PharmaSUG 2020 - Paper AP-22

It's All about the Base—Procedures, Part 2

Jane Eslinger, SAS Institute Inc.

ABSTRACT

"It's All about the Base—Procedures" (a PharmaSUG 2019 paper) explored the strengths and challenges of commonly used Base SAS® procedures. It also compared each procedure to others that could accomplish similar tasks. This paper takes the comparison further, focusing on the FREQ, MEANS, TABULATE, REPORT, PRINT, and SQL procedures.

As a programmer, whether novice or advanced, it is helpful to know when to choose which procedure. The first paper provided best-use cases, and this paper takes it a step further in its discussion of when to choose one procedure over another. It also provides example code to demonstrate how to get the most out of the procedure that you choose.

INTRODUCTION

The SAS Global Forum 2019 paper, "It's All about the Base—Procedures," discussed fifteen procedures. It provided a description of each one, along with identifying the best uses cases and challenges. The paper also provided a brief comparison of two procedures that have similar functionality.

This paper provides a more in-depth comparison of procedures. The examples demonstrate how to produce the same output for shared functionality. The goal of this paper is to empower you. By understanding the best use cases, the challenges, and the multiple methods for creating the data that you need, you are empowered. You will be able to make choices about the procedure that works best and most efficiently for your program.

Not every use case and comparison are covered in this paper. However, the examples cover a wide range of common, everyday programming tasks using the FREQ, MEANS, TABULATE, REPORT, PRINT, and SQL procedures.

FREQ PROCEDURE

One job of PROC FREQ is to provide counts and percentages. Most programmers find it easy to use because the syntax for the TABLES statement is straightforward. The variables for which counts are needed are listed in the TABLES statement. An asterisk is used to separate variables when counts are needed for combinations.

Example 1 demonstrates how to generate a data set with the counts of the combination of ORIGIN, MAKE, and TYPE.

```
proc freq data=sashelp.cars;
   tables origin*make*type / out=example1 norow nocol;
run;
```

In the Example1 data set, the percent value represents the ratio of the count for a specific combination against all observations in SASHELP.CARS. In some circumstances they might be useful. However, if other percentages are needed, the variables might need to be arranged differently. For example, if you want a subtotal percent where a variable's percentages need to sum to 100, that variable must be moved to a BY statement. This requires the data to be sorted prior to the execution of the PROC FREQ step.

Example 2 demonstrates moving ORIGIN to the BY statement so that the percentages within each value of ORIGIN sum to 100.

```
proc sort data=sashelp.cars out=cars;
   by origin;
run;
```

```
proc freq data=cars;
    by origin;
    tables make*type / out=example2 norow nocol;
run;
```

FREQ is very good at providing counts and percentages with minimal syntax and a small learning curve. Using the BY statements provides the flexibility for the level at which the percentages are calculated. However, PROC FREQ is limited. The next two sections compare PROC FREQ to PROC SUMMARY and PROC TABULATE, respectively.

PROC SUMMARY

PROC SUMMARY is a useful alternative to PROC FREQ for generating counts at multiple levels. A CLASS statement specifies the variables whose values define the subgroup combinations. Within a PROC SUMMARY step, if multiple variables are in the CLASS statement, the default results will include the overall count for each individual variable along with counts for the combination of the variables. If three or more variables are listed, the results include the combinations for pairs as well as threes, and so on. The TYPES statement gives even more control, allowing you to specify the exact combination of class variables that you need.

The same counts can be obtained through PROC FREQ but would require multiple steps —either multiple TABLES statement and merging or multiple PROC FREQ steps.

Examples 3 and 4 generate an output data set with counts for the following:

- overall TYPE
- overall MAKE
- the combination of TYPE and MAKE
- overall ORIGIN
- the combination of TYPE and ORIGIN
- the combination of MAKE and ORIGIN
- the combination of ORIGIN, TYPE, and MAKE

Example 3 demonstrates the required PROC FREQ and DATA step code. Not all the combination requests are included in one TABLES statement because the OUT= data set would contain counts for only the last request. The DATA step is used to create one data set that contains the output from each individual TABLES statement.

```
proc freq data=sashelp.cars noprint;
    tables origin /out=f1;
    tables origin*type /out=f2;
    tables origin*make/out=f3;
    tables type /out=f4;
    tables type*make /out=f5;
    tables make /out=f6;
    tables origin*type*make /out=f7;
run;

data example3;
    set f1 f2 f3 f4 f5 f6 f7;
run;
```

PROC SUMMARY, in Example 4, will produce the same counts in one step and with less code.

```
proc summary data=sashelp.cars;
    class origin make type;
    output out=example3;
run;
```

Example 5 uses the TYPES statement to produce counts for overall ORIGIN, the combination of ORIGIN and TYPE, and the combination of ORIGIN and MAKE.

```
proc summary data=sashelp.cars;
    class origin make type;
    types origin origin*(make type);
    output out=example5;
run;
```

In this circumstance, PROC SUMMARY has the advantage over PROC MEANS. When the VAR statement is omitted, PROC SUMMARY produces a simple count of observations, whereas PROC MEANS tries to analyze all the numeric variables that are not listed in the other statements.

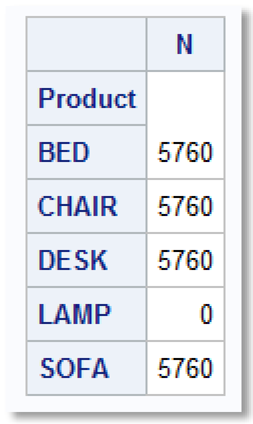
Remember that neither PROC SUMMARY nor PROC MEANS will calculate percentages. If percentages are required, they will have to be calculated with a DATA step after the procedure.

