SESUG Paper 084-2021

PROC REPORT: Tips and Customizations for Quickly Creating Customized Reports

Jonathan Duggins, NC State and James Blum, UNCW

Abstract

Producing high-quality reports is a cornerstone of data science, statistics, and statistical programming careers. While PROC REPORT has been around in SAS® since version 6, there is enough variety in what type of reports it can produce that students and practitioners alike are often unaware of some of its intricacies. We begin by reviewing the more similar usages (ORDER and GROUP) and some good practices for their use before going on to explore various applications of COMPUTE blocks. COMPUTE blocks provide multiple ways to customize the aesthetics of a report and so we conclude with a look at adjusting column, row, and cell styles based on columns that may or may not appear in the report. Each topic will be demonstrated via examples that show the code and results. Commented code and results will be available at jonathanduggins.com/conferences. Attendees should have a basic familiarity with the REPORT Procedure and some experience with conditional logic to get the most out of this presentation.

Introduction

Most SAS users are familiar with PROC PRINT as a tool for obtaining quick but simple inspections of a SAS data set. The PRINT procedure can even offer some simple markup features such as providing column sums or applying certain style options like changing the fill or text color for the cells. Nonetheless, the customization features in PROC PRINT are quite limited and many users turn to other procedures, such as TABULATE or REPORT, to produce more sophisticated output objects. Program 1 provides a reference point to start exploring alternatives to PROC PRINT.[1]

Program 1: Basic PROC PRINT

```
proc print data = SasHelp.Cars(obs = 5) label;
  var Type MPG_City MPG_Highway;
run;

proc print data = SasHelp.Cars(obs = 5) label;
  var MPG_City MPG_Highway;
run;
```

Output 1: Basic PROC PRINT

Obs	Type	MPG_City	MPG_Highway
1	SUV	17	23
2	Sedan	24	31
3	Sedan	22	29
4	Sedan	20	28
5	Sedan	18	24

Obs	MPG_City	MPG_Highway
1	17	23
2	24	31
3	22	29
4	20	28
5	18	24

[1] While not shown in the programs, all output objects produced by examples were delivered to a PDF via ODS PDF statements.

While simple, the tables in Output 1 provide a good starting point for exploring procedures that can produce reports of varying complexity. In particular, PROC REPORT offers substantial flexibility that can typically be traced to one of the following features.

1. Existence of roles for variables beyond simply printing every value in a column.
2. Columns do not have to be variables in the data set.
3. Rows do not have to be records in the data set.
4. Expanded aesthetic controls, including via conditional logic.

Of course, this flexibility comes at the cost of familiarizing yourself with the syntax of PROC REPORT. As an example of how the syntax differs and perhaps as a warning about the price we pay for that flexibility, Program 2 demonstrates the perils of simply replacing PROC PRINT with PROC REPORT without understanding the underlying differences.

Program 2: Naïve Attempt at REPORT

```
proc report data = SasHelp.Cars(obs = 5);❶
  column Type MPG_City MPG_Highway;❷
run;

proc report data = SasHelp.Cars(obs = 5);
  columns MPG_City MPG_Highway;❸
run;
```

❶ Unlike PROC PRINT, which required the LABEL option to use variable labels, the REPORT procedure automatically uses variable labels if they are available.

❷ The COLUMN statement in PROC REPORT is analogous to the VAR statement in PROC PRINT. It dictates what columns appear in the report and their order.

❸ COLUMN and COLUMNS are aliases, so using either one is recognized as valid syntax by the REPORT procedure.

After glancing at the code, it is understandable to expect to get duplicates of the results found in Output 1. However, an inspection of the results in Output 2 are often surprising for first-time users of PROC REPORT. In particular, the results from the first REPORT procedure match expectations, but the second table is unexpected at first. However, some a review of the documentation will confirm your quick, deductive reasoning – the values printed are the sums of the MPG_City and MPG_Highway variables.

Output 2: Naïve Attempt at REPORT

Type	MPG (City)	MPG (Highway)
SUV	17	23
Sedan	24	31
Sedan	22	29
Sedan	20	28
Sedan	18	24

MPG (City)	MPG (Highway)
101	135

This highlights several of the “flexibility features” mentioned above. First, the layout is not determined by simply printing every value in the selected columns; this results in a table that has one row instead of one row per record. Second, the row shown in the table is not a row in the data set; this row has been derived by PROC REPORT.

