------------------------------;
%rowcntl(data    = heart,
         out     = rowcntl,
         rowvars = agechddiag gender abn_chol nchol_status cholesterol
         );
```

After execution, you can see the row control file ROWCNTL as shown in Display 3:

	Analysis Data Set	Row Number	Row Variable	Row Format	Row Label	Row Statistics	N Evaluated Row
1	heart	1	agechddiag		Age CHD Diagnosed	Continuous	Yes
2	heart	2	gender	GENDER.	Gender	Categorical	No
3	heart	3	cholesterol		Cholesterol, mg/dL	Continuous	Yes
4	heart	4	nchol_status	CHOL.	Cholesterol Status	Categorical	No
5	heart	5	abn_chol	YESNO.	Abnormal Cholesterol	Indicator	Yes

Display 3. The Row Control Data Set: ROWCNTL

Similar to the column control file, the row control file puts the row variables in the order specified in the ROWVARS parameter from the %ROWCNTL macro call. The row variable formats and labels are also determined automatically by the macro using %GETFORMAT and %GETLABEL.

You probably notice that the row variables AGECHDDIAG and CHOLESTEROL do not have formats. If you have made a habit of formatting all of your categorical variables with custom formats, then this should mean that the variable is a continuous numeric variable. In fact, if you look at the “Row Statistics” column (ROWSTATS), you will see that these variables are both listed as “Continuous” variables. The %ROWCNTL macro uses the row variable format to make an educated guess about what type of statistics you would like to calculate. It assumes that variables with missing, COMMA, DOLLAR or BEST formats are continuous variables. Then it looks for any variables with custom format YESNO, which is defined as follows:

```
value yesno 1 = "Yes"
            0 = "No"
            9 = "Unknown"
            ;
```

In our group, we commonly use this format for indicator variables where we are only interested in reporting the “Yes” responses in the table, so the macro assigns “Indicator” as the row statistics measure for any variables with this format. If you are interested in reporting both “Yes” and “No” values, you can manually change the ROWSTATS value to “Categorical” with a DATA step similar to how we modified the column control file. Any other format is assumed to be for a categorical variable. You may need slightly different definitions for selecting the correct row statistics in your typical work flow, such as differentiating between nominal and ordinal continuous variables, so adapt the macros as needed.

Finally, our group’s standard descriptive statistics macros have an optional parameter to report the number of patients evaluated for a given variable in cases of missing data. The %ROWCNTL macro automatically evaluates whether the number of non-missing values for the variable is less than the total sample and if so, turns on this option in the “N Evaluated Row” (NEVAL) column. Of course, you can turn the option off manually when editing the row control file, but I don’t recommend it! Your descriptive statistics macros may have other or additional options that you may want to incorporate in the control file macro definitions.

After you have reviewed and edited your column and row control files to meet your specifications, you are now ready to make your table. The only change I make to the example row control file is adding the unit “years” to the row label for age at CHD diagnosis (AGECHDDIAG) in the final row control data set T01_ROWCNTL. (For details see Appendix G: Example Program below.)

MAKING THE TABLE: %MKTABLE AND %MKREPORT

MAKING THE TABLE WITH CALL EXECUTE: %MKTABLE_V1

If you make descriptive tables often, you probably already have macros to calculate the statistics you typically report for continuous and categorical variables. Our group has three macros that we commonly use for these statistics: %CONTSTATS for continuous variables, %CATSTATS for categorical variables and %FLAGSTATS for indicator variables. The macros have options built in to calculate P-values and standardized differences as well. I have slightly modified these macros to work with %MKTABLE_V1 (in Appendix C: Full SAS Code for %MKTABLE_V1), and you might need to adapt yours as well. %MKTABLE_V1 combines the information from the column and row control files and then uses this information to construct a macro call for %CONTSTATS, %CATSTATS or %FLAGSTATS. It then runs each macro call in the code below using CALL EXECUTE in a null DATA step:

```
%* Choose the descriptive statistics macro based on the row type ;
if rowstats = "Continuous" then macro = '%contstats';
else if rowstats = "Categorical" then macro = '%catstats';
else if rowstats = "Indicator" then macro = '%flagstats';

%* Mask any special characters in the column and row labels ;
collbl = cats('%bquote(',collbl,')');
rowlbl = cats('%bquote(',rowlbl,')');

%* Construct the macro call for each column and row combination based on
   the information in the control file ;
runstats = cats(macro,
                "(data=",   data,
                ",colnum=", colnum,
                ",colvar=", colvar,
                ",colval=", colval,
                ",collbl=", collbl,
                ",rownum=", rownum,
                ",rowvar=", rowvar,
                ",rowlbl=", rowlbl,
                ",rowfmt=", rowfmt,
                ",neval=",  neval,
                ")");

%* Use CALL EXECUTE to run the macro calls enclosed within %NRSTR
   to prevent any macro variable definitions using CALL SYMPUTX
   or INTO with PROC SQL from resolving prematurely ;
call execute(cats('%nrstr(',runstats,')'));
```

MAKING THE TABLE WITH PROC TABULATE: %MKTABLE_V2

If you do not have these types of macros in your library or are unsure if your macros can be easily adapted to these methods, I have an alternate version of %MKTABLE. The second macro called %MKTABLE_V2 uses PROC TABULATE to create the table with some commonly reported descriptive statistics (see Appendix D: Full SAS Code for %MKTABLE_V2). This version is not currently set up to calculate P-values or other statistics not included in PROC TABULATE, so if you are interested in reporting these types of measures, I suggest adapting %MKTABLE_V1 to suit your needs. However, %MKTABLE_V2 runs much more quickly than %MKTABLE_V1, so if you only need basic statistics and have many large tables to run, %MKTABLE_V2 might be the better option. The efficiency comes from calculating all of the table statistics within a single PROC TABULATE run:

```
%* Use PROC TABULATE to calculate all continuous/categorical statistics
   simultaneously -- The MISSING option here is important to make sure
   that no observations are dropped from the analytic data set ;
```

```
proc tabulate data=&_data missing;

     %* Class variables include column, categorical and indicator
        variables -- PRELOADFMT option makes sure all possible row
        variable category values are reported, even if missing/zero
        value (with the PRINTMISS option below) ;
      class &_colvars;
      class &_catvars &_flagvars / preloadfmt;

     %* Continuous variables (where means/SD/median are needed) ;
     var &_contvars;

     %* Construct a "table" for each variable, based on its row type ;
     %do _i = 1 %to &_nrow;
         %if %upcase(&&_rowstats&_i) = CONTINUOUS %then %do;
             table &_colvars, &&_rowvar&_i, n mean std median min max
                   / printmiss;
         %end;
         %else %do;
             table &_colvars, &&_rowvar&_i, n colpctn / printmiss;
         %end;
     %end;

     %* Output the raw statistics for formatting ;
     ods output table = _stats0;

     run;
```

The macro variables in the SAS code above came from using the INTO clause of the SQL procedure. Here is an example:

```
proc sql noprint;
     %* Name of the analysis data set ;
     select distinct(data) into :_data from &colcntl;

     %* Row variables and requested statistics (save the number of rows) ;
     select rowvar, rowstats into :_rowvar1-, :_rowstats1- from &rowcntl;
     %let _nrow = &sqlobs;

     quit;
```

The second SELECT statement demonstrates a neat feature of the INTO clause. You can see that the macro variable names following the colons both end in “1-,” which allows PROC SQL to create a sequence of macro variables, one for each observation of the FROM data set. Just save the number of observations that PROC SQL captures automatically in the SQLOBS macro variable, and you can easily %DO loop your way through each value, as shown in the PROC TABULATE code above.

After calculating the statistics comes the arduous task of formatting the sprawling PROC TABULATE output data set. If you have been wanting to practice your array programming, I suggest you take a close look at the %MKTABLE_V2 program in Appendix D: Full SAS Code for %MKTABLE_V2 below. Otherwise, the macro does the work for you!

RUNNING %MKTABLE_V1 OR %MKTABLE_V2

Whether using %MKTABLE_V1 or %MKTABLE_V2, you call the macro using either method in the same way:

