

The Fantastic Four: Running Your Report Using the TABULATE, TRANSPOSE, REPORT, or SQL Procedure

Joshua M. Horstman, Nested Loop Consulting, Indianapolis, IN, USA

ABSTRACT

Like all skilled tradespeople, SAS® programmers have many tools at their disposal. Part of their expertise lies in knowing when to use each tool. In this paper, we use a simple example to compare several common approaches to generating the requested report: the TABULATE, TRANSPOSE, REPORT, and SQL procedures. We investigate the advantages and disadvantages of each method and consider when applying it might make sense. A variety of factors are examined, including the simplicity, reusability, and extensibility of the code in addition to the opportunities that each method provides for customizing and styling the output. The intended audience is beginning to intermediate SAS programmers.

INTRODUCTION

SAS software provides a wealth of tools for data manipulation and analysis. New procedures and options are being added constantly. The choices can be mind-boggling for the new programmer.

In this paper, we use a simple reporting task as a vehicle for exploring some of the most common SAS procedures. Our purpose is not to single out one method as being universally superior. Rather, we explore how each method can be used to solve a basic programming problem.

We begin by presenting the data and describing the task to be performed. Next, we present four detailed ways to accomplish the task. The first uses a three-step process combining the FREQ, TRANSPOSE, and REPORT procedures. The second invokes the REPORT procedure alone. The third relies on the TABULATE procedure, while the last method employs the SQL procedure. Finally, we end with a discussion comparing and contrasting the methods.

THE SETUP

Before we dig into the details of each of the methods to be presented, let us begin by looking at the data we have and the report we wish to produce.

THE DATA

In this paper, we are working with adverse event data collected during the course of a clinical trial for a pharmaceutical product. Adverse events are negative effects experienced by patients during the course of the trial, whether or not they are related to the drug being studied.

Adverse event data sets in real clinical trials include many variables. However, for the purposes of this example, we narrow the data set to the following four variables:

- SUBJID – Subject identifier: a unique number assigned to each subject
- TRTGRP – Treatment group: the treatment assigned to a subject (Drug A” or “Drug B”)
- AETERM – Standardized description of the event (e.g. “Headache”, “Nausea”)
- AESEV – Severity of the event on a scale from 1 (Mild) to 4 (Life-Threatening)

Each record in our data set need not be unique. A given subject may experience the same adverse event with the same severity more than once during the course of the trial. In a real-world adverse events data set, these might be distinguished by other variables such as the start date of the event or another unique identifier. We omit such variables here as they are not pertinent to our discussion.

Also, note that the adverse event term (AETERM) is standardized. In other words, it is selected from a pre-defined list of allowable terms. This makes it relatively easy for us to summarize events by grouping those with like terms together. Such a task might be greatly complicated if this variable were free text.

The AE data set contains 80 records. The first few are shown below in Figure 1.

SUBJID	TRTGRP	AETERM	AESEV
1	Drug A	Hypercalcemia	2
1	Drug A	Pyrexia	2
1	Drug A	Abdominal Pain	2
1	Drug A	Nausea	1
1	Drug A	Fatigue	2
2	Drug B	Headache	2
2	Drug A	Headache	1
2	Drug A	Hypotension	1
...

Figure 1. First few rows of the AE data set

THE TASK

Our goal is to create a summary table displaying the number of events for each adverse event term occurring at each severity for each treatment group. We wish to have one row per adverse event term with separate columns for each combination of severity and treatment group as shown below in Figure 2.

Adverse Event	Drug A				Drug B			
	Mild	Moderate	Severe	Life-Threatening	Mild	Moderate	Severe	Life-Threatening
Abdominal Pain	0	1	0	0	1	1	0	0
Anemia	0	3	0	0	3	1	1	0
Constipation	1	0	1	0	1	0	0	0
Diarrhea	0	1	0	0	0	0	0	0
Dizziness	1	0	0	0	0	2	0	0
Dyspepsia	0	1	0	0	1	0	1	0
Epistaxis	1	0	1	0	0	0	2	0
Fatigue	1	2	0	0	1	0	0	0
Headache	2	0	1	0	1	1	2	0
Hyperbilirubinemia	0	4	1	0	0	3	1	0
Hypercalcemia	0	3	0	0	1	3	0	0
Hyponatremia	0	1	0	0	1	0	0	0
Hypotension	1	2	1	0	1	0	0	0
Musculoskeletal pain	2	4	0	0	0	0	0	0
Nausea	2	0	0	0	0	1	0	0
Pyrexia	0	1	2	0	0	1	0	0
Tachycardia	0	0	0	0	0	0	1	1
Vomiting	0	2	0	0	2	0	2	0