No one paper or presentation can convey the nearly endless possibilities provided by PROC REPORT. The following section provides examples that highlight ways to leverage these flexibility features to produce reports well beyond what PROC PRINT could produce. After these examples, we go on to show what can be done with a more complete understanding of PROC REPORT. For further examples and in-depth explanations of the PROC REPORT, readers are encouraged to take advantage of a tutorial session at a SAS conference, use an online training provided by SAS, or review the provided Recommended Reading at the end of this paper. Both of the recommended readings include examples with explanations and opportunities for hands on experience with PROC REPORT.

Report Item Usages

Program 2 introduced one of the first features programmers need to be familiar with in order to use PROC REPORT – the concept of **usages**. Unlike PRINT, REPORT allows you to provide instructions on how the values of a variable should be used and/or created within REPORT. Recall from Output 2 that when excluding the character variable, Type, from the report definition, the report layout changed. Specifically, the resulting report was simply a summary – using the sum – of the numeric variables.

Default Usages

The change in layout described above is because PROC REPORT applies a default usage to each variable in the COLUMN statement.

Properties 1: Default Usages

DISPLAY: Each observation of the variable is printed in its own row in the report. This is the default usage for character variables from the data set.

ANALYSIS: Calculates a statistic for a group of observations from the data set. If only ANALYSIS variables are included, the group consists of all records in the report. The default statistic is SUM. This is the default usage for numeric variables from the data set.

Armed with information on default usages, Output 2 makes more sense. In the first report shown, Type is a character variable and so has the DISPLAY usage. Thus, every observation gets printed in its own row. Now, even though MPG_City and MPG_Highway are numeric and get the default ANALYSIS usage, each observation must be its own row because of the DISPLAY variable, Type. Hence, it appears as if each value of the two MPG variables are printed, when in fact they are summed – but, of course, the sum of a single number is itself, so the report appears to display the data values.

In the second report, omitting Type means there is no longer a DISPLAY variable in the report. With only two numeric variables remaining, the default ANALYSIS usage with its default statistic of SUM now provides the summary for the entire set of observations. After identifying and understanding the issue, providing DISPLAY as the explicit usage solves the problem. Program 3 demonstrates explicit usages to produce the desired reports to mirror those from PROC PRINT shown in Program 1.

Program 3: Introducing Usages

```
proc report data = SasHelp.Cars(obs = 5);
  column Type MPG_City MPG_Highway;
  define Type ❶ / display; ❷
  define MPG_City / display; ❷
  define MPG_Highway / display; ❷
run;

proc report data = SasHelp.Cars(obs = 5);
  column MPG_City MPG_Highway; ❸
  define MPG_City / display; ❸
  define MPG_Highway / display; ❸
run;
```

❶ The DEFINE statement allows you to provide custom instructions for the named report item. A DEFINE statement may only include a single name. If a report item has multiple DEFINE statements associated with it, only the last one is applied.

❷ The DISPLAY keyword applies that usage to each of the variables in this report definition.

❸ It is considered a good programming practice to always include a DEFINE statement with an explicit usage to clearly communicate the desired report structure and how removal of a report item may affect the report structure. Here, removing Type no longer has an effect on the report.

Output 3: Introducing Usages

Type	MPG (City)	MPG (Highway)
SUV	17	23
Sedan	24	31
Sedan	22	29
Sedan	20	28
Sedan	18	24

MPG (City)	MPG (Highway)
17	23
24	31
22	29
20	28
18	24

Additional Options

Of course, PROC REPORT would be pretty boring if these were the only two usages and there were no other options. To support the creation of a variety of report definitions, there are four other possibilities: ACROSS, COMPUTED, GROUP, and ORDER. After looking at examples of each, along with some additional useful options, we summarize these additional usages. Program 4 begins by demonstrating a few basic options in the DEFINE statement.

Program 4: IDs, FORMATs, LABELS, and NOPRINT

```
proc format;❶
  value mpg low - 20 = "Terrible"
            20 <- 35 = "OK"
            35 <- 45 = "Good"
            45 <- high = "Great"
  ;
run;

proc report data = SasHelp.Cars(obs = 5);
  column Type MPG_City MPG_Highway;
  define Type / display id; ❷
  define MPG_City / display format = mpg. ❸ 'City Mileage' ❹;
  define MPG_Highway / display noprint ❺;
run;
```

❶ This step creates a numeric format, mpg, used here and throughout the examples to format the mileage variables, MPG_City and MPG_Highway.

❷ The ID option is analogous to the ID statement in PROC PRINT - designating the report item as an ID variable causing the column to appear at the beginning of every row associated with the record. This connects records when a report definition is too wide to fit on a single page.