```
* Create the table using CALL EXECUTE (V1) and PROC TABULATE (V2)
 ------------------------------------------------------------------------
```

```
  OUT     = Desired name for the output data set containing the table text
  COLCNTL = Name of the column control data set
  ROWCNTL = Name of the row control data set
 ------------------------------------------------------------------------
%mktable_v1(out     = t01_v1,
            colcntl = t01_colcntl,
            rowcntl = t01_rowcntl
            );

%mktable_v2(out     = t01_v2,
            colcntl = t01_colcntl,
            rowcntl = t01_rowcntl
            );
```

Here is the table as SAS data set T01_V1 from %MKTABLE_V1 in Display 4:

	Row Number	Statistic Number	Row Label	All Patients~N = 5,209	Alive~N = 3,218	Dead~N = 1,991	P-value for Patient Status
1	1	L	Age CHD Diagnosed				
2	1	N	N evaluated	1,449	555	894	
3	1	1	Mean (SD)	63.3 (10.0)	62.9 (10.1)	63.5 (10.0)	0.29
4	1	2	Median (Range)	63 (32 - 90)	63 (32 - 88)	63 (33 - 90)	
5	2	L	Gender, n (%)				
6	2	1	Male	2,336 (44.8%)	1,241 (38.6%)	1,095 (55.0%)	<0.0001
7	2	2	Female	2,873 (55.2%)	1,977 (61.4%)	896 (45.0%)	
8	3	L	Cholesterol, mg/dL				
9	3	N	N evaluated	5,057	3,135	1,922	
10	3	1	Mean (SD)	227 (44.9)	222 (43.1)	236 (46.5)	<0.0001
11	3	2	Median (Range)	223 (96 - 568)	217 (115 - 435)	232 (96 - 568)	
12	4	L	Cholesterol Status, n (%)				
13	4	1	Desirable	1,405 (27.0%)	998 (31.0%)	407 (20.4%)	<0.0001
14	4	2	Borderline	1,861 (35.7%)	1,186 (36.9%)	675 (33.9%)	
15	4	3	High	1,791 (34.4%)	951 (29.6%)	840 (42.2%)	
16	4	9	Unknown	152 (2.9%)	83 (2.6%)	69 (3.5%)	
17	5	1	Abnormal Cholesterol, n (%)	1,791 / 5,057 (35.4%)	951 / 3,135 (30.3%)	840 / 1,922 (43.7%)	<0.0001

Display 4. The Table in SAS Format Created Using CALL EXECUTE: T01_V1

The data set T01_V2 from %MKTABLE_V2 in Display 5 is identical except that it lacks the P-value column:

	Row Number	Statistic Number	Row Label	All Patients~N = 5,209	Alive~N = 3,218	Dead~N = 1,991
1	1	L	Age CHD Diagnosed			
2	1	N	N evaluated	1,449	555	894
3	1	1	Mean (SD)	63.3 (10.0)	62.9 (10.1)	63.5 (10.0)
4	1	2	Median (range)	63 (32 - 90)	63 (32 - 88)	63 (33 - 90)
5	2	L	Gender			
6	2	1	Male	2,336 (44.8%)	1,241 (38.6%)	1,095 (55.0%)
7	2	2	Female	2,873 (55.2%)	1,977 (61.4%)	896 (45.0%)
8	3	L	Cholesterol, mg/dL			
9	3	N	N evaluated	5,057	3,135	1,922
10	3	1	Mean (SD)	227 (44.9)	222 (43.1)	236 (46.5)
11	3	2	Median (range)	223 (96 - 568)	217 (115 - 435)	232 (96 - 568)
12	4	L	Cholesterol Status			
13	4	1	Desirable	1,405 (27.0%)	998 (31.0%)	407 (20.4%)
14	4	2	Borderline	1,861 (35.7%)	1,186 (36.9%)	675 (33.9%)
15	4	3	High	1,791 (34.4%)	951 (29.6%)	840 (42.2%)
16	4	9	Unknown	152 (2.9%)	83 (2.6%)	69 (3.5%)
17	5	1	Abnormal Cholesterol	1,791 / 5,057 (35.4%)	951 / 3,135 (30.3%)	840 / 1,922 (43.7%)

Display 5. The Table in SAS Format Created Using PROC TABULATE: T01_V2

Both versions of the %MKTABLE macro use picture formats to select the number of significant digits reported for means, standard deviations, percentages and p-values (if applicable). You can see the definition for these formats in Appendix G: Example Program below.

ADDING CUSTOM LABELS: %ADDLABEL

While the table already looks quite presentable, we could perhaps improve it a bit by adding another row label to group the various cholesterol measures. You can use the %ADDLABEL macro to do just that (see Appendix E: Full SAS Code for %ADDLABEL):

```
* Add custom row label
 ------------------------------------------------------------------------
  DATA     = Name of the data set containing the table text
  OUT      = Desired name for the output table data set (optional, uses
             DATA parameter if missing)
  ROWCNTL  = Name of the row control data set
  LABEL    = Label text
  STARTVAR = First row variable name under the new label
  STOPVAR  = Last row variable name under the new label (optional, only
             applies to the START parameter if missing)
 ------------------------------------------------------------------------;
%addlabel(data     = t01_v1,
          out      = t01,
          rowcntl  = t01_rowcntl,
          label    = Cholesterol Measurements,
          startvar = cholesterol,
          stopvar  = abn_chol
          );
```

If your LABEL parameter has special characters like commas, quotes or percentage signs, be sure to put the text inside the %BQUOTE macro function to mask these characters. The STARTVAR and STOPVAR parameters tell the macro where to put the label and which rows fall under the new label so the macro can increase the indentation of their row labels. I use row variable names for these parameters rather than row numbers so you can easily reorder the variables in the row control file without necessarily changing the %ADDLABEL call. Of course, if you change the order of the row variables within the added label then you may need to make modifications here. Display 6 shows the table after adding the row label specified above:

	Row Number	Statistic Number	Row Label	All Patients~N = 5,209	Alive~N = 3,218	Dead~N = 1,991	P-value for Patient Status
1	1	L	Age CHD Diagnosed				
2	1	N	N evaluated	1,449	555	894	
3	1	1	Mean (SD)	63.3 (10.0)	62.9 (10.1)	63.5 (10.0)	0.29
4	1	2	Median (Range)	63 (32 - 90)	63 (32 - 88)	63 (33 - 90)	
5	2	L	Gender, n (%)				
6	2	1	Male	2,336 (44.8%)	1,241 (38.6%)	1,095 (55.0%)	<0.0001
7	2	2	Female	2,873 (55.2%)	1,977 (61.4%)	896 (45.0%)	
8	3	K	Cholesterol Measurements				
9	3	L	Cholesterol, mg/dL				
10	3	N	N evaluated	5,057	3,135	1,922	
11	3	1	Mean (SD)	227 (44.9)	222 (43.1)	236 (46.5)	<0.0001
12	3	2	Median (Range)	223 (96 - 568)	217 (115 - 435)	232 (96 - 568)	
13	4	L	Cholesterol Status, n (%)				
14	4	1	Desirable	1,405 (27.0%)	998 (31.0%)	407 (20.4%)	<0.0001
15	4	2	Borderline	1,861 (35.7%)	1,186 (36.9%)	675 (33.9%)	
16	4	3	High	1,791 (34.4%)	951 (29.6%)	840 (42.2%)	
17	4	9	Unknown	152 (2.9%)	83 (2.6%)	69 (3.5%)	
18	5	1	Abnormal Cholesterol, n (%)	1,791 / 5,057 (35.4%)	951 / 3,135 (30.3%)	840 / 1,922 (43.7%)	<0.0001

Display 6. Adding a Custom Row Label to the Table: T01

PRINTING THE TABLE IN EXCEL: %MKREPORT

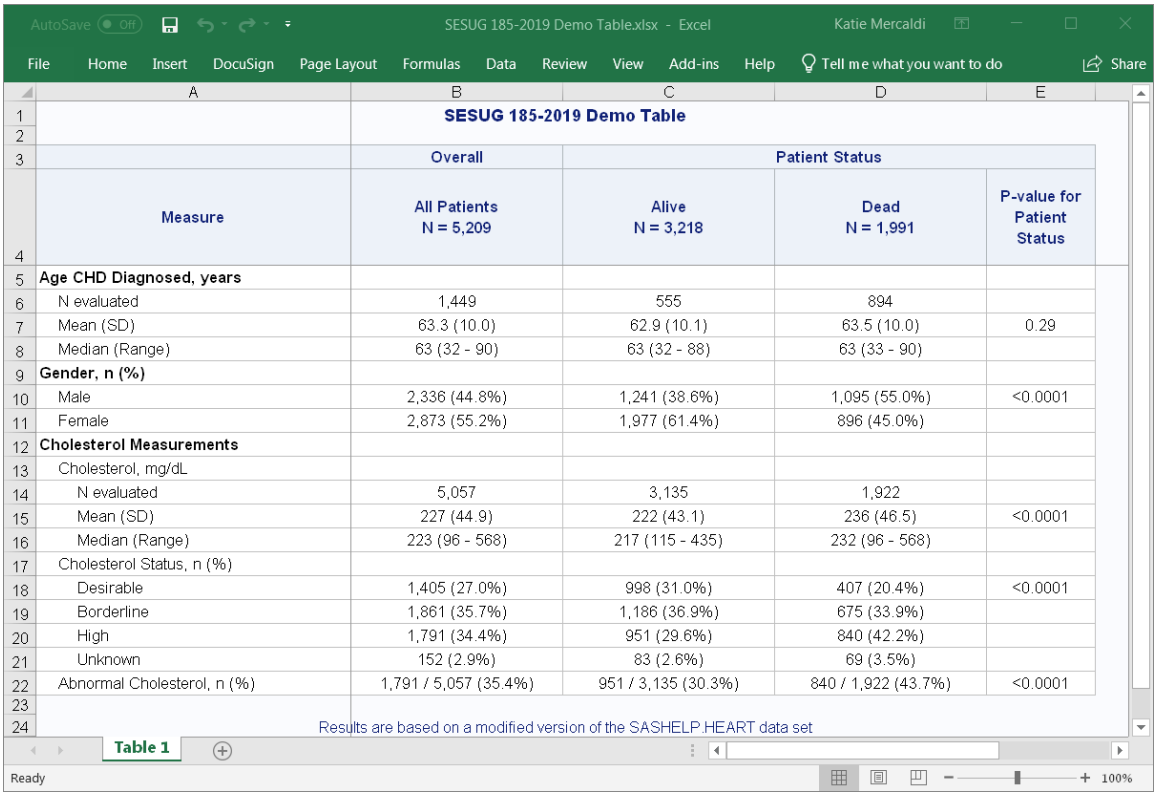
When you are satisfied with the appearance of the table as a SAS data set, you can then print it in a more palatable format to share with your non-SAS using colleagues. Our group typically delivers study tables in an Excel workbook, so the %MKREPORT macro is designed to work with the EXCEL output destination. With some minor tweaks, you can also use it with TAGSETS.EXCELXP if you are running earlier versions of SAS and, although not tested, possibly other destinations of interest. Execute as follows between ODS EXCEL opening and closing statements (details in Appendix F: Full SAS Code for %MKREPORT):

