PharmaSUG 2014 - Paper BB15

Atypical Application of PROC SUMMARY

John Henry King, Ouachita Clinical Data Services, Inc., Caddo Gap AR

Abstract

This paper describes using PROC SUMMARY combined with a few other DATA and PROC steps to produce stacked frequency tables of a large or small number of categorical variables. This technique uses a single pass of the analysis data preserving the variable order and the order of the levels of each variable, while providing complete rows and complete columns of zero counts (something from nothing) and BIGN. We will discuss applications of PROC FORMAT options NOTSORTED and MULTILABEL; PROC SUMMARY options CHARTYPES, PRELOADFMT, COMPLETETYPES, DESCENDTYPES, LEVELS, WAYS, ORDER=DATA, and the TYPES statement; the PROC FREQ WEIGHT statement, ZEROS option and ODS table CROSSTABFREQS; a data step view; PROC TRANSPOSE statements ID and IDLABEL; interesting functions FINDC, VNAME, VLABEL, VVALUE, VFORMATD, CATX and CATS all in the context of a simple extensible program that you can use every day.

Introduction

The goal is to write a program to produce two-way tables of any number, a few hundred at least, of discrete variables (row variables), crossed with a treatment variable (column variable) to ultimately be displayed in a summary table or used in a programmed QC. PROC SUMMARY is a good tool for this task it is fast and allows for processing character and numeric discrete variables, to establish an order for the levels of each variable, to create counts for levels that do not exist in the data, and it allows multi-level groupings. The following is a template for the PROC SUMMARY we will use -

```
proc summary completetypes chartype descendtypes missing;
   class list-of-discrete-variables  treatment-variable / preloadfmt order=data mlf;
   format class-variable1 format1 class-variable2 format2 …;
   types treatment-variable (list-of-discrete-variables)*treatment-variable;
   output out=sas-data-set(index=(_way_)) / levels ways;
   run;
```

it is the basic PROC SUMMARY needed to generate counts for hundreds of discrete analysis variables, in one pass of the data.

Details

- PROC SUMMARY when we use PROC SUMMARY with no VAR statement it implies that we are counting observations for each level of the CLASS variables. The same thing we could do with PROC FREQ but PROC FREQ does not support pre-loaded and multi-label formats. We will use PROC FREQ when it's time to calculate percents.
    - COMPLETETYPES interacts with PRELOADFMT to give zero counts for CLASS levels that do not exist in the data. This saves us the headache of trying to add rows or columns of zeros after the fact.

  - CHARTYPE requests the _TYPE_ variable created by PROC SUMMARY as a character variable instead of numeric. This is required with many CLASS variables and suites our processing later on.
  - DESCENDTYPES requests PROC SUMMARY to sort the output data by descending order of _TYPE_. This will make the row variables in our analysis ordered in the same order they appear on the class statement from left to right. This is useful because this order of the discrete variables is the order we want in the output.
  - MISSING is needed to insure no data is discarded for CLASS variables with missing values. How missing values are handled in the summary table depends on the application, in this example missing levels will not be included in the final summary and will not contribute to the denominator for a CLASS variable. The final output will include a row of n as the first summary row for each CLASS variable, showing the denominator used for each CLASS variable.
- CLASS this is the list of discrete variables to form groups for counting, these variables may be either character or numeric. We will list the column or treatment variable last, so we can identify it using _TYPE_ it will be associated with last byte of _TYPE_.
  - PRELOADFMT directs PROC SUMMARY to build a table for each CLASS variable using the associated FORMAT. This allows PROC SUMMARY to create observations with zeros counts.
  - ORDER=DATA specifies that the ORDER of summary rows be in the order defined by the pre loaded format. This allows us to define the order of the rows by the way we order the rows of the format (details on that to follow).
  - MLF specifies processing for multi-label formats, formats with overlapping groups, for example a “Total” column. MLF has two side effects that we will exploit even if we don’t actually have any multi-label formats we will use the option. Those side effects are
    - All CLASS variables are converted to character
    - Significantly faster and is especially noticeable when there are hundreds of CLASS variables.
- FORMAT this is the regular format statement we all know and love. If the formats have already been associated with the variables this statement can be omitted. The important part is that all variables have a value format. CLASS variables can share formats but all CLASS variables should have a format and it should be pre-loadable to achieve the desired effect of PROLOADFMT and ORDER=DATA. Following are example VALUE statements that use the NOTSORTED and MULTILABEL options for the column variable. We will not use multi-label formats for the row variables.

```
value trt(notsorted multilabel) 1='Placebo' 2='Active' 1,2='Total';
value $sex(notsorted) 'F'='Female' 'M'='Male';
value race(notsorted) 1='White' 2='Black' 3='Hispanic' 4='Other';
```

Without the NOTSORTED option the class levels are sorted by the formatted values which may not be the order we want.