PROC TABULATE

One drawback of using PROC FREQ is that it will not insert categories that are not present in the data with a count of zero, even when you apply a format that contains the category that does not exist in the data. PROC TABULATE has an advantage over PROC FREQ. The PRELOADFMT option in PROC TABULATE will insert values that are present in the format but are not present in the data set.

Example 6 creates a format and uses PRELOADFMT to include a count for lamp in the output, a value not present in the data set. The results are in Output 1. If an output data set is generated with the OUT= option, it will also contain a record for lamp.

```
proc format;
    value $prod 'BED'='BED' 'CHAIR'='CHAIR'
          'DESK'='DESK' 'SOFA'='SOFA' 'LAMP'='LAMP';
run;

options missing='0';
proc tabulate data=sashelp.prdsal2;
    format product $prod.;
    class product / preloadfmt;
    table product,n / printmiss;
run;
```



	N
Product	
BED	5760
CHAIR	5760
DESK	5760
LAMP	0
SOFA	5760

Output 1. Category with N=0 from PROC TABULATE

PROC FREQ provides overall percentages for an entire data set or within a BY group, as demonstrated in Example 2 at the beginning of this section. However, using a BY variable means that the printed output is spread across multiple tables. PROC TABULATE will provide the same values but keep them condensed in one table. The key to PROC TABULATE is specifying the proper denominator. In Example 7, the individual percentages represent the proportion of each TYPE within MAKE for each ORIGIN. The denominator is specified within the angle (< >) brackets.

```
proc tabulate data=sashelp.cars;
   class origin make type;
   tables origin*(make all), (type all)*(n pctn<make*type make*all
        all*type all>);
run;
```

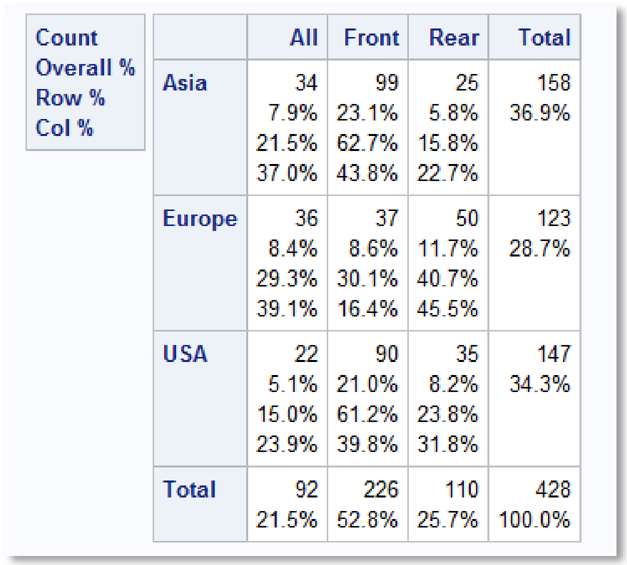
In the References section of this paper, you will find a link to a paper that gives much more detail about calculating percentages with PROC TABULATE and with further detail about specifying the proper denominator.

Finally, changing the format for the count and percent values from PROC FREQ can be challenging. For multi-way tables, the FORMAT= option in the TABLES statement changes the format for counts. It does not apply to percentages. Formatting for percentages must be changed in the table template. Modifying the format for either counts or percentages of one-way tables must be changed in a table template.

Example 8 contains the PROC TEMPLATE code that is required to assign a picture format to the various percentages generated by PROC FREQ. The results are in Output 2.

```
ods path(prepend) work.templat(update);

/* Create a picture format that contains percent sign */
proc format;
   picture pctfmt (round) other='009.9%';
run;

/* Apply format to the CrossTabFreqs table */
proc template;
define crosstabs Base.Freq.CrossTabFreqs;
   cellvalue frequency percent rowpercent colpercent;

   define frequency;
     format=8.;
     header='Count';
   end;
```

```
      define percent;
         format=pctfmt.;
         header='Overall %';
      end;

      define rowpercent;
         format=pctfmt.;
         header='Row %';
      end;

      define colpercent;
         format=pctfmt.;
         header='Col %';
      end;
   end;
   run;

   proc freq data=sashelp.cars;
      tables origin*drivetrain;
   run;

   /*Delete temporary table template*/
   proc template;
      delete base.freq.crosstabfreqs;
   run;
```



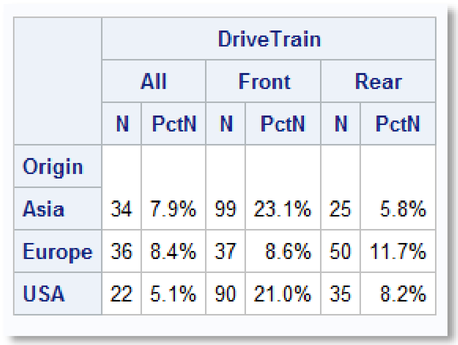
Count
Overall %
Row %
Col %

	All	Front	Rear	Total
Asia	34 7.9% 21.5% 37.0%	99 23.1% 62.7% 43.8%	25 5.8% 15.8% 22.7%	158 36.9%
Europe	36 8.4% 29.3% 39.1%	37 8.6% 30.1% 16.4%	50 11.7% 40.7% 45.5%	123 28.7%
USA	22 5.1% 15.0% 23.9%	90 21.0% 61.2% 39.8%	35 8.2% 23.8% 31.8%	147 34.3%
Total	92 21.5%	226 52.8%	110 25.7%	428 100.0%

Output 2. Formatted Percentages with PROC FREQ

The same format can be applied directly to the PCTN statistic in the TABLE statement of PROC TABULATE, without any extra procedure steps. Example 9 demonstrates the syntax and the results that are shown in Output 3.

```
   proc tabulate data=sashelp.cars;
      class origin drivetrain;
      table origin, drivetrain*(n pctn*f=pctfmt.);
   run;
```



	DriveTrain					
	All		Front		Rear	
	N	PctN	N	PctN	N	PctN
Origin						
Asia	34	7.9%	99	23.1%	25	5.8%
Europe	36	8.4%	37	8.6%	50	11.7%
USA	22	5.1%	90	21.0%	35	8.2%

Output 3. Formatted Percentages with PROC TABULATE

Although the previous examples focused on formatting the percentages, the same techniques can be used for counts.

