Paper BB-12-2013

Life Imitates Art: ODS Output Data Sets that Look like Listing Output

Dylan Ellis, Mathematica Policy Research, Washington DC

ABSTRACT

Confusingly, the output data set from a summary procedure looks nothing like the display in our output destination. Have you ever wished for an output data set that looks like the cross-tabulation you see in the listing window? Have you ever run a series of one-way frequencies on a list of variables and wished for a more compact tabular display? This presentation will show how we can leverage string functions and the automatic _TYPE_ variable from PROC TABULATE to reshape the output data set into a more intelligible and practical table visualization. Readers should be familiar with the basics of how to specify a categorical frequency table using Proc Freq or Proc Tabulate.

INTRODUCTION

Sometimes it is not possible to produce the tables we would like to see from standard summary procedures. This paper will demonstrate how to use array functions and PROC TRANSPOSE to manipulate the output data set into a form which mimics the desired table. The transformed data set can then be displayed using PROC PRINT or PROC REPORT. Because tables and data sets appear very similar on paper – and the point of this paper is that we can make them look similar – I have color-coded the header rows. Blue indicates output tables, grey an output data set.

The examples in this paper use the SASHELP.BWEIGHT data set, with formats attached as specified on the SAS website. Most examples utilize PROC TABULATE, but comparisons are made to the other summary procedures where possible. This paper will not be an exhaustive introduction to PROC FREQ, MEANS, TABULATE or REPORT. There are other venues and sources from which to learn about the procedures in depth. Rather this paper will focus on some important concepts and techniques for manipulating the output data sets and creating clean output tables.

SUMMARY PROCEDURES: OUTPUT TABLE VERSUS OUTPUT DATA SET

Each of the main summary procedures, FREQ, MEANS, TABULATE and REPORT, can produce an output display as well as an output data set – the former by default, the latter through use of an ODS OUTPUT statement or the OUT= option (for some procedures, such as PROC FREQ, the ODS OUTPUT data set is different than OUT=). However with the exception of PROC REPORT, the output data set looks very different from the output table displayed in our ODS destination. For example, here is a two-way table and associated output data set from PROC TABULATE.

```
Proc Tabulate MISSING
Data=bweight
 Out=outequals(drop=_:);
 Class ed  smoke;
Tables ed, smoke /
       box="(Table in ODS Destination)";
Run;
```

```
*Proc Print of the output data set
 so we can see what it looks like;

Proc Print
Data=outequals noobs;
Run;
```

(Table in ODS Destination)	Indicator: 1= Smoking Mother	
	0	1
	N	N
Mother's Education Level		
high school	14484	2965
some college	10982	1147
college	12137	312
less than high school	5864	2109

Contents of *outequals* data set.

ed	smoke	N
high school	0	14484
high school	1	2965
some college	0	10982
some college	1	1147
college	0	12137
college	1	312
less than high school	0	5864
less than high school	1	2109

Figure 1. A two-way table in PROC TABULATE: the output table (left) and output data set (right).

We can see that both the table and the data set contain the same eight cells, defining all observed combinations of mother's education level and smoking status. Whereas the output table displays the cells in a two-dimensional array, the output data set contains one observation for each cell in the table. The output data set is equivalent to the display we see when we use the list option in PROC FREQ. Values of the class variables are specified explicitly for each cell rather than inferred from the column or row headings as in the output table.

Each form has a purpose: while the output table is more readable, the output data set is better for merging summary counts onto by-groups in a detail-level data set. If desired, we can use PROC TRANSPOSE to rearrange the output data set to make it look like the output table, with observations grouped by row and a variable for each column value.

```
Proc Transpose
Data=outequals
 Out=twowaytable(drop=_:)
     prefix=smoke;
  By ed;
  ID smoke;
 Var N;
Run;
```

Contents of *twowaytable* data set.

ed	smoke0	smoke1
high school	14484	2965
some college	10982	1147
college	12137	312
less than high school	5864	2109

Figure 2. Result of PROC TRANSPOSE: the output data set (right) matches the original output table (above).

Sometimes it is helpful for the structure of the data set to mirror the original output table. For example, if we want to automatically update a chart range in Excel using DDE, we need the data to be in this rectangular form. However it may seem a futile exercise given that we could reach the same end with one call to PROC REPORT. Output data sets and tables from PROC REPORT display similar structure, and the procedure can handle any two-way table.

```
Proc Report NOWD
Data=bweight
 Out=reportout(keep=ed _C:);
Column ed smoke,N;
Define ed / group order=internal;
Define smoke / across;
Run;
```

Contents of *reportout* data set.

ed	_C2_	_C3_
high school	14484	2965
some college	10982	1147
college	12137	312
less than high school	5864	2109

Figure 3. Result of PROC REPORT: both the output table and output data set share the same structure. (the output table looks very similar to the table created by PROC TABULATE in Figure 1 above)

If we only ever need to cross two variables, we could end the paper here and always use PROC REPORT. But as we know, our tables typically get much more complicated, and a single transpose will rarely be sufficient to reshape the output data set to the form of the output table. The point is that all that matters are the cells the procedure can create. If we have the desired cells, we can manipulate them via some type of post-process to generate the desired table.