❸ In addition to the FORMAT statement, PROC REPORT allows you to define a format for a report item directly in the DEFINE statement. This is necessary since the FORMAT (or ATTRIB)

statement only applies to data set variables, but report items may not come from the data set - in those cases the FORMAT= option allows you to provide a temporary format.

❹ Temporary labels are included by placing the label in a quoted string in the DEFINE statement. Just like with formats, PROC REPORT supports the ATTRIB (or LABEL) statement, but that only applies to data set variables. This allows for custom labels for any report item.

❺ The NOPRINT option suppresses the report item when creating the output. Note that while it may seem that DISPLAY and NOPRINT are contradictory, DISPLAY is about how the report item should contribute to the layout of the report while NOPRINT is about whether it should be printed.

Output 4 shows that MPG_City has a custom label and format and that the MPG_Highway column is not printed. While this report definition is quite narrow, if the report were to break across multiple pages, all rows would be prepended with the Type column due to the ID option. Note that all columns to the left of this report item in the COLUMN statement will be treated as ID variables as well, though in this case Type was the first variable in the COLUMN statement. While this is different than the behavior in PROC PRINT – which requires you to list all ID variables in the ID statement – good programming practice still dictates including the ID option in each of the appropriate DEFINE statements so that removal of one report item does not affect the status of other report items as ID variables.

Output 4: IDs, FORMATs, LABELS, and NOPRINT

Type	City Mileage
SUV	Terrible
Sedan	OK
Sedan	OK
Sedan	Terrible
Sedan	Terrible

Additional Usages

Beyond the default usages of DISPLAY and ANALYSIS, the ORDER usage is one of the most commonly used for its ability to help arrange the rows in a report definition. In fact, the effects of the ORDER usage are very similar to those of PROC SORT and it uses hierarchical sorts in much the same way – if you want to sort by multiple variables, their order in the COLUMN statement dictates the hierarchy of the sort. Program 5 demonstrates a simple application of the ORDER usage and compares its affect on the layout of the report to the DISPLAY usage.

Program 5: Intro to ORDER Usage

```
proc report data = SasHelp.Cars(obs = 5);
  column Type MPG_City MPG_Highway;
  define Type / order ❶ descending ❷;
  define MPG_City / order order = internal format = mpg.; ❸
  define MPG_Highway / display;
run;

proc report data = SasHelp.Cars(obs = 5);
  column Type MPG_City MPG_Highway;
  define Type / order;
  define MPG_City / order format = mpg.;
  define MPG_Highway / order format = mpg.;
run;
```

❶ The keyword ORDER is sufficient to apply the required usage. Depending on what SAS product you are using, be aware the editor color coding may not render correctly. (E.g., in SAS 9.4 [windowing environment] the keyword remains black.)

❷ As with PROC SORT, the default is to sort values in ascending sequence. The DESCENDING keyword reverses the sort order.

❸ Unlike other procedures in SAS, the current default is to sort based on the formatted values. The same options available elsewhere in SAS (e.g., PROC FREQ) allow you to modify this by specifying the sorting criteria as INTERNAL|FORMATTED|FREQ|DATA. *Because of the inconsistency with other procedures, it is good programming practice - and recommended by the SAS documentation - to explicitly declare a sort criteria in case the REPORT procedure is updated to align with other procedures.*

The first report in Output 5 demonstrates that Type has been sorted in descending order; recall, sorting is based on ASCII and so 'Sedan' would normally come *after* 'SUV' since the capital 'U' would be sorted before the 'e' since it is lower case. Further, note that repeated values of 'Sedan' are not printed. This is a designed feature of ORDER variables – they suppress repeated printing of the same value. The use of ORDER = INTERNAL now forces the formatted value "Terrible" to appear before the formatted value "OK" because the values associated with the groups – below 20 and from 20 to 35, respectively – are arranged in ascending order based precisely on those values.

Output 5: Intro to ORDER Usage

Type	MPG (City)	MPG (Highway)
Sedan	Terrible	28
		24
	OK	31
		29
SUV	Terrible	23

Type	MPG (City)	MPG (Highway)
SUV	Terrible	OK
Sedan	OK	OK
	Terrible	OK

Output 5 includes a second report where all variables are declared with the ORDER usage, with sorting based on formatted values, and where the format is applied to both mileage columns. In this case, blank lines appear because every value in the record is a duplicate of the preceding record's values. This report is included to emphasize a key point regarding the ORDER usage – it produces one row per observation just like the DISPLAY usage.