- TYPES this statement directs PROC SUMMARY to only create the table crossings we specify. The default for PROC SUMMARY is to create tables for all possible crossings of all variables listed in the CLASS statement which is impossible when there are a large number of class variables, but with TYPES we are able to restrict the analysis to the tables of interest. The syntax is very similar to the PROC FREQ TABLES statement.
    - *Treatment-variable* listed alone for a one-way table, this is BIGN. Number of subjects.
    - *(list-of-discrete-variables)*treatment-variable* requests two-way tables for each discrete analysis variable crossed with the treatment variable.
- OUTPUT as the statement name implies describes the output SAS® data set we want PROC SUMMARY to create.
    - out=*sas-data-set(index=(_way_))* names the output data set and in our example we will create a simple index. The index allows us to retrieve the data for BIGN slightly faster when there are a lot of analysis variables.
    - LEVELS is an option that creates an index variable for the class levels. It is useful to our program for processing BIGN.
    - WAYS is an option that creates a variable that indicates the number of variables in a summary in our program the variable _WAY_ will have value 1 or 2.

The output data set created from this single PROC SUMMARY can be easily manipulated to produce a data set formatted for display as a summary table or used in a programmed QC. Understanding the output data produced here and the options used and their interaction is essential to the process.

- Variables in the output data
    - Each of the variables listed in the CLASS statement. Numeric variables are converted to character due to the use of MLF option.
    - _TYPE_ a character variable which encodes information about the variable(s) summarized in each observation. Each byte of _TYPE_ is associated with a CLASS variable in the same order the class variables are specified in the CLASS statement(s). The first CLASS variable being associated with byte 1 of _TYPE_ and the second CLASS variable associated with byte 2 and so on for all n CLASS. The length of _TYPE_ being equal to the total number of CLASS variables. When
    - _FREQ_ a numeric variable of counts.
    - _WAY_ a variable to indicate if the summary is a 1 way or 2 way table.
    - _LEVEL_ an index that will allow us to preserve the order of the rows should we decide to mix them up.
- Observations
    - One observation for each level of each discrete variable crossed with the treatment variable (_WAY_=2), including any levels that don't exist in the data where _FREQ_=0, as defined by the format associated with the variable.
    - One observation for each level of the treatment variable for the _WAY_=1 observations. This is BIGN.

For example consider the following output using SASHELP.CLASS with a couple of variables added RACE and TRT to make it look somewhat like a subject level data set ADSL. Each CLASS variable SEX RACE AGE and TRT were given value formats to define the order and value labels for the codes associate with each. TRT also uses a MULTILABEL format to create "Total".

Here in Figure 1 we can examine and understand how _TYPE_ is related to the CLASS variables. In this example there are four CLASS variables implying _TYPE_ will have length character 4. The value of each byte of _TYPE_ is either 1 or 0; indicating that the i^{th} CLASS variable is in the summary (1) or not in the summary (0). For example the first three observations where _TYPE_='0001' involve only one CLASS variable in this case the 4^{th} CLASS variable because the 4^{th} byte is 1.

Obs	Sex	Race	Age	Trt	_WAY_	_TYPE_	_LEVEL_	_FREQ_
1				Placebo	1	0001	1	6
2				Active	1	0001	2	13
3				Total	1	0001	3	19
4			10 and Under	Placebo	2	0011	1	0
5			10 and Under	Active	2	0011	2	0
6			10 and Under	Total	2	0011	3	0
7			Pre-teen	Placebo	2	0011	4	2
8			Pre-teen	Active	2	0011	5	5
9			Pre-teen	Total	2	0011	6	7
10			Teen	Placebo	2	0011	7	4
11			Teen	Active	2	0011	8	8
12			Teen	Total	2	0011	9	12
13		White		Placebo	2	0101	1	4
14		White		Active	2	0101	2	7
15		White		Total	2	0101	3	11
16		Black		Placebo	2	0101	4	1
17		Black		Active	2	0101	5	6
18		Black		Total	2	0101	6	7
19		Hispanic		Placebo	2	0101	7	1
20		Hispanic		Active	2	0101	8	0
21		Hispanic		Total	2	0101	9	1
22		Other		Placebo	2	0101	10	0
23		Other		Active	2	0101	11	0
24		Other		Total	2	0101	12	0
25	Female			Placebo	2	1001	1	2
26	Female			Active	2	1001	2	7
27	Female			Total	2	1001	3	9
28	Male			Placebo	2	1001	4	4
29	Male			Active	2	1001	5	6
30	Male			Total	2	1001	6	10

Figure 1

Similarly observations 13-24 being the 2-way table of RACE*TRT is encoded in _TYPE_ as '0101', in other words the observations are associated with the summary of CLASS variable 2 and 4.