The reason I start with this simple example is to show that there is more than one way to reach the desired display. Even if we cannot create our desired table with any summary procedure or special option, it may still be possible to construct in an automatic fashion by transforming the output data set with a few additional steps. We will look at a few specific examples but first let us review what each summary procedure allows us to do out of the box.

WHAT _TYPE_S OF OUTPUT DATA SETS CAN WE CREATE?

SAS summary procedures can be used for many purposes, including performing statistical tests, but for this paper we will consider only how they allow us to group data into tables and calculate descriptive statistics within each cell. There is a general form to the output data sets created by PROC FREQ, PROC MEANS, and PROC TABULATE:

Class1	Class2	Class3	Class4	_TYPE_	N	Pct	CumPct	Mean_Wt	Mean_Age
m	.	yes	0	1011	#	#	#	#	#
Values of classification variables that define group represented in the cell				Group Label	Count	Percentages		Descriptive Statistics	

Figure 4. General form of output data set, with classification variables on the left and statistics on the right.

- PROC FREQ will provide counts (Frequency), percentages and cumulative percentages within each group. The _TYPE_ variable is replaced by a variable called Table that describes the group in complete words.
- PROC MEANS will provide counts (_FREQ_) within each group and statistics based on analytic variables. The _TYPE_ variable is either a numeric index or a bitwise indicator if the 'chartype' option is selected. If no statistics are requested on the output statement, the structure of the output data set is somewhat different. There is an additional _STAT_ variable which essentially becomes part of the group definition, and the analytic variables themselves name the subsequent columns (rather than a variable_statistic combination).
- PROC TABULATE will provide counts (N), percentages or summary statistics for each group. N is provided by default, but summary statistics must be requested explicitly. Each combination of analysis variable and the associated statistic gets its own column. _TABLE_ and _PAGE_ distinguish between compound tables.
- PROC REPORT is able to produce counts, percentages and summary statistics for each group. Uniquely, the procedure can also provide counts within categories of other variables, known as 'across' variables, in the output data set. However columns under across variables are named by column number, such as _C#_. A _BREAK_ variable is also created, which for summary lines will contain the name of the break variable.

Defining groups to appear in the table

The method of specifying groupings of classification variables to include is similar for each procedure. A single variable gives a one-way tabulation. Two variables separated by a space give independent one-way tabulations. Variables crossed with an asterisk are tabulated for all combinations of the values of the two variables. Parentheses can be used to group variables, and crossing with a parenthetical group will distribute as in arithmetic.

`married`

married	_TYPE_	N
0	1	14369
1	1	35631

`married smoke`

married	smoke	_TYPE_	N
0	.	10	14369
1	.	10	35631
.	0	01	43467
.	1	01	6533

`married*smoke`

married	smoke	_TYPE_	N
0	0	11	11149
0	1	11	3220
1	0	11	32318
1	1	11	3313

Figure 5. The output data set for a one-way tabulation, two one-way tabulations, and a two-way tabulation.

In PROC FREQ, the table specification takes place on the 'Tables' statement, whereas in PROC MEANS it is on the 'Types' statement. PROC TABULATE uses commas to separate the rows and columns, but with respect to the output data set these are essentially crossings of page, row and column dimensions: equivalent to (page)*(rows)*(columns). Similarly, in PROC REPORT, the implied table specification is (group variables)*(across variables).

Therefore the following four approaches all produce the same categories and counts in the output data set:

```
Proc Freq Data=bweight;
Tables black*married*smoke/ out=pt1;
Tables black*married*boy  / out=pt2;
Run;
   Data Freq_out;
   Set pt1 pt2;
   *unnecessary if use ODS Output;
   Run;

Proc Means Data=bweight CHARTYPE;
Class black married smoke boy;
Types black*married*smoke
      black*married*boy;
  Var cigsper weight;
Output Max=maximum Out=Means_out
       / autoname;
Run;
```

```
Proc Tabulate Data=bweight MISSING
  Out=Tabulate_out;
Class black married smoke boy;
  Var cigsper;
Tables black*married, smoke*(N cigsper*median) boy;
Run;


Proc Report Data=bweight NOWD
Out=Report_out;
Column black married smoke boy,(N weight,mean);
Define black   / group;
Define married / group;
Define smoke   / across;
Define boy     / across;
Define weight  / analysis;
Run;
```

Figure 6. Code to produce four output data sets containing equivalent categories and counts.

Note that incorporating statistics does not affect the groups involved or the categories in the output data set.

The classification variables are what determine the structure of the output table and the number of observations in the output data set. Analysis variables only affect the *contents* of the output table and the columns in the output data set.

How to uniquely identify a cell in a table:

When we read an output table, we identify the group a cell represents by looking at the row variable(s) and values, and the column variable(s) and value. More generally, we look at *which* classification variables are involved and *what* their values are for this particular cell. These questions are answered by the _TYPE_ and classification variables in the output data set. Let us look at an example using the output data set from PROC TABULATE in Figure 6 above.

black	married	smoke	boy	_TYPE_	_PAGE_	_TABLE_	N	cigsper_Median
0	0	0	.	1110	1	1	6484	0
0	0	1	.	1110	1	1	2569	10
0	1	0	.	1110	1	1	29633	0
0	1	1	.	1110	1	1	3172	10
1	0	0	.	1110	1	1	4665	0
1	0	1	.	1110	1	1	651	7
1	1	0	.	1110	1	1	2685	0
1	1	1	.	1110	1	1	141	7
0	0	.	0	1101	1	1	4443	.
0	0	.	1	1101	1	1	4610	.
0	1	.	0	1101	1	1	15786	.
0	1	.	1	1101	1	1	17019	.
1	0	.	0	1101	1	1	2584	.
1	0	.	1	1101	1	1	2732	.
1	1	.	0	1101	1	1	1395	.
1	1	.	1	1101	1	1	1431	.

