SESUG Paper 218-2018

Using PROC SQL to Generate Shift Tables More Efficiently

Jenna Cody, IQVIA

ABSTRACT

Shift tables display the change in the frequency of subjects across specified categories from baseline to post-baseline time points. They are commonly used in clinical data to display the shift in the values of laboratory parameters, ECG interpretations, or other ordinal variables of interest across visits. The “into:” statement in PROC SQL can be used to create macro variables for the denominators used in these tables. These macro variables can be accessed throughout the program, allowing for easy computation of percentages and the ability to call the same macro variable to display the subject count value in the header.

This paper outlines the steps for creating a shift table using an example with dummy data. It describes the process of creating macro variables in PROC SQL using the “into:” step, creating shift table shells using the DATA step, conducting frequency tabulations using PROC FREQ, calling the macro variables to calculate and present the incidence and percent, and using the macro variables for the subject count value in the headers. It then discusses the efficiency of the use of PROC SQL to create macro variable denominators over other methods of calculating denominators, such as in the PROC FREQ step. Code examples are provided to compare shift table generation techniques.

INTRODUCTION

Shift tables display changes in the distribution of ordinal clinical data across visits and usually provide a comparison between treatment groups. This paper describes an efficient method of programming shift tables in SAS® using PROC SQL. It shows how several programming steps can be combined into a single PROC SQL query and describes the syntax used in this query. This paper specifically applies this technique to a shift table example but is also useful for those looking to use PROC SQL to code more efficiently.

The intended audience of this paper is programmers and statisticians who display ordinal clinical data in the form of shift tables. However, these techniques could be generalized to other applications where the user aims to present a shift in ordinal data over time. Those looking to learn more about the creation of macro variables using PROC SQL will benefit from this paper as well. Prior knowledge of basic SAS programming (PROCs and DATA steps) and familiarity with PROC SQL is assumed.

SHIFT TABLES

Shift tables are used to display the change in the frequency of ordinal data over time. Subjects are tabulated in each combination of treatment group/visit/parameter/category and these frequencies are displayed in layout that makes it easy to make comparisons and see trends. A comparison is typically made between treatment groups across time periods, with the values from the *Comparison Visit* represented as column headers and values from the other visits represented as row headers. Typically, the *Comparison Visit* is the baseline visit and the row headers display values from one or more study visits.

Common applications of shift tables are seen in laboratory data range flags, electrocardiography (ECG) data, and other ordinal variables. Laboratory shift tables are typically displayed with multiple parameters in each table (e.g. Alkaline Phosphatase, Creatinine, Hemoglobin, etc.) with general categories such as *Low*, *Normal*, and *High* based on each laboratory reference range. In many cases, a separate category for *Total* displays the total count of subjects in that timepoint and parameter across categories. ECG shift tables typically display ECG data in a *Normal*, *Abnormal Not Clinically Significant*, and *Abnormal Clinically Significant* ranges. Other shift tables can be based on any ordinal assessment, such as a questionnaire classifying an illness into categories of *None*, *Mild*, *Moderate*, *Severe*, *Very Severe*, etc. When many categories or treatment groups are presented, it is often more feasible to format

the treatment groups as row headers rather than column headers, as shown in Figure 1. In other cases where all the columns can fit on one page, the format used in Figure 2 is more common.

The shift table examples, Figure 1 and Figure 2, display simple formatted examples with one parameter, three visits (one comparison visit), two treatment groups, and three categories. In practice, the italicized headers are replaced with the actual parameter, group, visit, comparison visit, or category referenced. *Parameter* can refer to a clinical assessment, ECG test, laboratory test (i.e. Glucose, Platelet Count, Lymphocytes, etc.), organ function test, or other assessment to be displayed. *Group* refers to the treatment groups that are presented and compared in the table, such as Placebo and Treatment or Groups A, B, & C. *Visit* refers to the analysis visit, such as Baseline, Month X, or Endpoint. *Comparison visit* is usually Baseline but can also refer to End of Study or another time point of interest. *Category* refers to the categorical classifications for the assessment, such as *Low*, *Normal*, or *High*. This example shows a shift table formatted so one denominator is constant throughout the table and missing values are presented in a separate column. Another way to present this would be to display a new subject count per visit based on the subjects with non-missing observations at baseline and that visit.

