PharmaSUG 2013 - Paper TF18

The Bylaws of By Group Processing

Tracee Vinson-Sorrentino, Optum Life Sciences, Minneapolis, MN

ABSTRACT

One of the big challenges with health care claims data is that they contain multiple rows of transactions, e.g. rows, per patient, per visit, depending on the services rendered that day. Usually patients and their diseases are studied over time, and for this reason by group processing is a mandatory tool. This paper will showcase the power of by-group processing and will also cover what happens within SAS, behind the scenes. The process will be illustrated with an example of its usefulness by creating a physician targeting list, with metrics, where physicians practice in more than one location.

INTRODUCTION

This paper is for those who have spent some time in SAS, understand the basics of the Program Data Vector (PDV) and are ready to move forward in their SAS experience. By Group processing is a leap forward in synthesizing data, from simple QC steps to complex calculations and analysis. This paper will show one of those QC steps, as well as a way to use by grouping to create metrics. To illustrate, I've extracted parts of two of my recent projects to create a hypothetical project with a hot demand item – the Physician Targeting List With Metrics. I will cover the basics of using an Array and introduce several SAS Functions.

THE PRINCIPLE

By Group processing is a powerful tool that is really helpful in customizing our data. It's often used without the user realizing they've entered a by group situation. Merge, Proc Sort, Proc Summary, Proc Means, Proc Print and Proc Freq are some ways programmers can easily use a by group. A very common use, outside of the sort of course, is within a merge.

This brings us to our first ByLaws:

Bylaw # 1: You must always sort on the By variable prior to applying a By Group to a Procedure or a Datastep.
Bylaw # 2: By variables must be the same: Same size, type and variable name.

```
proc sort data=step1; by unique_id;
proc sort data=step2; by unique_id;
run;

data final;
merge step1 (in=a) step2 (in=b); by unique_id;
if a and b;
run;
```

For this example I'm merging step1 and step2 together by unique_id, where the unique_id is present in both datasets.

The merge step above is a very simple example. The power of the by group can be leveraged to check your match rate, or create some other useful variable. I use it all the time for QC steps.

For this example, the unique_id might be in only one of the datasets and I'd like to know what my match rate looks like:

```
data final;
merge step1 (in=a) step2 (in=b); by unique_id;
if a | b;
attrib Match length=$12;
if a & b then do;
        match='Both';
end;
if a &^ b then do;
        match='Step 1 Only';
end;
if b &^ a then do;
        match='Step 2 Only';
end;
proc freq; table match / list missing;
run;
```

Let's take a moment to examine the "in=" portion of the code. in= creates a variable that is used by PDV and it could be classified as either a *pseudo variable* or an *automatic variable*, because it is not sent to the output dataset. It is used for tracking whether the dataset you've attached it to is contributing to the current observation. You could call the IN variable anything you want, within the SAS variable naming conventions – as long as it is not the name of an already defined variable. In other words, it does not have to be in=a or in=b; It could very well be in=name for one and in=address for the other, if you prefer additional clarity in your code.

Another use of the By Group processing is within the Datastep. This example uses the automatic variables called first. and last., which are only available when you set the dataset with a by statement.

```
data final;
set step1;
 by unique_id;
```

PRACTICAL APPLICATION – A REPLACEMENT FOR PROC TRANSPOSE

For practical application, let's examine this in a real world scenario:

In creating a physician targeting list, the client wants a list of physicians who have treated patients with a food allergy. They want to see a patient segmentation within the physician list and they want several metrics applied: Average treatment time (in weeks), patient count, and a breakdown of patient count based on whether they were treated for anaphylactic shock prior to skin testing, or post skin testing, or not at all.

I've decided to break this up into two sections, first tackling the address issues and then creating the metrics. Later they will be joined back together to create the final deliverable.

I know my addresses are not going to be perfectly uniform. Some will have extra blanks and some will utilize the address_2 field, which is usually used for suite numbers.