We will use _TYPE_ and an ARRAY constructed from the list of class variables to process the CLASS variables without having to use the names of the CLASS variables, making our program dynamic with regards their number and order. While PROC SUMMARY does the majority of the heavy lifting we still need to process the output into a data set that we can pass to PROC FREQ to calculate percentages and finally transposed on the treatment columns for display with PROC REPORT.

Example

The following is a full working example with description of the steps and details of the important and interesting features.

Value format for analysis variables.

For each variable in the analysis a value label format is created, to define the levels of each and to provide descriptive row labels in the summary table. The NOTSORTED option allows the format to communicate the order of each level in the output, for example in the TRT format below the value 2='Placebo' is listed first followed by 1='Active' the NOTSORTED option preserves this ordering and allows this order to determine the order in the output. The default being to order the output by the collating sequence of the formatted values

```
proc format;
   value trt(notsorted multilabel) 2='Placebo' 1='Active' 1,2='Total';
   value $sex(notsorted) 'F'='Female' 'M'='Male';
   value race(notsorted) 1='White' 2='Black' 3='Hispanic' 4='Other';
   value age(notsorted) low-10 = '10 and Under' 11-12='Pre-teen' 13-high='Teen';
   value bmi(notsorted) low-<17='Slight' 17-<18.5='Medium' 18.5-high='Husky';
   value height(notsorted) low-<59.8 = 'Short' 59.8-<65.3='Average' 65.3-high='Tall';
   value weight(notsorted) low-<85='Light' 85-<102.5='Average' 102.5-high='Heavy';
   run;
```

Analysis data ADSL.

I like to use SASHELP.CLASS as a model for ADSL, by adding treatment and a couple of other variables to make it a bit more interesting. Each analysis variable is given a descriptive label and this label will become the break label beginning each section in the stacked two-way table summary.

```
data ADSL;
   Trt  = rantbl(12345,.5);
   set sashelp.class;
   Race = rantbl(12345,.5,.4);
   Bmi  = (weight*703) / height**2;
   if      _n_ eq 3 then call missing(race);
   else if _n_ eq 4 then call missing(sex);
   else if _n_ eq 6 then call missing(weight,bmi);
   attrib age    format=age.    label='Age Group';
   attrib height format=height. label='Height Range';
   attrib weight format=weight. label='Weight Range';
   attrib sex    format=$sex.   label='Gender';
   attrib race   format=race.   label='Ethnic Origin';
   attrib bmi    format=bmi.    label='BMI Class';
   attrib trt    format=trt.    label='Treatment';
   run;
```

Parameterization.

Macro variables DATA, TRT, and DVARS can be thought of as parameters to this dynamic program. They provide the entire variable part of the code.

```
%let data = adsl;
%let trt = trt;
%let dvars = Sex Age Height Weight Race Bmi;
```

See my paper "Atypical Applications of Proc Transpose" for details on how to process DVARS so that it can accept a "SAS Variable List" as input.

PROC SUMMARY to obtain counts.

Following is the call to PROC SUMMARY to get counts for the entire set of two-way tables and the one-way table to use for BIGN. This is the same as we discussed in detail above, it produces the data set shown in Figure 2 output from proc summary, 20 observations..

```
proc summary data=&data completetypes chartype missing descendtypes;
   class &dvars &trt / preloadfmt order=data mlf;
   types &trt (&dvars)*&trt;
   output out=counts(index=(_way_)) / levels ways;
   run;
```

Hopefully, from the discussion above, you can see how _TYPE_ relates to the CLASS variables. This data is sorted by descending _TYPE_ due to the PROC SUMMARY statement option DESCENDTYPES, but this does not alter the relationship between _TYPE_ and the variables listed on the CLASS statement.