Just as the ORDER usage is similar to PROC SORT, another popular PROC – PROC TRANSPOSE – is similar to the ACROSS usage because they both take values of a variable and turn them into columns in your output. The ACROSS usage is a prime example of why usages are often described as affecting the layout of your report since it turns rows into columns. Program 6 shows a simple application of the ACROSS usage as well as an incorrect attempt to combine it in a report with other usages.

Program 6: Intro to ACROSS Usage

```
proc report data = SasHelp.Cars;
  column Type;
  define Type / across ❶ order = freq ❷ descending ❷;
run;

proc report data = SasHelp.Cars(obs = 5);
  column Type Mpg_City;
  define Type / across descending;
  define Mpg_City / display; ❸
run;
```

❶ As with previous usages, the keyword – ACROSS – declares the usage for this report item.

❷ Because the variable values are used to create new columns, you can declare the order in which those columns are created using the same options as with the ORDER usage. Here, ORDER=FREQ puts the newly-created columns in order based on frequency and DESCENDING puts them in descending sequence.

❸ Including a DISPLAY (or ORDER) variable forces each of the five included observations to have their own row in the report, interfering with the ability of ACROSS to summarize the data.

As Output 6 shows, the ACROSS usage does exactly what it sounds like – it spreads out the values of the variables across the report to create new columns. This is our first example of the flexibility feature that allows columns to exist in the report that are not in the data set. In this case, by using a data variable – Type – in the COLUMN statement but giving it the ACROSS usage. By default, the value of the cells under an ACROSS variable is the frequency of that value in the data set; for example, there are 262 sedans and 3 hybrids in our data set. Other summary values are possible, but require combining the ACROSS usage with the ANALYSIS usage.

As a prelude to more examples about the mixing of usages, Output 6 also shows the effects of a poor combination: an ACROSS usage with a DISPLAY usage. Because of DISPLAY's requirement that each observation get its own row, the ACROSS usage can only provide frequencies within a row – producing not only an uninteresting report, but one where the information could be conveyed in a much more compact and professional manner by changing the report definition.

Output 6: Intro to ACROSS Usage

Type					
Sedan	SUV	Sports	Wagon	Truck	Hybrid
262	60	49	30	24	3

Type		
Sedan	SUV	MPG (City)
.	1	17
1	.	24
1	.	22
1	.	20
1	.	18

In Properties 1, the ANALYSIS usage was described as operating on groups with the default of summarizing the entire data set. If a DISPLAY variable was present, it treated each row as a group with a single item. ACROSS is performing much the same action – counting the frequency within the data set until forced to treat each row as its own group. To allow you to define custom groups based on

variables in the data set, PROC REPORT provides the GROUP usage as demonstrated in Program 7 which also shows how to request various statistics besides SUM from the ANALYSIS usage.

Program 7: Intro to GROUP Usage

```
proc report data = SasHelp.Cars;
  column Type MPG_City MPG_Highway;
  define Type / group descending order = freq; ❶
  define MPG_City / analysis mean format = 4.1 'Mean City MPG'; ❷
  define MPG_Highway / n 'Record Count'; ❸
run;
```

❶ The GROUP usage defines what observations should be collapsed into a single row. This usage also sorts the groups, so the usual options related to order are available.

❷ When declaring an ANALYSIS usage, you can also specify a keyword to select a single statistic. Most statistics keywords available in procedures like MEANS and UNIVARIATE are valid in PROC REPORT. (E.g., CLM is not valid since it calculates a confidence interval which is not a single statistic.)

❸ When specifying a statistic, such as N, it is not necessary to also specify ANALYSIS as the usage. However, it is considered good programming practice to explicitly state the usage, and statistic when applicable, since it makes the code more accessible for less experienced programmers. Thus, the approach shown in this DEFINE statement is not considered a good practice.

The output from Program 7 shows that only one row exists for each group, i.e., each level of Type. Within those groups, the ANALYSIS usage of the remaining report items computes the requested statistics, mean and count, for the respective variables, MPG_City and MPG_Highway.

Output 7: Intro to GROUP Usage

Type	Mean City MPG	Record Count
Sedan	21.1	262
SUV	16.1	60
Sports	18.4	49
Wagon	21.1	30
Truck	16.5	24
Hybrid	55.0	3

Programmers new to using PROC REPORT sometimes confuse the ORDER and GROUP usages because they both sort rows by default. However, as comparing Outputs 5 and 7 show, and as the names imply, the ORDER usage only arranges the rows and does not collapse the records with common values into a group of records. The GROUP usage does just that – it collects observations with the same value and treats them as a single group. Be aware that using DISPLAY or ORDER would prevent the creation of groups since they require one line be printed for each observation. This is similar to the interferance between DISPLAY and ACROSS shown in Program 6.