There can be as many parameters, groups, visits, and categories as needed. Horizontal space limitations must be taken into consideration for studies when the number of categories times the number of treatment groups cannot be displayed in the column headers. In this case, the table must be reformatted. This can be achieved by changing the format to that shown in Figure 1, collapsing categories, or displaying fewer treatment groups and splitting into multiple tables, e.g., displaying Placebo and Treatment 1 in one table and Treatments 2 and 3 in another table.

Parameter 1

			Comparison Visit			
Treatment Group	Visit		*Cat 1*	*Cat 2*	*Cat 3*	*Missing*
Group A (N=XXXX)	*Visit 1*					
		Cat 1	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
		Cat 2	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
		Cat 3	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
		Missing	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
	Visit 2					
		Cat 1	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
		Cat 2	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
		Cat 3	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
		Missing	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
Group B (N=XXXX)	*Visit 1*					
		Cat 1	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
		Cat 2	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
		Cat 3	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
		Missing	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
	Visit 2					
		Cat 1	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
		Cat 2	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
		Cat 3	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
		Missing	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)

Figure 1. Shift table layout with treatment groups as row headers.

	Comparison Visit							
	Group A (N=XXXX)				Group B (N=XXXX)			
Visit/ Parameter	*Cat 1*	*Cat 2*	*Cat 3*	*Missing*	*Cat 1*	*Cat 2*	*Cat 3*	*Missing*
Visit 1								
Parameter 1								
Cat 1	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
Cat 2	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
Cat 3	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
Missing	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
Visit 2								
Parameter 1								
Cat 1	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
Cat 2	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
Cat 3	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)
Missing	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)	X (XXX%)

Figure 2. Shift table layout with treatment groups as column headers.

TRADITIONAL APPROACH TO PROGRAMMING SHIFT TABLES

Traditionally, a shift table is programmed with a series of PROC FREQ steps to obtain the tabulations for the numerators and population counts. Next, the data is sorted and merged together to compute the percentages needed for the shift table. A CALL SYMPUT step may be used to convert the population counts to macro variables that can be called later in the program, such as in the PROC REPORT step or in a TITLE statement. The following sample code demonstrates how the body of a shift table is typically programmed for a simple example like the one in Figure 2:

```
proc freq data=adlb noprint;
    tables trtan/list missing out=tot_n(drop=percent rename=(count=total));
    where avisitn=1;
run;

proc freq data=adlb noprint;
    tables trtan*avisitn*paramn*avalca1n*baseca1n/
          list missing out=num(drop=percent rename=(count=ct));
    where avisitn ne 1;
run;

proc sort data=num out=num1;
    by trtan avisitn paramn avalca1n baseca1n;
run;

data trt (drop=ct);
    merge num1 tot_n;
    by trtan;
    length count $14;
    count=put(ct,5.)||" ("||put(100*ct/total,5.1)||"%)";
run;

data tot (keep=trtan total);
  set tot_n;
  call symput("tot"||compress(put(trtan,best.)),
    trim(left(put(total,best.))));
run;
```

This code displays the computation steps for the body of a shift table. Modifications to code that uses this approach can sometimes be difficult because an update to one step may impact the subsequent steps. This may make future edits time-consuming and error-prone, as the code may need to be updated in multiple locations. To simplify the code and allow for easier updates, these steps can be combined into one PROC SQL statement.