	NPI	Physician	address_1	address_2	city	state	zip
1	00023468	Jones, James	1234 Main St.	Suite 101	Sas Township	CA	18104
2	00023468	Jones, James	1234 Main St.	Suite 101	Sas Township	CA	18104
3	00023468	Jones, James	5987 Back St		Sas Township	CA	18105
4	00023468	Jones, James	4443 Side St	Suite 1A	Pharma Village	CA	18106
5	00023468	Jones, James	5688 Hospital Wy	Suite 1300	Sas Township	CA	18104
6	00023468	Jones, James	420 Vacation House Dr.		Mountains	CA	18104

I've decided to create a key so I can ignore those issues - the key is to be used for a later by group step. I'm also going to clean up the address using some call routines. This leads us to:

Bylaw # 3: Make sure your By Groups are clean – spaces, commas, ect - every byte matters during processing.

```
data place_holder med_docs (keep=npi physician address_key address);
*the dataset "place_holder" is necessary because I'm creating a lot of temporary variables
 that I do not want to write out to the med_docs file. Without "place_holder", the step will
 fail due to missing variables;
set treating_physicians;

attrib address_key  length=$30
       add_key      length=$30
       address      length=$65
       address_prep length=$65;

*catx will concatenate two or more strings, removing the leading and trailing blanks,
     while inserting one or more characters of your choice between strings. This call
     routine is creating a new variable called address_prep;
call catx(', ',address_prep,address_1||' '||address_2,city,state,zip);

*cats is being used to create a match key for later use, creating a new variable called
     address_key;
call cats(add_key,npi,substr(address_1,1,10),zip);

*The function "compbl" will correct the addresses with multiple spaces, changing those
 to one space and create a new variable called address;
address=compbl(address_prep);
address_key=compress(add_key);
output med_docs;
run;
```

This is what we see now:

	NPI	Physician	address_key	address
1	00023468	Jones, James	000234681234Main18104	1234 Main St. Suite 101, Sas Township, CA, 18104
2	00023468	Jones, James	000234681234Main18104	1234 Main St. Suite 101, Sas Township, CA, 18104
3	00023468	Jones, James	000234685987Back18105	5987 Back St, Sas Township, CA, 18105
4	00023468	Jones, James	000234684443Side18106	4443 Side St Suite 1A, Pharma Village, CA, 18106
5	00023468	Jones, James	000234685688H18104	5688 Hospital Wy Suite 1300, Sas Township, CA, 18104
6	00023468	Jones, James	00023468420Vacati18104	420 Vacation House Dr., Mountains, CA, 18104

Now we're ready to work some By Group magic. Before we implement the code for that, we'll take a quick look at how the first. and last. will appear.

```
proc sort data=med_docs (keep=npi address address_key) nodupkey;
by npi address_key;
run;

data prep_docs; set med_docs; by npi address_key;
first_npi=first.npi;
last_npi=last.npi;
first_add=first.address_key;
last_add=last.address_key;
run;
```

	NPI	address_key	address	first_npi	last_npi	first_add	last_add
1	00023468	000234681234Main18104	1234 Main St. Suite 101, Sas Township, CA, 18104	1	0	1	0
2	00023468	000234681234Main18104	1234 Main St. Suite 101, Sas Township, CA, 18104	0	0	0	1
3	00023468	00023468420Vacati18104	420 Vacation House Dr., Mountains, CA, 18104	0	0	1	1
4	00023468	000234684443Side18106	4443 Side St Suite 1A, Pharma Village, CA, 18106	0	0	1	1
5	00023468	000234685688H18104	5688 Hospital Wy Suite 1300, Sas Township, CA, 18104	0	0	1	1
6	00023468	000234685987Back18105	5987 Back St, Sas Township, CA, 18105	0	1	1	1

First.NPI will equal 1 every time the PDV recognizes a new value in the field NPI and Last.NPI will equal 1 where it is the last occurrence of the value in the field NPI. Because the data is grouped by NPI first and then address_key, the First.Address_key will only equal 1 where there is a new address_key *within* the group of NPI. Same applies to last. You can see on line two where last.address_key=1. That's because we have a duplicate of that particular address, so the first and last automatic variables are different.