COMPUTED Usage and COMPUTE Blocks

The sixth and final usage, COMPUTED, allows you to create new variables within the PROC REPORT step. As Program 8 shows, computing a new variable in PROC REPORT seems straightforward at first, but does require some planning.

Program 8: Intro to COMPUTED Usage

```
proc report data = SasHelp.Cars(obs = 5);
  column Type MPG_City MPG_Highway MPG_Ratio;❶
  define Type / display ;
  define MPG_City / display;
  define MPG_Highway / display;
  define MPG_Ratio / computed ❷ 'City/Highway';
  compute ❸ MPG_Ratio ❹;
    MPG_Ratio = MPG_City/MPG_Highway; ❺
  endcomp; ❻
run;

proc report data = SasHelp.Cars(obs = 5) split = "*"; ❼
  column Type MPG_Ratio ❽ MPG_City MPG_Highway ;
  define Type / display ;
  define MPG_City / display;
  define MPG_Highway / display;
  define MPG_Ratio / computed 'City/Highway';
  compute MPG_Ratio;
    MPG_Ratio = MPG_City/MPG_Highway;
  endcomp;
run;
```

❶ The variable you plan to derive must still appear in the COLUMN statement. Here, MPG_Ratio is added at the end so that it will be the last column in the output.

❷ The COMPUTED usage tells PROC REPORT that this report item is not a data set variable and to take the value from the provided computation.

❸ A COMPUTED variable must be created in a COMPUTE block. Within a COMPUTE block, PROC REPORT activates the DATA step compiler so most DATA step elements are legal here as well. This includes do loops, conditional logic, arrays but excludes data processing statements like SET or INPUT. While SAS documentation indicates that all DATA step functions are available, some functions are not legal within a COMPUTE block.

❹ Each COMPUTE block must have a *location* that instructs PROC REPORT when to execute the COMPUTE block. In this case, MPG_Ratio is listed as the location, so in each row, the COMPUTE block is executed once PROC REPORT reaches the MPG_Ratio column.

❺ Within the COMPUTE block, place the instructions for deriving the new variable. Note, this is just an assignment statement exactly like you would use in a DATA step.

❻ Every COMPUTE block closes with an ENDCOMP statement.

❼ While the SPLIT= option is unrelated to computing, it is necessary to keep the label 'City/Highway' from breaking because the default split character is the forward slash. Here '*' is chosen since it does not appear in any of the column headers.

❽ MPG_Ratio has been moved earlier in the report, now appearing as the second column.

The output from Program 8 shows the desired results in the first report – the MPG_Ratio variable has been correctly computed and displayed. Recall, this is not a data set variable, so any formats or labels must be applied in the DEFINE statement since MPG_Ratio cannot be placed in attribute-related statements like LABEL, FORMAT, or ATTRIB. However, the second report reveals an unwanted result – MPG_Ratio is always missing. In fact, the log for this program includes an unwelcome note: "Missing values were generated as a result of performing an operation on missing values."

Output 8: Intro to COMPUTED Usage

Type	MPG (City)	MPG (Highway)	City Highway
SUV	17	23	0.7391304
Sedan	24	31	0.7741935
Sedan	22	29	0.7586207
Sedan	20	28	0.7142857
Sedan	18	24	0.75

Type	City/Highway	MPG (City)	MPG (Highway)
SUV	.	17	23
Sedan	.	24	31
Sedan	.	22	29
Sedan	.	20	28
Sedan	.	18	24

Why did moving the location of the column in the report definition result in missing values? PROC REPORT builds the output one row at a time, moving cell by cell from left to right across the columns. Thus, just as in the DATA step where you cannot use a variable in a computation *before* its value is loaded into the program data vector (PDV), you cannot use a variable in a COMPUTE block if it is to the *left* of the COMPUTE location. In this case, MPG_Highway is the compute location, so only Type is available when the COMPUTE block is processed, since it is the only variable to the left in the COLUMN statement. Of course, you don't want your computations to dictate your column order, so Program 9 demonstrates two methods for achieving the desired result.