A MORE EFFICIENT APPROACH

The body of the shift table can be generated using PROC SQL in the following manner:

```
proc sql noprint;
  select count(distinct usubjid) into:tot1-:tot2
    from adlb
      group by trtan;
  create table trt as
    select baseca1n, avisitn, paramn, avalca1n, trtan,
(case when trtan=1 then put(count(distinct usubjid),5.)||" ("||
put(100*count(distinct usubjid)/&tot1,5.1)||"%)"
when trtan=2 then put(count(distinct usubjid),5.)||" ("||
put(100*count(distinct usubjid)/&tot2,5.1)||"%)" end) as count
      from adlb
        where avisitn ne 1
          group by trtan, baseca1n, avisitn, paramn, avalca1n
            order by trtan, avisitn, paramn, avalca1n, baseca1n;
quit;
```

The first SELECT statement uses the INTO: feature of PROC SQL to create the denominators for the shift table in the form of macro variables, &tot1. and &tot2. It selects the distinct number of subjects in the ADLB dataset in each treatment group, specified in the variable TRTAN. Because there are two possible values of TRTAN in this data, two values are output when a count of the subjects is performed by treatment group. The macro variables assigned to hold these values are specified in the :tot1-:tot2 statement. Just like macro variables created through a %LET statement or CALL SYMPUT step, these new macro variables may be referenced throughout the program at any point after the end of the PROC SQL query. In this example, &tot1 and &tot2 are referenced in the next PROC SQL query, and can later be referenced when needed in a title, “N=XXX” header display, or additional computation.

The second PROC SQL query creates the body of the shift table. It selects the variables of interest from the same ADLB dataset, only examining visits after the baseline visit, denoted AVISITN=1. This visit is already captured in the variable BASECA1N and is redundant. The variables of interest include the baseline category, analysis visit number, parameter, analysis (i.e. non-baseline) category, treatment group, and tabulation of the count and percent of subjects in each category.

The final variable in the SELECT statement uses the macro variables created in the first SELECT statement to compute the percentages to be displayed in the table. The data is read in, computed into counts and percentages, and formatted all in one step! The CASE WHEN option is used to selectively compute the percentage based on the treatment group. The number of subjects typically differs between treatment groups, and this option allows for flexibility in the choice of denominator for this computation.

The GROUP BY statement acts similarly to the crosstabulation option in PROC FREQ and allows the counts to be computed within each combination of these subcategories. The ORDER BY statement sorts the dataset into the desired order needed for the subsequent PROC TRANSPOSE step.

Output 1 shows the format of the shift table after using either of the approaches discussed. The simulated data presents the shift from the baseline categorization to the categorization in visits 2 and 3 for the 4 subjects in each of the 2 treatment groups in 1 parameter.

trtan	baseca1n	avisitn	paramn	avalca1n	count
1	2	2	1	2	2 (50.0%)
1	2	2	1	3	1 (25.0%)
1	2	2	1	4	1 (25.0%)
1	2	3	1	2	3 (75.0%)
1	2	3	1	4	1 (25.0%)
2	1	2	1	2	2 (50.0%)
2	2	2	1	2	1 (25.0%)
2	3	2	1	2	1 (25.0%)
2	2	3	1	1	1 (25.0%)
2	1	3	1	2	2 (50.0%)
2	3	3	1	2	1 (25.0%)

Output 1. Output from the PROC SQL Statement

FORMATTING THE SHIFT TABLE

Once the body of the shift table has been programmed, either using the traditional approach or the PROC SQL approach, several additional steps are needed to change the layout of the shift table to the desired format. First, the treatment/visit/parameter/baseline category/analysis category combinations that are not populated need to be filled in with a 0 count. The easiest way to do this is to create a dummy shell dataset and merge it with the existing data, ensuring that the dummy dataset does not overwrite the analysis dataset and only blank cells are filled in with “0”:

```
data dummy;
  length count $ 14;
  do trtan=1 to 2;
    do avisitn=2 to 3;
      do paramn=1;
        do avalca1n=1 to 4;
          do baseca1n=1 to 4;
            count="0";
            output;
          end;
        end;
      end;
    end;
  end;
run;

data merged;
  merge dummy trt;
  by trtan avisitn paramn avalca1n baseca1n;
run;
```