```
* Print the table with PROC REPORT
 ------------------------------------------------------------------------
  DATA    = Name of the data set containing the table text
  COLCNTL = Name of the column control data set
  SHEET   = Desired Excel sheet name
  BOLDROW = Row label levels to bold ("all" for all, 1 for top level
            labels, 2 for top and second level labels and so on)
 ------------------------------------------------------------------------;
title1 "SESUG 185-2019 Demo Table";
footnote1 "Results are based on a modified version of the SASHELP.HEART
data set";
%mkreport(data    = t01,
          colcntl = t01_colcntl,
          sheet   = Table 1,
          boldrow = 1

          );
```

In the code above, the TITLE and FOOTNOTE statements are not included in the %MKREPORT macro definition to give maximum flexibility for the number of titles and footnotes that you can add to the table.

Display 7 shows how the final unmodified table looks in Excel:



SESUG 185-2019 Demo Table

Measure	Overall: All Patients N = 5,209	Patient Status: Alive N = 3,218	Patient Status: Dead N = 1,991	P-value for Patient Status
Age CHD Diagnosed, years				
N evaluated	1,449	555	894	
Mean (SD)	63.3 (10.0)	62.9 (10.1)	63.5 (10.0)	0.29
Median (Range)	63 (32 - 90)	63 (32 - 88)	63 (33 - 90)	
Gender, n (%)				
Male	2,336 (44.8%)	1,241 (38.6%)	1,095 (55.0%)	<0.0001
Female	2,873 (55.2%)	1,977 (61.4%)	896 (45.0%)	
Cholesterol Measurements				
Cholesterol, mg/dL				
N evaluated	5,057	3,135	1,922	
Mean (SD)	227 (44.9)	222 (43.1)	236 (46.5)	<0.0001
Median (Range)	223 (96 - 568)	217 (115 - 435)	232 (96 - 568)	
Cholesterol Status, n (%)				
Desirable	1,405 (27.0%)	998 (31.0%)	407 (20.4%)	<0.0001
Borderline	1,861 (35.7%)	1,186 (36.9%)	675 (33.9%)	
High	1,791 (34.4%)	951 (29.6%)	840 (42.2%)	
Unknown	152 (2.9%)	83 (2.6%)	69 (3.5%)	
Abnormal Cholesterol, n (%)	1,791 / 5,057 (35.4%)	951 / 3,135 (30.3%)	840 / 1,922 (43.7%)	<0.0001

Results are based on a modified version of the SASHELP.HEART data set

Display 7. Final Table in Excel Format

The %MKREPORT macro uses information from the column control file to group columns derived from the same variable under one heading. The macro also dynamically determines an appropriate width for each column of the table based on the maximum character length of like columns: one length for the row label column, another for all columns containing summary statistics, and the last for all P-value columns. You may need to adjust the calculation of the column widths if you use a different style template than the default Excel template shown. You can also adjust which row labels are bolded using the BOLDROW parameter. %MKREPORT then generates the necessary PROC REPORT syntax using this information to print the table in a visually appealing manner.

Run the %MKREPORT macro as many times as you need with appropriate titles and footnotes within the ODS EXCEL definition to create an Excel workbook containing all of your study's descriptive tables in this format.

CONCLUSION

While writing these macros took a good deal of upfront work, they can drastically reduce the amount code you need to construct a professional looking table. If you format and label your analytic data set with the final result in mind, you can make the table in as little as 3 macro call statements:

```
%colcntl(data=, out=, colvars=, pvars=);
%rowcntl(data=, out=, rowvars=);
%mktable(out=, colcntl=, rowcntl=);
```

Then printing the table in Excel requires only the ODS EXCEL opening and closing statements, any desired TITLE and FOOTNOTE statements and the %MKREPORT macro call for the table. Besides reducing the amount of code needed to generate results, these macros used together should also add consistency and reliability to all of your descriptive tables.

Although this paper only applies the data-driven methods presented within to basic summary statistics, these techniques can be extrapolated to formatting the output of more advanced procedures, such as GLM, LOGISTIC, LIFETEST, PHREG and many others to meet all of your reporting needs.

REFERENCES

Carpenter, Art. 2012. Carpenter's Guide to Innovative SAS Techniques. Cary, NC: SAS Institute, Inc.

ACKNOWLEDGMENTS

The author would like to thank Kathy Fraeman for her invaluable contributions to this work.

CONTACT INFORMATION

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

Katie Mercaldi
katie.mercaldi@evidera.com
github.com/kmercaldi/SESUG-2019-Paper-185

APPENDIX A: FULL SAS CODE FOR %COLCNTL

```
/*-------------------------------------------------------------------------
  Program:      ColCntl.sas

  Platform:     SAS 9.4

  Description:  Macro to create a column control file to be used in
                conjunction with an associated row control file to
                dynamically create study tables