MEANS PROCEDURE

PROC MEANS generates some of the same statistics as UNVARIATE, TABULATE, REPORT, and SQL. PROC MEANS is chosen over these procedures for a couple of reasons. The first reason is that it does not require much code to get the numbers that you need. Secondly, there is not a learning curve to PROC MEANS like there is with TABULATE, REPORT, and SQL. Also, PROC MEANS output is more condensed than output from PROC UNIVARIATE.

This easy-to-use procedure is often used to double-check sums and means from other sources, and it is used as a sanity check for numbers calculated within a DATA step. Example 10 has code for the common task of generating categorical counts and merging them into detail data to perform a calculation.

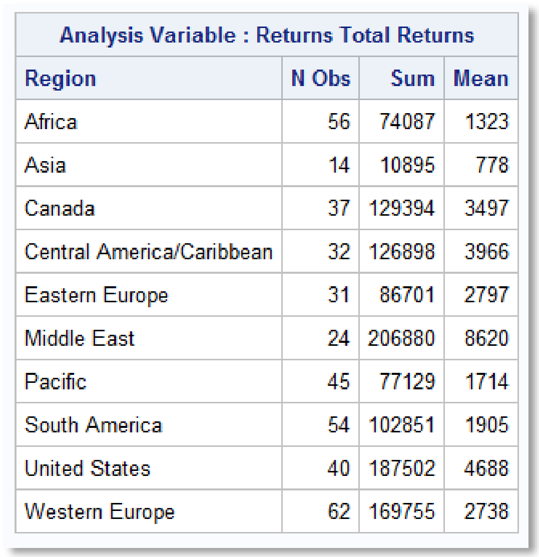
```
proc means data=sashelp.shoes noprint nway;
    class region subsidiary;
    var sales;
    output out=sums(drop=_type_ _freq_) sum=subsid_sales;
run;

data shoes;
    merge sashelp.shoes sums;
    by region subsidiary;
    subsid_perc = round(sales/subsid_sales*100,.01);
run;
```

PROC TABULATE

As mentioned above, PROC TABULATE is one of the procedures that calculates the same statistics as PROC MEANS. Programmers will often choose to use PROC MEANS when they need to feed the numbers into another DATA step or procedure. They will use PROC TABULATE when generating visible output that they want to share. A main driver for using TABULATE instead of MEANS for this task is because TABULATE gives more control over the formatting of the numbers.

In the default PROC MEANS printed output, the only option for formatting statistics is the MAXDEC= option, which controls the number of decimal places. Example 11 shows MAXDEC=, with the result in Output 4.

```
proc means data=sashelp.shoes sum mean maxdec=0;
    class region;
    var returns;
run;
```

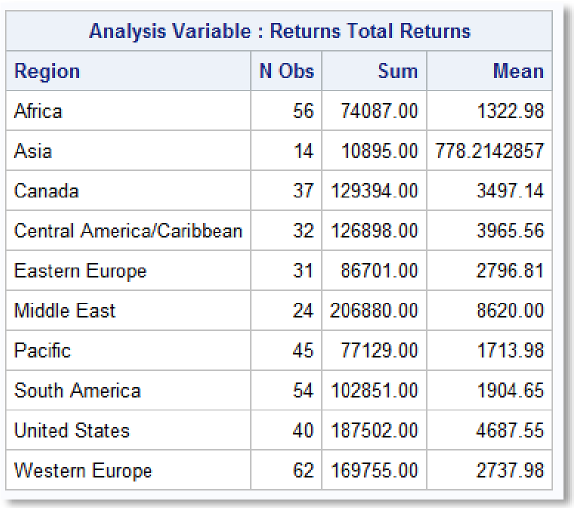


Analysis Variable : Returns Total Returns			
Region	N Obs	Sum	Mean
Africa	56	74087	1323
Asia	14	10895	778
Canada	37	129394	3497
Central America/Caribbean	32	126898	3966
Eastern Europe	31	86701	2797
Middle East	24	206880	8620
Pacific	45	77129	1714
South America	54	102851	1905
United States	40	187502	4688
Western Europe	62	169755	2738

Output 4. MAXDEC= in PROC MEANS

In the printed output, a FORMAT statement, included in Example 12, is ignored. There are no dollar signs in Output 5.

```
proc means data=sashelp.shoes sum mean;
    format returns dollar10.2;
    class region;
    var returns;
run;
```

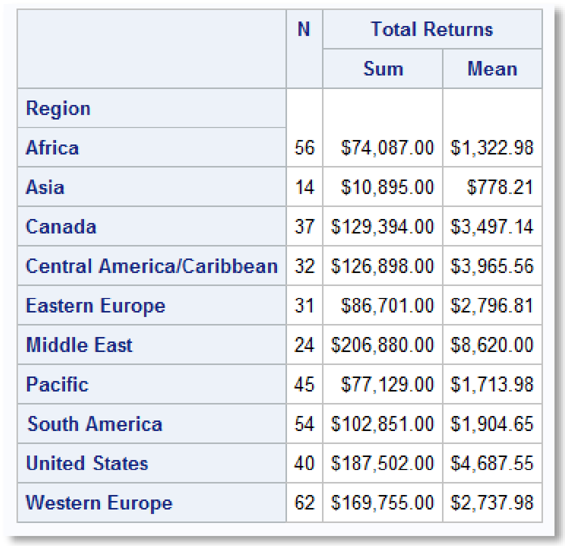


Analysis Variable : Returns Total Returns			
Region	N Obs	Sum	Mean
Africa	56	74087.00	1322.98
Asia	14	10895.00	778.2142857
Canada	37	129394.00	3497.14
Central America/Caribbean	32	126898.00	3965.56
Eastern Europe	31	86701.00	2796.81
Middle East	24	206880.00	8620.00
Pacific	45	77129.00	1713.98
South America	54	102851.00	1904.65
United States	40	187502.00	4687.55
Western Europe	62	169755.00	2737.98