Figure 2. Desired output

Our output will be written to an external file using the ODS RTF destination. Note that the severity levels in the column headers have been replaced with text descriptions not found in the data set. These are applied using the following format:

```
proc format;  
  value sev  
    1 = 'Mild'  
    2 = 'Moderate'  
    3 = 'Severe'  
    4 = 'Life-Threatening'  
  ;  
run;
```

METHOD #1: PROCS FREQ AND TRANSPOSE

The first method we use to create the desired report is a three-step process:

1. Use the FREQ procedure to produce frequency counts.
2. Use the TRANSPOSE procedure to rearrange the frequency counts in the desired manner.
3. Produce the printed output using the REPORT procedure.

STEP 1: PROC FREQ

To construct our summary table, we need a frequency count for each adverse event term for each treatment group for each severity level. Thus, the three variables which appear on the TABLES statement in PROC FREQ are AETERM, TRTGRP, and AESEV. Since we need to perform additional manipulation of the frequency counts before they are written to the output, we use the OUT= option on the TABLES statement to create an output data set and the NOPRINT option on the PROC FREQ statement to suppress the printed output.

In addition, we use the SPARSE option to request PROC FREQ include zero counts for those combinations of TRTGRP, AETERM, and AESEV that do not appear in the data. Had there been levels of one of those variables that did not appear in the data at all (for example, if we had no severity 4 events at all), we would need more complex logic to ensure that those levels still appear in the output. Refer to Phillips and Klein (2005) or Gupta (2015).

Thus, we have the following PROC FREQ code:

```
proc freq data=ae noprint;  
  tables aeterm*trtgrp*asev / out=freqout sparse;  
run;
```

This code produces the output data set shown below in Figure 3. Note that the values of the variable COUNT are the frequency counts we want for our table. The FREQ procedure also create the variable PERCENT which we don't need for the task at hand.

AETERM	TRTGRP	AESEV	COUNT	PERCENT
Abdominal Pain	Drug A	1	0	0
Abdominal Pain	Drug A	2	1	1.25
Abdominal Pain	Drug A	3	0	0
Abdominal Pain	Drug A	4	0	0
Abdominal Pain	Drug B	1	1	1.25
Abdominal Pain	Drug B	2	1	1.25
Abdominal Pain	Drug B	3	0	0
Abdominal Pain	Drug B	4	0	0
Anemia	Drug A	1	0	0
Anemia	Drug A	2	3	3.75
Anemia	Drug A	3	0	0
Anemia	Drug A	4	0	0
...

Figure 3. PROC FREQ output data set (FREQOUT)

STEP 2: PROC TRANSPOSE

The output data set from the FREQ procedure contains all of the frequency counts in a single column. For our report, we need a separate column for each treatment group (Drug A and Drug B) for each severity (1 through 4).

This is easily accomplished using PROC TRANSPOSE. We want the result to have one row per adverse event term, so we list the variable AETERM on the BY statement. In addition, we want a separate column for each combination of TRTGRP and AESEV, so we list these two variables on the ID statement. Finally, we want the values of COUNT to appear in these columns, so we place COUNT on the VAR statement.

Note that having multiple variables on the ID statement was not allowed until version 9.2 of SAS. To perform this transpose with a version of SAS prior to 9.2, more complex logic must be used to invoke the TRANSPOSE procedure twice with an intermediary data step. This approach is described in detail by Horstman (2014).

```
proc transpose data=freqout out=transout;
  by aeterm;
  var count;
  id trtgrp aesev;
run;
```

This code produces the output data set shown below in Figure 4. Note that PROC TRANSPOSE concatenates the values of TRTGRP and AESEV to generate variable names for each column. Blank spaces are replaced by underscore characters since a space is not valid in a SAS variable name. For example, the column associated with “Drug A” and severity 1 is named DRUG_A1.

