Paper 3068-2019

It's All about the Base—Procedures

Jane Eslinger, SAS Institute Inc.

ABSTRACT

As a Base SAS® programmer, you spend your day manipulating data and creating reports. You know there is a procedure that can give you what you want. As a matter of fact, there is probably more than one procedure to accomplish the task. Which one should you use? How do you remember which procedure is best for which task?

This paper is all about the Base procedures. It explores the strengths of the commonly used, nongraphing procedures. It discusses the challenges of using each procedure and compares it to other procedures that accomplish similar tasks. The first section of the paper looks at utility procedures that gather and structure data: APPEND, COMPARE, CONTENTS, DATASETS, FORMAT, SORT, SQL, and TRANSPOSE. The next section discusses the Base SAS procedures that work with statistics: FREQ, MEANS/SUMMARY, and UNIVARIATE. The final section provides information about reporting procedures: PRINT, REPORT, and TABULATE.

INTRODUCTION

The beauty of SAS is that usually there is more than one method to get what you need. That is a good thing! It can also be a challenge—you might not know if there is a more efficient method or a method that requires a lot less code.

This paper provides brief descriptions of the most frequently used Base SAS procedures. More importantly, it provides ideas for the best way to use the procedure and a comparison of when another procedure or technique might be better to use. Not every functionality of every procedure can be listed here. Nor can every comparison be made. Hopefully this paper will provide ideas on how to accomplish the tasks you need to accomplish, and it might encourage you to try multiple methods for accomplishing those tasks.

Please note that, due to space considerations, this paper focuses on using these procedures for SAS®9 data sets (and not for database tables).

UTILITY PROCEDURES

For the purposes of this paper, the utility procedures are those that enable you to get information about the data or to complete pre-analysis tasks such as restructuring the data.

APPEND PROCEDURE

Description: PROC APPEND is designed to add the observations in one data set (DATA= data set) to the bottom of another data set (BASE= data set). This process is also known as concatenating data sets.

Challenges:

- The data structure of the two data sets must match. The variable names, lengths, and types must be the same in the two data sets. By default, if the structures do not match, PROC APPEND will not append the observations. The FORCE option will force the append process to occur if the structure is different. However, in this case, variables in the DATA= data set that are not in the BASE= data set are not in the resulting data set.

- You cannot make changes to the observations in the BASE= data set or to the observations being added. The observations are simply appended.

Best Use Case: PROC APPEND has a very narrow focus—to add observations from one data set to another. It does not open the data set being added to. Therefore, it uses a limited amount of I/O. The ideal circumstance for using PROC APPEND is when you have routine or cyclical data and you need to add the latest data to the historical data.

Example Code:

```
data work.prdsal2;
   set sashelp.prdsal2;
run;

data sale2000;
   length country $10 state $22 county $20 prodtype product $10;
   input country $ state $ county actual predict prodtype $ product
   $ year
         quarter monyr monyy.;
   month=month(monyr);
   datalines;
U.S.A California . 1702.50 40.00 FURNITURE BED 2000 1 JAN00
;
run;

proc append base=prdsal2 data=sale2000;
run;
```

Comparisons: A DATA step that uses the SET statement can accomplish the same task as PROC APPEND. With PROC APPEND, the I/O to produce the bigger data set is a lot lower than when using a DATA step. The advantages of using a SET statement in a DATA step include the ability to rename the variables from each data set, change lengths or other attributes, or manipulate the data.

Example Code:

```
data allsale;
   set prdsal2(rename=(predict=orig_predict))
       sale2000(rename=(predict=orig_predict));

   predict=orig_predict*1.01;
run;
```

COMPARE PROCEDURE

Description: PROC COMPARE compares the variable values and attributes of two data sets and generates a report of the differences.

Challenges:

- The reports that are generated by PROC COMPARE contain the differences between the two data sets. It does not display values that are the same.
- When specified, the COMPARE procedure compares the data sets according to the BY and ID values. However, it does not provide a simple list of BY or ID values that are in one data set but not the other, or that are in both.
- There are no options with PROC COMPARE that will prevent the numeric values from being displayed in scientific notation or character values from being truncated at 20

characters. PROC COMPARE does not have an underlying table template that can be modified with PROC TEMPLATE to change the formatting of the printed output. An output data set must be created and then printed to control the formatting for long values.