  Macros Used:  %GETLABEL
                %GETFORMAT
  -------------------------------------------------------------------------
  Parameter Definitions:
  -------------------------------------------------------------------------
  DATA    = SAS data set containing the table analysis variables

  OUT     = Desired name for SAS dataset containing column control
            information

  COLVARS = Space-delimited list of column variables

  PVARS   = Optional parameter for space-delimited list of column
            variables for which P-values should be calculated,
            or specify "all" for all column variables
  -------------------------------------------------------------------------*/
%macro colcntl(data    = ,
               out     = ,
               colvars = ,
               pvars   = ,
               );

  %local _i _j _ncolvars _npvars _colvar _pvar _pval _nvals;

  %* Get the number of column variables and number of variables where
     P-values  are requested ;
  %let _ncolvars = %sysfunc(countw(&colvars));
  %let _npvars   = %sysfunc(countw(&pvars));

  %* Loop through each column variable ;
  %do _i = 1 %to &_ncolvars;
      %let _colvar = %scan(&colvars,&_i);

      %* Get values of the column variable present in the data
         and the sample size for each column;
      proc sql noprint;
           select distinct &_colvar format=8. into :_val1- from &data;
           select count(&_colvar) into :_n1- from &data
           group by &_colvar;
           %let _nvals = &sqlobs;
           quit;

      %* Determine whether P-value column is requested ;
      %* Initialize P-value indicator to zero ;
      %let _pval = 0;
```

```
    %* Only include P-value column if the variable has at least 2
       values ;
    %if &_nvals > 1 %then %do;

        %* If the ALL option is given, then set the P-value indicator
           to one ;
        %if %upcase(&pvars) = ALL %then %let _pval = 1;

        %* Otherwise search the PVARS parameter for this column
           variable name ;
        %else %do;
            %let _j = 1;
            %do %until(_pval = 1 or &_j > &_npvars);
                %let _pvar = %scan(&pvars,&_j);
                %if &_colvar = &_pvar %then %let _pval = 1;
                %let _j = %eval(&_j+1);
            %end;
        %end;

    %end;

    %* Construct the initial control file information for this variable ;
    data _colcntl&_i;
         length colvar $ 32;

         colvarnum = &_i;
         colvar = "&_colvar";

         %* Create a record for each value of the column variables ;
         %do _j = 1 %to &_nvals;
             colval = &&_val&_j;
             coln = &&_n&_j;
             output;
         %end;

         %* Create a record for P-value column if requested ;
         %if &_pval %then %do;
             colval = .p;
             output;
         %end;

         run;
%end;

%* Combine the control information for all column variables ;
data &out (drop = coln);
     length data $ 50
            colnum colvarnum colval 8
            colvar colfmt $ 32
            collbl colhdr $ 200
            ;

     set _colcntl:;

     label data      = "Analysis Data Set"
           colnum    = "Column Number"
           colvarnum = "Column Variable Number"
```

```
            colval    = "Column Variable Value"
            colvar    = "Column Variable"
            colfmt    = "Column Variable Format"
            collbl    = "Column Label"
            colhdr    = "Column Header"
            ;

        * Define additional column information, using %GETFORMAT and
          %GETLABEL macros to retrieve the column variable format and
          label from the analysis data set ;
        colnum + 1;
        data   = "&data";
        colfmt = resolve(cats('%getformat(var=',colvar,", data=",data,")"));
        colhdr = resolve(cats('%getlabel(var=',colvar,", data=",data,")"));
        if colval ne .p then collbl = cat(strip(putn(colval,colfmt)),
                                          "~N = ",
                                          strip(put(coln,comma20.)));
        else collbl = catx(" ","P-value for ",colhdr);

        run;

  proc sort data=&out;
        by colnum;
        run;

  %* Delete intermediate data sets ;
  proc datasets nolist;
        delete _colcntl:;
        run;
        quit;

%mend colcntl;
```

APPENDIX B: FULL SAS CODE FOR %ROWCNTL

```
/*-------------------------------------------------------------------------
  Program:      RowCntl.sas

  Platform:     SAS 9.4

  Description:  Macro to create a row control file to be used in
                conjunction with an associated column control file to
                dynamically create study tables

  Macros Used:  %GETLABEL
                %GETFORMAT
  -------------------------------------------------------------------------
  Parameter Definitions:
  -------------------------------------------------------------------------
  DATA    = SAS data set containing the table analysis variables

  OUT     = Desired name for SAS dataset containing row control
            information

  ROWVARS = Space-delimited list of row variables
  -------------------------------------------------------------------------*/
%macro rowcntl(data    = ,
               out     = ,
               rowvars = ,
               );

  %local _i _j _nrowvars _rowvar;

  %* Get the number of row variables ;
  %let _nrowvars = %sysfunc(countw(&rowvars));

  %* Loop through each row variable ;
  %do _i = 1 %to &_nrowvars;
      %let _rowvar = %scan(&rowvars,&_i);

      %* Get the non-missing count for this variable ;
      proc sql noprint;
           select count(&_rowvar) into :_count&_i trimmed from &data
           %if %getformat(var=&_rowvar, data=&data) = YESNO.
               %then where &_rowvar in (0 1);;
           quit;

      %* Construct the initial control file data for this variable ;
      data _rowcntl&_i;
           rownum = &_i;
           rowvar = "&_rowvar";
           rowcnt = &&_count&_i;
           run;
  %end;

  %* Get the observation count for the full data set;
  proc sql noprint;
       select count(*) into :_allcnt from &data;
       quit;
```

```
  %* Combine the control information for all row variables ;
  data &out (drop = rowcnt);
       length data $ 50
              rownum 8
              rowvar rowfmt $ 32
              rowlbl $ 200
              rowstats $ 32
              neval $ 3
              ;
       set _rowcntl:;
       label data      = "Analysis Data Set"
             rownum    = "Row Number"
             rowvar    = "Row Variable"
             rowfmt    = "Row Format"
             rowlbl    = "Row Label"
             rowstats  = "Row Statistics"
             neval     = "N Evaluated Row"
             ;

       %* Define additional row information, using %GETFORMAT and
          %GETLABEL macros to retrieve the row variable format and label
          from the analysis data set ;
       data   = "&data";
       rowfmt = resolve(cats('%getformat(var=',rowvar,", data=",data,")"));
       rowlbl = resolve(cats('%getlabel(var=',rowvar,", data=",data,")"));

       %* Assign row type based on the variable format to determine the
          type of analysis
         ------------------------------------------------------------
          Continuous  = missing, COMMA, DOLLAR or BEST format
          Indicator   = YESNO format
          Categorical = all other formats
         ------------------------------------------------------------;
       if upcase(rowfmt) in: ("" "COMMA" "DOLLAR" "BEST")
          then rowstats = "Continuous";
       else if upcase(rowfmt) =: "YESNO" then rowstats = "Indicator";
       else rowstats = "Categorical";

       %* If the non-missing count for the variable is less than the
          total count for the data set, then set the option to print
          "N evaluated" row to "Yes";
       if rowcnt < &_allcnt then neval = "Yes";
       else neval = "No";

       run;

  proc sort data=&out;
       by rownum;
       run;

  %* Delete intermediate data sets ;
  proc datasets nolist;
       delete _rowcntl:;
       run;
       quit;

%mend rowcntl;
```

APPENDIX C: FULL SAS CODE FOR %MKTABLE_V1

```
/*-------------------------------------------------------------------
  Program:      MkTable_V1.sas

  Platform:     SAS 9.4

  Description:  Macro to create a table in SAS format based on column
                and row control files designed by %COLCNTL and %ROWCNTL
                macros using CALL EXECUTE
  -------------------------------------------------------------------
  Parameter Definitions:
  -------------------------------------------------------------------
  OUT     = Desired name for SAS dataset containing the table text

  COLCNTL = Name of the column control file for this table

  ROWCNTL = Name of the row control file for this table
  -------------------------------------------------------------------*/
%macro mktable_v1(out     = ,
                  colcntl = ,
                  rowcntl = ,
                  );