AETERM	DRUG_ A1	DRUG_ A2	DRUG_ A3	DRUG_ A4	DRUG_ B1	DRUG_ B2	DRUG_ B3	DRUG_ B4
Abdominal Pain	0	1	0	0	1	1	0	0
Anemia	0	3	0	0	3	1	1	0
Constipation	1	0	1	0	1	0	0	0
Diarrhea	0	1	0	0	0	0	0	0
Dizziness	1	0	0	0	0	2	0	0
Dyspepsia	0	1	0	0	1	0	1	0
Epistaxis	1	0	1	0	0	0	2	0
Fatigue	1	2	0	0	1	0	0	0
Headache	2	0	1	0	1	1	2	0
Hyperbilirubinemia	0	4	1	0	0	3	1	0
Hypercalcemia	0	3	0	0	1	3	0	0
Hyponatremia	0	1	0	0	1	0	0	0
Hypotension	1	2	1	0	1	0	0	0
Musculoskeletal pain	2	4	0	0	0	0	0	0
Nausea	2	0	0	0	0	1	0	0
Pyrexia	0	1	2	0	0	1	0	0
Tachycardia	0	0	0	0	0	0	1	1
Vomiting	0	2	0	0	2	0	2	0

Figure 4. PROC TRANSPOSE output data set (TRANSOUT)

STEP 3: PROC REPORT

The frequency counts in the TRANSOUT data set are arranged exactly the way we want them in our report output. Now we just need to generate the report with appropriate formatting. Since we want to have row headers that span multiple rows, the PRINT procedure won't be sufficient for the task. We need the much more powerful REPORT procedure.

On the COLUMN statement, we list the variables we want in our output in the order we want them. We are able to add spanning headers to print the names of the treatment groups above the headers of the four corresponding severity columns. Unfortunately, we have to hard-code these treatment group titles directly into our code.

We can use the colon notation to abbreviate the list of variable names on the COLUMN statement. Entering "drug_a:" is the same as listing all variables in the data set that begin with "drug_a". In our case, that matches all four of the severity columns corresponding with that treatment group.

We also include a DEFINE statement for each variable. For the eight columns containing frequency counts, we use the CENTER option on the DEFINE statement to center the counts within each column. We also specify that the labels associated with each severity level (e.g. "Mild", "Moderate", etc.) be printed as column headers, but again these must be hard-coded.

In the DEFINE statement for the AETERM variable, we need not specify a justification option since we want the values to be left-justified, which is the default. However, column headers are centered by

default, so we use a style element to left justify the corresponding header. In addition, we remove the extraneous border line just above the header by setting its color to match the background color, a medium grey which happens to have a hexadecimal color code of BBBB BB.

We also make use of style elements on the PROC REPORT statement itself. We select the font face and font size desired for each element of the report, both the column headers and the column contents themselves.

Finally, the entire call to PROC REPORT is surrounded with the ODS RTF statements needed to direct the output to an RTF file.

```
ods rtf file='myfile.rtf';

proc report data=transout nowindows
  style(header)=[font_face="Arial" font_size=9pt]
  style(column)=[font_face="Arial" font_size=9pt]
;

column aeterm ("Drug A" drug_a:) ("Drug B" drug_b:);

define aeterm / "Adverse Event"
  style(header)=[just=1 bordertopcolor=cxBBBBBB];
define drug_a1 / center "Mild";
define drug_a2 / center "Moderate";
define drug_a3 / center "Severe";
define drug_a4 / center "Life-Threatening";
define drug_b1 / center "Mild";
define drug_b2 / center "Moderate";
define drug_b3 / center "Severe";
define drug_b4 / center "Life-Threatening";

run;

ods rtf close;
```

The resulting output, shown on the next page in Figure 5, is exactly the report requested.

METHOD #2: PROC REPORT ALL BY ITSELF

While the method above is straightforward enough, we can actually produce the exact same report using the REPORT procedure all by itself. We use ANALYSIS and ACROSS variables to replicate the work performed by the FREQ and TRANSPOSE procedures, respectively.

First, we define the AETERM column as a GROUP variable. This ensures that the report contains only one row per value of AETERM. Next, we define both the TRTGRP and AESEV columns as ACROSS variables. This produces a separate column in the report for each combination of values of these variables.

