MWSUG 2019 - Paper RF-017

3 ways to get Pretty Excel-Style Tables: PROC REPORT, PROC TABULATE, and Help from SAS Enterprise Guide ®

Brooke Ellen Delgoffe, Marshfield Clinic Research Institute, Marshfield, WI

ABSTRACT

SAS programmers with a variety of experience levels may be asked to provide tabular or summary data in an Excel-style format (stacked headers, colored headers, bolded total lines, etc.). This paper explores 3 different ways to produce and export Microsoft Excel®-style tables into Output Delivery System (ODS) locations or the results window using SAS® 9.4 or SAS® Enterprise Guide ® 7.15. Addition of ODS style elements will help readers to apply aesthetically pleasing colors and formatting to their output. The use of Enterprise Guide will help new users of SAS perform these tasks with little to no SAS programming knowledge, while helping more versed SAS programmers utilize pre-written code as a starting point. Exploration of how to use the SQL procedure in combination with the REPORT procedure to display distinct counts and other uniquely formatted summary statistics will give programmers a succinct way to display summary statistics in the presence of repeated measurement.. For example, a commonly used measure of body size is Body Mass Index (BMI), and multiple entries for a single patient may be needed to provide average BMI per patient per period, but distinct patient counts may still be desired. A series of examples like this one will be used to cover each of the methods.

INTRODUCTION

As the language of SAS advances, its uses expand, and its users change. It is important to consider a solution from multiple perspectives not only in terms of knowledge base of the programmer but also uniqueness of the data being used. The way a programmer uses the language of SAS can be almost entirely unique, just like the way a person uses spoken language. The key is to explore multiple ways to achieve the desired goal and determine which is best for any given situation. Since there is rarely only one “right” way to program a given solution, instead consider how you might exploit each of these methods for their unique contributions and also how you might combine them. This paper focuses on the production of “Pretty Excel-style” tables that can be exported directly into Microsoft Excel® native files using the ODS Excel Destination location. We use two easily reproducible examples that showcase situations for using each of the three techniques explored. A SAS program containing full code and SAS Enterprise Guide Project files for producing the results seen in this paper are included in the supplemental materials submitted with the paper.

PROC SQL WITH PROC REPORT

EXAMPLE: DISTINCT PATIENT COUNTS

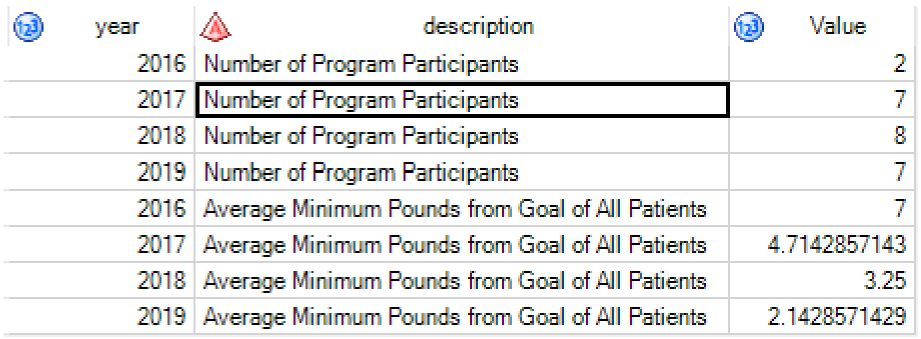
An annual progress report in excel-style has been requested for patients enrolled in a 5-year weight loss program where patients are required to be seen for measurements once a quarter. A new record (row) of data results from each visit to the clinic with variables: patient identifier (PID), quarter of visit (quarter), year of visit (year), program (program_id), goal (target_bmi), and body mass index (BMI). We will create an intermediate data set for each of two tables we seek to create: “Counts of Interest” and “Average Pounds from Goal”, use PROC REPORT to make the “Pretty Table”, and export into an EXCEL Destination spreadsheet using ODS EXCEL.

NOTE: This example is conducted using data set WORK.WEIGHT_LOSS, available in the DATA step of supplemental program “RF-017 Patient Counts Example Code.sas” (which does not contain actual patient data). The example is entirely fictitious and not based on a real scenario.

USING PROC SQL TO OBTAIN AND FORMAT COUNTS OF INTEREST

First, we use the unique summary power of PROC SQL to create the input data sets:

```
PROC SQL;
create table Counts_of_Interest as
select distinct year “Year”
    ,“Number of Program Participants” as description “Description”
    ,count(distinct pid) as Value
    from weight_loss
    group by year
union all
select distinct year
    ,“Average Minimum Pounds from Goal of All Patients” as description
    ,mean(min_pounds_from_goal) as Value
    From (select year
                ,PID
                ,min(bmi-target_bmi) as min_pounds_from_goal
          From weight_loss
          group by year, PID) find_min
    group by year
;QUIT;
```

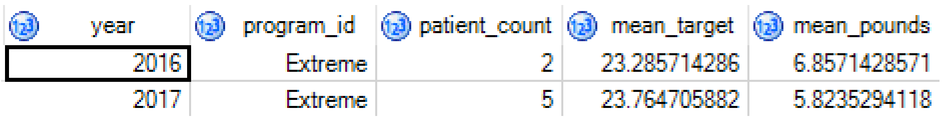


year	description	Value
2016	Number of Program Participants	2
2017	Number of Program Participants	7
2018	Number of Program Participants	8
2019	Number of Program Participants	7
2016	Average Minimum Pounds from Goal of All Patients	7
2017	Average Minimum Pounds from Goal of All Patients	4.7142857143
2018	Average Minimum Pounds from Goal of All Patients	3.25
2019	Average Minimum Pounds from Goal of All Patients	2.1428571429

Figure 1. Counts_of_Interest data set preview

Error! Reference source not found.

```
PROC SQL;
create table Average_Pounds_from_Goal as
select year “Program Year”
    ,program_id format=program_desc. “Program”
    ,count(distinct pid) as patient_count “Number of Patients”
    ,mean(target_bmi) as mean_target “Average Target BMI”
    ,mean(pounds_from_target) as mean_pounds “Average Pounds from Target”
    From (select pid, year, quarter, program_id, target_bmi
                , mean(bmi-target_bmi) as pounds_from_target
                From weight_loss
                group by pid, year, quarter) avg_per_quarter
    group by year, program_id
;QUIT;
```



year	program_id	patient_count	mean_target	mean_pounds
2016	Extreme	2	23.285714286	6.8571428571
2017	Extreme	5	23.764705882	5.8235294118

Figure 2. Average_Pounds_from_Goal data set preview

Distinct Value Counts

One of the biggest advantages of using PROC SQL is that it provides number of distinct values instead of number of rows as needed to provide the summary value of interest. In the above code for the average_pounds_from_goal data set we consider all values of pounds_from_target (all rows from subquery) to find the average pounds from target for the whole group. In the same query we only count the distinct values of PID to provide the number of unique patients (only rows with a unique value are counted; not a row count). As of now, PROC REPORT cannot provide number of unique values by itself.