Obs	Sex	Age	Height	Weight	Race	Bmi	Trt	_WAY_	_TYPE_	_LEVEL_	_FREQ_
1	Female						Placebo	2	1000001	1	2
2	Female						Active	2	1000001	2	6
3	Female						Total	2	1000001	3	8
4	Male						Placebo	2	1000001	4	4
5	Male						Active	2	1000001	5	6
6	Male						Total	2	1000001	6	10
7							Placebo	2	1000001	7	0
8							Active	2	1000001	8	1
9							Total	2	1000001	9	1
10		10 and Under					Placebo	2	0100001	1	0
11		10 and Under					Active	2	0100001	2	0
12		10 and Under					Total	2	0100001	3	0
13		Pre-teen					Placebo	2	0100001	4	2
14		Pre-teen					Active	2	0100001	5	5
15		Pre-teen					Total	2	0100001	6	7
16		Teen					Placebo	2	0100001	7	4
17		Teen					Active	2	0100001	8	8
18		Teen					Total	2	0100001	9	12
19			Short				Placebo	2	0010001	1	2
20			Short				Active	2	0010001	2	4

Figure 2 output from proc summary, 20 observations.

Now we need to calculate percents and I like to use PROC FREQ for this because it is easy. But we need to change the data somewhat so that it the information in the CLASS variables is normalized into a single variable. We could skip this and go straight to PROC FREQ but it will be too inefficient when there are many CLASS variables and we need to end up with the normalized format for the summary table anyway.

Normalize the COUNTS data obtained from PROC SUMMARY

This step normalizes the discrete analysis variables into a set of information variables that will facilitate the processing going forward. We will create an order variable from _TYPE_ and retrieve the variable name and label using functions VNAME and VLABEL.

```
data normalized;
   length _VARNUM_ 8 _VARNAME_ $32 _VARLABEL_ $128 _VALUE_ $16;
   set counts;
   where _way_ eq 2;
   array _d[*] &dvars;
   _varnum_   = findc(_type_,'1');
   _varname_  = vname(_d[_varnum_]);
   _varlabel_ = vlabel(_d[_varnum_]);
   _value_    = left(_d[_varnum_]));
   if _value_ in(' ','.') then delete;
   drop &dvars _type_;
   run;
```

Details of data normalized:

- LENGTH to define the information variables we are creating, you should modify length of _VARLABEL_ and _VALUE_ if applicable, make them big enough but don't go crazy.
- WHERE to subset the data to the two way tables. _WAY_ is create by the PROC SUMMARY output statement option WAYS.
- ARRAY we are able to create an array of the discrete analysis variables even when as in this example some were numeric and others were character because the MLF option on the CLASS statement converts all numeric class variables to character. This is fine because we are no longer interested in working with the underlying variables just the values. The array allows us to reference the unknown and variable number of discrete analysis variables with an index freeing the program from the further burden of knowing anything about the variable names.
- _VARNUM_ is created by finding the position of the first 1 in _TYPE_. This is the index to the variable in the array (in CLASS statement order) and provides an order variable for the set of 2-way tables. For example if _TYPE_ equals '001001' the value returned is 3 to reference element 3 of array _D.
- _VARNAME_ is simply the name of the variable for which the 2-way table summarizes. The VNAME function is used to lookup this name from the array.
- _VARLABEL_ is the variable label and is derived with the VLABEL function. This variable can be used in a report as the label for each 2-way table.
- _VALUE_ is the formatted value of the discrete analysis variable. These values have been converted to character by the MLF option. Missing numeric variables will be identified by '.'
- DELETE missing observations those with _VALUE_ blank or '.'. Missing values will not be summarize and displayed in the output and will not contribute to the denominator.
- DROP the class variables and _TYPE_, we no longer need these.

Figure 3 normalized data, shows the output created by the data step where former variables are now values of variables that we control; with the name saved in _VARNAME_, the list order saved in _VARNUM_, the LABEL saved in _VARLABEL_ and the values saved in _VALUE_.

Obs	_VARNUM_	_VARNAME_	_VARLABEL_	_VALUE_	Trt	_FREQ_
1	1	Sex	Gender	Female	Placebo	2
2	1	Sex	Gender	Female	Active	6
3	1	Sex	Gender	Female	Total	8
4	1	Sex	Gender	Male	Placebo	4
5	1	Sex	Gender	Male	Active	6
6	1	Sex	Gender	Male	Total	10
7	2	Age	Age Group	10 and Under	Placebo	0
8	2	Age	Age Group	10 and Under	Active	0
9	2	Age	Age Group	10 and Under	Total	0
10	2	Age	Age Group	Pre-teen	Placebo	2
11	2	Age	Age Group	Pre-teen	Active	5
12	2	Age	Age Group	Pre-teen	Total	7
13	2	Age	Age Group	Teen	Placebo	4
14	2	Age	Age Group	Teen	Active	8
15	2	Age	Age Group	Teen	Total	12
16	3	Height	Height Range	Short	Placebo	2
17	3	Height	Height Range	Short	Active	4
18	3	Height	Height Range	Short	Total	6
19	3	Height	Height Range	Average	Placebo	2
20	3	Height	Height Range	Average	Active	5

Figure 3 normalized data

PROC FREQ to obtain column percents for each 2-way table.