On the COLUMN statement, we include only the three variables AETERM, TRTGRP, and AESEV. However, we insert a comma between TRTGRP and AESEV to indicate that these variables are to be stacked.

Despite there being only 3 items listed on the COLUMN statement, our report has 9 columns: AETERM plus one for each combination of the 2 treatment groups and the 4 severity levels. Had one of our stacked variables been an analysis variable, a computed variable, a group variable, or a statistic, then its values would fill the cells beneath the stacked headers. Since we have none of these items, PROC REPORT simply gives us frequency counts, which happens to be exactly what we want.

Adverse Event	Drug A				Drug B			
	Mild	Moderate	Severe	Life-Threatening	Mild	Moderate	Severe	Life-Threatening
Abdominal Pain	0	1	0	0	1	1	0	0
Anemia	0	3	0	0	3	1	1	0
Constipation	1	0	1	0	1	0	0	0
Diarrhea	0	1	0	0	0	0	0	0
Dizziness	1	0	0	0	0	2	0	0
Dyspepsia	0	1	0	0	1	0	1	0
Epistaxis	1	0	1	0	0	0	2	0
Fatigue	1	2	0	0	1	0	0	0
Headache	2	0	1	0	1	1	2	0
Hyperbilirubinemia	0	4	1	0	0	3	1	0
Hypercalcemia	0	3	0	0	1	3	0	0
Hyponatremia	0	1	0	0	1	0	0	0
Hypotension	1	2	1	0	1	0	0	0
Musculoskeletal pain	2	4	0	0	0	0	0	0
Nausea	2	0	0	0	0	1	0	0
Pyrexia	0	1	2	0	0	1	0	0
Tachycardia	0	0	0	0	0	0	1	1
Vomiting	0	2	0	0	2	0	2	0

Figure 5. Output from PROC REPORT

We also must make a few formatting adjustments to achieve the desired result. We apply the SEV format created earlier to format the AESEV column headers so they display the severity descriptions instead of the corresponding numerical values. That, in turn, requires that we add ORDER=INTERNAL to the DEFINE statement for AESEV so that the columns remain ordered by the internal numerical values and not by the formatted values.

In addition, we set the column labels for TRTGRP and AESEV to an empty character string in order to suppress the printing of an additional header row containing variable labels. The only headers we want are the variable levels themselves.

Next, we modify the style elements associated with the column contents in order to center the contents as we no longer can use the CENTER option on a DEFINE statement. Note that this does not affect the AETERM column because its positioning is overridden by the style elements on the corresponding DEFINE statement.

Finally, we precede our call to PROC REPORT with an OPTIONS statement that changes the system option MISSING to 0 so that report cells associated with frequency counts of zero display the number 0 instead of a dot, which is the default indicator for missing.

The completed code appears below:

```

ods rtf file='myfile.rtf';

option missing=0;

proc report data=ae nowindows
    style(column)=[font_face="Arial" font_size=9pt just=c]
    style(header)=[font_face="Arial" font_size=9pt]
    ;

    column aeterm trtgrp, aesev;

    define aeterm / group "Adverse Event"
        style=[just=1]
        style(header)=[just=1 bordertopcolor=cxBBBBBB];
    define trtgrp / across '';
    define aesev / across order=internal format=sev. '';

run;

ods rtf close;

```

The resulting report is identical to the one produced using method #1, which is shown above in Figure 5.

METHOD #3: PROC TABULATE

Next, we consider the same problem using the TABULATE procedure instead of PROC REPORT. The workhorse of PROC TABULATE is the TABLE statement. The TABLE statement expects a page expression (optional), a row expression (also optional), and a column expression (required), separated by commas.

In our case, we have no need of a page expression so this is omitted. For our row expression, we use `aeterm=' '`, which specifies that we want one row per value of AETERM, and that we do not desire a separate header for that portion of the output. Our column expression is slightly more complex. In order to specify that we wish to have a column for every combination of TRTGRP and AESEV, we list these two variables together with an asterisk between them. We also request frequency counts by appending another asterisk and the keyword N. As with our row expression, we suppress the printing of separate headers for these elements by assigning a null string as the label for each. Thus, our final column expression is `trtgrp=' '*aesev=' '*n=' '`.

