PharmaSUG 2015 - Paper HA06

The Path To Treatment Pathways

Tracee Vinson-Sorrentino, IMS Health, Plymouth Meeting, PA

ABSTRACT

Refills, switches, restarts, and continuation are valuable and necessary metrics when analyzing pharmaceutical treatment paths. Calculating duration of therapy is an important piece of getting to those data points and a business need for pharmaceutical clients. The nature of rx claims give us many overlapping incidents of therapy and those overlapping dates can skew the duration of therapy calculation, if the calculation is done incorrectly.

This paper will show one very successful way of calculating duration of therapy. Additionally, as a prelude to that process, I will show you how to make SAS tell you what value you need for the arrays that will go into the step and force SAS to put that value right into the array code.

INTRODUCTION

All markets have different business rules. Sometimes it's acceptable to simply add the days_supply of each rx across all time periods to count the days of therapy for each patient. Unfortunately, this simple equation can lead to trouble. For example, if a patient had two rxs on the same day and each had a 30 day supply, you could erroneously declare them as having 60 days of therapy when they actually only had 30. Deduping could handle that problem if the patients always fill the two rxs on the same day, but consider the patient who fills one rx on a given day, find that it didn't work and switches to another rx 10 days later. Adding days_supply in that example would net you 70 days on therapy, when it should only be 40. The techniques shown in this paper will offer an alternative method, and a few other tips and tricks.

THE DATA

The data that will be used in this paper is a mocked up dataset containing 85 rx claims spread out over 4 patients.

Patient_ID	Claims Count
1	17
2	13
3	3
4	52

COHORT PREPARATION

When we get to the meat of this technique, it will involve By Group processing. By Group processing is a powerful tool that allows you to customize data in ways no other programming language allows. It's done in a data step, and requires sorting, so it will result in huge resource usage. As such, it's very important to bring the smallest amount of data into the process, and limit the variables to only those you'll need to accomplish it. Extraneous variables can be reattached once the treatment pathway is determined.

Setting macros will help you in later iterations, and limits the editing to a small section:

```
%let data_set    = conf.rxs;      *<----Dataset for transactions;
%let start_date = '01JAN2010'd; *<----Sets begin date for Study TRxs;
%let stop_date  = '31DEC2013'd; *<----Sets end date for Study TRxs;
%let gap_time   = 30; *<---Allowable days between therapy before considered Gap;
```

Limit data to the transactions within the study time period, using the `order by` code to sort. Also important is a "stop date", so I've added a simple calculation of Rx Date + Days Supplied, which are both readily available variables in claims data.

```
proc sql;
create table cohort
as select distinct
                patient_id,
                rx_date               as start,
                rx_date+days_supplied as stop format date9.,
```

```
                days_supplied
from &data_set
where rx_date between &start_date. and &stop_date.
order by 1,2;
quit;
```

DECLARING A MAX ARRAY VALUE

Step 1: The goal of the first part of this two step process is to calculate a single value – the maximum count of rxs for each patient.

```
proc sql; create table max as select
      patient_id,
      count(start) as TRx_Count
from cohort
group by 1;
quit;
```

Result:

	Patient_ID	TRx_Count
1	1	17
2	2	13
3	3	3
4	4	52

Step 2: The next step will look at the column TRx_Count across all patients to find the maximum value, and then it will put that value into a SAS Macro we're calling maxrx. The value we'll expect to see is 52, as patient 4 has the most rxs with a count of 52. This value will become a "user defined macro" and will be available throughout the session and between multiple concurrent programs.

```
proc sql;
select max(TRx_Count) into :maxrx
from max;
quit;
```

To view the value of the macro created in this step, I've added a `%put` statement, which will print the value of our macro `&maxrx` in the output window:

```
%put &maxrx;
title 'max count of trxs';
run;
```

Result:

max count of trxs
52

A QUICK GUIDE TO MACROS

Macro variables are a fantastic way to eliminate code through symbolic substitution. You can assign small or large amounts of text, both character and numeric, to a macro and then you need only reference the macro to retrieve the text.

Macros have a maximum length of 65,534 characters and length is determined by the amount of text declared. When defining a macro, you do not use quotes unless it's a date value – and only then to declare it as a date (see example below).

There are two kinds of macros in SAS: those defined by the programmer are called "User-defined macro variables" and those defined by the macro processor are called "Automatic macro variables". Macro variables can be used anywhere in a SAS program, except within data lines. `%put` is an example of an automatic macro variable.

Most of the macro variables that are being declared within this program are all user-defined macros.