- It is difficult to look among all the differences for many records and variables that could all be different.

Best Use Case: Use PROC COMPARE when you are confident that the data sets are very nearly the same and you want to find the one or two variables that might have differences.

When developing the program or when expecting a large number of differences, it might be best to suppress the output generated by PROC COMPARE by using the NOPRINT option. Instead, create a data set and then print that.

Example Code:

```
proc compare data=sashelp.prdsal2 comp=allsale noprint outdif
             outcomp outnoequal outbase out=diffs;
run;
```

Comparisons: There is no other procedure that has similar functionality to PROC COMPARE. It is possible to merge data sets and use assignment logic to compare the values of two data sets. That would be a substantial programming challenge, based on the number of variables and their type along with the desired structure of the final data set.

Alternatively, in theory, you can compare the two files by converting the data sets to text files and then using a comparison tool that is not offered by SAS.

CONTENTS PROCEDURE

Description: PROC CONTENTS provides data set attribute information and engine and host information, as well as a list of variables within the data set. The attribute information includes the number of observations, the number of variables, whether a data set is sorted, whether it has indexes, and what its encoding is. The engine and host information section provides the file size and the version of SAS and the operating system that created the data set. For the list of variables, PROC CONTENTS provides details about types, lengths, formats, informats, and labels.

PROC CONTENTS does not read or process the data. It reads only the data set header information.

Challenges:

- PROC CONTENTS does not report the number of observations for a DATA step view.
- The **Created by** and **Last Modified by** characteristics are unique to z/OS sequential access bound libraries. PROC CONTENTS does not provide these characteristics in UNIX or Microsoft Windows environments.
- PROC CONTENTS does not provide the number of observations that are contained in an external database, such as an Oracle table.

Best Use Case: Use PROC CONTENTS to check that your data set has the variables that you expect it to have, including their labels, types, and formats (Output 1 shows an example of a list of variables). The generated report is also a good way to see the SAS version and the encoding that was used to create the data set, which is critical information if you get a cross-environment data access error after you try to use the data set.

Example Code:

```
proc contents data=sashelp.prdsal2 out=contents;
run;
```

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat	Label
4	ACTUAL	Num	8	DOLLAR12.2		Actual Sales
1	COUNTRY	Char	10	$CHAR10.		Country
3	COUNTY	Char	20	$CHAR20.		County
10	MONTH	Num	8	MONNAME3.		Month
11	MONYR	Num	8	MONYY.	MONYY.	Month/Year
5	PREDICT	Num	8	DOLLAR12.2		Predicted Sales
6	PRODTYPE	Char	10	$CHAR10.		Product Type
7	PRODUCT	Char	10	$CHAR10.		Product
9	QUARTER	Num	8	8.		Quarter
2	STATE	Char	22	$CHAR22.		State/Province
8	YEAR	Num	8	4.		Year

Output 1. Variable List Created with PROC CONTENTS

Comparisons: A PROC SQL step that references a DICTIONARY table (columns, indexes, members, and tables) can provide much of the same information that PROC CONTENTS does.

DICTIONARY.INDEXES provides more information about indexes than PROC CONTENTS. The below code demonstrates the PROC SQL step.

Example Code:

```
ods output indexes=indexzip1;
proc contents data=sashelp.zipcode out=contentszip;
run;

proc sql;
   create table indexzip2
   as select *
   from dictionary.indexes
   where libname='SASHELP' and memname='ZIPCODE';
quit;
```

DATASETS PROCEDURE

Description: PROC DATASETS enables you to change variable attributes, rename variables, create indexes, and see the contents of a data set.

Challenges:

- PROC DATASETS enables you to copy and rename variables only. It cannot drop variables.

- It does not provide the number of observations that are contained in an external database table.

Best Use Case:

- PROC DATASETS is often used to delete all the data sets in a library. However, be careful when you do this, because once deleted, you cannot get them back.

Example Code:

```
proc datasets library=work kill;
quit;
```

- Use PROC DATASETS to modify formats and labels without having to run a DATA step—it is less resource-intensive (Output 2 shows a log that confirms that modification is successful).