Output 5. Statistics Are Not Formatted

To see formatted statistics, you need to create a data set and print it with the desired formats, or you must use another procedure like PROC TABULATE. The PROC TABULATE code in Example 13 creates the same table except that its statistics are formatted, as shown in Output 6.

```
proc tabulate data=sashelp.shoes;
    class region;
    var returns;
    table region, n returns*(sum mean)*f=dollar12.2;
run;
```



	N	Total Returns	
		Sum	Mean
Region			
Africa	56	$74,087.00	$1,322.98
Asia	14	$10,895.00	$778.21
Canada	37	$129,394.00	$3,497.14
Central America/Caribbean	32	$126,898.00	$3,965.56
Eastern Europe	31	$86,701.00	$2,796.81
Middle East	24	$206,880.00	$8,620.00
Pacific	45	$77,129.00	$1,713.98
South America	54	$102,851.00	$1,904.65
United States	40	$187,502.00	$4,687.55
Western Europe	62	$169,755.00	$2,737.98

Output 6. Formatted Statistics with PROC TABULATE

TABULATE PROCEDURE

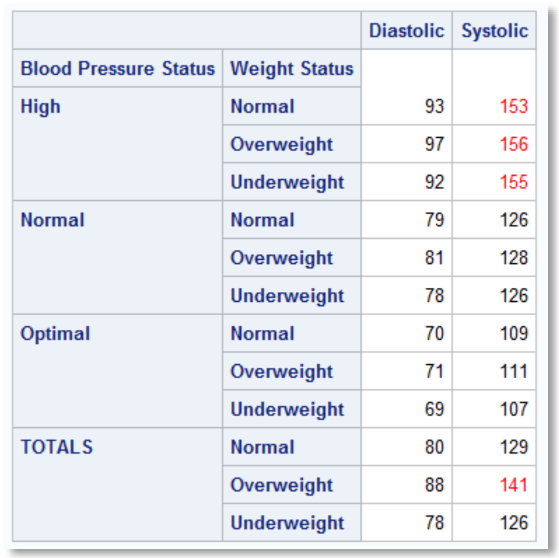
PROC TABULATE is considered a reporting procedure and it is often used to generate final tables that contain various numbers like counts, means, and percentages. The TABLE statement allows you to provide column labels, formats, and style enhancements.

Example 14 demonstrates multiple benefits of using PROC TABULATE:

1. calculating statistics for the combination of class variables
2. calculating statistics for one class variable that is independent of the other
3. adding trafficlighting
4. overriding the original format of the analysis variable.

```
proc format;
   value sysp 131-high = 'red';
run;

proc tabulate data=sashelp.heart;
   class bp_status weight_status;
   var diastolic systolic;
   table bp_status*weight_status❶ all='TOTALS'*weight_status=''❷,
         (diastolic systolic*[s={foreground=sysp.}]❸)*mean*f=8.❹;
   keylabel mean=' ';
run;
```

The results are shown in Output 7.



		Diastolic	Systolic
Blood Pressure Status	Weight Status		
High	Normal	93	153
	Overweight	97	156
	Underweight	92	155
Normal	Normal	79	126
	Overweight	81	128
	Underweight	78	126
Optimal	Normal	70	109
	Overweight	71	111
	Underweight	69	107
TOTALS	Normal	80	129
	Overweight	88	141
	Underweight	78	126

Output 7. PROC TABULATE Results

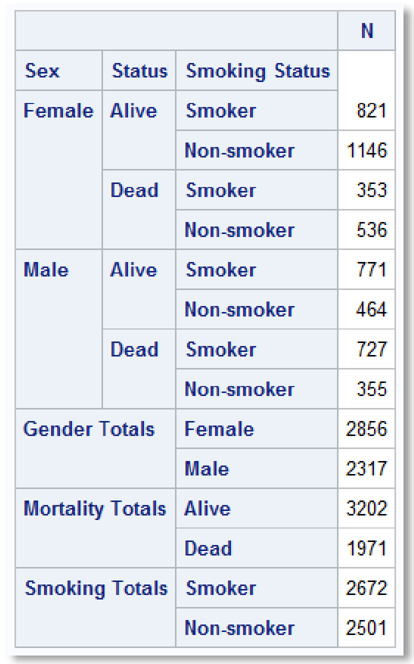
Refer to earlier sections in this paper for comparisons of PROC TABULATE to both PROC MEANS and PROC FREQ.

PROC REPORT

PROC TABULATE is also often compared to PROC REPORT. Both are reporting procedures, both calculate statistics, and both allow style overrides. This section will discuss when PROC TABULATE is better to use than PROC REPORT. The PROC REPORT section, later in the paper, will discuss when REPORT is more appropriate.