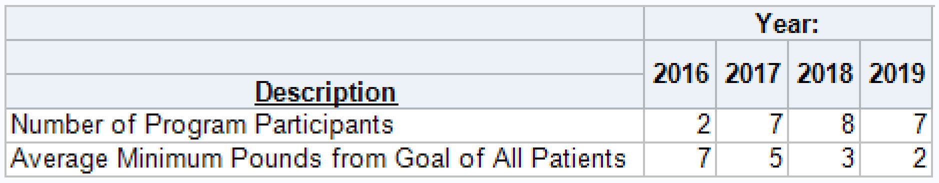
Subqueries for Controlled Summarization of Multiple Values

The use of sub-queries is also very helpful in determining how multiple values for a grouping are handled. In the “find_min” subquery used to help calculate “Average Minimum Pounds from Goal of All Patients” we take the minimum difference between Body Mass Index (bmi) at visit and Goal BMI (target_bmi) per patient. The subquery provides one value per patient, the minimum, to be used in finding the average for the group. In the “avg_per_quarter” subquery used to help calculate “Average Pounds from Target” we specify that the average of multiple values is given (mean difference in bmi and target_bmi). Similar to the “find_min” subquery, “avg_per_quarter” provides one value per patient (per quarter per year of the program) to the main query. Unlike the above query it deals with multiple values per group by taking the average instead of the minimum. This allows us to deal with different numbers of observations per patient and forces each patient to only contribute one value. This way, a patient who comes in 3 times in a quarter does not contribute to the average more than the patient who only comes in once. One can argue that with consistent weight loss assumed the first value might be a better fit since the patients with multiple measurements will have a lower average than their one measurement peers, but the mean was chosen for demonstration.

USING PROC REPORT TO DISPLAY GROUPED DATA WITH STACKED HEADERS

Next, we will use the REPORT procedure to create the Excel-style table and utilize the STYLE(location) argument to add pretty formatting:

```
proc report DATA=counts_of_interest style(header)={foreground=black};
column description ("Year:" year, (value));

define description / 'Description' group descending
style(header)={text_decoration=underline};
define year / '' across;
define value / '' analysis format=2.;
run;
```



	Year:			
Description	2016	2017	2018	2019
Number of Program Participants	2	7	8	7
Average Minimum Pounds from Goal of All Patients	7	5	3	2

Output 1. PROC REPORT: Style added with Across Variable

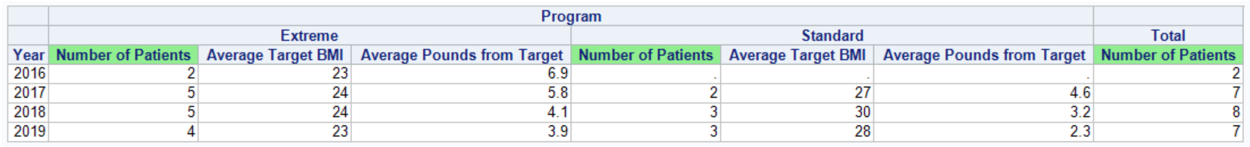
Using Across Variables in PROC REPORT

One great reason to use PROC REPORT to create your excel-style table is for the built-in “across variable” functionality. This allows you to use a grouping variable to create individual columns for levels of that variable (like is done above with the variable year). We identify a variable as an across variable by placing a comma after the variable name in the column statement and adding “across” as the variable type in the define statement. For more reading on across variables in PROC REPORT, see *Sailing Over the ACROSS Hurdle in PROC REPORT* (Zender, 2014).

```
column description ("Year:" year, (value));
define year / '' across;
```

When you use this in conjunction with a data set that already has a count/value variable, identify the count/value variable as an “analysis” variable; telling SAS to use this in calculations. Since there is only one value per combination of the grouping variables and we are not using this variable to add a summary line or column, this could also be left as a display variable.

```
proc report data=average_pounds_from_goal;
column year program_id, (patient_count mean_target mean_pounds) ("Total"
patient_count);

define year           / 'Year' group;
define program_id     / 'Program' across;
define patient_count / analysis format=2.
style(header)={background=lightgreen};
define mean_pounds    / format=3.1;
define mean_target / format=3.1;
run;
```



	Program						
	Extreme			Standard			Total
Year	Number of Patients	Average Target BMI	Average Pounds from Target	Number of Patients	Average Target BMI	Average Pounds from Target	Number of Patients
2016	2	23	6.9	.	.	.	2
2017	5	24	5.8	2	27	4.6	7
2018	5	24	4.1	3	30	3.2	8
2019	4	23	3.9	3	28	2.3	7

Output 2. PROC REPORT Stacked Headers and Summary Column

Stacked Headers = Commas and Parentheses

Breaking down the COLUMN statement is the most important part of understanding how stacked headers, side by side stacked headers, and the rows are differentiated. When we look at the column statement, we should first notice the commas and spaces. A space means the next variable will be next to the value of the current variable. A comma means the next variable will be stacked under the value of the current variable. Parentheses are used to group variables when we want more than one variable to be placed in a certain location and/or we want a single header to apply to all variables in the group.

```
column year program_id, (patient_count mean_target mean_pounds) ("Total"
patient_count);
```

Some times it is easier to read these statements backwards: patient_count will have the label “Total”. Total will be placed NEXT TO (space) patient_count, mean_target, and mean_pounds, which are grouped together and will be UNDER (comma) the value of program_id. Year will be placed NEXT TO (space) them.

The green header in Output 2 is used to point out the use of the same variable multiple times in the table. Since ACROSS variables are a special type of grouping variable, the values at their intersections will be grouped values. However, if the variable patient_count is used only with the grouping variable year then values will be summed across to produce the desired “Total” row. Since the patient can only be part of one program, this is okay, but care should be taken in making sure that the total row is not inflated by patients being in more than one group count.

CUSTOMIZING PROC REPORT OUTPUT WITH STYLE ELEMENTS

When you use style(header) on the PROC REPORT line, it applies this style to all headers in the output; hence all headers in Output 1 have black text (usually they would have navy blue to comply with HTMLBlue style attribution; default in SAS Enterprise Guide ™ for results window).

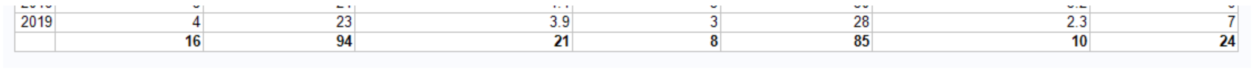
```
proc report DATA=counts_of_interest style(header)={foreground=black};
define description / 'Description' group descending
style(header)={text_decoration=underline};
```

When the same style(header) argument is used as part of the define statement(s), it only applies to the specific variable's header, hence only the header "Description" is underlined. In either position, they add attributes to a column's header (Description is both black and underlined).

If a RBREAK statement was added to provide a column summary using the following statement:

```
rbreak after / summarize style(summary)={font_weight=bold};
```

we would see the resulting line in the output. The location "summary" refers to the summary line created by the RBREAK statement:



2019	4	23	3.9	3	28	2.3	7
	16	**94**	**21**	**8**	**85**	**10**	**24**

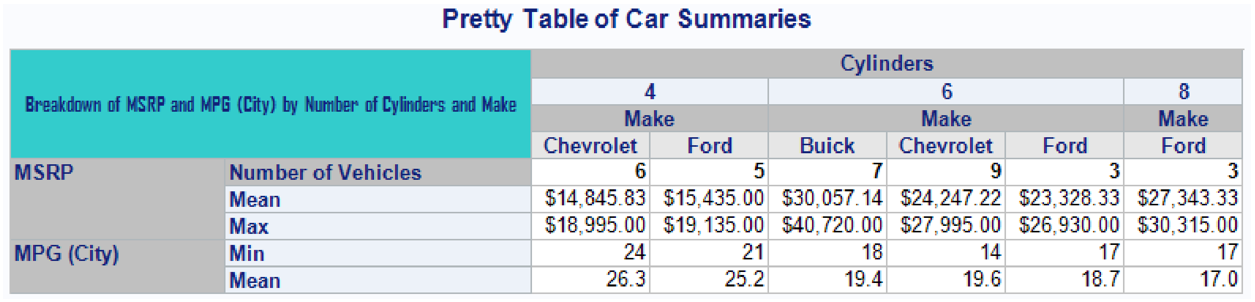
Output 3. Example of RBREAK in PROC REPORT

As a note of caution, PROC REPORT does not perpetuate distinct patient counts when it provides its summary line. The RBREAK statement was not included in the original code for the table because it provides inaccurate summarization in this case.