Example Code:

```
data work.prdsal2;
   set sashelp.prdsal2;
run;

proc datasets library=work nolist;
   modify prdsal2;
   label product='Product Name';
   format country $char20.;
quit;
```

```
5
6    /****Best Use Case****/
7    /*Modify labels and formats*/
8    proc datasets library=work nolist;
NOTE: Writing HTML Body file: sashtml.htm
9       modify prdsal2;
10      label product='Product Name';
11      format country $char20.;
12   quit;

NOTE: MODIFY was successful for WORK.PRDSAL2.DATA.
NOTE: PROCEDURE DATASETS used (Total process time):
      real time           0.20 seconds
      cpu time            0.12 seconds
```

Output 2. Log Showing That a PROC DATASETS Modification Is Successful

Comparisons:

- PROC DATASETS and PROC CONTENTS provide the same information about either a data set or all the data sets in a library. For content information, use the procedure that you are most comfortable with.

Example Code:

```
proc datasets library=work;
   contents data=prdsal2;
quit;

proc contents data=prdsal2;
run;
```

- PROC SQL provides data set information and enables you to change a variable name at the same time. Keep in mind that PROC SQL will process the entire data set, not just the header information.

Example Code:

```
proc datasets library=work nolist;
   modify prdsal2;
   index create state / nomiss;
   format monyr yymon7.;
quit;

proc sql noprint;
   create table prdsal2b as
   select country, state, county, actual, predict, prodtype, product,
          year, quarter as qtr, month, monyr format=yymon7.
   from sashelp.prdsal2;
   create index state on prdsal2b(state);
quit;
```

FORMAT PROCEDURE

Description: PROC FORMAT creates informats, which are instructions for how Base SAS should read and interpret raw values. It also creates formats and picture formats, which are instructions for how to display data values.

Formats are created either by hardcoding the values or by using a CNTLIN= data set, which contains variables with the desired value-label combinations.

Challenges: Forethought is necessary: you must know how you want your data values displayed.

Best Use Case:

- Generally, PROC FORMAT is used to create a format so that raw values, such as **zero** and **one**, are displayed as another value, such as **No** and **Yes**.

Example Code:

```
data work.prdsal2;
   set sashelp.prdsal2;
run;

proc format;
   value $ typ
         'FURNITURE'='Residential'
         'OFFICE'='Commercial'
         other='Other';
run;
```

(code continued)

```
data prdsal2a;
   length prodtype $12;
   format prodtype $12.;
   set prdsal2;
   prodtype=put(prodtype,$typ.);
run;
```

- Creating a format is also needed when you want to reassign values (in a DATA step); to group values or categories when calculating statistics or creating reports in other procedures; or to generate rows or placeholders for categories that are not present in the data. The code below demonstrates using a format to group values in PROC FREQ output.

Example Code:

```
proc format;
   value $ func
         'BED', 'SOFA'='Soft'
         'DESK','CHAIR'='Hard';
run;

proc freq data=sashelp.prdsal2;
   format product $func.;
   table product;
run;
```

Comparisons: There is no other procedure that has similar functionality to PROC FORMAT. In a DATA step, you can choose to use IF-THEN or IF-THEN/ELSE statements to conditionally assign a new value based on an old value. To an extent, one method is not better than the other, because you still must type out both the new and old values.

Example Code:

```
data work.prdsal2;
   set sashelp.prdsal2;
run;

data prdsal2b;
   length prodtype $12;
   set prdsal2;
   if prodtype='FURNITURE' then prodtype='Residential';
   else if prodtype='OFFICE' then prodtype='Commercial';
   else prodtype='Other';
run;
```

SORT PROCEDURE

Description: The SORT procedure sorts data and can remove duplicate observations.

Challenges: Sorting data is a common, necessary practice. However, PROC SORT will create a temporary utility file that is approximately twice the size of the original data set. The amount of memory and disk space used by PROC SORT can be problematic for large data sets.

Best Use Case: You should sort your data with PROC SORT when the other procedures that you are using for analytics and reporting require sorting. However, data that is sorted or grouped prior to being brought into SAS does not need re-sorting. In most procedures, you can take advantage of the NOTSORTED option in the BY statement. This option will group

together observations that have the same BY value. However, it does not put them into any specific order, such as by numeric value. Also, using a CLASS statement within a procedure does not require the data to be presorted.

Example Code:

```
proc sort data=sashelp.prdsal2 out=prdsal2;
   by year product country;
run;
```

Comparisons: PROC SQL can sort data sets. The code below demonstrates using both PROC SORT and PROC SQL to sort a data set and keep only the unique combinations of COUNTRY, STATE, and PRODUCT. The two steps generate identical data sets.