Program 9: Selecting the COMPUTE Location

```
proc report data = SasHelp.Cars(obs = 5) split = "*";
  column Type MPG_Ratio MPG_City MPG_Highway;
  define Type / display ;
  define MPG_City / display;
  define MPG_Highway / display;
  define MPG_Ratio / computed 'City/Highway' format = 4.2;
  compute MPG_Highway; ❶
    MPG_Ratio = MPG_City/MPG_Highway;
  endcomp;
run;

proc report data = SasHelp.Cars(obs = 5) split = "*";
  column Type MPG_Ratio MPG_City MPG_Highway dummy ❷;
  define Type        / display ;
  define MPG_City    / display;
  define MPG_Highway / display;
  define MPG_Ratio   / computed 'City/Highway' format = 4.2;
  define Dummy       / computed noprint; ❸
  compute dummy; ❹
    MPG_Ratio = MPG_City/MPG_Highway;
  endcomp;
run;
```

❶ The COMPUTE location has been updated to be MPG_Highway because it is the last report item in the COMPUTE block.

❷ A new report item, Dummy, is added to the COLUMN statement.

❸ The new item, Dummy, is a COMPUTED variable that will be hidden due to the NOPRINT option.

❹ The COMPUTE block is now pointed to the Dummy column, ensuring the computation occurs in the last column in the report definition.

Both PROC REPORT steps shown in Program 9 produce the table shown in Output 9. In this case, both solutions are equally effective; however, there are times where the dummy approach is preferred even though it requires more code. Both solutions work because the COMPUTE location does not have to be the same column that is being derived. (Even though SAS documentation states the COMPUTE block must be associated with the variable, that is false.) Moreover, a COMPUTE block may contain the definitions for multiple new variables and include other customizations of the report. However, A report item cannot be the location of multiple COMPUTE blocks. As a result, in more complex reports you may want to use the dummy approach so that the report items of interest to you are still available as the target for other COMPUTE blocks.

Output 9: Selecting the COMPUTE Location

Type	City/Highway	MPG (City)	MPG (Highway)
SUV	0.74	17	23
Sedan	0.77	24	31
Sedan	0.76	22	29
Sedan	0.71	20	28
Sedan	0.75	18	24

Summarizing Usages

The previous subsections demonstrated the six usages and provide examples and some common pitfalls new programmers face when applying the usages. The list below summarizes all six usages for easy reference. While there are many more interactions between the usages and features to help you get the most out of them, there are also other aspects of PROC REPORT that make it the valuable and flexible report-generating tool that it is. The sections that follow provide some examples of the power of PROC REPORT.

Properties 2: Summary of Usage Descriptions

ACROSS: Creates a new column for each formatted unique value of of the ACROSS variable. Displays the frequency of that value by default, but can be modified by applying other usages to additional report items.

ANALYSIS: Calculates a statistic for a group of observations from the data set. If no explicit or implicit grouping is done, then the statistic is calculated based on all values. The default statistic is SUM. This is the default usage for numeric variables from the data set.

COMPUTED: Variables that do not come in with the data set but which are defined by you in the report. COMPUTED variables must be created in a COMPUTE block.

DISPLAY: Each observation of the variable is printed in its own row in the report. This is the default usage for character variables from the data set.

GROUP: Collapses observations with the same formatted value of the GROUP variable into a single group. If multiple GROUP variables are present, then rows in the resulting table represent unique combinations of levels of the GROUP variables. Sequences the groups based on sort-related options such as DESCENDING and ORDER=. Suppresses repeated printing of the GROUP value within the group.

ORDER: Each observation of the variable is assigned to its own row in the resulting table. Sequences the rows of the report based on either the default sort criteria or the sort-related options such as DESCENDING and ORDER= that you provide. Suppresses repeated printing of the ORDER value.

An Example of the Power of PROC REPORT

While understanding the usages individually is a cornerstone of controlling the layout of a report definition in PROC REPORT, being able to implement the usages in complementary ways allows you to greatly expand the reports you can produce. In the Introduction we said that most of the flexibility in PROC REPORT could be traced to one of four features: layout options, un-linking columns from data set variables, un-linking rows from data set observations, and expanded aesthetics.

The usages alone provide examples of some various layouts, the ability to create new columns either via usage (e.g., ACROSS or COMPUTED), and the ability to display rows that are not in the original data set via usages (e.g., GROUP and/or ANALYSIS). However, even with all their applications, usages alone cannot unlock the full flexibility of PROC REPORT. Through additional statements (e.g., BREAK, RBREAK, LINE, and CALL DEFINE) and options (e.g., STYLE) you can produce highly customized reports that provide further examples of the first three flexibility features described in the Introduction as well as examples of changing the report aesthetics.