Figure 7. The output data set from PROC TABULATE as specified in Figure 6.

For PROC TABULATE, or PROC MEANS using option 'chartype', the _TYPE_ variable will indicate in bitwise fashion which classification variables were used to define the cell. _TYPE_ contains one byte for each classification variable, in the exact same order as the variables are listed on the class statement (or statements). A '1' indicates the variable was involved, in which case the value of the variable is in the respective column. A '0' means the variable was not involved and the value in the column for that variable will be missing. [That said, we will see later that if the missing option is not used, variables with a _TYPE_ of '0' may still be involved – indirectly – in defining the count in a cell].

The _TYPE_ variable also indicates the dimension of the table. A single '1' is a one-way tabulation. Two '1's indicate a two-way tabulation. If the _TYPE_ comprises all '1's then it is an nway table, where n is the number of classification variables. We see in Figure 7 that our output actually comprises two different three-way tables (this is not the same as the _TABLE_ variable). Output data sets for PROC TABULATE and PROC FREQ will contain only the tables and dimensions specified in the Tables statement. PROC MEANS and PROC REPORT operate slightly different:

- PROC MEANS – by default, all classification variables are used in all possible combinations, from one-way to nway, to form the final output table. The 'ways' option allows you to request all possible two-way tables, for instance, without having to spell out every possible pairing. Alternatively, the nway option produces only the table which involves all of the classification variables, e.g. a three-way table for three class variables.
- PROC REPORT – by default, multiple group variables function akin to the nway option in PROC MEANS. Rows are only created which involve all of the group variables, as if the group variables had been crossed.

…

SAS's summary procedures are very powerful, but there are still some output tables which they cannot produce automatically. Now that we have reviewed composition of output data sets, let us take a look at a few examples.

DISPLAY ONE-WAY TABULATIONS OF SIMILAR VARIABLES

Suppose we want to summarize multiple variables that share a common range of values. This is a frequent problem with survey data where many items are on a common (often numeric) scale, such as a likert scale. Here is an excerpt from one such table – copied from the CDC website – summarizing responses to the 2012 HealthStyles survey.

Question	# Analyzed Respondents	Agree	Neither Agree/ Disagree	Disagree
Feeding a baby formula instead of breast milk increases the chances the baby will get sick.	4,158	25.66%	40.86%	33.48%
Breast milk is specially designed to meet a baby's nutritional needs.	4,152	82.85%	14.31%	2.84%
If a child is not breastfed, she/he will be more likely to become overweight.	4,154	13.00%	47.76%	39.24%

Figure 8. Excerpt of table summarizing 2012 HealthStyles survey on breastfeeding, taken from CDC website.

We can frame the problem similarly for our data set SASHELP.BWEIGHT. Variables “black” “married” and “smoke” are indicator variables – sharing the common range of values 0 and 1. We want our output table to have the variable names as rows and common values as the columns [the following approach would also work for character variables with a common range of values, such as “agree”, “neither”, and “disagree”].

indicator	value_0	value_1
Black	41858	8142
Married	14369	35631
Smoke	43467	6533

Figure 9. The output table we would like to see.

We can use PROC FREQ or PROC MEANS to produce one-way frequency tables, but these each yield a long and unwieldy display. Alternatively, PROC TABULATE and PROC REPORT can also produce “wide” displays, with the variables in columns rather than rows, but the output table is still excessively complicated and not easy to read.

```
Proc Tabulate MISSING
Data=bweight
 Out=onewayfreqs;
Class black married smoke;
Table black married smoke; *below;
Table black married smoke, ALL; *right;
Run;
```

Indicator: 1 = Black Mother		Indicator: 1= Married Mother		Indicator: 1= Smoking Mother	
0	1	0	1	0	1
N	N	N	N	N	N
41858	8142	14369	35631	43467	6533

	All
	N
Indicator: 1= Black Mother	
0	41858
1	8142
Indicator: 1= Married Mother	
0	14369
1	35631
Indicator: 1= Smoking Mother	
0	43467
1	6533

Figure 10. The output we get instead from PROC TABULATE, in both wide and long form.

From the perspective of SAS's summary procedures, the problem is that the columns in the desired display do not represent values of a single categorical variable. We cannot simply cross all of our similar variables by some type of 'scale' variable, since that would essentially entail crossing each variable by itself.

On the bright side, we can see that our standard summary procedures will generate all of the necessary cells for our desired output table. They are just not in the right configuration. Consequently, our table problem reduces from "why can't my summary procedure do that" to "what data manipulations are necessary to take us the rest of the way".

As we will see, the answer could be as simple as a DATA step, PROC TRANSPOSE, and PROC PRINT.