  %local _ncol _nrow _c _r;

  %* Combine information from the column and row control files ;
  proc sql noprint;
       create table _tcntl (drop = rdata) as
       select * from
       &colcntl as c full join &rowcntl (rename = (data=rdata)) as r
       on c.data = r.rdata
       order by colnum, rownum;
       quit;

  %* Get the number of columns and rows in the table ;
  proc sql noprint;
       select max(colnum), max(rownum)
       into :_ncol trimmed, :_nrow trimmed
       from _tcntl;
       quit;

  %* Data step to run the macros that calculate and format descriptive
     statistics ;
  data _null_;
       set _tcntl;
       length runstats $ 1000;

       %* Choose the descriptive statistics macro based on the row type ;
       if rowstats = "Continuous" then macro = '%contstats';
       else if rowstats = "Categorical" then macro = '%catstats';
       else if rowstats = "Indicator" then macro = '%flagstats';

       %* Mask any special characters in the column and row labels ;
       collbl = cats('%bquote(',collbl,')');
       rowlbl = cats('%bquote(',rowlbl,')');
```

```
      %* Construct the macro call for each column and row combination
         based on the information in the control file ;
      runstats = cats(macro,
                      "(data=",   data,
                      ",colnum=", colnum,
                      ",colvar=", colvar,
                      ",colval=", colval,
                      ",collbl=", collbl,
                      ",rownum=", rownum,
                      ",rowvar=", rowvar,
                      ",rowlbl=", rowlbl,
                      ",rowfmt=", rowfmt,
                      ",neval=",  neval,
                      ")");

      %* Use CALL EXECUTE to run the macro calls enclosed within %NRSTR
         to prevent any macro variable definitions using CALL SYMPUTX
         or INTO with PROC SQL from resolving prematurely ;
      call execute(cats('%nrstr(',runstats,')'));
      run;

  %* Combine all rows for each column ;
  %do _c = 1 %to &_ncol;
      data col&_c;
           set %do _r = 1 %to &_nrow;
                   col&_c.row&_r
               %end;
               ;
           run;
  %end;

  %* Combine all columns to create the final table ;
  data &out;
       merge %do _c = 1 %to &_ncol;
                 col&_c
             %end;
             ;
       by rownum statnum rowlabel;
       run;

  %* Delete intermediate data sets ;
  proc datasets nolist;
       delete _tcntl
              %do _c = 1 %to &_ncol;
                  col&_c
                  %do _r = 1 %to &_nrow;
                      col&_c.row&_r
                  %end;
              %end;
              ;
       run;
       quit;

%mend mktable_v1;
```

APPENDIX D: FULL SAS CODE FOR %MKTABLE_V2

```
/*-------------------------------------------------------------------
  Program:      MkTable_V2.sas

  Platform:     SAS 9.4

  Description:  Macro to create a table in SAS format based on column
                and row control files designed by %COLCNTL and %ROWCNTL
                macros using PROC TABULATE
  -------------------------------------------------------------------
  Parameter Definitions:
  -------------------------------------------------------------------
  OUT     = Desired name for SAS dataset containing the table text

  COLCNTL = Name of the column control file for this table

  ROWCNTL = Name of the row control file for this table
  -------------------------------------------------------------------*/
%macro mktable_v2(out     = ,
                  colcntl = ,
                  rowcntl =
                  );

  %local _i _data _colvars _null _nrow _ncol _contvars _catvars
         _ncont _contvar _contstats;

  %* Extract needed information for creating the table from the column
     and row control files ;
  proc sql noprint;
       %* Name of the analysis data set ;
       select distinct(data) into :_data from &colcntl;

       %* Column variable names, ordered as in the column control file ;
       select distinct(colvar), colvarnum
       into :_colvars separated by " ", :_null separated by " "
       from &colcntl order by colvarnum;

       %* Column labels, ordered as in the column control file ;
       select collbl, colnum into :_collbl1-, :_null
       from &colcntl order by colnum;

       %* Row variables and requested statistics (save the number of rows) ;
       select rowvar, rowstats into :_rowvar1-, :_rowstats1- from &rowcntl;
       %let _nrow = &sqlobs;

       %* Row variables names by analysis type ;
       select rowvar into :_contvars separated by " " from &rowcntl
       where upcase(rowstats) = "CONTINUOUS";
       select rowvar into :_catvars separated by " " from &rowcntl
       where upcase(rowstats) = "CATEGORICAL";
       select rowvar into :_flagvars separated by " " from &rowcntl
       where upcase(rowstats) = "INDICATOR";

       quit;

  %* Use PROC TABULATE to calculate all continuous/categorical statistics
```

```
   simultaneously -- The MISSING option here is important to make sure
   that no observations are dropped from the analytic data set ;
ods listing close;

proc tabulate data=&_data missing;

     %* Class variables include column, categorical and indicator
        variables -- PRELOADFMT option makes sure all possible row
        variable category values are reported, even if missing/zero
        value (with the PRINTMISS option below) ;
      class &_colvars;
      class &_catvars &_flagvars / preloadfmt;

     %* Continuous variables (where means/SD/median are needed) ;
     var &_contvars;

     %* Construct a "table" for each variable, based on its row type ;
     %do _i = 1 %to &_nrow;
         %if %upcase(&&_rowstats&_i) = CONTINUOUS %then %do;
             table &_colvars, &&_rowvar&_i, n mean std median min max
                   / printmiss;
         %end;
         %else %do;
             table &_colvars, &&_rowvar&_i, n colpctn / printmiss;
         %end;
     %end;

     %* Output the raw statistics for formatting ;
     ods output table = _stats0;

     run;

ods listing;

%* Create a list of continuous variable names with "_:" appended to each
   for array processing ;
%let _contstats = %str();
%let _ncont = %sysfunc(countw(&_contvars));

%do _i = 1 %to &_ncont;
    %let _contvar = %scan(&_contvars,&_i);
    %let _contstats =  &_contstats &_contvar._:;
%end;

%* Condense the PROC TABULATE output statistics ;
data _stats1 (keep = colvar colval rowvar statnum n pct mean std
                     median min max);
     set _stats0;
     length colvar rowvar revname $ 200;

     array colvars{*} &_colvars;
     array contvars{*} &_contstats;
     array catvars{*} &_catvars &_flagvars;
     array pctn{*} pctn:;

     %* Get the column variable names ;
     do i = 1 to dim(colvars);
```

```
      if colvars{i} ne . then do;
         colvar = vname(colvars{i});
         colval = colvars{i};
      end;
   end;

   %* Get categorical variable names and statistic number (for each
      category value) ;
   do i = 1 to dim(catvars);
      if catvars{i} ne . then do;
         rowvar = vname(catvars{i});
         statnum = catvars{i};
      end;
   end;

   %* Condense percentages for categorical/indicator variables into
      one percentage variable called PCT ;
   do i = 1 to dim(pctn);
      if pctn{i} ne . then pct = pctn{i};
   end;

   %* Condense continuous variable statistics creating one variable
      for each statistic ;
   do i = 1 to dim(contvars);
      revname = upcase(reverse(vname(contvars{i})));
      if contvars{i} ne . then do;
         if revname =: reverse("_N") then do;
            rowvar = strip(lowcase(reverse(substr(revname,3))));
            n = contvars{i};
         end;
         else if revname =: reverse("_MEAN") then mean = contvars{i};
         else if revname =: reverse("_STD") then std = contvars{i};
         else if revname =: reverse("_MEDIAN")
            then median = contvars{i};
         else if revname =: reverse("_MIN") then min = contvars{i};
         else if revname =: reverse("_MAX") then max = contvars{i};
      end;
   end;

   %* if N is missing then set to zero ;
   if n = . then n = 0;

   run;

%* Get the number of decimal places for continuous variable ;
proc contents data = &_data (keep = &_contvars)
              out  = _fmtd (keep = name formatd)
              noprint;
   run;

%* Combine information from the column and row control files ;
proc sql noprint;
   create table _tcntl0 (drop = rdata) as select *
   from &colcntl as c full join
        &rowcntl (rename = (data=rdata)) as r
   on c.data = r.rdata;
```