Covering each of these additional features in detail is beyond the scope of this paper, but it is worthwhile to see an example of what you can produce when these features are used together. Program 10 demonstrates how to use many of these programming elements simultaneously. Due to the length of the program and the visual nature of the report, the results are presented first to aide with relating the callouts in the code to the relevant portions of the output.

Output 10: Styling a Report

Origin	DriveTrain	Count	%	Mean	Median
USA	All	22	15.0%	$33,972	$32,448
	Front	90	61.2%	$25,095	$23,115
	Rear	35	23.8%	$33,301	$30,835
		147	34.3%	$28,377	$25,520
Europe	All	36	29.3%	$45,103	$39,445
	Front	37	30.1%	$34,980	$34,845
	Rear	50	40.7%	*$60,581*	*$52,243*
		123	28.7%	*$48,350*	*$40,590*
Asia	All	34	21.5%	$28,982	$26,898
	Front	99	62.7%	$20,687	$19,560
	Rear	25	15.8%	$35,028	$31,045
		158	36.9%	$24,741	$23,033
		428		*$32,775*	*$27,635*

Mean and median differ by more than 15%

As you can see from Output 10, we have added numerous customizations to this report definiton. The header row is styled using a specific shade of blue, while rows corresponding to groups of records banded using progressive shades of yellow. Furthermore, summary rows have been added to provide additional summaries of Origin and the overall data set and those rows are given their own shade of blue. In addition, certain cells have been flagged using a red, italic font and a footnote is attached to

the bottom of the report to explain the flag.

In addition to the usages shown earlier – Origin and DriveTrain are GROUP variables; Count, Mean, and Median are ANALYSIS variables associated with MSRP, and % is a COMPUTED variable – additional code beyond simple aesthetics are clearly involved since multiple statistics are presented for a single variable. The callouts following the program provide a high-level overview of the code responsible for this report.

Program 10: Styling a Report

```
%let FlagStyle = color = cxe41a1c
                 fontstyle=italic
                 fontweight=extra_bold;

proc report data = SasHelp.Cars
            style(lines) = [background = grayEE] ❶
            style(summary) = [background = cx8DD3C7] ❶
            style(header) = [background = cx80B1D3] ❶
            style(column) = [fontfamily = "Arial"]; ❶
  column Origin DriveTrain MSRP=num pct MSRP=Mean MSRP=Median; ❷
  define Origin / group descending;
  define DriveTrain / group;
  define Num / analysis n "Count" format = F4.; ❸
  define Pct / computed "%" format = misspct.; ❹
  define Mean / analysis Mean "Mean"; ❸
  define Median / analysis Median "Median"; ❸

  rbreak after / summarize; ❺
  break after Origin / summarize suppress; ❺

  compute before; ❻
    Tot = num;
  endcomp;

  compute before Origin; ❻
    grpTot = num;
  endcomp;

  compute Pct;❼
    if missing(_break_) then pct = num/grpTot;
      else if upcase(_break_) eq "ORIGIN" then pct = grpTot/Tot;
      else if upcase(_break_) eq "_RBREAK_" then call missing(pct);
  endcomp;
```

Program 10 (cont.): Styling a Report

```
*Continued PROC REPORT code;
  compute Median;
    ratio = Mean/Median;
    if Ratio <0.85 or Ratio > 1.15 then do;
        call define(_col_, "style", "style=[&FlagStyle]");❽
        call define("Mean", "style", "style=[&FlagStyle]");❽
      end;

    if not missing(_break_) then c=0; ❾
    if not missing(DriveTrain) then c+1; ❾
    if not missing(DriveTrain) and mod(c,3) eq 1 then
        call define(_row_,"style", "style=[background=cxFFFFCC]"); ❾
     else if not missing(DriveTrain) and mod(c,3) eq 2 then
        call define(_row_,"style", "style=[background=cxFFEDA0]"); ❾
     else if not missing(DriveTrain) and mod(c,3) eq 0 then
        call define(_row_,"style", "style=[background=cxFED976]"); ❾
  endcomp;

  compute after / style = [&FlagStyle just=right]; ❿
    line "Mean and median differ by more than 15%"; ❿
  endcomp;
run;
```

❶ The STYLE option allows you to adjust the aesthetics associated with specific regions of the report. Valid syntax places the locations inside parentheses but multiple locations, e.g., STYLE(LINES SUMMARY), is valid syntax if you want to apply the same style to multiple locations. See the SAS documentation or a recommended reading for a discussion of the various locations and the style attributes that you can modify. Here, we modify background colors and font family.