STEP 1. COLLAPSE THE OUTPUT DATA SET USING A DATA STEP

black	married	smoke	_TYPE_	_PAGE_	_TABLE_	N
0	.	.	100	1	1	41858
1	.	.	100	1	1	8142
.	0	.	010	1	1	14369
.	1	.	010	1	1	35631
.	.	0	001	1	1	43467
.	.	1	001	1	1	6533

Varname	VarValue	N
black	0	41858
black	1	8142
married	0	14369
married	1	35631
smoke	0	43467
smoke	1	6533

Figure 11. The original output data set from PROC TABULATE on the left and the collapsed form on the right.

If we know that all of our classification variables are of the same type – character or numeric – and same length, it is unnecessary to store them in separate columns in the output data set. We can collapse our categorical columns to just two: varname and varvalue. 'Varname' will capture the name of the classification variable, and 'Varvalue' will hold the level of the classification variable. To collapse the columns we will use a DATA step and set the output data set.

Create VarName variable to store variable names

As described earlier, an output data set of one-way frequencies is like many individual one-way output data sets appended together. The _TYPE_ variable indicates which classification variables contributed to the count in that row, but since these are one-way frequencies there will be exactly one '1'. Since bytes in the _TYPE_ variable correspond in a one-to-one fashion with the list of class variables, we can use string functions to retrieve varname.

```
VarName=scan("black married smoke",findc(_type_,'1'));
```

The findc() function will identify the position of the first '1' in the _TYPE_ variable, and scan() will return the variable name in the same corresponding position in the list of classification variables. We can take the same approach with output from PROC MEANS if we use the 'chartype' option. Most likely we will implement this transformation as part of a macro, so we could substitute a macro variable or macro parameter for the list of class variables.

```
%let classlist = black married smoke;
VarName=scan("&classlist.",findc(_type_,'1'));
```

Or, if we define the list of classification variables as an array, we can use the vname() array function:

```
array classlist &classlist.;
VarName=vname(classlist[findc(_type_,'1')]);
```

Findc() will select an array element, and vname() will return the name of the variable. This method is slightly more precise in that it preserves capitalization of the variable name, rather than a manually entered &classlist parameter.

For one-way frequencies created by PROC FREQ, the output data set will contain a variable called 'Table' of the form "Table X" where X is the classification variable involved. To retrieve the variable name we can use the scan function:

```
VarName = scan(Table,2, " ");
```

Define VarValue variable to store variable values

We know we can define VarName as a character variable with length 32, since that will accommodate any valid SAS variable name. However if we are going to collapse the values of several columns into one, we need to know what type of column we are working with. We might assume a numeric type when dealing with indicators or survey items, or if we know that all of the variables are the same type we could use the vtypex() function on an arbitrary variable.

But for a macro solution we will want to determine the variable type programmatically, as well as confirm that all of the classification variables are in fact of the same type before proceeding. For this I employed a dictionary table.

```
Proc SQL;
Create View classvars As
Select type,   count(distinct type)   as ntypes,
       length, count(distinct length) as nlengths
  From dictionary.columns
 Where LIBNAME="SASHELP" and MEMNAME="BWEIGHT"
   and findw("&classlist",strip(name), ' ', 'i')
;
Quit;

Data _null_;
 Set classvars end=lastvar;

if ntypes>1 or nlengths>1 then do;
  put "All classification variables are not of same length and type";
  abort cancel;
end;
else if lastvar then do;
  call symput('type',type);
  call symput('length',length);
end;

Run;

*in our macro we can then define our new variables as follows;
length VarName $ 32;

%if "&type."="char" %then %do; length VarValue $ &length.; %end;
%if "&type."="num"  %then %do; length VarValue   &length.; %end;
```

Populate VarValue variable with variable values

There are several ways we can populate the 'Varvalue' variable with the actual values of each classification variable. For data sets from PROC MEANS or PROC TABULATE, we can use an array and again employ the findc() function.

```
array classlist &classlist.;
VarValue=classlist[findc(_type_,'1')];
```

Using this method we technically do not need the type and length of classification variables. SAS will implicitly define the type and length of VarValue to be the same as the first variable assigned to it. Having confirmed that the type and length of each classification variable are consistent, there will be no risk of conflict. Still, the risk of subsequent human misinterpretation suggests we should be explicit in defining the type of the VarValue variable and the array.

Alternatively, since we have already determined the variables' type, we could use the coalesce() and coalesceC() functions. These retrieve the first non-missing value from an array. In this case we know there will be exactly one.

```
*this option will also work for one-way tables created by PROC FREQ;
%if "&type."="char" %then %do; VarValue=coalescec(of classlist[*]); %end;
%if "&type."="num"  %then %do; VarValue=coalesce(of classlist[*]);  %end;
```

Finally, I would like to note a potential problem with another serviceable function, vvaluex(). This function accepts an expression that evaluates to a variable name, and retrieves the *formatted* value of that variable. We might try to use:

```
VarValue= vvaluex(VarName);
```

If the classification variable does not have a format and VarValue has already been defined to be numeric, then an automatic type conversion will take place - the formatted value will be character and get converted back to a number. We could make this conversion explicit by using an input() function on the result of vvaluex(). The risk, however, is that classification variables may have different formats. Suppose one of the indicators is formatted as yes/no – the resulting VarValue column will be character and contain a combination of 'yes', 'no', '0' and '1'. We could avert this by first removing formats from the output data set via PROC DATASETS, but it seems best to use another function.