The example below (and at the top of my program) shows several ways to define macros, and keeping them at the top helps make edits easier for future program usage with different projects. All definitions (right side of equal sign) are shown without quotation marks, except the date. The date needs the single quote so it can be defined and treated as a date when called. When values appear to be numeric, SAS will treat them as such and the like happens for values that appear to be character.

```
%let data_set    = conf.rxs;      *<----Dataset for transactions;
%let start_date = '01JAN2010'd; *<----Sets begin date for Study TRxs;
%let stop_date  = '31DEC2013'd; *<----Sets end date for Study TRxs;
%let gap_time  = 30; *<---Allowable days between therapy before considered Gap;
```

HORRIZONTAL DATA

In order to prepare our cohort for removal of overlapping dates, we must first turn it into a horizontal dataset. The goal is to turn this:

	Patient_ID	Rx_Date	stop
1	1	01JAN2010	31JAN2010
2	1	28JAN2010	27FEB2010
3	1	05MAR2010	04APR2010
4	1	15APR2010	15MAY2010
5	1	13MAY2010	12JUN2010
6	1	10JUN2010	10JUL2010
7	1	08JUL2010	07AUG2010
8	1	06AUG2010	05SEP2010
9	1	07SEP2010	07OCT2010
10	1	15DEC2010	14JAN2011

Into this:

	Patient_ID	start1	start2	start3	start4	start5	
1	1	01JAN2010	28JAN2010	05MAR2010	15APR2010	13MAY2010	10J
2	2	16MAY2011	12JUN2011	18JUL2011	28AUG2011	20NOV2011	19D
3	3	03MAY2013	15AUG2013	10OCT2013	.	.	
4	4	02FEB2010	02MAR2010	30MAR2010	27APR2010	25MAY2010	22J

We'll do this with arrays, and this is why we need to know the maximum number of claims (or dates) per patient.

Note that we have set our data `by patient_id.` This is because we will be doing by-group processing in this step: `if first.patient_id then do;`

We begin with a `retain` statement. This will allow the variables declared in the retain statement to be retained with each iteration – essentially moving the start and stop dates from below to beside.

The `format` statement is a quick housekeeping step that maintains the format of our dates.

The arrays will loop the data through 52 times and rename the start and stop variables to include the count; for example, start1, stop1, start2, stop2, etc.

Our previously declared macro that equates to 52 is invoked with `&maxrx.` Using `%trim` forces it to attach to the variable name. All patients will loop through 52 times, as declared with the code `do i=1 to &maxrx;`. For patients with less than 52 rxs, their start and stop dates beyond valid values will be blank due to the code `e{i} =.;` and `d{i} =.;`

Stepwise, our patients are all first being given the variables start1-start52 and stop1-stop52. Next those variables are all set to missing. And finally the patients loop through 52 times and those values get filled, where appropriate.

The variables start_count and stop_count will tell us how many dates each patient actually had.

stop52	start_count	stop_count
.	17	17
.	13	13
.	3	3
30JAN2014	52	52

The last line of code tells SAS to give us the last line per patient. This is because everytime the patient loops through the array, a new line of data is generated. So per patient, line one has the first set of dates and start_count/stop_count=1. Line two shows the first and second set of dates and start_count/stop_count=2, and so on.

```
data overlap_prep; set cohort; by patient_id;
retain start1-start%trim(&maxrx) stop1-stop%trim(&maxrx);
format start1-start%trim(&maxrx) stop1-stop%trim(&maxrx) date9.;

array e{*} start1-start%trim(&maxrx);
array d{*} stop1-stop%trim(&maxrx);

    if first.patient_id then do;
              do i=1 to &maxrx;
                e{i} =.;
                d{i} =.;
              end;
          start_count=0;
          stop_count=0;
    end;

start_count + 1;
stop_count + 1;
e{start_count} = start;
d{stop_count} = stop;
if last.patient_id then output;
keep patient_id start1-start%trim(&maxrx)stop1-stop%trim(&maxrx)
start_count stop_count;
run;
```

REMOVING THE OVERLAPPING DATES

Now we're ready for some patient level adjustments to the dates – the nitty gritty of our project.

We start with our retain statement, holding our newly created total_duration variable. This will ultimately be an "adjusted count" that each patient was on therapy, after removing overlapping periods.

I've also introduced a new trick in the array declaration: the Colon Modifier. The colon in this usage will tell SAS to go through the PROC CONTENTS in variable position order and select any variable that it finds with a name that begins with the characters preceding the colon. We've previously declared our start and stop variables in numeric order, so that is the order they now appear in our contents. This means that the array start will now be start1-start52 and stop will now be stop1-stop52. We could have invoked our macro as we did previously, but then I couldn't show you the cool, and underutilized, colon modifier!