There is any number of ways to calculate the parentages, using PROC FREQ is easy and the output produced by ODS OUTPUT CROSSTABFREQS makes further processing easy. Notice that with the normalized data we no longer need be concerned with the names of the variables, this step is same for one variable or 100.

```
ods select none;
proc freq data=normalized order=data;
   by _varnum_ _varname_ _varlabel_;
   tables _value_ * &trt / norow nopercent;
   weight _freq_ / zeros;
   ods output CrossTabFreqs=CrossTabFreqs(index=(_type_));
   run;
ods select all;
```

- ODS SELECT NONE allows us to turn off all printed output from PROC FREQ without having to know what destinations are open.
- The PROC FREQ statement includes the ORDER=DATA option to preserved the order of the table variable _VALUE_ and TRT as established by the options used with PROC FORMAT and PROC SUMMARY.
- The BY statement includes the information variables created in the previous step.
- The TABLES statement specifies two options to discards percentages that we are not interested in.
- We use the WEIGHT statement because we have summary data. _FREQ_ was created by PROC SUMMARY and contains the counts, the ZEROS options is included to have PROC FREQ process cells with zero counts. ZEROS is a relatively new option and is important when there are zero counts in the data.

Obs	_VARNUM_	_VARNAME_	_VARLABEL_	Table	_VALUE_	Trt	_TYPE_	_TABLE_	Frequency	ColPercent	Missing
1	1	Sex	Gender	Table _VALUE_ * Trt	Female	Placebo	11	1	2	33.3333	.
2	1	Sex	Gender	Table _VALUE_ * Trt	Female	Active	11	1	6	50.0000	.
3	1	Sex	Gender	Table _VALUE_ * Trt	Female	Total	11	1	8	44.4444	.
4	1	Sex	Gender	Table _VALUE_ * Trt	Female		10	1	16	.	.
5	1	Sex	Gender	Table _VALUE_ * Trt	Male	Placebo	11	1	4	66.6667	.
6	1	Sex	Gender	Table _VALUE_ * Trt	Male	Active	11	1	6	50.0000	.
7	1	Sex	Gender	Table _VALUE_ * Trt	Male	Total	11	1	10	55.5556	.
8	1	Sex	Gender	Table _VALUE_ * Trt	Male		10	1	20	.	.
9	1	Sex	Gender	Table _VALUE_ * Trt		Placebo	01	1	6	.	.
10	1	Sex	Gender	Table _VALUE_ * Trt		Active	01	1	12	.	.
11	1	Sex	Gender	Table _VALUE_ * Trt		Total	01	1	18	.	.
12	1	Sex	Gender	Table _VALUE_ * Trt			00	1	36	.	0
13	2	Age	Age Group	Table _VALUE_ * Trt	10 and Under	Placebo	11	1	0	0.0000	.
14	2	Age	Age Group	Table _VALUE_ * Trt	10 and Under	Active	11	1	0	0.0000	.
15	2	Age	Age Group	Table _VALUE_ * Trt	10 and Under	Total	11	1	0	0.0000	.
16	2	Age	Age Group	Table _VALUE_ * Trt	10 and Under		10	1	0	.	.
17	2	Age	Age Group	Table _VALUE_ * Trt	Pre-teen	Placebo	11	1	2	33.3333	.
18	2	Age	Age Group	Table _VALUE_ * Trt	Pre-teen	Active	11	1	5	38.4615	.
19	2	Age	Age Group	Table _VALUE_ * Trt	Pre-teen	Total	11	1	7	36.8421	.
20	2	Age	Age Group	Table _VALUE_ * Trt	Pre-teen		10	1	14	.	.

Figure 4 CrossTabFreqs stacked tables from PROC FREQ

The output show in Figure 4 is the ODS CROSSTABFREQS data which contains the same two-way tables we created in the previous step with the addition of COLPERCENT the percentages we need for our report. The counts are now called FREQUENCY and there are some new variables TABLE, _TABLE_ and MISSING that we will not use and it also includes the denominator used for the percents identified as _TYPE_=10. _TYPE_ is also a new variable but should not be confused with _TYPE_ created by PROC SUMMARY. We will use the _TYPE_=10 data to create a row of denominator counts in our report.