PROC TABULATE FOR PRESENTING MULTIPLE STATISTICS ON THE SAME VARIABLE

EXAMPLE: STACKED HEADERS USING SASHELP.CARS

One of the most characteristic features of Excel is the ability to use stacked headers to cover multiple statistics for a single variable; like in Output 4 below. While we've already seen one way to create stacked headers, the TABULATE procedure handles multiple statistics for a single variable with ease. Similar to PROC REPORT, the TABULATE procedure can utilize the style(location)= option to add formatting. As you see in Output 4, this formatting can be quite detailed in PROC TABULATE.



Pretty Table of Car Summaries

Breakdown of MSRP and MPG (City) by Number of Cylinders and Make		Cylinders					
		4		6			8
		Make		Make			Make
		Chevrolet	Ford	Buick	Chevrolet	Ford	Ford
MSRP	Number of Vehicles	**6**	**5**	**7**	**9**	**3**	**3**
	Mean	$14,845.83	$15,435.00	$30,057.14	$24,247.22	$23,328.33	$27,343.33
	Max	$18,995.00	$19,135.00	$40,720.00	$27,995.00	$26,930.00	$30,315.00
MPG (City)	Min	24	21	18	14	17	17
	Mean	26.3	25.2	19.4	19.6	18.7	17.0

Output 4. PROC TABULATE Output with Stacked Headers and Formatting

This example was created using the following as the final code:

```
TITLE1 "Pretty Table of Car Summaries";
PROC TABULATE DATA=SASHELP.CARS;
    WHERE (Type = 'Sedan' AND Make IN ('Buick','Ford','Chevrolet'));
    VAR MSRP MPG_City;
    CLASS Make Type Cylinders/ ORDER=UNFORMATTED MISSING;
    TABLE /* Row Dimension */
MSRP={STYLE={BACKGROUND=#C0C0C0}}*(
           N={LABEL="Number of Vehicles"
           STYLE={BACKGROUND=#C0C0C0}}*F=COMMA9.*{STYLE={FONT_WEIGHT=BOLD}}
           Mean*F=DOLLAR10.2
           Max*F=DOLLAR10.2)
MPG_City={STYLE={BACKGROUND=#C0C0C0}}*(  Min*F=3.
                                          Mean*F=4.1),
/* Column Dimension */
Cylinders={STYLE={BACKGROUND=#C0C0C0}}*
```

```
Make={STYLE={FONT_WEIGHT=BOLD BACKGROUND=#C0C0C0}
STYLE(CLASSLEV)={BACKGROUND=#E6E6E6}}
/*Table Options */
/ BOX={LABEL="Breakdown of MSRP and MPG (City) by Number of Cylinders and
Make" STYLE={FONT_FACE='Agency FB' BACKGROUND=#33CCCC JUST=CENTER
VJUST=MIDDLE}};
RUN;
```

CREATING A GROUPED TABLE OF SUMMARY STATISTICS WITH STACKED HEADERS

When using PROC TABULATE, it is important to first specify the variables of interest using CLASS (grouping variables), CLASSLEV (levels of grouping variables), or VAR (analysis variables) statements. To use unique options for a given variable, you can use multiples of the same type of statement. For example:

```
CLASS Make /      ORDER=UNFORMATTED MISSING;
CLASS Type /      ORDER=UNFORMATTED MISSING;
CLASS Cylinders / ORDER=UNFORMATTED MISSING;
```

But you can also specify them on the same line like so:

```
CLASS Make Type Cylinders/ ORDER=UNFORMATTED MISSING;
```

Once you have placed all the variables of interest on the CLASS, CLASSLEV, or VAR statement, it's time to tell SAS how to arrange them in the table. This is done using the TABLE statement. Like PROC REPORT's COL statement, the TABLE statement uses commas, spaces, and parentheses to define how the table should appear. Differently from PROC REPORT, PROC TABULATE uses an asterisk to link variables (ex. Stacked headers) and an equal sign is used to apply formatting headers. The organization of the TABLE statement is a little different than the COLUMN statement and it allows addition of formats, labels, and style elements.

There are only 2 commas that can be used in the TABLE statement: one to signify the end of the row dimension and one to specify the end of the column dimension. A page dimension is not required. Spaces are still interpreted as "next to", parentheses are still used to group variables, but an asterisk is interpreted as "to left of" in the row dimension ("below" in the column dimension), when read right to left. Asterisks can also be used to attach formatting and style elements to values of variables. The equals sign attaches formatting, style elements, and labels to variable headers. Table options are specified after a back slash on the TABLE statement.

The following statement (which defines the row dimension) can be interpreted by reading it backwards:

```
TABLE /* Row Dimension */
MSRP={STYLE={BACKGROUND=#C0C0C0}}*(
         N={LABEL="Number of Vehicles"
         STYLE={BACKGROUND=#C0C0C0}}*F=COMMA9.*{STYLE={FONT_WEIGHT=BOLD}}
         Mean*F=DOLLAR10.2
         Max*F=DOLLAR10.2)
MPG_City={STYLE={BACKGROUND=#C0C0C0}} * (Min*F=3.
                                         Mean*F=4.1),
```

Interpretation: Starting at the row dimension comma (highlighted above), the mean (formatted using numeric format 4.1) will come NEXT to the min (formatted using numeric format 3.). These will both come TO LEFT OF MPG_City (formatted with background color #C0C0C0). That grouping will come NEXT TO mean (formatted using currency format DOLLAR10.2) which will be NEXT TO Mean (also formatted with DOLLAR10.2) which will be NEXT TO N. N will have column label "Number of Vehicles" which is formatted with background color #C0C0C0. This is attached to the value formatting for N using an asterisk. Values of N will be formatted using numeric format comma9 and will appear in bold. The parentheses around variables N, Mean, and Max group these variables together. The asterisk attaches this group TO LEFT OF MSRP (formatted with background color #C0C0C0).

You may be wondering how variables n, min, mean, and max are used without being present in the input data set. These summary statistics are available for use with any data set variable in PROC TABULATE (and a subset within PROC REPORT). Summary variables are observation-based statistics (duplicate rows will create inflation in the data) and do not account for overlap inflation (ex. People with cats plus people with dogs will be inflated by people with both). As demonstrated in the cars example, multiple statistics for a given variable can be presented with very custom formatting in very few lines of code. Similar to PROC REPORT, the same variable can be used multiple times in the same table with the same or differing statistics and custom formatting. For a link to a list of available automatic variables see the product documentation for PROC TABULATE in Recommended Reading.

EXPORTING TO THE ODS EXCEL LOCATION

Once the coding for the creation of the tables is complete, a possible last step is to export into an external document. In the case of Excel-Style tables it seems fitting to cover the excel destination. By using the Excel output destination statements, the reports can be exported into a native Excel file without the use of Microsoft Add-in extensions.

The basic wrapper code is:

```
ODS EXCEL FILE="[Name of File with Full Path and Excel extension]";

[Add table coding here]

ODS EXCEL CLOSE;
```

This is added around the code that creates the Excel-style table and tells SAS to create a file with an Excel extension (ex. csv, xls, xlsx) compatible with Microsoft Office 2010 or newer in the specified directory. Then SAS populates it with the output of the code before the ODS EXCEL CLOSE statement. There is a long list of options that can be used in this statement to modify how the file is populated, but here are the go-to options you'll most likely want to use. For a full list of options see the product documentation for the Output Delivery System in the Recommended Reading.