We also add two options to the TABLE statement. The PRINTMISS option ensures that the output includes columns for all combinations of TRTGRP and AESEV, even if some of those combinations don't actually appear in the data. This is necessary for our example because our data does not contain any severity 4 events for Drug A, but we still want to have a column showing zero counts. Even if our data did contain all combinations, it's a good practice to consider how our data may change in the future or in what other situations we might want to reuse this code.

The other option we add on the TABLE statement is BOX. This allows us to customize the area in the upper-left of the output where the row headers and column headers overlap. In order to make our output match the PROC REPORT output as closely as possible, we add the label "Adverse Event" here and use in-line style elements to control the positioning and font.

In order for this all to work, we also must add AETERM, TRTGRP, and AESEV to the CLASS statement so that PROC TABULATE recognizes them as class variables. In addition, we use two CLASSLEV statements so that we can apply additional styling to these variables. By default, PROC TABULATE formats the rows headers using bold text on a grey background just like the column headers. For our report, we want the row headers (the values of AETERM) to look just like every other column in the table,

so we set the background color to white and the font weight to medium. We use style elements on both CLASSLEV statements to specify the font for the row and column headers.

While the above styling takes care of the row and column headers as well as the upper-left box, the body of the table also needs some minor formatting adjustments. We add style elements on the PROC TABULATE statement itself to adjust the font and center the values within each column. Since PROC TABULATE considers the first column to be a row header and not part of the table body, it is not affected by this styling. This is fortunate since we wish it to remain left-justified.

Next, we add a FORMAT statement so that the column header shows the severity descriptions instead of the numerical values of AESEV. Unlike with PROC REPORT, this does not alter the ordering of the columns, so no additional logic is needed for that purpose.

Finally, we preface our call to PROC TABULATE by setting the system option MISSING to the number 0, just like we did previously for PROC REPORT. This causes empty cells to display frequency counts of zero.

The complete code appears below:

```
ods rtf file='myfile.rtf';

option missing=0;

proc tabulate data=ae style=[font_face="Arial" font_size=9pt just=c];

    class aeterm trtgrp aesev;

    classlev aeterm / style=[
        font_face="Arial"
        font_size=9pt
        fontweight=medium
        backgroundcolor=white];

    classlev trtgrp aesev / style=[
        font_face="Arial"
        font_size=9pt];

    table aeterm='', trtgrp=''*aesev=''*n='' / printmiss
        box = [label='Adverse Event'
            style=[font_face="Arial"
                font_size=9pt
                just=l
                verticalalign=bottom]
        ];

    format aesev sev.;

run;

ods rtf close;
```

Running this code produces a report identical to the ones produced with the previous two methods, shown above in Figure 5.

METHOD #4: PROC SQL

We have now produced the exact same report using three different methods: one consisting of the `FREQ`, `TRANSPOSE`, and `REPORT` procedures in a three-step process, `PROC REPORT` alone, and `PROC TABULATE`. Next, we attempt to match this output using `PROC SQL`.

While `PROC SQL` is not exactly designed for this type of reporting, the entire report can be generated with a single select statement. The first item on the select statement is the `AETERM` variable, which also appears in the `GROUP BY` clause. Thus, the output has only one row per value of `AETERM`.

The other eight terms on the `SELECT` statement are each of this form:

```
sum(case when trtgrp='Drug A' and aesev=1 then 1 else 0 end)
```

There is one such statement for each of the eight combinations of `TRTGRP` and `AESEV`. The `CASE` statement shown above evaluates to 1 only for those records corresponding with Drug A and severity level 1. For all other records, it evaluates to zero. When `PROC SQL` executes the grouping function `SUM`, it adds up these values for all records associated with a particular value of `AETERM`. The net effect is to produce a frequency count of the number of such records.

Each of these eight results are assigned to variable names such as `DRUG_A1` and assigned an appropriate format and label (e.g. "Mild"), all within the `SELECT` statement. These labels become the column headers in the printed output.

Unfortunately, `PROC SQL` does not provide the extensive opportunities for in-line styling that other procedures like `REPORT` and `TABULATE` have. In order to obtain the desired output, we must resort to other methods. We employ two such strategies here, both of which involve the `TEMPLATE` procedure.

First, we create a custom style template which is applied to the entire RTF output file. This style template is defined in `PROC TEMPLATE` using the `STYLES.RTF` style as a parent and is called `MY.STYLE`. When we open our output file using the `ODS RTF` statement, we use the `STYLE=` option to apply this style to our output.