PROC TABULATE is easier to use than PROC REPORT in two distinct use cases. The first is the one mentioned in Example 14. PROC TABULATE can easily generate the combinations of class variables and individual class variables. Example 15 has three variables in the CLASS statement. The TABLE statement requests the combination of the three variables and each variable by itself. Results are in Output 8.

```
proc format;
   value $smk
     'Non-smoker'='Non-smoker'
     other='Smoker';

proc tabulate data=sashelp.heart;
   format smoking_status $smk.;
   class sex status smoking_status;
   table sex*status*smoking_status all='Gender Totals'*sex=''
     all='Mortality Totals'*status=''
     all='Smoking Totals'*smoking_status='', n;
run;
```



			N
Sex	Status	Smoking Status	
Female	Alive	Smoker	821
		Non-smoker	1146
	Dead	Smoker	353
		Non-smoker	536
Male	Alive	Smoker	771
		Non-smoker	464
	Dead	Smoker	727
		Non-smoker	355
Gender Totals		Female	2856
		Male	2317
Mortality Totals		Alive	3202
		Dead	1971
Smoking Totals		Smoker	2672
		Non-smoker	2501

Output 8. PROC TABULATE Grouping

PROC REPORT can generate those numbers, but it is not an easy task to get the individual group totals. It requires dummy grouping variables for each value of each class variable that needs to be shown independently. It also requires a temporary variable for each analysis variable for each value of each class variable. The code becomes long, cumbersome, and possibly confusing. The full code is included in the Appendix. Even if you want to use PROC REPORT for the final report, using PROC TABULATE to generate the numbers first might be preferable.

The second case where PROC TABULATE is easier than PROC REPORT is calculating percentages. Out-of-the box, PROC REPORT generates PCTN and PCTSUM only at the report level. Additional code is required to calculate percentages within groups.

Example 16 contains the PROC TABULATE code to calculate the PCTN and the cholesterol PCTSUM for status within each gender.

```
proc tabulate data=sashelp.heart;
    class sex status;
    var cholesterol;
    tables sex*status, pctn<status> cholesterol*pctsum<status>;
run;
```

Example 17 shows the PROC REPORT code that is required to generate the same percentages. It requires five times the number of statements, with multiple compute blocks and temporary variables.

```
proc report data=sashelp.heart;
    column sex status n cholesterol;
    define sex / group;
    define status / group;
    define n / format=8.2;
    define cholesterol / format=8.2;

    compute before sex;
       total_n = n;
       total_chol = cholesterol.sum;
    endcomp;
```

```
     compute n;
       if _break_='' then n = n / total_n * 100;
     endcomp;
     compute cholesterol;
       if _break_='' then
       cholesterol.sum = cholesterol.sum / total_chol * 100;
     endcomp;
  run;
```

REPORT PROCEDURE

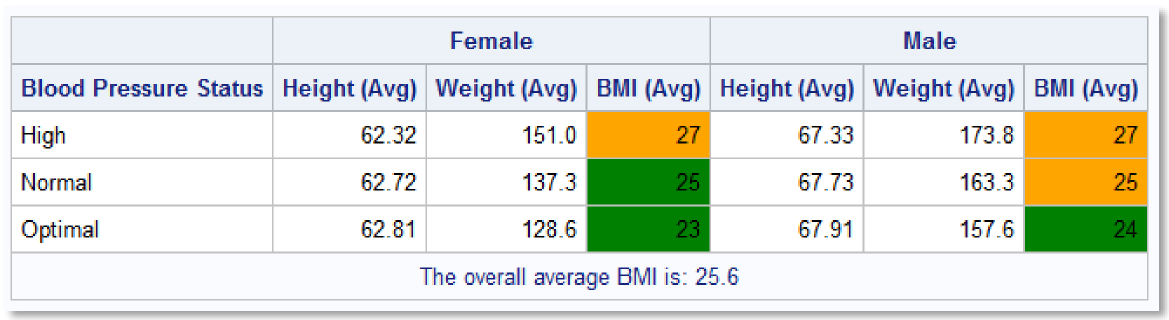
PROC REPORT is one of the most versatile Base SAS procedures. Its role is to create final, production quality reports. The types of reports that you can generate span the spectrum, depending on the data and the industry.

Example 18 demonstrates the primary strengths of PROC REPORT:

- transposing
- creating new columns
- inserting text
- incorporating style changes

The final table is in Output 9.

```
  proc format;
     value bmi
        low-18.4='yellow'
        18.5-24.9='green'
        25-29.9='orange'
        30-high='red';
  run;

  proc report data=sashelp.heart;
     column bp_status sex,(height weight bmi);
     define bp_status / group;
     define sex / across ' ';
     define height / mean 'Height (Avg)' f=8.2;
     define weight / mean 'Weight (Avg)' f=8.1;
     define bmi / computed 'BMI (Avg)' f=8. style(column)=[background=bmi.];

     compute bmi;
        _c4_ = 703 * (_c3_ / (_c2_*_c2_));
        _c7_ = 703 * (_c6_ / (_c5_*_c5_));
     endcomp;
     compute after;
        abmi = sum(_c4_,_c7_)/2;
        line "The overall average BMI is: " abmi 8.1;
     endcomp;
  run;
```



	Female			Male		
Blood Pressure Status	Height (Avg)	Weight (Avg)	BMI (Avg)	Height (Avg)	Weight (Avg)	BMI (Avg)
High	62.32	151.0	27	67.33	173.8	27
Normal	62.72	137.3	25	67.73	163.3	25
Optimal	62.81	128.6	23	67.91	157.6	24
The overall average BMI is: 25.6						

Output 9. PROC REPORT Results

PROC PRINT

PROC REPORT and PROC PRINT are naturally compared to each other. The main and best use case for both procedures is to generate a final report, although both procedures can also be used to preview the data.