ODS EXCEL OPTION[1]	**Go-to Value**	**Function**
SHEET_NAME='**[Name of File]**'	'[Custom Sheet Name]'	Sets the name of the Sheet. If multiple sheets are created and only one sheet name specified, this becomes a prefix (ex. SHEET_NAME='Sheet' will give "Sheet", "Sheet 2", etc.)
SHEET_INTERVAL= 'BYGROUP' | 'PAGE' | 'PROC' | 'NONE' | 'NOW' | **TABLE**'	'NONE' with individual use of 'NOW'	Tells SAS to place all output in a single sheet ('NONE') until a sheet break is specified ('NOW') or new value of 'SHEET_NAME'=

EMBEDDED_TITLES='**OFF**' | 'ON'
EMBEDDED_TITLES_ONCE='**OFF**' | 'ON'
	'ON'	Inserts the titles into the excel spreadsheet either once (_ONCE) or each time a new output is created.
		Otherwise titles are not shown in the file.

EMBEDDED_FOOTNOTES='**OFF**' | 'ON'
EMBEDDED_FOOTNOTES_ONCE='**OFF**' | 'ON'
	'ON'	Inserts the footnotes into the excel spreadsheet either once (_ONCE) or each time a new output is created.
		Otherwise footnotes are not shown in the file.

Table 1. ODS EXCEL "Go-To" Options

[1] Default value of this option is in bold.

These options are entered on the ODS EXCEL statement separated by spaces within the parentheses like so:

```
   ODS EXCEL options(EMBEDDED_TITLES=ON SHEET_NAME='SASHELP.CARS Example');
```

Options entered on the first ODS EXCEL statement will control the whole file if only one ODS EXCEL statement is used. However, additional ODS EXCEL statements can be used to change the options at different points in the document. For example, you might use the following code to add a second sheet named "SASHELP.CARS Example 2":

```
   ODS EXCEL options(SHEET_NAME='SASHELP.CARS Example 2');
```

This will only start a new page if there was output between the first ODS EXCEL statement and the current one. For example, PROC ODSTEXT should be used instead of title statements or ODS TEXT="" statements if a page with text only is desired. In subsequent ODS EXCEL statements any previous options will be overwritten; any options not changed or reset will stay in effect for the rest of the file.

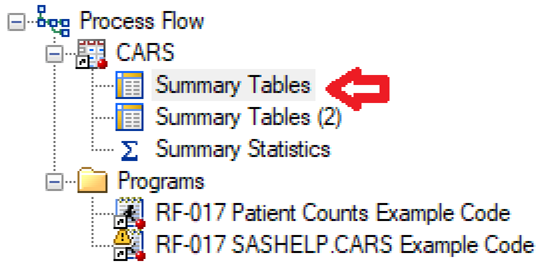
Here is code used to create an Excel file containing both example outputs:

SUMMARY TABLES BUILDER IN SAS ENTERPRISE GUIDE

CREATING THE SAME OUTPUT AS THE PROC TABULATE EXAMPLE USNG SAS ENTERPRISE GUIDE© 7.15 HF7

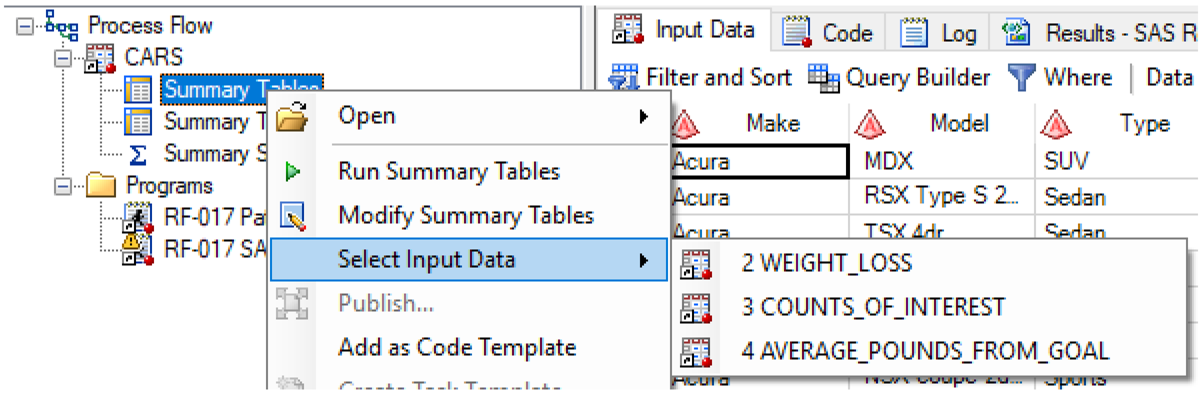
Yes, that's right, you can create the same highly customized table seen in EXAMPLE: Stacked Headers using SASHELP.cars using the point-and-click tasks built into SAS Enterprise Guide. Using the Summary Tables task requires zero code writing, uses drag-and-drop to build tables, utilizes nearly all functionality of PROC TABULATE, and provides real-time versions of the code it uses to create the end result. If navigating the table statement in PROC TABULATE is a pain point for you, use the Summary Table task to give you starter code that you can copy right into your program and edit from there. If you've never used PROC TABULATE, use the Summary Table task to get the desired product faster, provide code that you can use to understand how PROC TABULATE works, and watch how the code changes as you change the table.

Just like using PROC TABULATE in your code, you can run the Summary Table tasks multiple times. Once defined, the Summary Table task is added in the process flow. As more and more tables are created using the Summary Table task or other point-and-click tasks these will appear with automatically generated names using format: [Task Name] ({NULL, 2,3, etc.}) underneath the name of the table like this:



Display 1.Viewing Point-and-Click Tasks after Definition

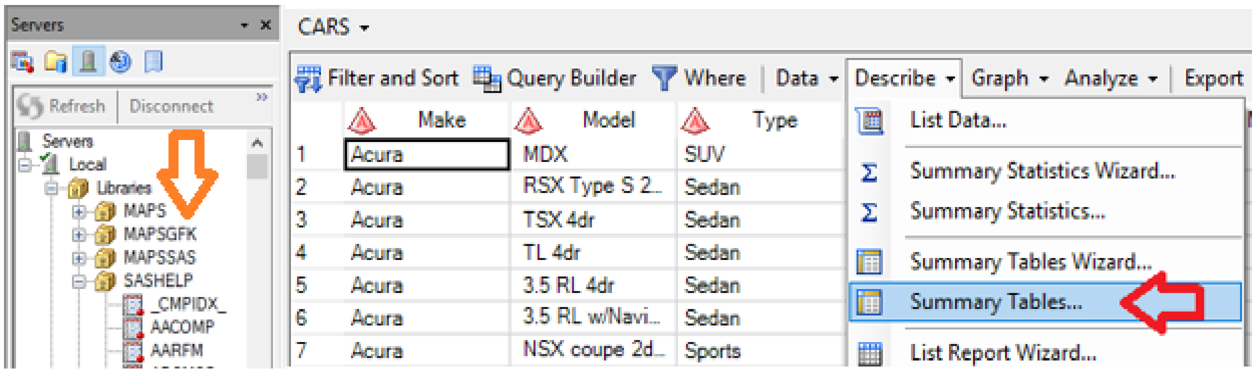
These names can be changed by right clicking on the name and selecting "Rename". This goes for all of the tasks. Right clicking will also give you quick access to running, modifying, copying, or changing the input data set (if the new one contains the same variables, not edits are needed).