In the style template definition, we set the body and header fonts, change the header background color, and center the eight columns of frequency counts. Because we do not want this centering to apply to the entire table body, it is applied specifically to each of the other columns using CSS-style syntax in a series of `CONTEXT` statements.

Secondly, we create a custom table template, also using the `TEMPLATE` procedure. The ODS table template that `PROC SQL` uses by default is called `BASE.SQL` and is located in the `SASHELP.TMPLMST` store. By creating a table template with the same name but located in the `SASUSER.TEMPLAT` store, we ensure that `PROC SQL` finds our table template first and uses it instead of the default template.

In our table template, we define two spanning headers. One has text reading "Drug A" and spans the columns `DRUG_A1` through `DRUG_A4`. The other has text reading "Drug B" and spans the columns `DRUG_B1` through `DRUG_B4`. More information about building table templates can be found in Smith (2007).

The full code appears below:

```
proc template;

  define style my.style;
    parent = styles.rtf;
    style header /
      backgroundcolor=cxBBBBBB
      fontweight = bold
      font_face = "Arial"
      font_size = 9pt;
```

```

style data /
  font_face = "Arial"
  font_size = 9pt;
context '.data[name="drug_a1"]' / just = c;
context '.data[name="drug_a2"]' / just = c;
context '.data[name="drug_a3"]' / just = c;
context '.data[name="drug_a4"]' / just = c;
context '.data[name="drug_b1"]' / just = c;
context '.data[name="drug_b2"]' / just = c;
context '.data[name="drug_b3"]' / just = c;
context '.data[name="drug_b4"]' / just = c;
end;

define table base.sql;
  header header_a header_b;
  define header_a;
    text 'Drug A';
    start="drug_a1";
    end="drug_a4";
  end;
  define header_b;
    text 'Drug B';
    start="drug_b1";
    end="drug_b4";
  end;
end;

run;

ods rtf style=my.style file='myfile.rtf';

proc sql;
  select
    aeterm "Adverse Event",
    sum(case when trtgrp='Drug A' and aesev=1 then 1 else 0 end)
      as drug_a1 format=best8.3 "Mild",
    sum(case when trtgrp='Drug A' and aesev=2 then 1 else 0 end)
      as drug_a2 format=best8.3 "Moderate",
    sum(case when trtgrp='Drug A' and aesev=3 then 1 else 0 end)
      as drug_a3 format=best8.3 "Severe",
    sum(case when trtgrp='Drug A' and aesev=4 then 1 else 0 end)
      as drug_a4 format=best8.3 "Life-Threatening",
    sum(case when trtgrp='Drug B' and aesev=1 then 1 else 0 end)
      as drug_b1 format=best8.3 "Mild",
    sum(case when trtgrp='Drug B' and aesev=2 then 1 else 0 end)
      as drug_b2 format=best8.3 "Moderate",
    sum(case when trtgrp='Drug B' and aesev=3 then 1 else 0 end)
      as drug_b3 format=best8.3 "Severe",
    sum(case when trtgrp='Drug B' and aesev=4 then 1 else 0 end)
      as drug_b4 format=best8.3 "Life-Threatening"
  from ae group by aeterm;
quit;
run;

ods rtf close;

```

Together, the custom table template, the custom style template, and the limited formatting options available within the SQL procedure are sufficient to accomplish the task. The result of this code is a report identical to the reports produced previously under the three other methods.

COMPARISON OF METHODS

It has often been noted that there is usually more than one way to perform a task in SAS, sometimes many ways. While this is frequently true, different methods may have advantages and disadvantages, and those may vary based on the situation at hand.

SIMPLICITY

All else being equal, simpler code should be preferred over more complex code. Generally, simpler code is easier to understand, easier to write and debug, and easier for someone else (or even yourself) to maintain later.

In terms of simplicity, method #2, which uses PROC REPORT alone, seems to be the winner among the four methods detailed in this paper. Method #3, PROC TABULATE is not far behind, although it requires a little more custom styling. Method #1 is somewhat more complex as it involves three separate steps. Method #4 turned out to be quite a nightmare. There probably aren't many situations where the latter approach makes much sense.