```
     %* Add format decimals to the control file information ;
     create table _tcntl (drop = name) as select *
     from _tcntl0 as t left join _fmtd  as f
     on upcase(t.rowvar) = upcase(f.name)
     order by colnum, rownum;

     quit;

%* Combine table control file information with the table statistics ;
proc sql noprint;
     create table _stats (drop = cvar cval rvar) as
     select * from
     _tcntl as t join
     _stats1 (rename = (colvar=cvar colval=cval rowvar=rvar)) as s
     on upcase(t.colvar) = upcase(s.cvar) and
        t.colval = s.cval and
        upcase(t.rowvar) = upcase(s.rvar)
     order by rownum, statnum, colnum;
     quit;

%* Get the number of columns reported ;
proc sql noprint;
     select max(colnum) into :_ncol trimmed from _stats;
     quit;

%* Create the final table ;
data &out (keep = rowlabel rownum statnum col1-col&_ncol);
     set _stats (where = (not (upcase(rowstats) = "INDICATOR" and
                              statnum = 9)));
     by rownum statnum colnum;

     label rowlabel = "Row Label"
           rownum   = "Row Number"
           statnum  = "Statistic Number"
           %do _i = 1 %to &_ncol;
               col&_i = "&&_collbl&_i"
           %end;
           ;

     length rowlabel col1-col&_ncol mean_std1-mean_std&_ncol
            median_rng1-median_rng&_ncol n_pct1-n_pct&_ncol $ 200
            ;

     retain n_eval1-n_eval&_ncol mean_std1-mean_std&_ncol
            median_rng1-median_rng&_ncol n_pct1-n_pct&_ncol;

     %* Arrays for column building ;
     array col{&_ncol} $;
     array n_eval{&_ncol};
     array mean_std{&_ncol} $;
     array median_rng{&_ncol} $;
     array n_pct{&_ncol} $;

     %* Use COMMA format for unformatted continuous variables and
        add number of decimal places from PROC CONTENTS ;
     if rowfmt = "" then rowfmt = "COMMA20.";
     if upcase(rowstats) = "CONTINUOUS" then rowfmt = cats(rowfmt,formatd);
```

```
%* Initialize statistics arrays at the start of each new row
   variable ;
if first.rownum then do;
   call missing(of n_eval{*});
   call missing(of mean_std{*});
   call missing(of median_rng{*});
   call missing(of n_pct{*});
end;

%* Calculate column N for each row variable ;
n_eval{colnum} + n;

%* Recalculate percentages for indicator variables after
   eliminating "Unknown" responses (see WHERE clause in
   SET statement for this data set);
if upcase(rowstats) = "INDICATOR"
   then pct = n / n_eval{colnum} * 100;

%* Get statistics for each column for continuous row variables ;
if upcase(rowstats) = "CONTINUOUS" then do;
   %* Use DOLLAR format instead of AUTODEC for cost variables ;
   if upcase(rowfmt) =: "DOLLAR" then
      mean_std{colnum} = cat(strip(putn(mean,rowfmt))," (",
                             strip(putn(std,rowfmt)),")"
                             );
   %* Otherwise use AUTODEC to automatically determine appropriate
      decimal places ;
   else mean_std{colnum} = cat(strip(put(mean,autodec.))," (",
                               strip(put(std,autodec.)),")"
                               );
   %* Use format from the data (or COMMA if missing) with original
      decimal places for median and range measures ;
   median_rng{colnum} = cat(strip(putn(median,rowfmt))," (",
                            strip(putn(min,rowfmt)),", ",
                            strip(putn(max,rowfmt)),")"
                            );
end;

%* Get statistics for each column for categorical and indicator
   row variables ;
else do;
   if upcase(rowstats) = "INDICATOR" and
      upcase(neval) = "YES" then
      n_pct{colnum} = cat(strip(put(n,comma20.))," / ",
                          strip(put(n_eval{colnum},comma20.)),"  (",
                          strip(put(pct,autopct.)),")"
                          );
   else n_pct{colnum} = cat(strip(put(n,comma20.))," (",
                            strip(put(pct,autopct.)),")"
                            );
end;

%* Create the final output columns ;
if upcase(rowstats) = "CONTINUOUS" and last.rownum then do;
   * Row variable label ;
   statnum = .1;
```

```
      rowlabel = rowlbl;
      output;

      * N evaluated row ;
      if upcase(neval) = "YES" then do;
         statnum = .n;
         rowlabel = 'A0A0A0A0'x || "N evaluated";
         do i = 1 to &_ncol;
            col{i} = strip(put(n_eval{i},comma20.));
         end;
         if last.rownum then output;
      end;

      * Mean and standard deviation ;
      statnum = 1;
      rowlabel = 'A0A0A0A0'x || "Mean (SD)";
      do i = 1 to &_ncol;
         col{i} = mean_std{i};
      end;
      output;

      * Median and range (min - max) ;
      statnum = 2;
      rowlabel = 'A0A0A0A0'x || "Median (range)";
      do i = 1 to &_ncol;
         col{i} = median_rng{i};
      end;
      output;
   end;

   else if upcase(rowstats) = "CATEGORICAL" then do;
      if last.statnum then do;
         %* Count and percentage ;
         rowlabel = 'A0A0A0A0'x || strip(putn(statnum,rowfmt));
         do i = 1 to &_ncol;
            col{i} = n_pct{i};
         end;

         %* If all columns for the unknown/missing value for a
            categorical variable have zero count then suppress the
            output ;
         if not (find(rowlabel,"Missing","i") or
                 find(rowlabel,"Unknown","i"))
                 %do _i = 1 %to &_ncol;
                     or n_pct&_i ^=: "0"
                 %end;
                 then output;
      end;

      if last.rownum then do;
         * Row variable label ;
         statnum = .l;
         call missing(of col{*});
         rowlabel = rowlbl;
         output;

         * N evaluated row ;
```

```
            if upcase(neval) = "YES" then do;
               statnum = .n;
               rowlabel = 'A0A0A0A0'x || "N evaluated";
               do i = 1 to &_ncol;
                  col{i} = strip(put(n_eval{i},comma20.));
               end;
               output;
            end;
         end;
      end;

      else if upcase(rowstats) = "INDICATOR" and last.statnum then do;
         %* Count and percentage for "Yes" values only ;
         rowlabel = rowlbl;
         do i = 1 to &_ncol;
            col{i} = n_pct{i};
         end;
         if statnum = 1 then output;
      end;

      run;

  %* Sort the output data set by the order specified in the row control
     file ;
  proc sort data=&out;
       by rownum statnum;
       run;

  %* Delete intermediate data sets ;
  proc datasets nolist;
       delete _stats:
              _tcntl:
              _fmtd
              ;
       run;
       quit;

%mend mktable_v2;
```

APPENDIX E: FULL SAS CODE FOR %ADDLABEL

```
/*-------------------------------------------------------------------
  Program:      AddLabel.sas

  Platform:     SAS 9.4

  Description:  Macro to add row labels to existing table data sets
  -------------------------------------------------------------------
  Parameter Definitions:
  -------------------------------------------------------------------
  DATA     = Name of the data set containing the table text

  OUT      = Optional name for the output data -- If blank then the DATA
             parameter will be used

  LABEL    = New row label text

  STARTVAR = The table row variable name above which the label will be
             placed

  STOPVAR  = The last row variable where the added label applies for
             adjusting indentation -- If blank, only the STARTROW will
             be adjusted
  -------------------------------------------------------------------*/
%macro addlabel(data     = ,
                out      = ,
                rowcntl  = ,
                label    = ,
                startvar = ,
                stopvar  =
                );

  %local _startnum _stopnum;

  %* Use the DATA parameter for the output data set and the STARTVAR
     variable for STOPVAR if these parameters are not specified ;
  %if not %length(&out) %then %let out = &data;
  %if not %length(&stopvar) %then %let stopvar = &startvar;