Display 2. Options for Defined Tasks

The Summary Tables task was available in earlier versions of SAS Enterprise Guide too. The following step-by-step process is similar to the earlier process documented in *Summary Tables in SAS Enterprise Guide* (Slaughter & Delwiche, Summary Tables in SAS® Enterprise Guide®, 2005) and expanded on in *Summary Tables Task in SAS® Enterprise Guide ® PROC TABULATE Made Easy* (Slaughter & Delwiche, Summary Tables Task in SAS® Enterprise Guide® PROC TABULATE Made Easy, 2006).

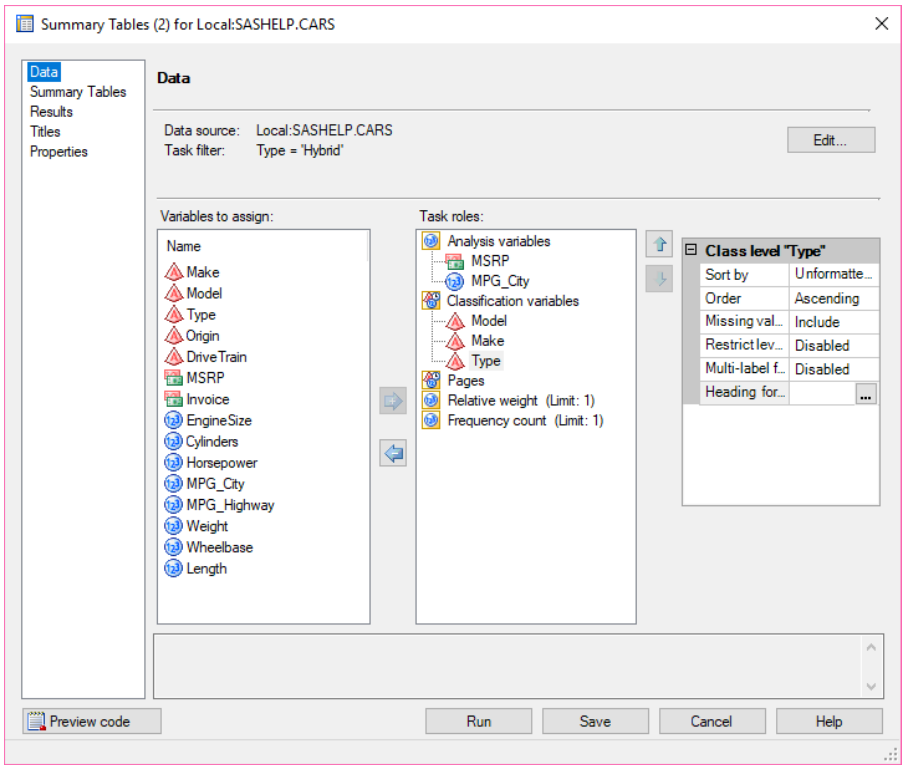
STEP-BY-STEP SCREENSHOTS



	Make	Model	Type
1	Acura	MDX	SUV
2	Acura	RSX Type S 2...	Sedan
3	Acura	TSX 4dr	Sedan
4	Acura	TL 4dr	Sedan
5	Acura	3.5 RL 4dr	Sedan
6	Acura	3.5 RL w/Navi...	Sedan
7	Acura	NSX coupe 2d...	Sports

Display 3. Servers Navigation Panel and Summary Tables Task in SAS Enterprise Guide

Start by navigating to the data you would like to create the table from by finding it in the Servers navigation panel (marked by orange arrow in Display 3) and double clicking on the data set name. Once the data set is open, navigate to the Describe menu and select "Summary Tables…" from the drop down (marked by red arrow in Display 3).


Summary Tables (2) for Local:SASHELP.CARS

Data

Data source: Local:SASHELP.CARS
Task filter: Type = 'Hybrid'

Variables to assign:

Name
Make
Model
Type
Origin
DriveTrain
MSRP
Invoice
EngineSize
Cylinders
Horsepower
MPG_City
MPG_Highway
Weight
Wheelbase
Length

Task roles:

Analysis variables
  MSRP
  MPG_City
Classification variables
  Model
  Make
  Type
Pages
Relative weight (Limit: 1)
Frequency count (Limit: 1)

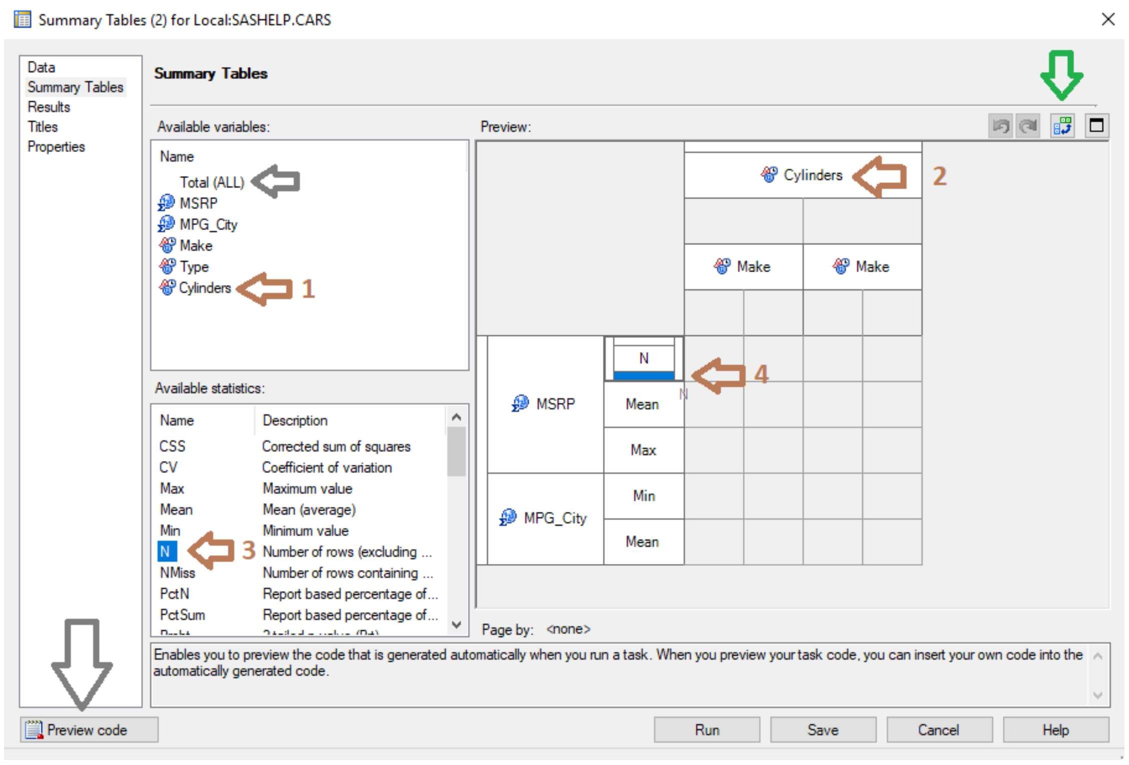
Class level "Type"

Sort by	Unformatte...
Order	Ascending
Missing val...	Include
Restrict lev...	Disabled
Multi-label f...	Disabled
Heading for...	

Display 4.Selecting the Variables for the Table

Next the variables being used in the table need to specified. Drag variables from the “Variables to assign” area to the “Task Roles” area. Numeric variables used in calculating statistics go under the analysis variables. Character variables are not allowed in this area. Grouping variables for the table go under classification variables role. Variables whose values define separate table go under pages role. Variables for which frequency counts are to be used in the table instead of observation-based counts (tells SAS that one observation actually represents “n_i” observations of the same value) go under the frequency count role. Similarly, variables that give the relative weight of an observation instead of the default equal weights (one observation should actually be given custom weight “w_i” instead of default weight) go under relative weight role.