So to collapse an output data set of one-way frequencies from PROC TABULATE, the full DATA step looks like this:

```
Data onewayfreqs_mod (keep=VarName VarValue N);
 Set onewayfreqs; *original output data set;

array classlist &classlist;

length VarName $32;
VarName =vname(classlist[findc(_type_,'1')]);

%if "&type."="char" %then
%do; length VarValue $ &length.; %end;
%if "&type."="num"  %then
%do; length VarValue   &length.; %end;

VarValue=classlist[findc(_type_,'1')];
run;
```

Contents of *onewayfreqs_mod* data set

Varname	VarValue	N
black	0	41858
black	1	8142
married	0	14369
married	1	35631
smoke	0	43467
smoke	1	6533

STEP 2. TRANSPOSE VALUE BY VARIABLE USING PROC TRANSPOSE

In this form, we are one transpose away from the desired table visualization. We simply use PROC TRANSPOSE to transpose values by variable name. We can also add a prefix to the value to define the new variable/column name.

```
Proc Transpose
Data=onewayfreqs_mod
 Out=desired_output
    prefix=value_;
  By VarName;
  ID VarValue;
 Var N;
Run;
```

Contents of final data set

VarName	value_0	value_1
Black	41858	8142
Married	14369	35631
Smoke	43467	6533

Alternatively, we could use PROC REPORT or PROC TABULATE (again) to perform what is essentially a transpose. However that requires requesting a 'fake' SUM statistic, summarizing the frequency of a single observation. I prefer using PROC TRANSPOSE because I know I am only rearranging values rather than inadvertently grouping the data.

Adding Statistics

I would like to note that we could have just as easily transposed the 'Percent' column, or even a statistic based on an analysis variable. PROC TABULATE and PROC MEANS allow us to calculate statistics such as MEAN, SUM, and MEDIAN within each grouping of the classification variables. These statistics occupy separate columns in the output data set, so to use them we change the 'Var' variable in the final PROC TRANSPOSE from 'N' to the desired statistic. The DATA step transformation only depends on the classification variables, so that remains unchanged.

```
Proc Tabulate MISSING
Data=bweight
 Out=onewayfreqs;
Class black married smoke;
  Var weight;
Table black married smoke,
      ALL*weight*MEAN;
Run;
```

```
Proc Transpose
Data=onewayfreqs_mod
 Out=desired_output
    prefix=mean_;
  By VarName;
  ID VarValue;
 Var weight_Mean;
Run;
```

VarName	mean_0	mean_1
Black	3411.23	3162.68
Married	3234.43	3425.73
Smoke	3402.31	3160.85

Figure 12. Example of transformation carrying through an analysis statistic rather than frequency count.

STEP 3. DISPLAY THE DATA SET USING PROC PRINT OR PROC REPORT

Because PROC TRANSPOSE does not display the table in ODS, we need to use PROC PRINT or PROC REPORT to view our table. While it may seem cumbersome to undertake yet another additional step, this will actually increase our options for formatting our table. PROC REPORT offers far more flexibility and control than the other procedures do over the various style elements in the displayed table.

For example, PROC TABULATE allows us to use formats to perform traffic-lighting of a cell based on its contents – i.e. assign color, style, or format based on value in the cell. However PROC REPORT allows the style and format of a cell to be based on *other* values in that row or column. Returning to the example of survey data, we could easily highlight items (rows) where a majority of the respondents agree or disagree, or simply use alternating row colors for enhanced readability. Also, since we have already performed most of the required summarization behind the scenes, we can just use style options to create a pretty table without having to learn the inner workings of PROC REPORT.

Many excellent papers have already been written on this topic. I would especially recommend papers by Allison McMahill Booth or Vince DelGobbo of SAS. Additionally, Art Carpenter has an entire book about PROC REPORT. If you have an exceptionally complex report, you can also use the report-writing interface with ODSOUT statements to generate a table with the DATA step. Dan O'Connor of SAS has several authoritative papers on this approach.

SHOW ONLY SELECT COLUMNS IN A SURVEILLANCE REPORT

Extending this idea, suppose we have several indicators we would like to track across various populations of interest, as specified by a by-group. For example, I defined the following indicators for the SASHELP.BWEIGHT data set:

- *Low birth weight* – infant birth weight less than 2500 grams [`low_bweight = (weight < 2500);`]
- *High maternal weight gain* – maternal weight gain of more than 35 lbs, the recommended maximum for women of normal BMI before pregnancy [`high_m_gain = (m_wtgain > 35);`]
- *Heavy smoker* – mother smokes more than a pack a day [`heavy_smoke = (cigsper > 20);`]
- *No prenatal visit* – no prenatal visit [`no_prenatal = (visit = 0);`]

We would like to display counts for these indicators by education, race and marital status. That is, we would like the indicators to comprise the columns of our report and the various subgroups to make up the rows. Here is an example of the output table we can get out of the box from PROC TABULATE. PROC REPORT could produce a similar table.