Example 19 demonstrates using PROC SORT and then PROC PRINT to print the SASHELP.CLASS data set in the order of the SEX variable.

```
proc sort data=sashelp.class out=class; by sex;
proc print data=class noobs;
   var sex name height weight;
run;
```

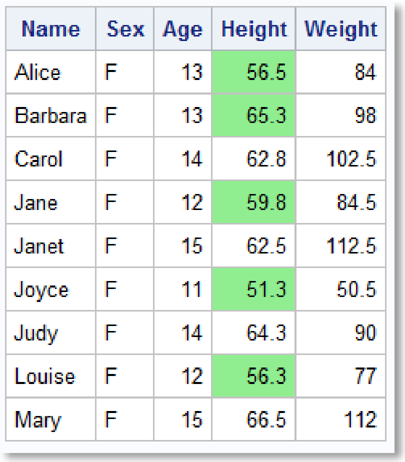
Example 20 generates the exact same output with PROC REPORT, but sorting is not required. The SEX variable is defined as ORDER, which means that its values are not necessarily printed on every row. The compute blocks ensure that the value of SEX is written on every row, matching what PROC PRINT does by default.

```
proc report data=sashelp.class;
   column sex name height weight;
   define sex / order;
   compute before sex;
     tsex = sex;
   endcomp;
   compute sex;
     if sex='' then sex=tsex;
   endcomp;
run;
```

PROC PRINT and PROC REPORT each provide style-override capabilities. For example, foreground and background colors can be applied to each variable (trafficlighting). PROC REPORT has an advantage over PROC PRINT because you can apply a style that is based on another variable.

Example 21 demonstrates changing the background color of the HEIGHT column based on the value of AGE. The results are in Output 10.

```
proc report data=sashelp.class(where=(sex='F'));
   define age / display;
   compute height;
     if age < 14 then
     call define('height.sum','style','style=[background=lightgreen]');
   endcomp;
run;
```

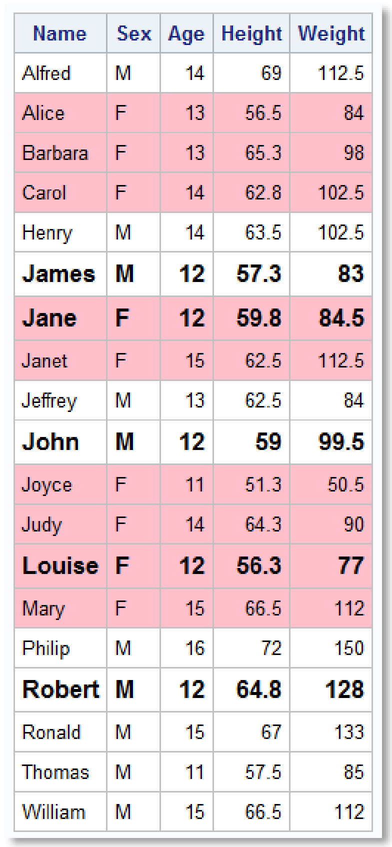


Name	Sex	Age	Height	Weight
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90
Louise	F	12	56.3	77
Mary	F	15	66.5	112

Output 10. Cell Color Based on Other Columns

REPORT can also do something else that PROC PRINT cannot do: apply a style at the row level. Example 22 demonstrates changing the background color, font weight, and font size based on the values of SEX and AGE. Output 11 shows the results.

```
proc report data=sashelp.class;
    define age / display;
    compute sex;
       if sex = "F" then
       call define(_row_,'style','style=[background=pink]');
   endcomp;
   compute age;
       if age = 12 then
       call define(_row_,'style/merge','style=[font_weight=bold
           font_size=12pt]');
    endcomp;
run;
```

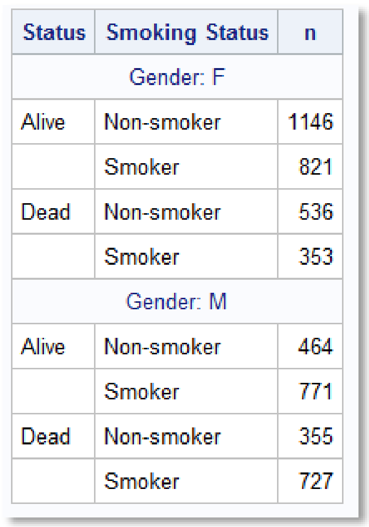


Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	**M**	**12**	**57.3**	**83**
Jane	**F**	**12**	**59.8**	**84.5**
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84
John	**M**	**12**	**59**	**99.5**
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90
Louise	**F**	**12**	**56.3**	**77**
Mary	F	15	66.5	112
Philip	M	16	72	150
Robert	**M**	**12**	**64.8**	**128**
Ronald	M	15	67	133
Thomas	M	11	57.5	85
William	M	15	66.5	112

Output 11. PROC REPORT Row-level Styling

Both PROC REPORT and PROC TABULATE can transpose on the fly, calculate statistics, and insert missing categories. Two things PROC REPORT can do that PROC TABULATE cannot do are hide columns and write text in the middle of a table. Any variable that is included in the TABLE statement for PROC TABULATE will appear in the table. With PROC REPORT, just because a variable is in the COLUMN statement does not mean it has to be present in the final table. It is common to suppress a column with the NOPRINT option but to display its value using a LINE statement. Example 23 suppresses the printing of the SEX column and instead prints it as part of a line of text at the top of each section. The results are shown in Output 12.

```
proc report data=sashelp.heart;
    column sex status smoking_status n;
    define sex / group noprint;
    define status / group;
    define smoking_status / group format=$smk.;
    compute before sex;
       line "Gender: " sex $1.;
    endcomp;
run;
```



Status	Smoking Status	n
Gender: F		
Alive	Non-smoker	1146
	Smoker	821
Dead	Non-smoker	536
	Smoker	353
Gender: M		
Alive	Non-smoker	464
	Smoker	771
Dead	Non-smoker	355
	Smoker	727

Output 12. Values Printed as Text

SQL PROCEDURE

It is hard to categorize PROC SQL because it does so many things. It sorts, merges, renames variables, and creates indexes. It also calculates statistics, creates macro variables, and more. Depending on your comfort level with PROC SQL, you might use it more than the DATA step or any other procedure. Or you might use it only to create macro variables.