BIGN.

```
data bign(index=(&trt) keep=&trt _colNum_ _bign_);
   set counts;
   where _way_ eq 1;
   rename _level_=_colNum_ _freq_=_bign_;
   run;
```

Obs	Trt	_colNum_	_bign_
1	Placebo	1	6
2	Active	2	13
3	Total	3	19

Using the single one-way table we requested in the TYPES statement, create data BIGN by sub-setting on _WAY_ and renaming variables _LEVEL_ and _FREQ_ as _COLNUM_ and _BIGN_ respectively. _COLNUM_ is the column order as defined by the value format associated with &TRT and will be used to name the column variables when the data are transpose. _BIGN_ is number of subjects in each treatment group, BIGN and will be added to the formatted value of &TRT to create labels for the column variables.

Row labels and formatted table cells.

At this point we start to get into actual table formatting and will need to be tailored to the requirements found in the table shells. Here I will format the two-way tables with the denominator as the first row follow by the category levels; this effectively handles missing and makes it clear how the percents are calculated.

```
data crossView / view=crossView;
   set crossTabFreqs(where=(_type_ eq '01')) crossTabFreqs(where=(_type_ eq '11'));
```

```
   by _varnum_ _varname_ _varlabel_ _type_;
   run;
```

- A data step view is used because we don't really need an intermediate data we just need to run the reorder the rows of CROSSTABFREQS and pass them through to the next step.
- The SET statement with WHERE data set option and BY are used to interleave the data with itself so that _TYPE_=01 can be moved ahead of _TYPE_=11, notice in Figure 4 that observation 4 is _TYPE_=01 but want it to precede the _TYPE_=11 observations. This is done for each _VARNAME_ with the self interleave.

```
data StackedTables;
   retain _VARNUM_ _VARNAME_ _VARLABEL_ _TYPE_ _VLEVEL_ _VALUE_;
   set CrossView;
   by _varnum_ _varname_ _varlabel_ _type_ _value_ notsorted;
   length _VLEVEL_ 8;
   if first._varnum_ then _vlevel_ = 0;
   if first._value_  then _vlevel_ + 1;
   if _type_ eq '01' then _value_ = 'n';
   set bign key=&trt/unique;
   length _IDLABEL_ $64;
   _idlabel_ = catx('~',&trt,cats('(N=',_bign_,')'));
   length _CELL_ $32;
   select(_type_);
      when('01') _cell_ = vvalue(frequency);
      when('11') select(frequency);
         when(0) _cell_ = vvalue(frequency);
         otherwise do;
            colPercent = round(colPercent,10**-vformatd(colPercent));
            _cell_ = catx(' ',vvalue(frequency),cats('(',vvalue(colPercent),'%)'));
            end;
         end;
      otherwise;
      end;
   format colPercent 8.1 frequency 8.0-L;
   label  _type_=' ';
   rename _type_=_STYPE_;
   drop table _table_ frequency colpercent missing _bign_;
   run;
```

- RETAIN is used to put the variables in desired order.
- SET with BY and NOTSORTED option used to create the set up FIRST DOT variables to create _VLEVEL_. Using NOTSORTED is part of the reason for using the data step view from the previous step as input to this step. The NOSTORTED option is only allowed on simple SET statements we could not have used it on the self interleave above.
- _VLEVEL_ is an index for each unique level of each discrete variable. If it created by incrementing a counter on FIRST._VALUE_. Remember the values of each discrete analysis variables have been ordered by the combination of NOTSORTED value statement option and the PROC SUMMARY options PRELOADFMT and ORDER=DATA.
- The denominators or little Ns are identified by _TYPE_=01 but _VALUE_ is blank coming from PROC FREQ. The label text is added here in the IF statement.
- BIGN is stored in a data set that is index by the value formatted value of &TRT. A SET statement with the KEY option is used to retrieve the values by the KEY. The value of BIGN will be used in

the treatment column labels and _COLNUM_ will be to name the treatment column variables. The UNIQUE option allows the same observation to be accessed more than one time.