Once you have placed a variable under one of the roles, a box with additional settings will appear to the right of the task roles. In Display 4 the additional options for Class levels of variable Type are shown. These options are either to help prepare the data for the table creation (sorting, filtering, and application of labels using a different variable) or tell SAS how to handle certain data values (missing values, restrict levels used to ones with a format). The task filter is covered later in the paper.



Name	Description
CSS	Corrected sum of squares
CV	Coefficient of variation
Max	Maximum value
Mean	Mean (average)
Min	Minimum value
N	Number of rows (excluding ...
NMiss	Number of rows containing ...
PctN	Report based percentage of...
PctSum	Report based percentage of...

Enables you to preview the code that is generated automatically when you run a task. When you preview your task code, you can insert your own code into the automatically generated code.

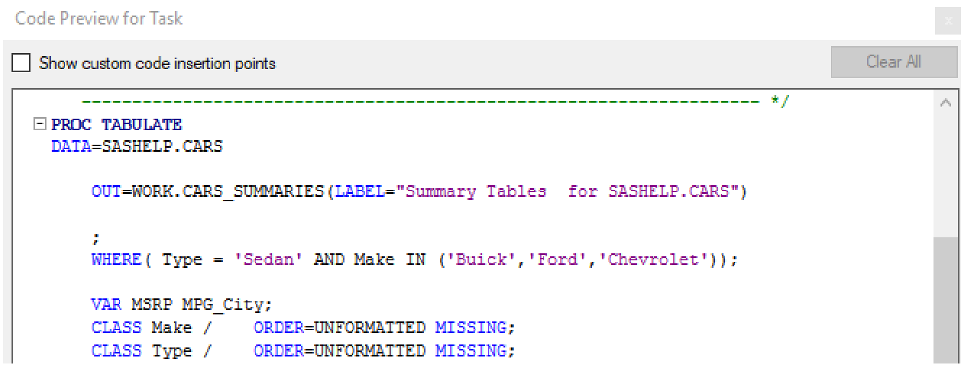
Display 5. Building Your Table in Summary Tables Builder

Next we will define the table of interest using the variables we’ve specified and available statistics for them. We will drag variables from the “Available variables” area to the “Preview” area. For example, variable Cylinders is moved from orange arrow 1 to orange arrow 2 in Display 5. You will notice 4 rectangles appear around the variable name when you try to drag another variable or a statistic over them (Arrow 4 in Display 5). This indicates the location of the new variable/statistic in relation to the selected variable. For example, to put a variable or statistic below a variable already in preview, left click and hold on the new variable, drag to hover over the variable already in preview without releasing, move your cursor to the bottom of the variable name until the rectangle on the bottom is highlighted, and then release the click.

The total variable is an automatic variable for getting values across all values in a row or within a column. When used in both column and row it creates a table total. Total is a observation based count and will calculate even if the value doesn’t make logical sense. For example, in a table with columns for average income and average distance from the city, a total column would put income values and distance values on the same number line and take the average; a value that has no logical meaning.

If you make a mistake or would like to tranpose your table you can use the buttons at the upper right of the preview panel (green arrow in Display 5) to Undo, Redo, Tranpose, or make the preview full screen. You can also drag variables off the preview screen back into the "Available variables" area to remove them or drag them around within the preview screen to move them.

At anytime during table creation you can push the "Preview Code" button in the lower left (gray arrow in Display 5) to view the code SAS will use to create the table you've defined as seen in Display 6.

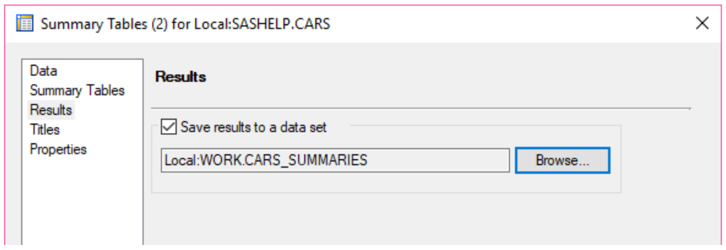


```
    ------------------------------------------------------------------ */
PROC TABULATE
DATA=SASHELP.CARS

    OUT=WORK.CARS_SUMMARIES(LABEL="Summary Tables  for SASHELP.CARS")

    ;
    WHERE( Type = 'Sedan' AND Make IN ('Buick','Ford','Chevrolet'));

    VAR MSRP MPG_City;
    CLASS Make /    ORDER=UNFORMATTED MISSING;
    CLASS Type /    ORDER=UNFORMATTED MISSING;
```

Display 6. Code Preview for Summary Table Builder

The code that SAS creates may look differently than you would've written it. Option values are often specified even when they are the default values. A class statement will exist for each class variable, even if all class variables share the same option values and formatting. Titles and footnotes are always specified and reset afterwards. Intermediate data sets may be created using PROC SQL to apply WHERE statements or order variables before creating the table. Newer users to SAS may enjoy this for learning what options exist and where to put them to change the desired attributes.

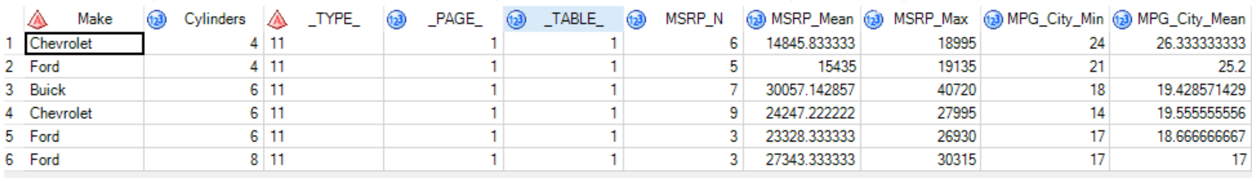
While the code is always available in preview, it is not able to be edited within the preview window. However, the code can be copied into any program and it will create the same table seen in your preview screen. Since the task doesn't cover all options available, it does allow you to enter "custom code" into the code it provides. A check box is provided in the top left of the code preview window for adding this code (see Display 6). More experienced users of SAS may enjoy using the task to create the bulk of the code and then editing it further either inside or outside of the task.



Save results to a data set [✓]

Local:WORK.CARS_SUMMARIES

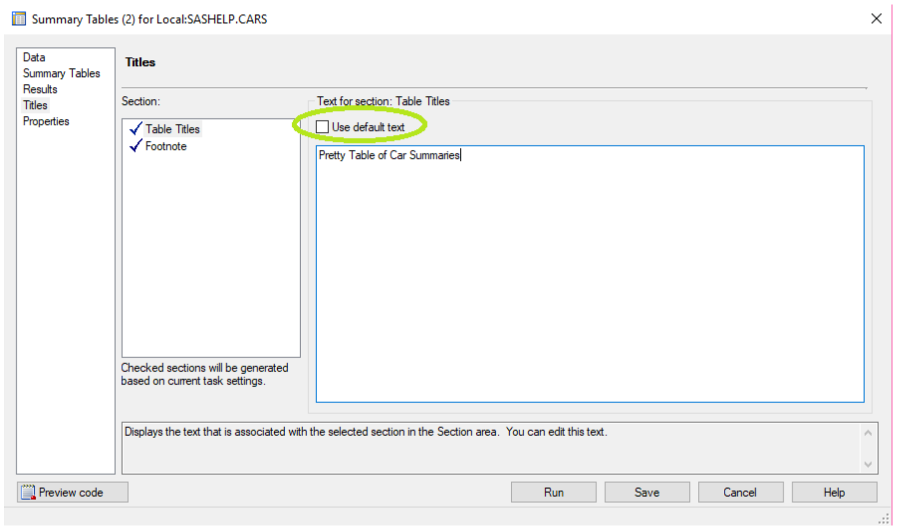
Display 7. Setting an Output Data set for Summary Values