ROBUSTNESS

Another consideration is the relative ease with which the code could accommodate changes to the data. It's worth thinking about how one might have to change the code if a Drug C were added, or if the number of levels of severity changed.

Both methods #2 (PROC REPORT) and #3 (PROC TABULATE) are very robust in this regard. Neither would require any change to the code to handle such changes, other than modifying the definition of the SEV format.

Method #1 (FREQ / TRANPOSE / REPORT), on the other hand, would require changes to the PROC REPORT call. Specifically, the COLUMN statement would need to be modified and DEFINE statements added or removed based on the new levels of TRTGRP and AESEV.

Method #4 (PROC SQL) would require the most extensive changes. Based upon the new levels of TRTGRP and AESEV, one would have to add or remove terms from the SELECT statement, add or remove CONTEXT statements from the style template definition, and modify the spanning headers in the table template definition.

CUSTOMIZATION

As we saw in our example, there is considerable variation in the ease with which output from different procedures can be customized. In some situations, this may help determine which method to use on a given project. We saw that the REPORT and TABULATE procedures generally provide more flexibility with formatting than PROC SQL. However, before selecting an approach, it helps to have some idea of the intended end result. If the goal is to produce an output data set, then formatting considerations may play no role at all in the process.

PERFORMANCE

Due to the extremely small size of the data set being used in this example, there is no practical difference in the speed of execution between any of these methods. However, if one were working with substantially larger data sets, or one had reason to expect their data might grow large in the future, it would be worth performing additional testing to compare the performance of the methods. This is beyond the scope of the current paper.

CONCLUSION

We were able to reproduce the desired output with each of the four methods, even including minor formatting details. This is a testament to the flexibility of SAS. While the PROC REPORT and PROC TABULATE methods seem best for this particular circumstance, there may be cases in which another method better accommodates some peculiarity of the situation. Understanding how and why each method works allows the SAS programmer to better judge each situation. Like any good craftsman, the savvy SAS programmer has many tools at his disposal and knows how and when to use each one.

REFERENCES

- Gupta, Ajay. 2015. "Simulate PRELOADFMT Option in PROC FREQ." *Proceedings of the Pharmaceutical Industry SAS Users Group 2015 Conference*. Available at <http://www.pharmasug.org/proceedings/2015/QT/PharmaSUG-2015-QT33.pdf>.
- Horstman, Joshua M. 2014. "Five Ways to Flip Flop Your Data" *Proceedings of the SAS Global Forum 2014 Conference*. Cary, NC: SAS Institute Inc. Available at <http://support.sas.com/resources/papers/proceedings14/1266-2014.pdf>.
- Phillips, Stacey D. and Gary Klein. 2005. "Oh No, a Zero Row: 5 Ways to Summarize Absolutely Nothing." *Proceedings of the Pharmaceutical Industry SAS Users Group 2015 Conference*. Available at <http://www.lexjansen.com/pharmasug/2005/CodersCorner/cc22.pdf>.
- Smith, Kevin D. 2007. "PROC TEMPLATE Tables from Scratch." *Proceedings of the SAS Global Forum 2007 Conference*. Cary, NC: SAS Institute Inc. Available at <http://www2.sas.com/proceedings/forum2007/221-2007.pdf>.

ACKNOWLEDGEMENTS

I wish to thank my colleague, Dr. Roger D. Muller, for asking a question which was the seed that grew into this paper.

I also wish to thank Kevin D. Smith of SAS Institute for providing invaluable assistance in working out the details of the PROC SQL code in this paper.

RECOMMENDED READING

- Carpenter, Art. 2007. *Carpenter's Complete Guide to the SAS® REPORT Procedure*. Cary, NC: SAS Institute Inc.
- Lafler, Kirk Paul. 2013. *PROC SQL: Beyond the Basics Using SAS®, Second Edition*. Cary, NC: SAS Institute Inc.
- Lake, Lauren Haworth and Julie McKnight. 2015. *PROC TABULATE by Example, Second Edition*. Cary, NC: SAS Institute Inc.
- Smith, Kevin D. 2007. *PROC TEMPLATE Made Easy: A Guide for SAS® Users*. Cary, NC: SAS Institute Inc.

CONTACT INFORMATION

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

Joshua M. Horstman
Nested Loop Consulting
317-721-1009
josh@nestedloopconsulting.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.