- CAT(X S) functions are used to create _IDLABEL_. These functions have features that make them a bit easier to use than the concatenation operator. The values of _IDLABEL_ will be used to create variable labels for the new variables created from the values of &TRT when the data are transposed into the final display data. Creating these labels this way is important because it allows the program to be dynamic. There is no need to change the program if the number of treatment columns changes. No messy macro variables and the treatment column labels flow right into PROC REPORT with no other reference to them needed.
- The select statement determines how the values of _CELL_, the formatted table cell data, will be formatted depending on the value of _TYPE_.
    - When _TYPE_=01 _CELL_ will receive the value of FREQUENCY. The VVALUE function returns the formatted value of FREQUENCY as a character string. In this example the format is 8.1-L where -L modifies the default justification of the numeric format from right to left justified.
    - When _TYPE_=11 _CELL_ will receive the count and percent in parenthesis unless the count is 0 when the percent of omitted. I think this is a bit less busy especially when there are a lot of zeros.
        - The value of COLPERCENT is rounded to the precision implied by the number of decimals of the associated format 8.1. The function VFORMATD returns the value of D from a format in this case 1. In the round function 10 is raised to the negative power of the value returned from VFORMATD to produce 0.1 as the rounding factor. Using VFORMATD in this way allows the program to derive the value from a common metadata object that can be easily varied.
        - Finally _CELL_ is formatted using a combination CATX and CATS to produce the count and percent in the style ‘ccc (ppp.p%)’.
    - FORMAT specifies the formats for COLPERCENT and FREQUENCY.
    - The label is removed from _TYPE_ and it is renamed. This variable encodes the summary type 01 is the denominator row labeled ‘n’. It could be used in PROC REPORT to change the attributes of the row with a CALL DEFINE to make is stand out in some way or it could be used to remove the row all together if it were not needed.

The data created by this step is shown in Figure 5 STACKEDTABLES. This data looks simple and for the most part it is but I think it is important to consider that the program to produce it is dynamic and requires no modification for different variables with completely different attributes and class levels. The only requirement is to define the value label formats for each variable which are required anyway to have the proper row and column labels for the final table.

Obs	_VARNUM_	_VARNAME_	_VARLABEL_	_VALUE_	Trt	_STYPE_	_colNum_	_IDLABEL_	_CELL_	_VLEVEL_
1	1	Sex	Gender	n	Placebo	01	1	Placebo~(N=6)	6	1
2	1	Sex	Gender	n	Active	01	2	Active~(N=13)	12	1
3	1	Sex	Gender	n	Total	01	3	Total~(N=19)	18	1
4	1	Sex	Gender	Female	Placebo	11	1	Placebo~(N=6)	2 (33.3%)	2
5	1	Sex	Gender	Female	Active	11	2	Active~(N=13)	6 (50.0%)	2
6	1	Sex	Gender	Female	Total	11	3	Total~(N=19)	8 (44.4%)	2
7	1	Sex	Gender	Male	Placebo	11	1	Placebo~(N=6)	4 (66.7%)	3
8	1	Sex	Gender	Male	Active	11	2	Active~(N=13)	6 (50.0%)	3
9	1	Sex	Gender	Male	Total	11	3	Total~(N=19)	10 (55.6%)	3
10	2	Age	Age Group	n	Placebo	01	1	Placebo~(N=6)	6	1
11	2	Age	Age Group	n	Active	01	2	Active~(N=13)	13	1
12	2	Age	Age Group	n	Total	01	3	Total~(N=19)	19	1
13	2	Age	Age Group	10 and Under	Placebo	11	1	Placebo~(N=6)	0	2
14	2	Age	Age Group	10 and Under	Active	11	2	Active~(N=13)	0	2
15	2	Age	Age Group	10 and Under	Total	11	3	Total~(N=19)	0	2
16	2	Age	Age Group	Pre-teen	Placebo	11	1	Placebo~(N=6)	2 (33.3%)	3
17	2	Age	Age Group	Pre-teen	Active	11	2	Active~(N=13)	5 (38.5%)	3
18	2	Age	Age Group	Pre-teen	Total	11	3	Total~(N=19)	7 (36.8%)	3
19	2	Age	Age Group	Teen	Placebo	11	1	Placebo~(N=6)	4 (66.7%)	4
20	2	Age	Age Group	Teen	Active	11	2	Active~(N=13)	8 (61.5%)	4

Figure 5 STACKEDTABLES

Transpose treatment rows to variables.