So what do we do with this information? Our objective is to move each unique occurrence of an address within an NPI group to a new variable, creating additional address fields. This can be done with some cumbersome Proc Transpose code, but this is a paper about By Groups, so the best way to handle this for our purpose is with an array, using our first. and last. automatic variables.

We need to take a quick side-step to decide what to set our array iterations count at. We need to know the maximum number of addresses a physician can have.

```
proc sort data=med_docs (keep=npi physician address address_key) nodupkey;
by npi address_key;
run;

data address_count; set med_docs; by npi address_key;
cnt + 1;
if first.npi then do;
       cnt=1;
end;
proc freq; table cnt / list missing;
run;
```

We now know the maximum number of addresses is 5, based on our frequency output. This is also a handy piece of information to pass on to the client – x% of their target physicians practice in 1 location, 2 locations, ect. But our goal has been achieved; we will set our array iteration count at 5, based on that being the highest value of cnt.

cnt	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	1	20.00	1	20.00
2	1	20.00	2	40.00
3	1	20.00	3	60.00
4	1	20.00	4	80.00
5	1	20.00	5	100.00

```
data prep_docs; set med_docs; by npi address_key;
attrib address1-address5 length=$65;
retain address1-address5;
array x{*} address1-address5;
if first.npi then do;
   do i = 1 to 10;
      x{i} = '                                                                    ';
   end;
   address_count = 0;
end;
address_count + 1;
x{address_count} = address;
```

The first few lines simply create and define the new address variables that will be filled within the array. Then we see the by grouping appear. When first.npi=1, the data will compile 5 times within the loop, creating a line for each iteration, and moving forward with each line the information from the previous line. The number of iterations has a value of 5, because that's what we declared in the array. If there had only been 4 addresses for Dr. Jones, the fifth iteration would have occurred, but the address5 field would have been blank.

You can see how this takes shape below:

	Physician	address1	address2	address3	address4	address5	address_count
1	Jones, James	1234 Main St. Suite 101, Sas Township, CA, 18104					1
2	Jones, James	1234 Main St. Suite 101, Sas Township, CA, 18104	420 Vacation House Dr., Mountains, CA, 18104				2
3	Jones, James	1234 Main St. Suite 101, Sas Township, CA, 18104	420 Vacation House Dr., Mountains, CA, 18104	4443 Side St Suite 1A, Pharma Village, CA, 18106			3
4	Jones, James	1234 Main St. Suite 101, Sas Township, CA, 18104	420 Vacation House Dr., Mountains, CA, 18104	4443 Side St Suite 1A, Pharma Village, CA, 18106	5688 Hospital Wy Suite 1300, Sas Township, CA, 18104		4
5	Jones, James	1234 Main St. Suite 101, Sas Township, CA, 18104	420 Vacation House Dr., Mountains, CA, 18104	4443 Side St Suite 1A, Pharma Village, CA, 18106	5688 Hospital Wy Suite 1300, Sas Township, CA, 18104	5987 Back St, Sas Township, CA, 18105	5

What we really want is that 5th line, so we add this piece of code to complete the datastep:

```
if last.npi then do;
        keep npi address1-address5 address_count;
        output;
end;
run;
```

This is what we end up with:

	NPI	Physician	address1	address2	address3	address4	address5	address_count
1	00023468	Jones, James	1234 Main St. Suite 101, Sas Township, CA, 18104	420 Vacation House Dr., Mountains, CA, 18104	4443 Side St Suite 1A, Pharma Village, CA, 18106	5688 Hospital Wy Suite 1300, Sas Township, CA, 18104	5987 Back St, Sas Township, CA, 18105	5

PRACTICAL APPLICATION – CALCULATIONS WITHIN A BY GROUP

The next part of the project focus' on creating the metrics. We have claims for 3 patients and the claims look like this:

	Patient_ID	NPI	Diagnosis_1	Diagnosis_2	Diagnosis_3	Diagnosis_4	Diagnosis_5	Procedure_Code	Svc_Dt
1	1	00023468	9956	4771				J0170	10NOV2010
2	1	00023468	4771	.				95015	05DEC2010
3	1	00023468	4771	.				95117	01JAN2011
4	1	00023468	4771	.				95117	07JAN2011
5	1	00023468	4771	.				95117	13JAN2011
6	1	00023468	4771	.				95117	19JAN2011
7	1	00023468	4771	.				95117	25JAN2011
8	1	00023468	4771	.				95117	31JAN2011
9	1	00023468	9956	4771				J0170	01FEB2011
10	1	00023468	4771	.				95117	06FEB2011
11	1	00023468	4771	.				95117	12FEB2011
12	1	00023468	4771	.				95117	18FEB2011
13	1	00023468	4771					95117	24FEB2011

For the first part, I want to calculate a treatment time: The number of weeks a physician treated a patient. I'll add a third group so I can use the first.svc_dt and last.svc_dt within the patient's record, within a particular physician.

```
proc sort data=physician; by patient_id npi svc_dt;
data treatment_time; set physician; by patient_id npi svc_dt;
retain first_date;
if first.npi then do;
       first_date=svc_dt;
```

```
end;
if last.npi then do;
        last_date=svc_dt;
end;
treatment_time=intck('weeks',first_date,last_date);
if last.patient_id then do;
        keep patient_id npi treatment_time;
        output;
end;
run;
```

Every time the patient appears with a new physician, the first.npi value will be 1. When that happens, I want to copy the svc_dt into a new field called “first_date”. I retain that date, which pulls it down in subsequent rows until I give it a reason to stop. In this case, every time the patient starts a new physician (NPI), that retention will stop, and start over – creating a new value for first_date. Then , when we get to the patient’s last record within the NPI, I copy the svc_dt over to a new field called “last_date” and apply the function intck to calculate the amount of weeks that passed between the first date and the last date. I keep that last record and send it to the output dataset with only the minimum variables needed for this step.

To finish out the project, I now need to do an analysis of skin testing and anaphylaxis in our patients. The first two datasteps are quick and dirty – just give me the patients with skin testing and give me the patients with anaphylactic shock.

```
data skin;
set physician;
if procedure_code='95015';
skin_date=svc_dt;
keep patient_id npi skin_date;
proc sort nodupkey; by patient_id npi skin_date;
run;

data anaph;
set physician;
if diagnosis_1=9956 | diagnosis_2=9956 | diagnosis_3=9956 | diagnosis_4=9956 | diagnosis_5=9956;
ana_date=svc_dt;
keep patient_id npi ana_date;
proc sort nodupkey; by patient_id npi ana_date;
run;
```

Now we apply our last by group process in a merge, and create a segmentation based on how they match. Our client wants everything done at the patient-physician level, so we’ll match on Patient_id and NPI. I’m creating the requested metrics using dichotomous variables, allowing for an easy sum later.

```
data test_anaph;
merge skin (in=a) anaph (in=b); by patient_id npi;
if a | b;

attrib Segment length=$37;
No_Anaphylaxis=0;

if a &^ b then do;
        Anaphylaxis_Before_Skin_Test=0;
        Anaphylaxis_After_Skin_Test=0;
        No_Anaphylaxis=1;
        segment='Skin Tested without Shock Treatment';
end;
if b &^ a then do;
        Anaphylaxis_Before_Skin_Test=0;
        Anaphylaxis_After_Skin_Test=0;
        segment='Shock Treatment without Skin Testing';
end;
```