The most complicated portion of this code though lies in the next portion: blah. Another new function is introduced

here, called the Dim Function. Dim is going to help us loop through our start array 51 times, because (start) has set the upper bounds of the function. The lower bounds is set by i=2 .

We now have two variables we will be comparing to each other: i and j. i will be looking at start/stop2 through start/stop 52, and j will be looking at start/stop1-start/stop51.

But the really cool part of this piece of code – the very functional part – is we're now going to be comparing dates to each other.

Here is the full piece of code, which is broken out below:

```
data overlap_removal; set overlap_prep;
retain total_duration;
array start{*} start:;
array stop{*} stop:;
total_duration=stop1-start1+1;

do i=2 to dim(start);
do j=1 to i-1;
      if . lt start(i) le stop(j) and stop(i) gt stop(j) gt . then do;
            start(i)=stop(j)+1;
      end;
end;
      if start(i) ne . and stop(i) ne . then dur=stop(i)-start(i)+1;
      else dur=0;
      total_duration+dur;
end;
run;
```

BREAKDOWN

The first section of the code tells SAS: If start2 is not blank and stop1 is not blank AND start2 is less than stop1 AND stop2 is greater than stop1, then do this: make start2 equal to stop1 plus 1 day. This way, the second rx begins the day after the first rx "ends" so days of therapy will not overlap – see red boxes below.

```
do i=2 to dim(start);
do j=1 to i-1;

      if . lt start(i) le stop(j) and stop(i) gt stop(j) gt . then do;
            start(i)=stop(j)+1;
      end;
```

VIEWTABLE: Work.Old

	Patient_ID	start1	start2	start3	stop1	stop2	stop3
1	1	01JAN2010	28JAN2010	05MAR2010	31JAN2010	27FEB2010	04APR2010
2	2	16MAY2011	12JUN2011	18JUL2011	15JUN2011	12JUL2011	17AUG2011
3	3	03MAY2013	15AUG2013	10OCT2013	02JUN2013	14SEP2013	09NOV2013
4	4	02FEB2010	02MAR2010	30MAR2010	04MAR2010	01APR2010	29APR2010

VIEWTABLE: Work.New

	Patient_ID	start1	start2	start3	stop1	stop2	stop3
1	1	01JAN2010	01FEB2010	05MAR2010	31JAN2010	27FEB2010	04APR2010
2	2	16MAY2011	16JUN2011	18JUL2011	15JUN2011	12JUL2011	17AUG2011
3	3	03MAY2013	15AUG2013	10OCT2013	02JUN2013	14SEP2013	09NOV2013
4	4	02FEB2010	05MAR2010	02APR2010	04MAR2010	01APR2010	29APR2010

The next section of code tells SAS to calculate the duration of therapy. It says if the equally numbered start and stop dates are not blank (start1/stop1, etc) then subtract start date (plus 1) from the stop date to get to the days on therapy.

```
if start(i) ne . and stop(i) ne . then dur=stop(i)-start(i)+1;
else dur=0;
total_duration+dur;
```

You have a few options after completing this step. You can either change it back to a vertical dataset (one line per claim), or you can remove all of the dates and keep just the fields you really need.

To change it back to a vertical dataset, you can run this code:

```
data final_vertical; set overlap_removal;
array start{*} start:;
array stop{*} stop:;
do i=1 to dim(start);
      begin=start(i);
      end=stop(i);
      if start(i) ne . then output;
end;
format begin end date9.;
keep patient_id begin end total_duration;
run;
```

	Patient_ID	total_duration	begin	end
10	1	485	15DEC2010	14JAN2011
11	1	485	15JAN2011	11FEB2011
12	1	485	12FEB2011	12MAR2011
13	1	485	01APR2011	01MAY2011
14	1	485	02MAY2011	30MAY2011
15	1	485	31MAY2011	27JUN2011
16	1	485	28JUN2011	15JUL2011
17	1	485	16JUL2011	11AUG2011
18	2	375	16MAY2011	15JUN2011
19	2	375	16JUN2011	12JUL2011
20	2	375	18JUL2011	17AUG2011
21	2	375	28AUG2011	27SEP2011

CONCLUSION

Business rules change all the time. New products enter the market, generics knock promoted brands off the shelves, and acquisitions change companies. It's important to use programming techniques that can evolve with the business and I think arrays with macros offer that solution. These steps can be further enhanced to count therapeutic episodes, identify a patient as chronic or acute, and by adding product levels, can lead to a patient-product pathway. The methods described in this paper leverage some of the best components of the SAS system.

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: IMS Health
Work Phone: +1 484.894.7443
E-mail: tsorrentino@us.imshealth.com
Web: www.linkedin.com/pub/sorrentino-tracee/4/a67/41a/

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