One last step to transpose the treatment rows to columns and we will have a nice data set that can be used for QC or passed to PROC REPORT to make a table.

```
Proc transpose data=StackedTables out=Display(drop=_name_) prefix=col;
   by _varnum_ _varname_ _varlabel_ _type_ _vlevel_ _value_;
   var _cell_;
   id _colnum_;
   idlabel _idlabel_;
   run;
```

- BY statement list all of the information variables we created above.
- VAR is the variable to transpose _CELL_ the formatted table cells.
- ID names the new variables using the PREFIX and the value of _COLUMN_.
- IDLABEL creates labels for the new variables using the values in _IDLABEL_.

Figure 6 CONTENTS shows the new COLn variables and their labels. If there were a different number of treatments we would have a different number of COLn variables but the other variables will remain the same with any number of discrete analysis variables with any variable and unknown number of levels for each.

Variables in Creation Order				
#	Variable	Type	Len	Label
1	_VARNUM_	Num	8	
2	_VARNAME_	Char	32	
3	_VARLABEL_	Char	128	
4	_STYPE_	Char	2	
5	_VLEVEL_	Num	8	
6	_VALUE_	Char	16	
7	col1	Char	32	Placebo~(N=6)
8	col2	Char	32	Active~(N=13)
9	col3	Char	32	Total~(N=19)

Figure 6 CONTENTS

In Figure 7 DISPLAY we have the PROC PRINT output of the last PROC TRANSPOSE step. This data is formatted and structured in such a way that it can be easily displayed using PROC REPORT or used in programmed QC. While the specifics of programmed QC are beyond the scope of this discussion I think we can all see how this data could be adapted for that purpose.

Obs	_VARNUM_	_VARNAME_	_VARLABEL_	_STYPE_	_VLEVEL_	_VALUE_	col1	col2	col3
1	1	Sex	Gender	01	1	n	6	12	18
2	1	Sex	Gender	11	2	Female	2 (33.3%)	6 (50.0%)	8 (44.4%)
3	1	Sex	Gender	11	3	Male	4 (66.7%)	6 (50.0%)	10 (55.6%)
4	2	Age	Age Group	01	1	n	6	13	19
5	2	Age	Age Group	11	2	10 and Under	0	0	0
6	2	Age	Age Group	11	3	Pre-teen	2 (33.3%)	5 (38.5%)	7 (36.8%)
7	2	Age	Age Group	11	4	Teen	4 (66.7%)	8 (61.5%)	12 (63.2%)
8	3	Height	Height Range	01	1	n	6	13	19
9	3	Height	Height Range	11	2	Short	2 (33.3%)	4 (30.8%)	6 (31.6%)
10	3	Height	Height Range	11	3	Average	2 (33.3%)	5 (38.5%)	7 (36.8%)
11	3	Height	Height Range	11	4	Tall	2 (33.3%)	4 (30.8%)	6 (31.6%)
12	4	Weight	Weight Range	01	1	n	6	12	18
13	4	Weight	Weight Range	11	2	Light	2 (33.3%)	3 (25.0%)	5 (27.8%)
14	4	Weight	Weight Range	11	3	Average	0	4 (33.3%)	4 (22.2%)
15	4	Weight	Weight Range	11	4	Heavy	4 (66.7%)	5 (41.7%)	9 (50.0%)
16	5	Race	Ethnic Origin	01	1	n	6	12	18
17	5	Race	Ethnic Origin	11	2	White	4 (66.7%)	7 (58.3%)	11 (61.1%)
18	5	Race	Ethnic Origin	11	3	Black	1 (16.7%)	5 (41.7%)	6 (33.3%)
19	5	Race	Ethnic Origin	11	4	Hispanic	1 (16.7%)	0	1 (5.6%)
20	5	Race	Ethnic Origin	11	5	Other	0	0	0
21	6	Bmi	BMI Class	01	1	n	6	12	18
22	6	Bmi	BMI Class	11	2	Slight	0	6 (50.0%)	6 (33.3%)
23	6	Bmi	BMI Class	11	3	Medium	3 (50.0%)	4 (33.3%)	7 (38.9%)
24	6	Bmi	BMI Class	11	4	Husky	3 (50.0%)	2 (16.7%)	5 (27.8%)

Figure 7 DISPLAY

Conclusion

We've looked at a method of using PROC SUMMARY to produce a large number of two-way tables and prepare the output for display. This technique requires only one pass of the raw analysis data and is extremely fast especially when compared to looping the over variables individually. I hope you can use this going forward in your work programming summaries for tables and programmed QC.

Contact Information

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

John King
Ouachita Clinical Data Services, Inc.
1769 Highway 240 West
Caddo Gap, AR 71935
870-356-3033
datanull@gmail.com