	Make	Cylinders	_TYPE_	_PAGE_	_TABLE_	MSRP_N	MSRP_Mean	MSRP_Max	MPG_City_Min	MPG_City_Mean
1	Chevrolet	4	11	1	1	6	14845.833333	18995	24	26.333333333
2	Ford	4	11	1	1	5	15435	19135	21	25.2
3	Buick	6	11	1	1	7	30057.142857	40720	18	19.428571429
4	Chevrolet	6	11	1	1	9	24247.222222	27995	14	19.555555556
5	Ford	6	11	1	1	3	23328.333333	26930	17	18.666666667
6	Ford	8	11	1	1	3	27343.333333	30315	17	17

Figure 3. Preview of Output Data set for Summary Values

Like many other tasks, the Summary Tables Task allows you to export your results to a SAS data set. All values shown in the table will be output to the data set using the same format PROC TABULATE would use, seen in Figure 3. Preview of Output Data set for Summary Values



Summary Tables (2) for Local:SASHELP.CARS

Data
Summary Tables
Results
Titles
Properties

Titles

Section:

✓ Table Titles
✓ Footnote

Checked sections will be generated based on current task settings.

Text for section: Table Titles

☐ Use default text

Pretty Table of Car Summaries

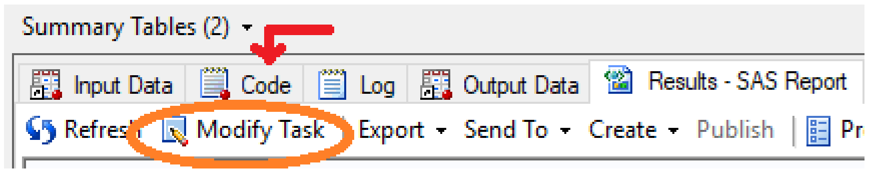
Displays the text that is associated with the selected section in the Section area. You can edit this text.

Preview code | Run | Save | Cancel | Help

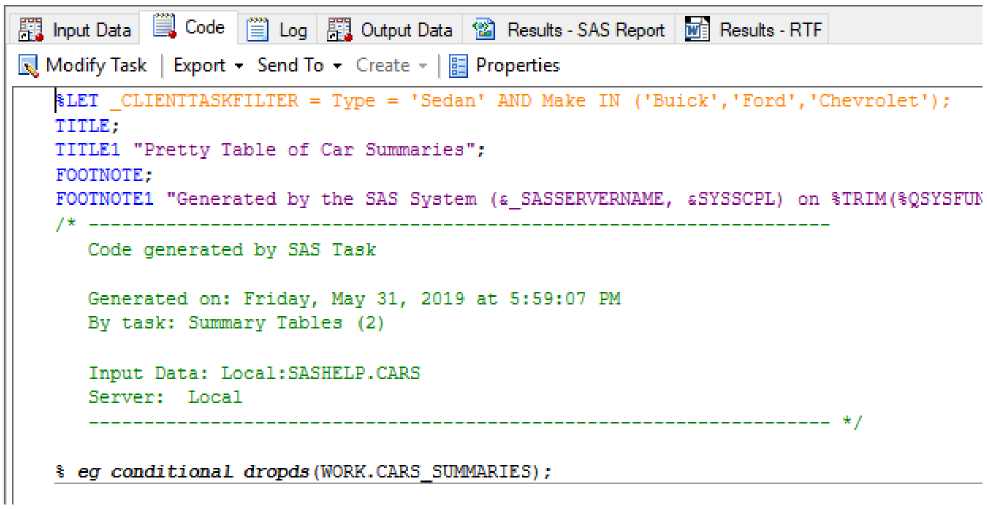
Display 8. End with Setting the Titles and Footnotes

The last step is to set the desired titles and footnotes. The default values given information about when the table was created in the footnote and “Summary Tables” as the title. To change these, unclick the default text box (Display 8. End with Setting the Titles and Footnotes.

Once you’ve completed this step you are ready to see your product by selecting the “Run” button. You can always go back into the Summary Table Tasks to make changes using the “Modify Task” button (Display 9. Modifying the Output. You can also view the final code (Display 10) used to create the output by navigating to the code tab (red arrow in Display 9). While this code cannot be edited directly, it can be copied into another program and run independently of the task. There may be macros used in the code it generates; these can also be run outside of the task but are not required.



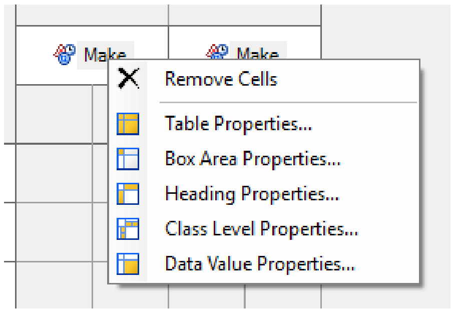
Display 9. Modifying the Output



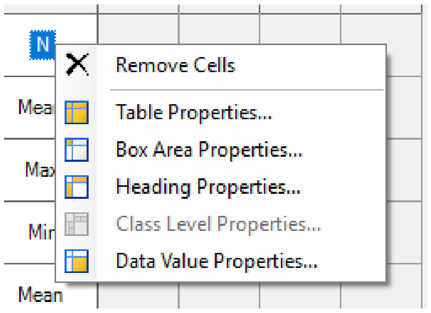
```
%LET _CLIENTTASKFILTER = Type = 'Sedan' AND Make IN ('Buick','Ford','Chevrolet');
TITLE;
TITLE1 "Pretty Table of Car Summaries";
FOOTNOTE;
FOOTNOTE1 "Generated by the SAS System (&_SASSERVERNAME, &SYSSCPL) on %TRIM(%QSYSFUN
/* -------------------------------------------------------------------
   Code generated by SAS Task

   Generated on: Friday, May 31, 2019 at 5:59:07 PM
   By task: Summary Tables (2)

   Input Data: Local:SASHELP.CARS
   Server:  Local
   ------------------------------------------------------------------- */

%_eg_conditional_dropds(WORK.CARS_SUMMARIES);
```

Display 10. Viewing the Final Code

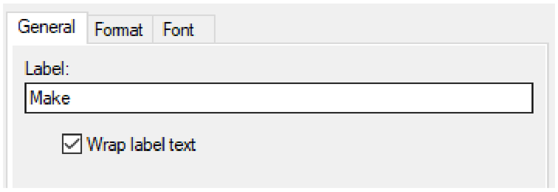
FORMATTING VALUES IN THE SUMMARY TABLES BUILDER



Display 11. Categorical Variable Format Options



Display 12. Numeric Variable Format Options



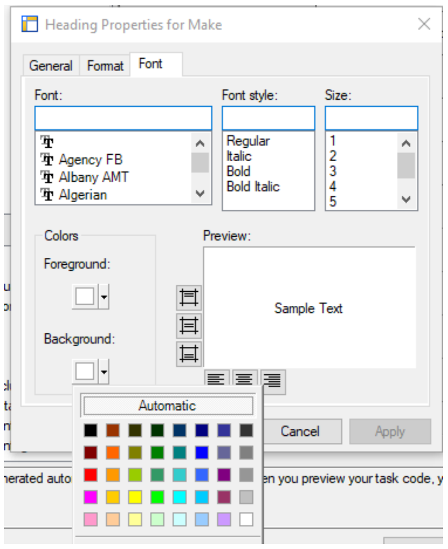
Label:
Make

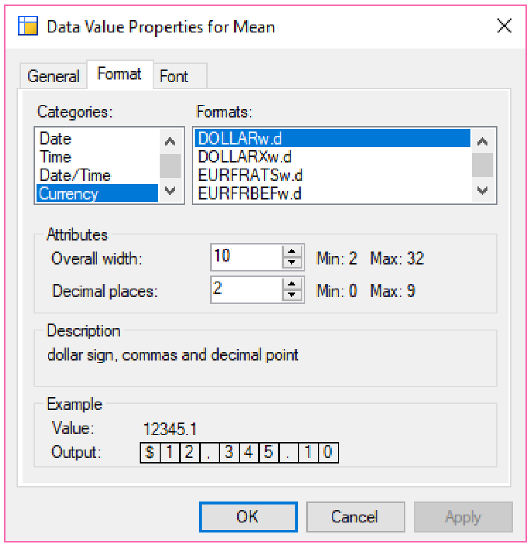
☑ Wrap label text

Display 13. Changing the Labels

One advantage of the Summary Tables task is its ability to provide users with an easy way to change style attributes, headings, formats, and class level properties within the table (Display 11Display 12).