Output 2 shows the format of the dataset after it has been merged with the dummy data.

count	trtan	avisitn	paramn	avalca1n	baseca1n
0	1	2	1	1	1
0	1	2	1	1	2
0	1	2	1	1	3
0	1	2	1	1	4
0	1	2	1	2	1
2 (50.0%)	1	2	1	2	2
0	1	2	1	2	3
0	1	2	1	2	4
0	1	2	1	3	1

1 (25.0%)	1	2	1	3	2
0	1	2	1	3	3
0	1	2	1	3	4
0	1	2	1	4	1
1 (25.0%)	1	2	1	4	2
0	1	2	1	4	3
0	1	2	1	4	4
0	1	3	1	1	1
0	1	3	1	1	2
0	1	3	1	1	3
0	1	3	1	1	4
0	1	3	1	2	1
3 (75.0%)	1	3	1	2	2
0	1	3	1	2	3
0	1	3	1	2	4
0	1	3	1	3	1
0	1	3	1	3	2
0	1	3	1	3	3
0	1	3	1	3	4
0	1	3	1	4	1
1 (25.0%)	1	3	1	4	2
0	1	3	1	4	3
0	1	3	1	4	4
0	2	2	1	1	1
0	2	2	1	1	2
0	2	2	1	1	3
0	2	2	1	1	4
2 (50.0%)	2	2	1	2	1
1 (25.0%)	2	2	1	2	2
1 (25.0%)	2	2	1	2	3
0	2	2	1	2	4
0	2	2	1	3	1
0	2	2	1	3	2
0	2	2	1	3	3
0	2	2	1	3	4
0	2	2	1	4	1
0	2	2	1	4	2
0	2	2	1	4	3
0	2	2	1	4	4
0	2	3	1	1	1
1 (25.0%)	2	3	1	1	2
0	2	3	1	1	3
0	2	3	1	1	4
2 (50.0%)	2	3	1	2	1
0	2	3	1	2	2
1 (25.0%)	2	3	1	2	3
0	2	3	1	2	4
0	2	3	1	3	1
0	2	3	1	3	2
0	2	3	1	3	3
0	2	3	1	3	4
0	2	3	1	4	1
0	2	3	1	4	2
0	2	3	1	4	3
0	2	3	1	4	4

Output 2. Output after merging with the dummy dataset

The next step for formatting the dataset is to transpose it. Depending on the requested format for the table, the PROC TRANSPOSE step may take several forms. This example shows a transformation of the table has into the format from Figure 2:

```
proc transpose data=merged out=trans prefix=basecat;
   by trtan avisitn paramn avalca1n;
   id baseca1n;
   var count;
run;
```

Output 3 shows the format of the dataset after it has been transposed.

trtan	avisitn	paramn	avalca1n	_NAME_	basecat1	basecat2	basecat3	basecat4
1	2	1	1	count	0	0	0	0
1	2	1	2	count	0	2 (50.0%)	0	0
1	2	1	3	count	0	1 (25.0%)	0	0
1	2	1	4	count	0	1 (25.0%)	0	0
1	3	1	1	count	0	0	0	0
1	3	1	2	count	0	3 (75.0%)	0	0
1	3	1	3	count	0	0	0	0
1	3	1	4	count	0	1 (25.0%)	0	0
2	2	1	1	count	0	0	0	0
2	2	1	2	count	2 (50.0%)	1 (25.0%)	1 (25.0%)	0
2	2	1	3	count	0	0	0	0
2	2	1	4	count	0	0	0	0
2	3	1	1	count	0	1 (25.0%)	0	0
2	3	1	2	count	2 (50.0%)	0	1 (25.0%)	0
2	3	1	3	count	0	0	0	0
2	3	1	4	count	0	0	0	0