You can often accomplish the same task with one PROC SQL step that takes multiple DATA steps and a procedure. Example 24 is a good example of this. With one step, the HEIGHT variable is summed within SEX and is merged back to the full data set.

```
proc sql;
    create table example24 as
    select * from sashelp.class as one
    join
    (select distinct sex, sum(weight) as overallweight
    from sashelp.class
    group by sex) as two
    on one.sex=two.sex;
quit;
```

Obviously, PROC SQL is capable of more than what Example 24 demonstrates. Due to space considerations, this paper touches only lightly on the capabilities of PROC SQL. See the Recommended Reading section for links to two papers to learn more about what PROC SQL can do.

PROC MEANS

PROC SQL calculates many of the basic statistics that PROC MEANS and PROC SUMMARY calculate. The best procedure to use depends on what else you are trying to do. For example, you might want to join (merge) two tables while you calculate statistics, or you might need to create macro variables that contain the calculated statistics. The circumstances and your preference determine the best procedure to use.

Example 25 demonstrates how PROC SQL can be used to sum a numeric value within two categorical variables, one of which is created by using a format.

```
proc sql;
    create table example25 as
    select put(date,year4.) as date1, product, sum(qtysold) as sold,
    mean(qtysold) as mean, max(qtysold) as max
    from sashelp.snacks
    group by product, date1
    order by product, date1;
quit;
```

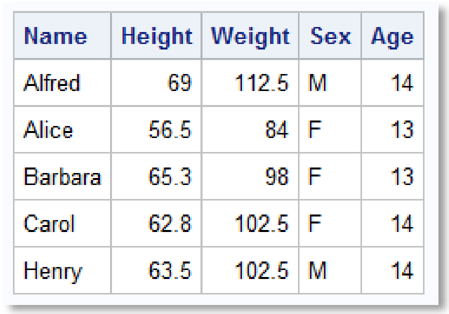
Example 26 uses PROC MEANS to generate the exact same data set that PROC SQL created in Example 25.

```
proc means data=sashelp.snacks nway noprint;
    class product date;
    format date year4.;
    var qtysold;
    output out=example26(drop=_type_ _freq_) sum=sold mean=mean max=max;
run;
```

PROC PRINT AND PROC REPORT

By default, a SELECT clause in PROC SQL generates printed output (assuming it does not follow a CREATE clause). The output is utilitarian, at best. The open ODS destination dictates the color scheme and the size of the columns. Example 27, along with Output 13, demonstrate the output generated by PROC SQL.

```
proc sql inobs=5;
    select name, height, weight, sex, age from sashelp.class;
quit;
```

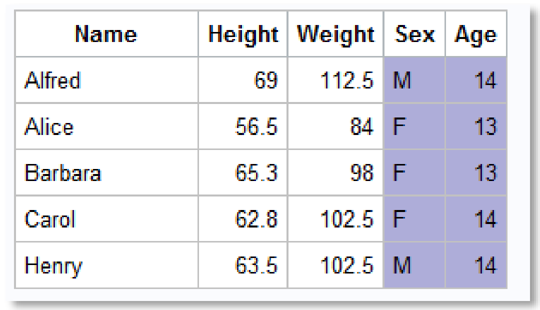


Name	Height	Weight	Sex	Age
Alfred	69	112.5	M	14
Alice	56.5	84	F	13
Barbara	65.3	98	F	13
Carol	62.8	102.5	F	14
Henry	63.5	102.5	M	14

Output 13. Default PROC SQL Results

PROC SQL does not have any style override capabilities. For generating more sophisticated reports, which include style overrides, PROC PRINT (Example 28) or PROC REPORT (Example 29) would be a better choice. You can see the effects of the overrides in Output 14.

```
proc print data=sashelp.class(obs=5) noobs
      style(header)=[background=white foreground=black font_weight=bold];
    var name /style(data)=[cellwidth=1in];
    var height weight;
    var sex age / style(data)=[background=vpab];
run;


proc report data=sashelp.class(obs=5)
      style(header)=[background=white foreground=black font_weight=bold];
    column name height weight sex age;
    define name /style(column)=[cellwidth=1in];
    define sex / style(column)=[background=vpab];
    define age / style(column)=[background=vpab];
run;
```



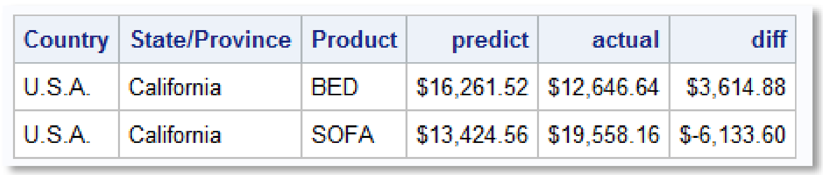
Name	Height	Weight	Sex	Age
Alfred	69	112.5	M	14
Alice	56.5	84	F	13
Barbara	65.3	98	F	13
Carol	62.8	102.5	F	14
Henry	63.5	102.5	M	14

Output 14. PROC PRINT and PROC REPORT Output with Style Overrides

The PharmaSUG 2019 paper contains a PROC REPORT example that calculates a new column. It is worth revisiting that example here because PROC REPORT and PROC SQL are the only Base SAS procedures that can create a new column for printing.

PROC SQL, in Example 30, requires the SUM function to produce a consolidated report with one row for each combination of COUNTRY, STATE, and PRODUCT. Notice that the SUM function is also placed around the calculation for the new column, DIFF. Output 15 shows the results.

```
proc sql inobs=20;
    select country, state, product,
    sum(predict) as predict format=dollar10.2,
    sum(actual) as actual format=dollar10.2,
    sum(predict-actual) as diff format=dollar10.2
    from sashelp.prdsal2
    group by country, state, product
    order by country, state, product;
quit;
```



Country	State/Province	Product	predict	actual	diff
U.S.A.	California	BED	$16,261.52	$12,646.64	$3,614.88
U.S.A.	California	SOFA	$13,424.56	$19,558.16	$-6,133.60

Output 15. Table with Calculated Column

Since PROC SQL can be intimidating, if you want to generate a column based on the values of other variables in the data set, then PROC REPORT is the way to do it. Example 31 generates the same table as Example 30.

```
proc report data=sashelp.prdsal2(obs=20);
    column country state product predict actual diff;
    define country / group ;
    define state / group;
    define product / group;
    define diff / computed 'Dif=Pred-Act' format=dollar10.2;
    compute diff;
      diff=predict.sum-actual.sum;
    endcomp;
run;
```

CONCLUSION

The goal of this paper was to empower you. The examples compared the FREQ, MEANS, TABULATE, PRINT, REPORT, and SQL procedures. Knowing the strengths of each of the procedures and when they might work better than another procedure is valuable. You can choose the procedure that works best for your programming needs and style.