Example Code:

```
proc sort data=sashelp.prdsal2 out=unqprd1(keep=country state product)
          nodupkey;
   by country state product;
run;

proc sql noprint;
   create table unqprd2 as
   select distinct country, state, product
   from sashelp.prdsal2
   order by country, state, product;
quit;
```

SQL PROCEDURE

Description: PROC SQL is kind of a super utility procedure. It lets you sort data, merge data sets, create indexes, and rename variables. You can also create macro variables and calculate statistics. The procedure accesses DICTIONARY tables, which are special read-only PROC SQL tables or views that contain information about all the SAS libraries and SAS data sets.

Challenges:

- It is not a standard SAS procedure that requires a certain set of statements or options. There is a learning curve to understand the syntax and take full advantages of its capabilities.
- It does not create a production-quality report, and you cannot manipulate the printed table with style overrides.

Best Use Case:

- For the average SAS programmer, PROC SQL is best used for generating macro variables in one step, such as a list of unique variable values, population totals, or minimum and maximum values.

Example Code:

```
proc sql noprint;
select count(country) into :usa from sashelp.prdsal2
       where country="U.S.A.";
select count(country) into :mex from sashelp.prdsal2
       where country="Mexico";
```

(code continued)

```
select count(country) into :can from sashelp.prdsal2
       where country="Canada";
quit;

proc sql noprint;
   select distinct(state) into :stlist1 separated by ', '
   from sashelp.prdsal2;
quit;
```

- Access to DICTIONARY tables is also very useful when you need information about libraries, data sets, and variables.

Example Code:

```
proc sql;
   select *
   from dictionary.tables
   where libname='WORK';
quit;
```

Comparisons: PROC SQL functionality overlaps that of the DATASETS, CONTENTS, FREQ, MEANS, and SORT procedures. It also has some of the same capabilities as the DATA step.

It is difficult to provide a set rule for when to use it over other procedures. If you need a Cartesian product, use PROC SQL. If you understand merging with the IN= operator more than joins, use a DATA step with the MERGE statement.

The code below demonstrates generating the same counts with both PROC SQL and PROC FREQ.

Example Code:

```
proc sql;
   select product, count(product) as n
   from sashelp.prdsal2
   group by product
   order by product;
quit;

proc freq data=sashelp.prdsal2;
   tables product /nopercent nocum;
run;
```

TRANSPOSE PROCEDURE

Description: PROC TRANSPOSE restructures data. It changes data sets from long and narrow to wide and short, or vice versa.

Challenges:

- It usually requires that data be sorted first.
- Controlling the resulting variable names can be challenging; to do that, sometimes you must use two PROC TRANSPOSE steps.

Best Use Case: Use PROC TRANSPOSE with a data set that is unique—either within a set of BY variables or over the entire data set—where you need to create new numeric variables named after the value of a categorical variable.

Example Code:

```
proc transpose data=sashelp.stocks out=test;
   where year(date)=2005;
   by stock;
   id date;
   var close;
run;
```

Comparisons: Using a DATA step can transpose data with the use of ARRAY, DO, and OUTPUT statements. It often provides more control over the resulting variables, and you can include any other DATA step functionality that you need. Using a DATA step as an alternative to PROC TRANSPOSE is daunting for some programmers because of the use of arrays.

Transposing multiple variables is a challenging task. The code below provides a comparison of the steps needed to use PROC TRANSPOSE and a DATA step that accomplishes the same task.

Transposing multiple variables with PROC TRANSPOSE requires multiple steps: a PROC SORT step, two PROC TRANSPOSE steps, and a DATA step.

Example Code:

```
proc sort data=sashelp.prdsale out=prdsale;
   where product='SOFA' and year=1993 and region='EAST';
   by country division prodtype;
run;

proc transpose data=prdsale out=sale1 prefix=predict_;
   by country division prodtype;
   id month;
   var predict;
run;

proc transpose data=prdsale out=sale2 prefix=actual_;
   by country division prodtype;
   id month;
   var actual;
run;

data all1;
   merge sale1 sale2;
   by country division prodtype;
run;
```

The DATA step code below accomplishes the same task of transposing the data as the previous four steps.