❷ If you want to use a data set variable in multiple column definitions, you can employ *aliases*. Here we alias MSRP three times: once as Num, once as Mean, and once as Median. Alias names must follow SAS naming conventions and there are potential limitations on their use in PROC REPORT. See the SAS documentation and/or the Recommended Readings for further details.

❸ You can reference an aliased variable in a DEFINE statement just as you can with a data set variable.

❹ The MISSPCT format is a user-defined format (not shown in the program) that blanks out any missing values and uses PERCENTN for the non-missing values. Details are in the full code provided online.

❺ The RBREAK and BREAK statements create rows using all records from the table (RBREAK) or from a level of a GROUP or ORDER report item (BREAK). The SUMMARIZE keyword places summaries from ANALYSIS report items into the BREAK/RBREAK row using the same statistic defined for that column. The AFTER keyword provides the location of the summary row: after the report for RBREAK AFTER and after the level of origin for BREAK AFTER Origin. The SUPPRESS keyword prevents printing the value of the break variable, which in this case is Origin.

❻ Like BREAK and RBREAK statements, COMPUTE blocks accept the BEFORE and AFTER keywords to indicate when REPORT should carry out the statement's actions. COMPUTE BEFORE instructs PROC REPORT to run the COMPUTE block before the report is generated. (Similar to how RBREAK BEFORE creates a summary row before the other rows in the report.) Similarly, COMPUTE BEFORE Origin directs the REPORT procedure to execute the COMPUTE block before each new value of ORIGIN. In both COMPUTE blocks, the value of Num – which contains the count – is copied into a variable we can use in the COMPUTE blocks.

❼ The Pct variable was given a COMPUTED usage, so this COMPUTE block derives its value depending on the value of the automatic variable _BREAK_ created by PROC REPORT. This variable indicates when a row is generated by a BREAK or RBREAK statement. Missing values indicate the row is not a summary row, values of _RBREAK_ indicate the row is generated by an RBREAK statement, and the name of a report item (e.g., ORIGIN) indicates the row is generated by a BREAK statement applied to that report item.

❽ While the DEFINE statement applies to an entire column, the CALL DEFINE statement is available inside COMPUTE blocks to allow you to apply a change to other parts of a report. The _COL_ keyword refers to the report item in the COMPUTE statement (Median, in this case) and is not placed in quotes. The quoted string “Mean” refers to the report item Mean. In both cases, the macro variable FlagStyle defined prior to the REPORT step is used to apply a consistent set of style changes to cells that meet the stated criteria: Ratio below 0.85 or above 1.15.

❾ A counter variable, c, is created to keep track of the rows within each level of Origin. Conditional logic is used to ensure the correct value is set at the beginning of each new level of Origin and then incremented for each new level of DriveTrain. Finally, the _ROW_ keyword is used in the CALL DEFINE statements to apply a change to an entire row.

❿ This COMPUTE block assigns the previously defined FlagStyle settings to the output created by the block. The LINE statement simply adds a line to the report with whatever contents you specify. In this case, COMPUTE AFTER places the line at the end of the report so it acts as a styled footnote included in the report.

Conclusion

The code in Program 10 is only one example of what can be done with PROC REPORT. From simple reports to dynamic styles to inline graphs such as sparkline or similar charts to hyperlinked reports that connect a set of documents into a larger suite of navigable reports – it is all possible with PROC REPORT. Of course, as Program 10 also shows, each of those tweaks and modifications requires understanding the various syntax available. In fact, as with most code you write, there are multiple ways to achieve this same report such as using code that does not rely on aliasing your variables in the COLUMN statement.

The introduction to usages provided in this paper are not sufficient to produce such a report, but producing such a report is not possible without understanding the basics of usages and the various options available to you in the DEFINE statement. For more hands on experience with PROC REPORT, consider attending a conference tutorial, taking advantage of a SAS training, or reading a general purpose SAS book or even a book dedicated to PROC REPORT. Program 10 is presented here to show the possibilities and hopefully convince you to use PROC REPORT for all your detail and summary reports in the future.

Recommended Reading

- Blum, James and Jonathan Duggins. *Fundamentals of Programming in SAS: A Case Studies Approach.* SAS, 2019.
- Eslinger, Jane. *The SAS Programmer's PROC REPORT Handbook: Basic to Advanced Reporting Techniques.* SAS, 2016.

Contact Information

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

Jonathan W. Duggins

North Carolina State University

jwduggin@ncsu.edu

https://jonathanduggins.com/