REFERENCES

McLawhorn, Kathryn. 2011. “Choosing the Road Less Traveled: Performing Similar Tasks with either SAS DATA Step Processing or with Base SAS Procedures.” Proceedings of the SAS Global Forum 2011 Conference. Cary, NC: SAS Institute Inc. https://support.sas.com/resources/papers/proceedings11/259-2011.pdf.

McLawhorn, Kathryn. 2013. “Tips for Generating Percentages Using the SAS TABULATE Procedure.” Proceedings of the SAS Global Forum 2013 Conference. Cary, NC: SAS Institute Inc. https://support.sas.com/resources/papers/proceedings13/134-2013.pdf.

McLawhorn, Kathryn. 2014. “Which Base procedure is best for simple statistics?” Available at https://blogs.sas.com/content/sgf/2014/05/09/which-base-procedure-is-best-forsimple-statistics/. Last modified October 1, 2014. Accessed on January 1, 2019.

SAS Institute Inc. “SAS Note 52520: Tips for determining which Base SAS procedure to choose.” http://support.sas.com/kb/52/520.html.

ACKNOWLEDGMENTS

The author is immensely grateful to Marcia Surratt, Kathryn McLawhorn, and Sharon Hamrick for contributing to this paper.

RECOMMENDED READING

- Eslinger, Jane. 2019. “It’s All about the Base—Procedures*.” Proceedings of the SAS Global Forum 2019 Conference*. Cary, NC: SAS Institute Inc. https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2019/3068-2019.pdf.
- Eslinger, Jane. 2015. “The REPORT Procedure: A Primer for the Compute Block.” *Proceedings of the SAS Global Forum 2015 Conference*. Cary, NC: SAS Institute Inc. https://support.sas.com/resources/papers/proceedings15/SAS1642-2015.pdf.
- McLawhorn, Kathryn. 2017. “Tables and Graphics That Will ‘FREQ’ You Out.” *Proceedings of the SAS Global Forum 2017 Conference*. Cary, NC: SAS Institute Inc. https://support.sas.com/resources/papers/proceedings17/SAS0404-2017.pdf.
- Lafler, Kirk Paul. 2009. “Exploring PROC SQL Joins and Join Algorithms.” *Proceedings of the SAS Global Forum 2009 Conference*. Spring Valley, CA: Software Intelligence Corporation. https://support.sas.com/resources/papers/proceedings09/035-2009.pdf.
- Lafler, Kirk Paul. 2017. “Advanced Programming Techniques with PROC SQL.” *Proceedings of the SAS Global Forum 2017 Conference*. Spring Valley, CA: Software Intelligence Corporation. https://support.sas.com/resources/papers/proceedings17/0930-2017.pdf.
- SAS Institute Inc. 2018. Base SAS 9.4 Procedures Guide, Seventh Edition. Cary, NC: SAS Institute Inc. https://go.documentation.sas.com/?docsetId=proc&docsetTarget=titlepage.htm&docsetVersion=9.4&locale=en.

CONTACT INFORMATION

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

Jane Eslinger
SAS Institute Inc.
Jane.Eslinger@sas.com
www.sas.com

APPENDIX

The PROC REPORT code below mimics the PROC TABULATE code from Example 15 and the numbers shown in Output 8.

```
data heart;
   length sex $20;
   set sashelp.heart;
   array d{6} dummy1-dummy6;
   do i=1 to 6;
     d{i} = 1;
   end;
run;

proc report data=heart;
   column dummy1-dummy6 sex status smoking_status n temp;
   define dummy1-dummy6 / group noprint;
   define sex / group;
   define status / group;
   define smoking_status / group format=$smk.;
   define temp / noprint;

   break after dummy1 / summarize;
   break after dummy2 / summarize;
   break after dummy3 / summarize;
   break after dummy4 / summarize;
   break after dummy5 / summarize;
   break after dummy6 / summarize;

   compute before sex;
     tsex = sex;
   endcomp;
   compute before status;
     tstatus = status;
   endcomp;

   compute temp;
     if _break_ = '' then do;
        if tsex = "Female" then sexf + n;
        else sexm + n;

        if tstatus = "Alive" then statusa + n;
        else statusd +n;

        if smoking_status = "Non-smoker" then smkn + n;
        else smko + n;
     end;
```

```
    endcomp;

    compute after dummy6;
       sex = "Gender Totals";
       smoking_status = "Female";
       n=sexf;
    endcomp;
    compute after dummy5;
      sex = "Gender Totals";
      smoking_status = "Male";
      n=sexm;
    endcomp;
    compute after dummy4;
      sex = "Mortality Totals";
      smoking_status = "Alive";
      n=statusa;
    endcomp;
    compute after dummy3;
      sex = "Mortality Totals";
      smoking_status = "Dead";
      n=statusd;
    endcomp;
    compute after dummy2;
      sex = "Smoking Totals";
      smoking_status = "Smoker";
      n=smko;
    endcomp;
    compute after dummy1;
      sex = "Smoking Totals";
      smoking_status = "Non-smoker";
      n=smkn;
    endcomp;
run;
```