				low_bweight		high_m_gain		heavy_smoke		no_prenatal	
			All	0	1	0	1	0	1	0	1
			N	N	N	N	N	N	N	N	N
Mother's Education Level	**Indicator: 1= Black Mother**	**Indicator: 1= Married Mother**									
high school	**0**	**0**	3999	3745	254	3936	63	3959	40	3944	55
		1	10010	9525	485	9879	131	9949	61	9966	44
	1	**0**	2508	2244	264	2480	28	2507	1	2447	61
		1	932	840	92	921	11	931	1	924	8
some college	**0**	**0**	1674	1581	93	1652	22	1662	12	1663	11
		1	8335	8006	329	8264	71	8310	25	8314	21
	1	**0**	1213	1085	128	1196	17	1211	2	1196	17
		1	907	831	76	896	11	906	1	902	5
…	**…**	**…**	…	…	…	…	…	…	…	…	…

Figure 13. Example of surveillance-type report created with the default options of PROC TABULATE.

There are several problems with the standard output which make it less than ideal for a surveillance report:

- *We are most interested in the cases where the indicator is 1.* Presumably the indicator is 0 or missing for all of the other observations, so it is redundant to display both counts. We can simply display the total number of observations in the by-group and assume that the count for '0' is the by-group total minus the count for '1'.
- *Indicators are often rare relative to the general population*. This will result in a pattern of small '1' totals and large '0' totals, drawing our attention away from the important cells and towards the many that are 'normal'.

Ideally we would restrict the table to show only the columns where the indicator is '1'.

SOLUTION: SUBSET OUTPUT DATA SET BEFORE TRANSPOSE

If we start by collapsing our classification variables (indicators) as we did for the one-way tabulations, we can restrict the data set to only keep where value=1. If we then transpose VarName by by-group we will have our desired table.

ed	black	married	low_ bweight	high_ m_gain	heavy_ smoke	no_ prenatal	_TYPE_	N	VarName	VarValue
high school	0	0	0	.	.	.	1000	3745	low_bweight	0
high school	0	0	1	.	.	.	1000	254	low_bweight	1
high school	0	0	.	0	.	.	0100	3936	high_m_gain	0
high school	0	0	.	1	.	.	0100	63	high_m_gain	1
high school	0	0	.	.	0	.	0010	3959	heavy_smoke	0
high school	0	0	.	.	1	.	0010	40	heavy_smoke	1
high school	0	0	.	.	.	0	0001	3944	no_prenatal	0
high school	0	0	.	.	.	1	0001	55	no_prenatal	1
high school	0	1	0	.	.	.	1000	9525	low_bweight	0
high school	0	1	1	.	.	.	1000	485	low_bweight	1
high school	0	1	.	0	.	.	0100	9879	high_m_gain	0
high school	0	1	.	1	.	.	0100	131	high_m_gain	1
high school	0	1	.	.	0	.	0010	9949	heavy_smoke	0
high school	0	1	.	.	1	.	0010	61	heavy_smoke	1
…	…	…	…	…	…	…	…	…	…	…

Figure 14. Output data set of one-way frequencies from PROC TABULATE with the additional VarName and VarValue columns as specified in the first example. Table has been trimmed for sake of space.

ed	black	married	N	VarName	VarValue
high school	0	0	254	low_bweight	1
high school	0	0	63	high_m_gain	1
high school	0	0	40	heavy_smoke	1
high school	0	0	55	no_prenatal	1
high school	0	1	485	low_bweight	1
high school	0	1	131	high_m_gain	1
high school	0	1	61	heavy_smoke	1

```
*can subset data set using
where option in DATA step
or even in PROC TRANSPOSE;

(where=(VarValue=1))

Proc Transpose
Data=onewayfreqs_mod
 Out=desired_output(drop=_:);
  By ed black married;
  ID VarName;
 Var N;
Run;
```

Figure 15. Data set where indicator value = 1, and final PROC TRANSPOSE to produce output data set.

But wait! We want to display by-group totals in the final report. We can add these subtotals with PROC TABULATE by using the ALL keyword.

```
Proc Tabulate MISSING
Data=bweight
 Out=onewayfreqs;
  By ed black married;
Class low_bweight high_m_gain heavy_smoke no_prenatal;
Table all low_bweight high_m_gain heavy_smoke no_prenatal;
Run;
```

The ALL keyword instructs the procedure to produce a count for the group ignoring all of the classification variables. This cell will appear in the output data set with all of the classification variables missing and _TYPE_="0000" (as there are four classification variables in this case). Since VarName will be undefined for these records – findc() returns 0 and vname(classlist[0]) is out of range – we need to adjust our DATA step to explicitly assign VarName to be 'Total'.

```
if findc(_type_,'1') then do;
  VarName =vname(classlist[findc(_type_,'1')]);
  VarValue=classlist[findc(_type_,'1')];
end;
else do;
  VarName = "Total";
end;
```

We also need to make sure that the where clause does not drop our 'Total' records. We can either assign the total records a VarValue of 1, or we can just add a second condition:

```
(where=(VarValue=1 or VarName="Total"))
```

When we are done, we execute the PROC TRANPOSE of VarName by by-group to get the following output data set.

ed	black	married	Total	low_bweight	high_m_gain	heavy_smoke	no_prenatal
high school	0	0	3999	254	63	40	55
high school	0	1	10010	485	131	61	44
high school	1	0	2508	264	28	1	61
high school	1	1	932	92	11	1	8
some college	0	0	1674	93	22	12	11
some college	0	1	8335	329	71	25	21
some college	1	0	1213	128	17	2	17
some college	1	1	907	76	11	1	5
college	0	0	446	28	3	2	2
college	0	1	11017	350	48	6	27
college	1	0	254	17	3	.	4
college	1	1	732	61	5	.	4
less than high school	0	0	2934	215	43	48	51
less than high school	0	1	3443	194	44	52	37
less than high school	1	0	1341	189	26	7	51
less than high school	1	1	255	20	7	1	5

Figure 16. Final surveillance report showing just the desired cells, with indicators by subgroup.