Example Code:

```
data all2;
   set prdsale;
   by country division prodtype;
```

(code continued)

```
   array pred{12} predict_Jan predict_Feb predict_Mar predict_Apr
                  predict_May predict_Jun predict_Jul predict_Aug
                  predict_Sep predict_Oct predict_Nov predict_Dec;
   array actl{12} actual_Jan actual_Feb actual_Mar actual_Apr
                  actual_May actual_Jun actual_Jul actual_Aug
                  actual_Sep actual_Oct actual_Nov actual_Dec;
   array vals{12} $ ('Jan' 'Feb' 'Mar' 'Apr' 'May' 'Jun' 'Jul' 'Aug'
                     'Sep' 'Oct' 'Nov' 'Dec');
   retain predict: actual_:;

   do i=1 to 12;
      if put(month,monname3.)=vals{i} then do;
            pred{i}=predict;
            actl{i}=actual;
      end;
   end;
   if last.prodtype;
   drop actual predict quarter year vals: i;
run;
```

STATISTICAL-ESQUE PROCEDURES

For the purposes of this paper, statistical-esque procedures are those that enable you to summarize your data and calculate various statistics. The results might or might not be in final, report-quality form.

FREQ PROCEDURE

Description: PROC FREQ gives counts and percentages. It can also generate statistics, like Fisher's exact test or the chi-square test.

Challenges:

- PROC FREQ 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.
- The OUT= and ODS output data sets do not contain the same variables or structure.
- It is not easy to change the format of the statistics for one-way tables. For example, you cannot display the counts with commas without making a change to the underlying table template.
- You can get overall percentages for a whole data set or within a BY group, but you cannot generate customized percentages.

Best Use Case:

- PROC FREQ is often used to create data sets with total counts that can be used later as denominators, when merged with detail data.
- PROC FREQ is also extremely useful for checking your data to ensure that it contains all expected categories. An example is shown in Output 3.

Example Code:

```
proc freq data=sashelp.prdsal2;
   tables country*product / list out=freqs;
run;
```

The FREQ Procedure

COUNTRY	PRODUCT	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Canada	BED	1152	5.00	1152	5.00
Canada	CHAIR	1152	5.00	2304	10.00
Canada	DESK	1152	5.00	3456	15.00
Canada	SOFA	1152	5.00	4608	20.00
Mexico	BED	1152	5.00	5760	25.00
Mexico	CHAIR	1152	5.00	6912	30.00
Mexico	DESK	1152	5.00	8064	35.00
Mexico	SOFA	1152	5.00	9216	40.00
U.S.A.	BED	3456	15.00	12672	55.00
U.S.A.	CHAIR	3456	15.00	16128	70.00
U.S.A.	DESK	3456	15.00	19584	85.00
U.S.A.	SOFA	3456	15.00	23040	100.00

Output 3. PROC FREQ Output

Comparisons: PROC REPORT and PROC TABULATE also calculate counts and percentages. They might require more coding to get the same information, but they enable you to have more control over formatting and table structure.

The PRELOADFMT option, in both the REPORT and TABULATE procedures, will display values that are present in the format but are not present in the data set.

PROC TABULATE can calculate subtotals, which PROC FREQ cannot.

Example Code:

```
proc tabulate data=sashelp.prdsal2 format=8.;
   class country product;
   table country*(product all='SubTotal') all='Total', N;
run;
```

MEANS AND SUMMARY PROCEDURES

Description: PROC MEANS calculates statistics, such as N, sum, mean, minimum, maximum, standard error, and percentiles. PROC SUMMARY is very similar to PROC MEANS: it computes the same descriptive statistics. By default, the SUMMARY procedure does not produce printed output. Also, if the VAR statement is omitted, PROC SUMMARY produces a simple count of observations.

Challenges:

- The procedures do not calculate percentages.
- The structure of the report output is either difficult or impossible to change.

Best Use Case: Use PROC MEANS when you want to generate data sets with statistics, totals, and overall. Simultaneously, use it to add records, with a count of zero, for values that are not present in the data. See Output 4 for an example of the PROC MEANS output.