  %* Get the row numbers associated with the starting and ending
     row variables for which the label applies ;
  proc sql noprint;
       select rownum into :_startnum trimmed from &rowcntl
       where rowvar = "&startvar";
       select rownum into :_stopnum trimmed from &rowcntl
       where rowvar = "&stopvar";
       quit;

  %* Assign special missing characters to order labels ;
  data _null_;
       set &data (keep = rownum statnum where = (rownum = &_startnum));
       by rownum statnum;
       if first.rownum;

       %* Determine the first statistic value for the row -- If a
          label already exists then set the special missing character
```

```
         to one letter before ;
      statchar = strip(lowcase(put(statnum,4.)));
      if anyalpha(statchar) then
         call symputx("_statnum",cats(".",byte(rank(statchar)-1)));

      %* Otherwise use .L to represent the label ;
      else call symputx("_statnum",".l");

      run;

  %* Add the label to the table ;
  data &out (drop = i);
      set &data;
      by rownum statnum;
      array col $ col: pval:;

      %* Increase the indentation by one for all rows included
         under the new label ;
      if &_startnum <= rownum <= &_stopnum then
         rowlabel = cat('A0A0A0A0'x,strip(rowlabel));
      output;

      %* Create the new label based on the first observation for
         the row and output ;
      if first.rownum and rownum = &_startnum then do;
         statnum = &_statnum;
         rowlabel = "&label";
         do i = 1 to dim(col);
            col{i} = "";
         end;
         output;
      end;
      run;

  proc sort data=&out;
      by rownum statnum;
      run;

%mend addlabel;
```

APPENDIX F: FULL SAS CODE FOR %MKREPORT

```
/*-------------------------------------------------------------------
  Program:      MkReport.sas

  Platform:     SAS 9.4

  Description:  Macro to generate PROC REPORT syntax for printing
                study tables in an Excel workbook
  -------------------------------------------------------------------
  Parameter Definitions:
  -------------------------------------------------------------------
  DATA    = Name of the data set containing the table text

  COLCTNL = Name of column control file for the table to get column
            header information

  SHEET   = Excel worksheet name for the table

  ROWHDR  = Header text for the row label column (default = "Measure")

  BOLDROW = Indentation level for bold row labels. Default is all row
            labels are bolded ("all"). Use integers (1, 2, 3...) to
            change which labels are bolded.
  -------------------------------------------------------------------*/
%macro mkreport(data    = ,
                colcntl = ,
                sheet   = ,
                rowhdr  = Measure,
                boldrow = all
                );

  %local _i _colnames _coldefs _ncol _col _rlen _slen _plen;

  %* Replace >= and <= with unicode characters in the row label column ;
  data _temp;
       set &data;
       rowlabel = tranwrd(rowlabel,"<=","^{unicode '2264'x}");
       rowlabel = tranwrd(rowlabel,">=","^{unicode '2265'x}");
       run;

  %* Get names of columns included in the final table ;
  proc contents data=&data (keep=col:) out=_colnames (keep=name) noprint;
       run;

  proc sql noprint;
       select quote(upcase(name)) into :_colnames separated by " "
       from _colnames;
       quit;

  %* Get column information from the column control file ;
  data _null_;
       set &colcntl (where = (cats("COL",colnum) in (&_colnames))) end = eof;
       by colvarnum colhdr;
       length coldef $ 1000;
       retain coldef;
```

```
      %* Identify P-value columns ;
      call symputx(cats("_colval",colnum),colval);

      %* Build the REPORT column definition from the column header text and
         column variable names ;
      if first.colhdr then do;
         coldef = strip(cat(strip(coldef)," (",
                            quote(strip(colhdr))," (",
                            cats("col",colnum)));
         if last.colhdr then coldef = cats(coldef,"))");
      end;
      else if last.colhdr
         then coldef = cat(strip(coldef)," ",cats("col",colnum),"))");
      else coldef = cat(strip(coldef)," ",cats("col",colnum));

      %* Put the column definitions into a macro variable to insert
         in the PROC REPORT syntax ;
      if eof then do;
         call symputx("_coldefs",coldef);
         call symputx("_ncol",colnum);
      end;

      run;

%* Determine column lengths based on the table data ;
data _collength;
      set _temp;
      array col{&_ncol} $ col:;

      %* Row label character length ;
      rlen = length(rowlabel);

      %* Text may wrap >90 characters so use 0 to "autofit" row height which
         may need some adjustment in Excel--otherwise set row height to 15 ;
      if rlen > 90 then call symputx("_rowht","0");
      else call symputx("_rowht","15");

      %* Statistic and P-value column character lengths ;
      do i = 1 to &_ncol;
         if symget(cats("_colval",i)) = "P"
            then plen = max(length(col{i}),plen);
         else slen = max(length(col{i}),slen);
      end;

      run;

%* Get maximum column length for row label, stats and P-value columns ;
proc sql noprint;
      select min(max(rlen),90), max(max(slen),10), max(max(plen),10)
      into :_rlen trimmed, :_slen trimmed, :_plen trimmed
      from _collength;
      quit;

%* Set the sheet name, row height and title/footnote width ;
ods excel options(sheet_name = "&sheet"
                  row_heights = "0,&_rowht,0,0,0,0,0"
                  title_footnote_width = "%eval(&_ncol+1)"
```

```
                    );

  %* Create the table report ;
  proc report data = _temp nowindows split = "~" missing
              style(header) = {font_weight = bold
                               font_size   = 10pt
                               just        = center
                               vjust       = m
                               protectspecialchars = off
                               }
              style(column) = {font_size = 10pt
                               just      = center
                               vjust     = m
                               };
       column rownum rowlabel &_coldefs c;

       %* Column definitions ;
       define rownum   / order noprint;
       define rowlabel / display "&rowhdr"
                         style(column) = {cellwidth = &_rlen.em
                                          just      = left
                                          };
       %* Assign width for each data column as calculated above ;
       %do _i = 1 %to &_ncol;
           %if &&_colval&_i = P %then %let _len = &_plen;
           %else %let _len = &_slen;
           define col&_i / display style = {cellwidth = &_len.em};
       %end;
       define c / noprint;

       compute c;
       %* Bold row labels at the requested level ;
       %if %upcase(&boldrow) = ALL or &boldrow > 0 %then %do;
           if col1 = ""
           %if %upcase(&boldrow) ne ALL %then %do;
              and index(rowlabel,
                        %do _i = 1 %to &boldrow;
                            'A0A0A0A0'x %if &_i ne &boldrow %then ||;
                        %end;
                        ) ne 1
           %end;
           then call define(_row_,"style","style = [font_weight = bold]");
       %end;
       endcomp;

       run;

  %* Delete intermediate data sets ;
  proc datasets nolist;
       delete _temp
              _colnames
              _collength
              ;
       run;
       quit;

%mend mkreport;
```

APPENDIX G: EXAMPLE PROGRAM

```
/*-----------------------------------------------------------------------
  Program:       SESUG 185-2019 Demo.sas

  Platform:      SAS 9.4

  Description:   Prepares the SASHELP.HEART data set for table creation
                 by:
                  - Adding/modifying labels to match text in the table
                  - Converting character to numeric variables to preserve
                    desired order in the output table
                  - Applying formats to variables with the same text as
                    the table

                 Demonstrates use of the included macros to create an Excel
                 table based on the modified version of SASHELP.HEART

  Macros Used:   %GETLABEL
                 %COLCNTL
                 %ROWCNTL
                 %MKTABLE_V1
                 %MKTABLE_V2
                 %ADDLABEL
                 %MKREPORT

  Input:         SASHELP.HEART

  Output:        SESUG 185-2019 Demo Table.XLSX
  -----------------------------------------------------------------------*/
%let projloc = /* Full path to your project folder */;
%let fmtloc = &projloc\SASFormats;
%let macloc = &projloc\SASMacros;
%let rptloc = &projloc\Reports;

libname projfmt "&fmtloc";
filename mac "&macloc";
filename xltables "&rptloc\SESUG 185-2019 Demo Table.xlsx";

options msglevel = i nodate minoperator mprint mautosource
        sasautos = ("." mac sasautos) fmtsearch = (projfmt work);