If a patient appeared in both skin and anaph, then they will get segmented into ‘Anaphylactic Shock and Skin Testing’. But for those particular patients, we need more information: We need to know how many were treated for anaphylactic shock prior to their skin testing, and how many were treated post skin testing. Some might even have both, so I’ve added this code, using the automatic variables first. and last., which will tell SAS when and how to apply my calculations:

```
if a & b then do;
```

```
  segment='Anaphylactic Shock and Skin Testing';
       if first.patient_id and last.patient_id then do;
              if ana_date le skin_date then do;
                            Anaphylaxis_Before_Skin_Test=1;
                            Anaphylaxis_After_Skin_Test=0;
              end;
              if ana_date gt skin_date then do;
                            Anaphylaxis_After_Skin_Test=1;
                            Anaphylaxis_Before_Skin_Test=0;
              end;
       end;
   retain Anaphylaxis_Before_Skin_Test Anaphylaxis_After_Skin_Test;
   if ana_date le skin_date then Anaphylaxis_Before_Skin_Test=1;
   if ana_date gt skin_date then Anaphylaxis_After_Skin_Test=1;
end;
keep patient_id npi segment Anaphylaxis_Before_Skin_Test Anaphylaxis_After_Skin_Test
No_Anaphylaxis;
if last.patient_id and last.npi then output;
run;
```

The resulting dataset:

	Patient_ID	NPI	Segment	No_Anaphylaxis	Anaphylaxis_Before_Skin_Test	Anaphylaxis_After_Skin_Test
1	1	00023468	Anaphylaxic Shock and Skin Testing	0	1	1
2	2	00023468	Anaphylaxic Shock and Skin Testing	0	0	1
3	3	00023468	Skin Tested without Shock Treatment	1	0	0

PRACTICAL APPLICATION – THE FINALE

Now the fun part: Put it all together!

We have three datasets: physician_info – that is our address information; treatment_time – that's the number of weeks each patient saw a particular physician; and test_anaph – the segmentation and additional metrics.

To put them together, I'm going to switch it up a little. I could do a simple merge and then a summary, but instead I'll join the tables and roll them up with a sql step and a Group By.

```
proc sql;
create table Final_List as
select b.segment,
       a.npi,
       a.physician,
       a.address1,
       a.address2,
       a.address3,
       a.address4,
       a.address5,
       a.address_count,
       mean(c.treatment_time)              as Avg_Treatment_Time_Weeks,
       count(distinct b.patient_id)        as Patient_Count,
       sum(b.Anaphylaxis_Before_Skin_Test) as Anaphylaxis_Before_Skin_Test_Ct,
       sum(b.Anaphylaxis_After_Skin_Test)  as Anaphylaxis_After_Skin_Test_Ct,
       sum(b.No_Anaphylaxis)               as No_Anaphylaxis_Ct
from physician_info a,
     test_anaph     b,
     treatment_time c
where a.npi=b.npi
  and b.npi=c.npi
  and b.patient_id=c.patient_id
group by 1,2,3,4,5,6,7,8,9,10,11;
quit;
```

I stripped out the address info, which we've already examined, to show the resulting deliverable:

Segment	NPI	Physician	Address Count	Avg Treatment Time Weeks	Patient Count	Anaphylaxis Before Skin Test Ct	Anaphylaxis After Skin Test Ct	No Anaphylaxis Ct
Anaphylaxic Shock and Skin Testing	00023468	Jones, James	5	25.5	2	1	2	0
Skin Tested without Shock Treatment	00023468	Jones, James	5	0	1	0	0	1

CONCLUSION

By group processing can be as simple or as complex as you'd like to make it. Mistakes are easy to make once you add secondary groups, but with proper preparation you can create some extraordinary deliverables, without a lot of code. Like most SAS techniques, much of what you can do with a by group can be done with other methods, but they will be cumbersome code heavy. I promise you that if you take the time to really understand what by group processing does, it will be well worth the time.

CONTACT INFORMATION

In case a reader wants to get in touch with you, please provide your contact information.

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

Name: Tracee Vinson-Sorrentino
Enterprise: Optum Life Sciences
Work Phone: +1 612.632.2892
E-mail: tracee.sorrentino@optum.com
Web: www.linkedin.com/pub/sorrentino-tracee/4/a67/41a/

The next two paragraphs are **required** and need to remain in the paper.