As before, this table could be easily modified to show percentages instead of counts. Also, we could add variable labels and formatting to cell values in a subsequent call to PROC REPORT. The point is that our output data set now contains only the desired columns, ready for display.

COMPARISON TO CLASSDATA AND PRELOADFMT OPTIONS

'Classdata' and 'preloadfmt' are options that can be used to specify the combinations of classification variables to be shown in the output table. However neither of these options will enable us to generate the above surveillance report.

- *Preloadfmt* uses a format to specify the values of a classification variable shown in the table. Available in PROC MEANS, PROC TABULATE and PROC REPORT.
- *Classdata* uses an external data set to specify combinations of classification variables shown in the table. Available in PROC MEANS and PROC TABULATE.

Each of these options defines a set of categories, as determined by the classification variables, which may or may not match up with the categories observed in the data. We can control what happens when there is an imbalance:

More: If there are *more* categories in the preloaded format or classdata data set than in the raw data set, we have options to force these categories to display even though counts will be missing for all subgroups: • Option 'printmiss' in PROC TABULATE • Option 'completetypes' in PROC MEANS • Option 'completerows' in PROC REPORT ('completecols' already defaults to yes)	**Less:** If there are *fewer* categories in the preloaded format or classdata set than in the original data set, we can use the 'exclusive' option to display only the desired categories in the output table. Sounds promising, right? Only the precise verbiage from the documentation says that all other records will be 'excluded from analysis'. This is the equivalent of using a where statement on the input data set and removing any observations that do not fall into one of the pre-specified categories.

Preloadfmt and classdata are very effective when we want to *expand* the categories shown in the table, ensuring that they stay consistent across all by groups and pages. However neither of these options is very helpful when *restricting* which categories are displayed. What 'exclusive' really does (whether in PROC TABULATE, PROC MEANS or PROC REPORT) is restrict the input data set, **changing the universe of records** which appear in our output table.

This is fundamentally different from what we did in the surveillance report. Implicitly we assume that the indicators are tabulated independently, e.g. the count for 'heavy smokers' will include both mothers who did and did not receive 'prenatal care'. We tabulated the counts on the entire data set and then rearranged the output data set to show only the desired cells. If we try to do this with a format that only displays '1's, or a classdata data set that only contains 1s for each of our indicators, then we will find that our counts are no longer independent. In fact they will all then be the same, as the only records that would be included in the universe for tabulation would have a 1 for every indicator.

…

Missing option `MISSING`

For those unaware, the 'missing' option also affects the input data set. This little nondescript option is all that prevents observations with missing classification variables from being 'excluded from analysis', i.e. dropped from the universe.

- PROC FREQ – the default behavior, without the missing option, is that observations with missing values for any variable in a table statement are dropped from the input data set before processing.
  Counts of missings are displayed below each output table.
- PROC MEANS – the default behavior, without the missing option, is that observations with missing values for any class variable are dropped from the input data set before processing
- PROC TABULATE – the default behavior, without the missing option, is that observations with missing values for any class variable (*even class variables that do not appear on any tables statements!*) will be dropped from the input data set before processing
- PROC REPORT – the default behavior, without the missing option, is that observations with missing values for any group, order or across variable will be dropped from the input data set before processing

If, like me, your assumption was that the universe for both your output table and output data set was the input data set you used (minus any observations dropped due to a where statement), then *your default* behavior should be to always use the missing option for any summary procedure. Especially when using PROC TABULATE, it is easy to forget that a variable on your class statement could be affecting your output data set even if it is not used in the table specification. These variables would be '0' in the _TYPE_ variable but missing values will restrict the table universe.

ALTERNATIVE METHOD FOR BY GROUPS

Use of by groups in a summary procedure requires the input data set to be sorted, or at least grouped if using the NOTSORTED option. By groups have different effects in terms of how the tables display in our ODS destination – often the tables are displayed on a separate page or tab for each by group. Yet in terms of the output data set, by groups are just extra variables, equivalent to having crossed the by variables with the entire table specification.

Therefore the following two forms produce equivalent categories in the output data set:

```
Proc Sort Data = bweight;
      By ed visit;
Run;

Proc Tabulate MISSING
Data=bweight
 Out=by_output;
Class black married smoke;
By ed visit;
Tables black married smoke;
Run;
```

```
Proc Tabulate MISSING
Data=bweight
 Out=x_output;
Class black married smoke;
Class ed visit;
Tables (ed*visit)*(black married smoke);
Run;

Proc Sort Data = x_output;
      By ed visit;
Run
```

Figure 17. Default by group approach, and an alternative method using a crossing of classification variables.