Remembering where style definitions need to appear to change the desired attribute can be one of the more difficult parts of using PROC TABULATE. The ability to edit style elements in a point and click atmosphere can be very helpful both in supplementing memory of all the style elements and their placements. Even experienced coders will appreciate the help with color codes given by selecting a color from a drop down (Display 14). The preview of the value given a chosen format (Display 15) can also be very helpful when determining the appropriate formats to use and which ones are available.




Data Value Properties for Mean

General | Format | Font

Categories: Date, Time, Date/Time, Currency (selected)

Formats: DOLLARw.d (selected), DOLLARXw.d, EURFRATSw.d, EURFRBEFw.d

Attributes
Overall width: 10 Min: 2 Max: 32
Decimal places: 2 Min: 0 Max: 9

Description
dollar sign, commas and decimal point

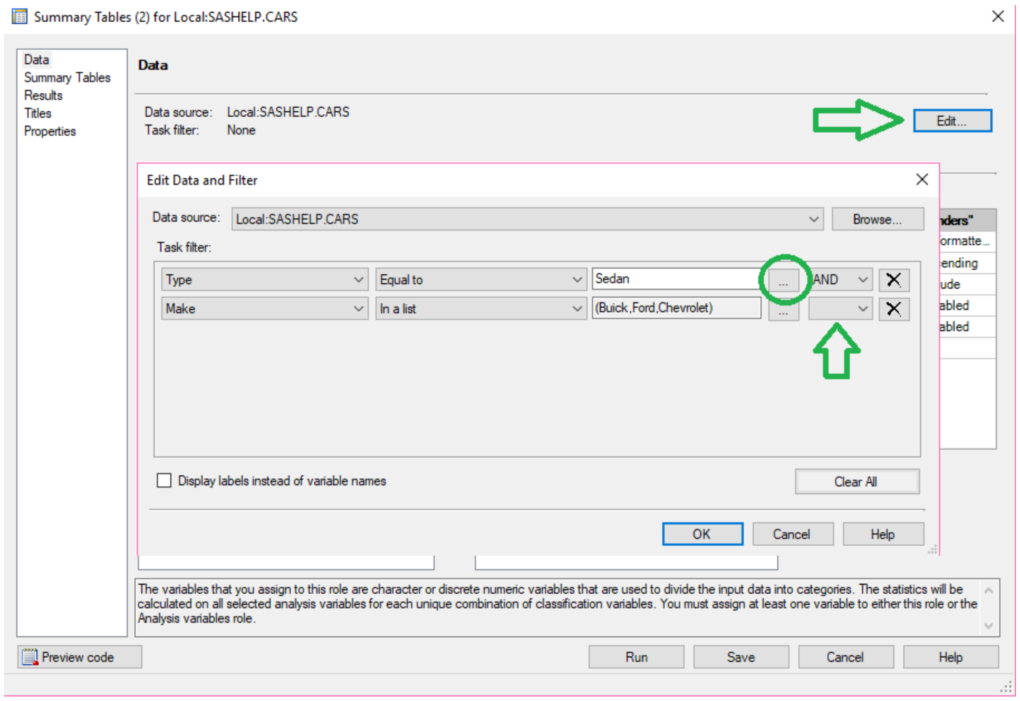
Example
Value: 12345.1
Output: $12,345.10

Display 14. Changing Font Characteristics

Display 15. Changing Formats Used

FILTERING DATA IN THE SUMMARY TABLES BUILDER

Just as WHERE statements can be used to limit the data being considered by PROC TABULATE or PROC REPORT, the task filter can be used to limit the data being considered by the Summary Tables Task. Similar to other portions of the task windows, this has been converted to a series of drop downs and free text fields and does not require writing any code. Navigate to the task filter by using the “Edit” button (Right green arrow in Display 16). Select variables from the data set and operators from the drop downs and enter values in the text box or select the ellipsis button (green circle in Display 16) to select values from a drop down. Connect multiple filtering statements with an AND or OR from the last dropdown (up green arrow in Display 16); this will automatically add a blank filter statement below.


Summary Tables (2) for Local:SASHELP.CARS

Data

Data source: Local:SASHELP.CARS
Task filter: None

Edit Data and Filter

Data source: Local:SASHELP.CARS

Task filter:

Type	Equal to	Sedan	AND
Make	In a list	(Buick,Ford,Chevrolet)	

Display labels instead of variable names

The variables that you assign to this role are character or discrete numeric variables that are used to divide the input data into categories. The statistics will be calculated on all selected analysis variables for each unique combination of classification variables. You must assign at least one variable to either this role or the Analysis variables role.

Display 16. Filtering Data in Summary Tables

CONCLUSION

SAS is an extensive programming language used by a diverse set of programmers to meet a variety of needs. With so many differing needs and users, presenting a variety of techniques to get similar outcomes is useful to those at all experience levels. The use of SAS Enterprise Guide Tasks can help those still learning to produce the same products as more experienced SAS programmers and at the same time help experienced programmers by providing code that can be built upon. Whether you are new to SAS, new to Enterprise Guide, or experienced in both, you can now create and export Pretty Excel-Style tables to the ODS Excel destination.

REFERENCES

Slaughter, S. J., & Delwiche, L. D. (2005). Summary Tables in SAS® Enterprise Guide®. *Proceedings of SAS Users Group International 30.* Philadelphia, PA: SAS Institute Inc. Retrieved from https://support.sas.com/resources/papers/proceedings/proceedings/sugi30/179-30.pdf

Slaughter, S. J., & Delwiche, L. D. (2006). Summary Tables Task in SAS® Enterprise Guide® PROC TABULATE Made Easy. *Proceedings of SAS Users Group International 31.* San Francisco, CA. Retrieved from https://support.sas.com/resources/papers/proceedings/proceedings/sugi31/113-31.pdf

Zender, C. L. (2014). Sailing Over the ACROSS Hurdle in PROC REPORT. *Proceedings of Midwest SAS Users Group 2014 Conference.* Cary, NC: SAS Institute. Retrieved from http://support.sas.com/resources/papers/proceedings14/SAS388-2014.pdf

RECOMMENDED READING

- *SAS® 9.4 Output Delivery System: Users Guide, Fifth Edition*
  - *ODS EXCEL Suboptions*
- *Base SAS® Procedures Guide, Seventh Edition*
  - *Statistics That Are Available in PROC TABULATE*
  - *Statistics That Are Available in PROC REPORT*

CONTACT INFORMATION

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

Brooke Ellen Delgoffe
Office of Research Computing and Analytics, Marshfield Clinic Research Institute
brooke_delgoffe@hotmail.com
https://www.linkedin.com/in/brookedelgoffe