*** Formats needed for table creation ***;
proc format library = projfmt;
  value allpt  1 = "All Patients"
               ;
  value yesno  1 = "Yes"
               0 = "No"
               9 = "Unknown"
               ;
  value status 0 = "Dead"
               1 = "Alive"
               9 = "Unknown"
               ;
  value gender 1 = "Male"
               2 = "Female"
```

```
              9 = "Unknown"
              ;
  value chol  1 = "Desirable"
              2 = "Borderline"
              3 = "High"
              9 = "Unknown"
              ;
  * Format for P-values ;
  picture pvalfmt (round)
            . = ""
            low-<0.0001 = "<0.0001" (noedit)
            0.0001-<0.000995 = "9.9999"
            0.000995-<0.00995 = "9.999"
            0.00995-<0.995 = "9.99"
            0.995-1.0 = "1.00" (noedit)
            ;
  * Percentage formats that automatically assigns decimal places ;
  picture autopct (round)
            . = ""
          low      - -99.95  = "0,000,000,000,000,009%" (prefix='-')
          -99.95<  -  -0.095 = "09.9%" (prefix='-')
           -0.095< -  -0.01  = "9.99%" (prefix='-')
           -0.01<  -  <0     = "-<0.01%" (noedit)
            0                = "0%" (noedit)
            0<      -  <0.01  = "<0.01%" (noedit)
            0.01    -  <0.095 = "9.99%"
            0.095   - <99.95  = "09.9%"
           99.95    - high     = "0,000,000,000,000,009%" (mult=1)
            ;
  * Decimal format that automatically assigns significant figures ;
  picture autodec (round)
            . = ""
          low        - -99.95    = "0,000,000,000,000,009" (prefix='-')
          -99.95<    -  -0.095   = "09.9" (prefix='-')
           -0.095<   -  -0.0095  = "9.99" (prefix='-')
           -0.0095<  -  -0.00095 = "9.999" (prefix='-')
           -0.00095< -  -0.0001  = "9.9999" (prefix='-')
           -0.0001<  -  <0       = "-<0.0001" (noedit)
            0                    = "0" (noedit)
            0<       -  <0.0001  = "<0.0001" (noedit)
            0.0001   -  <0.00095 = "9.9999"
            0.00095  -  <0.0095  = "9.999"
            0.0095   -  <0.095   = "9.99"
            0.095    - <99.95    = "09.9"
           99.95     - high      = "0,000,000,000,000,009"
            ;
  run;


*** Prepare the analytic data set ***;

* Reformat the SASHELP Framingham Heart Study data set to work with table
  creation macros ;
data heart;
  set sashelp.heart;

  * Add or relabel variables with text desired in the final table ;
```

```
  label  allpt        = "All Patients"
         nstatus      = "Patient Status"
         gender       = "Gender"
         cholesterol  = "Cholesterol, mg/dL"
         nchol_status = "%getlabel(var=chol_status, data=sashelp.heart)"
         abn_chol     = "Abnormal Cholesterol"
         ;

  * Apply formats with table text to the numeric categorical variables ;
  format allpt        allpt.
         nstatus      status.
         gender       gender.
         nchol_status chol.
         abn_chol     yesno.
         ;

  * Variable for "All Patients" column ;
  allpt = 1;

  *** Recode existing character variables in the table as numeric ***;
  * Patient status is used as a column stratification variable ;
  nstatus      = sum(0*(status = "Dead"),
                     1*(status = "Alive"),
                     9*(status = "")
                     );
  * Categorical row variables ;
  gender       = sum(1*(sex = "Male"),
                     2*(sex = "Female"),
                     9*(sex = "")
                     );
  nchol_status = sum(1*(chol_status = "Desirable"),
                     2*(chol_status = "Borderline"),
                     3*(chol_status = "High"),
                     9*(chol_status = "")
                     );
  * Create an indicator variable for abnormal cholesterol ;
  abn_chol     = sum(1*(nchol_status = 3),
                     9*(nchol_status = 9)
                     );
  run;


*** Create the control files for the table ***;

* Run the macro to create the column control file
 ----------------------------------------------------------------------
  DATA    = Name of the analytic data set
  OUT     = Desired name for the column control data set
  COLVARS = Space-delimited list of column variable names
  PVARS   = Space-delimited list of variables for statistical tests
            (optional)
 ----------------------------------------------------------------------;
%colcntl(data    = heart,
         out     = colcntl,
         colvars = allpt nstatus,
         pvars   = nstatus
         );
```

```
* Modify the column control file as desired ;
data t01_colcntl;
     set colcntl;

     * Change column header for "All Patients" column ;
     if colvar = "allpt" then colhdr = "Overall";

     * Swap order for patient status values ;
     if collbl =: "Alive" then colnum = 2;
     else if collbl =: "Dead" then colnum = 3;

     run;

proc sort data=t01_colcntl;
     by colnum;
     run;

proc print data=t01_colcntl noobs label;
     run;

* Run the macro to create the row control file
 ------------------------------------------------------------
  DATA    = Name of the analytic data set
  OUT     = Desired name for the column control data set
  ROWVARS = Space-delimited list of row variable names
 ------------------------------------------------------------;
%rowcntl(data    = heart,
         out     = rowcntl,
         rowvars = agechddiag gender cholesterol nchol_status abn_chol
         );

* Modify the row control file as needed ;
data t01_rowcntl;
     set rowcntl;
     if upcase(rowvar) = "AGECHDDIAG" then rowlbl = cats(rowlbl,", years");
     run;

proc print data=t01_rowcntl noobs label;
     run;


*** Make the table specified by the control files ***;

* Create the table
 ------------------------------------------------------------------------
  OUT     = Desired name for the output data set containing the table text
  COLCNTL = Name of the column control data set
  ROWCNTL = Name of the row control data set
 ------------------------------------------------------------------------;

* Using CALL EXECUTE ;
%mktable_v1(out     = t01_v1,
            colcntl = t01_colcntl,
            rowcntl = t01_rowcntl
            );
```

```
* Using PROC TABULATE ;
%mktable_v2(out     = t01_v2,
            colcntl = t01_colcntl,
            rowcntl = t01_rowcntl
            );

* Add custom row label
-------------------------------------------------------------------------
  DATA     = Name of the data set containing the table text
  OUT      = Desired name for the output table data set (optional, uses
             DATA parameter if missing)
  ROWCNTL  = Name of the row control data set
  LABEL    = Label text
  STARTVAR = First row variable name under the new label
  STOPVAR  = Last row variable name under the new label (optional, only
             applies to the START parameter if missing)
 -------------------------------------------------------------------------;
%addlabel(data     = t01_v1,
          out      = t01,
          rowcntl  = t01_rowcntl,
          label    = Cholesterol Measurements,
          startvar = cholesterol,
          stopvar  = abn_chol
          );


*** Print the table in an Excel workbook ***;
ods escapechar = "^";
ods listing close;

ods excel file = xltables
    options(fittopage = 'no'
            embedded_titles = 'yes'
            embedded_footnotes = 'yes'
            flow = 'tables'
            zoom = '100'
            orientation='Landscape'
            row_repeat = 'header'
            pages_fitheight = '100'
            center_horizontal = 'yes'
            center_vertical = 'no'
            frozen_headers = "yes"
            frozen_rowheaders = "1"
            );

* Print the table with PROC REPORT
 -------------------------------------------------------------------------
  DATA    = Name of the data set containing the table text
  COLCNTL = Name of the column control data set
  SHEET   = Desired Excel sheet name
  ROWHDR  = Header text for the row label column (default is "Measure")
  BOLDROW = Row label levels to bold ("all" for all, 1 for top level
            labels, 2 for top and second level labels and so on)
 -------------------------------------------------------------------------;
title1 "SESUG 185-2019 Demo Table";
footnote1 "Results are based on a modified version of the SASHELP.HEART data
set";
```

```
%mkreport(data    = t01,
          colcntl = t01_colcntl,
          sheet   = Table 1,
          boldrow = 1);

ods excel close;
ods listing;
```