Example Code:

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

proc means data=sashelp.prdsal2 completetypes nway;
   format product $prod.;
   class product / preloadfmt;
   var actual predict;
   output out=means mean= sum= n= /autoname;
run;
```

The MEANS Procedure

Product	N Obs	Variable	Label	N	Mean	Std Dev	Minimum	Maximum
BED	5760	ACTUAL	Actual Sales	5760	644.1409447	654.5324375	0.0615385	3440.80
		PREDICT	Predicted Sales	5760	678.7320144	672.2739927	0	3605.70
CHAIR	5760	ACTUAL	Actual Sales	5760	641.5596563	640.5110277	0	3376.20
		PREDICT	Predicted Sales	5760	694.1324808	691.6989891	0	3564.90
DESK	5760	ACTUAL	Actual Sales	5760	645.5484375	656.2886439	0	3415.30
		PREDICT	Predicted Sales	5760	696.0894495	687.8018861	0	3609.10
SOFA	5760	ACTUAL	Actual Sales	5760	674.3381707	673.9880104	0	3515.60
		PREDICT	Predicted Sales	5760	695.4524014	688.4784458	0	3634.60
TABLE	0	ACTUAL	Actual Sales	0	.	.	.	.
		PREDICT	Predicted Sales	0	.	.	.	.

Output 4. PROC MEANS Output

Comparisons:

- PROC TABULATE calculates the same statistics as the MEANS and SUMMARY procedures, as well as percentages.

Example Code:

```
proc tabulate data=sashelp.prdsal2;
   class year country;
   var actual;
   table country='', year*actual=''*(mean median);
run;
```

- PROC SQL calculates statistics and has the additional capability of generating macro variables with the calculated values.

Example Code:

```
proc sql noprint;
   select max(actual) into :actmax from sashelp.prdsal2;
quit;
```

UNIVARIATE PROCEDURE

Description: PROC UNIVARIATE calculates summary statistics, generates quantiles and percentiles, performs tests for goodness of fit, and produces probability plots.

Challenges:

- The default output contains a large number of tables and data. To limit the generated output, you have to use an ODS SELECT statement or an ODS EXCLUDE statement.
- There are no options in the PROC UNIVARIATE statement for subsetting to get only the statistics that you are interested in. You have to create an output data set and manipulate it to get only the statistics that you need.

Best Use Case: Use it to generate custom (nondefault) percentiles.

Example Code:

```
proc univariate data=sashelp.prdsal2;
   var actual;
   output out=univ pctlpts=15 25 35 pctlpre=actual_p;
run;
```

Comparisons: PROC MEANS calculates the same summary statistics as PROC UNIVARIATE. If you do not need the statistical testing or plotting capabilities, either procedure can be used. It is a matter of programmer preference.

Example Code:

```
proc means data=sashelp.prdsal2 min max mean mode;
   class product;
   var predict actual;
   output out=stats mean= sum= /autoname;
run;
```

REPORTING PROCEDURES

For the purposes of this paper, reporting procedures are those that are used to create final, publication-quality report tables.

PRINT PROCEDURE

Description: The PRINT procedure displays data. Summary totals can be requested for numeric variables.

Challenges:

- The amount of customization in the final table is limited. For example, PROC PRINT cannot either apply formatting based on another column or include additional text.
- The values of numeric columns are simply displayed. Their values cannot be added, subtracted, multiplied, or divided against each other.
- An output data set cannot be created. There is no OUT= data set option, and PROC PRINT does not support the ODS OUTPUT statement.

Best Use Case:

- Use PROC PRINT when you need a quick view of your data to ensure it has the variables and values that you expect.

Example Code:

```
proc print data=sashelp.prdsal2(obs=30);
   var country state product actual predict;
run;
```

- PROC PRINT is the easiest way to get a report when your data set contains everything you need and it does not require additional formatting, or you have a small amount of data and want a grand total. Output 5 provides an example with a summary row at the end.

Example Code:

```
proc print data=sashelp.class grandtotal_label='Grand Total';
   var name age height weight;
   sum height weight;
run;
```

Obs	Name	Age	Height	Weight
1	Alfred	14	69.0	112.5
2	Alice	13	56.5	84.0
3	Barbara	13	65.3	98.0
4	Carol	14	62.8	102.5
5	Henry	14	63.5	102.5
6	James	12	57.3	83.0
7	Jane	12	59.8	84.5
8	Janet	15	62.5	112.5
9	Jeffrey	13	62.5	84.0
10	John	12	59.0	99.5
11	Joyce	11	51.3	50.5
12	Judy	14	64.3	90.0
13	Louise	12	56.3	77.0
14	Mary	15	66.5	112.0
15	Philip	16	72.0	150.0
16	Robert	12	64.8	128.0
17	Ronald	15	67.0	133.0
18	Thomas	11	57.5	85.0
19	William	15	66.5	112.0
Grand Total			1184.4	1900.5

Output 5. PROC PRINT Output with a Summary Row

Comparisons: PROC REPORT can simply display data, but it can also apply custom formatting and transpose the data.

Example Code:

```
proc report data=sashelp.prdsal2;
   column country product predict actual;
   define country / group;
   define product / group;
   break after country / summarize;
   compute actual;
      if actual.sum > predict.sum then
      call define('actual.sum','style',
      'style=[foreground=green]');
   endcomp;
run;
```

REPORT PROCEDURE

Description: PROC REPORT generates both detailed and summary reports, meaning it can print all observations from your data set or group them into categories. It can calculate statistics, create new variables, and transpose data.

Challenges:

- There is a learning curve to using the compute block, the most powerful aspect of PROC REPORT.
- It does not provide a subtotals breakdown like PROC TABULATE can generate.

Best Use Case: Use it to create a final report that includes customized headers and text, coloring based on certain values (trafficlighting), and calculations based on various columns. See Output 6 for an example of a report that uses a customized header.

Example Code:

```
proc report data=sashelp.prdsal2;
   column country state product predict actual diff;
   define country / group noprint;
   define state / group;
   define product / group;
   define diff / computed 'Dif=Pred-Act' format=dollar10.2;
   break after country / summarize;

   compute before country;
      line 'Section for ' country $20.;
   endcomp;
   compute after country;
      state='Total';
   endcomp;
   compute diff;
      diff=predict.sum-actual.sum;
   endcomp;
run;
```

State Province	Product	Predicted Sales	Actual Sales	Dif=Pred-Act
Section for Canada				
British Columbia	BED	$225,369.00	$197,706.60	$27,662.40
	CHAIR	$204,264.00	$200,905.20	$3,358.80
	DESK	$194,682.60	$186,262.20	$8,420.40
	SOFA	$230,587.20	$216,282.60	$14,304.60
Ontario	BED	$210,796.20	$194,493.60	$16,302.60
	CHAIR	$206,632.80	$179,892.00	$26,740.80
	DESK	$206,857.80	$208,778.40	$-1,920.60
	SOFA	$221,410.80	$196,882.20	$24,528.60
Quebec	BED	$219,006.00	$204,737.40	$14,268.60
	CHAIR	$225,808.20	$199,751.40	$26,056.80
	DESK	$215,080.20	$202,555.80	$12,524.40
	SOFA	$227,037.60	$200,777.40	$26,260.20
Saskatchewan	BED	$195,429.60	$193,568.40	$1,861.20
	CHAIR	$210,690.00	$201,580.20	$9,109.80
	DESK	$228,223.80	$208,481.40	$19,742.40
	SOFA	$209,140.20	$205,178.40	$3,961.80
Total		$3431016.00	$3197833.20	$233182.80

Output 6. PROC REPORT Output (Partial)

Comparisons: PROC REPORT combines a lot of the capabilities of the PRINT and TABULATE procedures. PROC PRINT should be used for small, simple reports. PROC TABULATE easily generates subtotals and percentages that require a large amount of code in PROC REPORT.

Example Code:

```
proc format;
   picture mypct low-high='009.9%';
run;

proc tabulate data=sashelp.prdsal2;
   class prodtype product;
   var actual;
   table prodtype*product, actual*(sum*f=dollar12.
         pctsum<product>*f=mypct.);
run;
```

TABULATE PROCEDURE

Description: The TABULATE procedure calculates statistics and percentages. It is capable of summarizing at a categorical level and generating overall totals. It can create columns from values in the input data set.

Challenges:

- The syntax for the TABLE statement has a learning curve. The statement can be long, and it might be confusing due to the asterisks, brackets, and denominator definitions.
- The procedure cannot perform basic mathematical calculations, such as adding, subtracting, multiplying, or dividing analysis variables. However, there is a special case where you can calculate a ratio by using two analysis variables and the PCTSUM statistic.
- You can apply formatting to a variable, but that formatting cannot be based on the value of another column.
- You cannot insert text in the middle of the table.

Best Use Case: PROC TABULATE generates percentages with very little code. It can also calculate totals and subtotals for various combinations of class variables. Output 7 shows an example.

Example Code:

```
proc tabulate data=sashelp.prdsal2;
   class country product;
   var actual;
   tables country*product all='TOTALS'*product='',
          actual*(sum*f=dollar12. pctsum<product>);
run;
```

		Actual Sales	
		Sum	PctSum
Country	Product		
Canada	BED	$790,506	24.72
	CHAIR	$782,129	24.46
	DESK	$806,078	25.21
	SOFA	$819,121	25.61
Mexico	BED	$535,408	25.37
	CHAIR	$515,390	24.42
	DESK	$523,075	24.78
	SOFA	$536,870	25.44
U.S.A.	BED	$2,384,338	24.58
	CHAIR	$2,397,864	24.72
	DESK	$2,389,207	24.63
	SOFA	$2,528,197	26.06
TOTALS	BED	$3,710,252	24.72
	CHAIR	$3,695,384	24.62
	DESK	$3,718,359	24.78
	SOFA	$3,884,188	25.88

Output 7. PROC TABULATE Output

Comparisons: PROC REPORT generates the same statistics as PROC TABULATE. Its additional capabilities are manipulating cell values and formatting based on other columns.

Example Code:

```
proc report data=sashelp.prdsal2;
   column country product predict actual diff;
   define country / group;
   define product / group;
   define diff / computed 'Dif=Pred-Act' format=dollar10.2;
   break after country / summarize;
   compute diff;
      diff=predict.sum-actual.sum;
      if _break_ ^= '' then do;
            if country='U.S.A.' then call define('diff','style',
               'style=[background=lightblue]');
            else if country='Mexico' then call define('diff','style',
               'style=[background=lightgreen]');
            else if country='Canada' then call define('diff','style',
               'style=[background=lightred]');
      end;
   endcomp;
run;
```

CONCLUSION

This paper focused on commonly used Base SAS procedures. The capability of most procedures overlaps with at least one other procedure. Each procedure has its strengths and its challenges.

Use the information in this paper to help you decide whether it would be easier to accomplish a task with a different procedure. Or you can confirm that you are already doing the right thing!

REFERENCES

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

McLawhorn, Kathryn. 2013. "Tips for Generating Percentages Using the SAS® TABULATE Procedure." In *Proceedings of the SAS Global 2013 Conference*. Cary, NC: SAS Institute Inc. Available at **support.sas.com/resources/papers/proceedings13/134-2013.pdf**.

McLawhorn, Kathryn. 2014. "Which Base procedure is best for simple statistics?" Available at **blogs.sas.com/content/sgf/2014/05/09/which-base-procedure-is-best-for-simple-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." Available at **support.sas.com/kb/52/520.html**.

ACKNOWLEDGMENTS

The author is immensely grateful to Marcia Surratt, Kathryn McLawhorn, Sharon Hamrick, and Jerry Leonard for contributing to this paper, and to Chris Dahlin, who edited the paper.

RECOMMENDED READING

- Eslinger, Jane. 2015. "The REPORT Procedure: A Primer for the Compute Block." In *Proceedings of the SAS Global 2015 Conference*. Cary, NC: SAS Institute Inc. Available at **support.sas.com/resources/papers/proceedings15/SAS1642-2015.pdf**.
- McLawhorn, Kathryn. 2017. "Tables and Graphics That Will 'FREQ' You Out." In *Proceedings of the SAS Global 2017 Conference*. Cary, NC: SAS Institute Inc. Available at **support.sas.com/resources/papers/proceedings17/SAS0404-2017.pdf**.
- Olson, Diane. 2014. "Nitty Gritty Data Set Attributes." In *Proceedings of the SAS Global 2014 Conference*. Cary, NC: SAS Institute Inc. Available at **support.sas.com/resources/papers/proceedings14/SAS164-2014.pdf**.
- SAS Institute Inc. 2018. *Base SAS® 9.4 Procedures Guide, Seventh Edition.* Cary, NC: SAS Institute Inc. Available at **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.
SAS Campus Drive
Cary, NC 27513
Email: **support@sas.com**
Web: **support.sas.com/en/support-home.html**