Output 3. Output after PROC TRANSPOSE step

Next, a DATA step is used to ensure the data is in the desired format. The formats VISITS, PARAMETERS, and CATEGORIES were created ahead of time using the FORMAT procedure. In practice, these would take on the actual names of these categories:

```
data final;
     length col1-col11 $200;
     merge trans(where=(trtan=1) drop=_NAME_ rename=
     (basecat1=col4 basecat2=col5 basecat3=col6 basecat4=col7))
     trans(where=(trtan=2) drop=_NAME_ rename=
     (basecat1=col8 basecat2=col9 basecat3=col10 basecat4=col11));
     by avisitn paramn avalca1n;
     col1=put(avisitn,visits.);
     col2=put(paramn,parameters.);
     col3=put(avalca1n,categories.);
     keep col1-col11;
run;
```

Output 4 shows the format of the dataset after the DATA step.

col1	col2	col3	col4	col5	col6	col7	col8	col9	col10	col11
Vis2	Param1	Cat 1	0	0	0	0	0	0	0	0
Vis2	Param1	Cat 2	0	2 (50.0%)	0	0	2 (50.0%)	1 (25.0%)	1 (25.0%)	0
Vis2	Param1	Cat 3	0	1 (25.0%)	0	0	0	0	0	0
Vis2	Param1	Missing	0	1 (25.0%)	0	0	0	0	0	0
Vis3	Param1	Cat 1	0	0	0	0	0	1 (25.0%)	0	0
Vis3	Param1	Cat 2	0	3 (75.0%)	0	0	2 (50.0%)	0	1 (25.0%)	0
Vis3	Param1	Cat 3	0	0	0	0	0	0	0	0
Vis3	Param1	Missing	0	1 (25.0%)	0	0	0	0	0	0

Output 4. Output after final formatting DATA step

The dataset is now ready to be output. This outputting step varies based on the style of the report. If using PROC REPORT for this step, the &tot1 and &tot2 macro variables that were created in the PROC SQL step can be referenced for the “N=XXX” section.

CONCLUSION

The use of this technique for programming shift tables allows programmers to gain efficiency and prevent potential programming errors. An obvious advantage of generating a shift table using PROC SQL over the PROC FREQ and DATA step method is a reduction in the amount of code that needs to be written. Once a programmer is familiar with the syntax and format of PROC SQL, it can be faster to write the code for shift tables using this format. Fewer steps are required because PROC SQL allows users to combine features of multiple DATA and PROC steps into one query. It saves computation time, which is especially beneficial if the data set for the shift table is large.

As discussed earlier, another advantage of programming shift tables using PROC SQL is to allow future edits to be made more easily. Because the body of the table is programmed in one step, it is easy for a programmer to make modifications quickly. In a series of PROC and DATA steps, it is easy to make a mistake when updates are performed to one PROC or DATA step and are omitted from subsequent steps. For example, the programmer may change the order of the variables in the SORT procedure but may forget to update the BY statement in the following DATA step. Using PROC SQL, these errors are reduced because there are fewer programming steps that need to be changed for each modification.

Computations can be performed more easily and with fewer mistakes when population denominators are stored in macro variables. When using the :INTO feature in PROC SQL or the CALL SYMPUT step to dynamically assign population counts to macro variables, programmers know exactly how the percentages are being computed and mistakes from improper merges can be avoided. It is common for mistakes to be made when merging several frequency outputs, or when performing a merge with a different number of observations in each dataset. This step is avoided when using macro variables instead of a frequency output for the population counts because they can be called from any step without the need for a merge.

Lastly, an advantage to storing denominators in macro variables instead of using values from a PROC FREQ step is the ability to call these denominators in future steps. This is especially useful when programming shift tables because in most cases, the row or column headers display the population counts in the form “N=XXX”. When stored in macro variables, the denominators are automatically updated for any changes in the input data set each time the program is run. The use of macro variables in these header calls is easier and more dynamic that manually inputting the population counts.

CONTACT INFORMATION

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

Jenna Cody
IQVIA
Jenna.Cody@IQVIA.com
https://www.linkedin.com/in/jennacody/