This generalized form of the table specification will work with PROC FREQ, PROC MEANS and PROC TABULATE. For PROC REPORT, the equivalent would be turning by variables into group variables. The output data set on the right will not be sorted, but we can easily do so afterwards. It is far more efficient to sort the summarized data set.

Crossing the by variables

To make this approach work we need to cross all of our by variables in the by group portion of the table specification. If we are implementing this in a macro context, suppose we have a macro parameter to store the list of by variables:

```
%let bygroup = ed visit;
```

Then we can automatically create the asterisk-delimited list `ed*visit` by using the translate function:

```
%let by_spec = %sysfunc(translate(&bygroup.,'*',' '));
```

The translate function replaces all spaces in the string with asterisks [on a side note, the %sysfunc(translate()) trick is also good for converting by groups to comma-delimited lists for PROC SQL]. We can use this in the table statement:

```
Tables (&by_spec.)*(<original table specification>);
```

Accommodating the new _TYPE_ variable

Using the crossing technique for by groups will produce the same cells in your output data set, but the _TYPE_ variable will be different. By variables are ignored when producing the _TYPE_ variable, but if we add by variables to the class statement they will become '1's in the _TYPE_ statement. I recommend putting all of your by variables in the beginning or end of your class statement (or, as above, in a subsequent class statement). That way the '1's for by group variables will be together and can be easily ignored when searching the _TYPE_ variable to identify the variables that contributed to the cell. For example, _TYPE_ in the output data set from figure 17 contains five bytes: three 0/1 indicators for black, married and smoke, and two '1's at the end indicating by group variables ed and visit.

TYPE
100

TYPE
10011

Figure 18. Equivalent values of the _TYPE_ variable from the by_output and x_output data sets in Figure 17.

To accommodate this, I add a startpos argument to the findc() function used to determine VarName and VarValue:

```
VarName =vname(indicators[findc(_type_,'1',-3)]);
VarValue=indicators[findc(_type_,'1',-3)];
```

This starts the search in the third position and goes left, so only the first three bytes (our three actual class variables) are searched. Alternatively, if the by variables were first in our class list, we could use a startpos argument of one plus the number of by group variables (in this case, with two by variables, it would be `findc(_type_,'1',3)`).

Using formatted values for by groups

There is another major advantage of using a class variable approach for by groups. This allows you to use formats to create by groups. By default, summary procedures create by groups from the *unformatted* values of the by variables.

This is no trivial benefit, as it means we can also use ***multilabel*** formats to form our by groups. PROC TABULATE, PROC MEANS and PROC REPORT all support use of multilabel formats, which allow the same value to be assigned multiple labels. These are very helpful for variables with nested categories such as age or geographic region.

For example, when using PROC REPORT with the ExcelXP tagset we can export a report for each by group onto a separate tab of the workbook. Multilabel formats are supported for group variables, but not for by groups. However if we use a multilabel format to first summarize the data set with the by variables as class (or group) variables, then the column in the output data set will contain the formatted values for all categories. We can use this new column as the by variable for a subsequent PROC REPORT step, and create a report tab for each label of the multilabel format.

CONCLUSION

Although it is convenient to use a single procedure to both summarize our data and generate an output table, these are really two distinct processes. As long as we like the looks of the output table, we may neglect the output data set lying underneath. But when we find ourselves trying to make a table that the standard summary procedures cannot create automatically, we are stuck. We should instead view summary procedures as the first step in table creation.

By employing PROC TRANSPOSE and selected array and string functions, we can often manipulate the output data set into something that resembles our desired output table. While using multiple steps to generate a table may seem cumbersome, it allows us to take full advantage of the best each procedure has to offer. For example, rather than try to create multiple rows of summary statistics with PROC REPORT, which requires creating several dummy variables, we can simply use PROC MEANS and append the statistics to the original detail report. Once we have rearranged our output data set into the form we need, we can use the extensive style and formatting options of PROC REPORT to bring our output table to life.

REFERENCES

National Center for Chronic Disease Prevention and Health Promotion. “HealthStyles Survey — Breastfeeding Practices: 2012.” CDC Website. August 16, 2013. Available at http://www.cdc.gov/breastfeeding/data/healthstyles_survey/survey_2012.htm#2012

SAS Institute. “Example 72.3 Quantile Regression Analysis of Birth-Weight Data.” SAS/STAT(R) 9.2 User's Guide, Second Edition. August 16, 2013. Available at http://support.sas.com/documentation/cdl/en/statug/63033/HTML/default/viewer.htm#statug_qreg_sect033.htm

RECOMMENDED READING

This paper from Westat describes a comprehensive approach to standardizing output from statistical procedures.

- Long, Stuart, Ed Heaton and Dan Scharf. 2011. “ODS Output Datasets that Work for You.” *Proceedings of the SAS Global 2011 Conference.* Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings11/249-2011.pdf

CONTACT INFORMATION

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

Dylan Ellis
Mathematica Policy Research
1100 1st Street NE, 12th Floor
Washington, DC 20002-4221
(202) 554-7542
djellis@mathematica-mpr.com
www.linkedin.com/in/dylanellis