SAS®
GLOBAL FORUM
2020
MARCH 29 - APRIL 1
WASHINGTON, DC

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.
Abstract

Analysis and research on administrative healthcare data often involve the manipulation of multiple records with contiguous or overlapping date fields. Claims data contain transaction records with multiple records for each individual across multiple dates of service, while enrollment data often contain multiple records per person reflecting that individual's enrollment change history. In many cases, it is necessary to roll up the date fields in these types of files in order to establish continuous enrollment, episodes of care, and the gaps in coverage for those periods.

In this presentation, a flexible SAS® program is introduced that rolls up multi-record date fields to establish continuous segments, segment lengths, and segment gap lengths, and to calculate the number of segments and gaps for an individual ID variable. Using SAS functions such as LAG, RETAIN, and INTCK, as well as procedures such as PROC SQL and PROC TRANSPOSE, the target data are transformed from long to wide. This enables the user to more easily process hospitalization data into episodes of care or evaluate claims and enrollment records against standardized performance measures such as the Healthcare Effectiveness Data and Information Set (HEDIS).
Introduction

Administrative medical claims data, such as Medicare, Medicaid, and commercial coverage plans like Blue Cross and Blue Shield are often used to study healthcare utilization and outcomes. However, these types of claims were designed to facilitate billing, not research. When processing this type of medical claims data, it is often necessary to roll up an individual’s experience along dates of service or dates of enrollment for the purposes of checking for gaps between services or gaps between insurance coverage dates. One of the most common uses for this type of roll up is calculating HEDIS measures.

HEDIS measures are a comprehensive set of standardized performance measures designed to evaluate and compare health plan performance. Each measure will typically have allowable gap criteria, such as:

- “No more than one gap in enrollment of up to 45 days during each year of continuous enrollment.”
- “The member may not have more than a 1-month gap in coverage during each year of continuous enrollment.”
- “No gaps in enrollment during the continuous enrollment period.”

The SAS program discussed here — although designed for use with this type of administrative medical enrollment and claims data — could easily be adapted to roll up any time series data containing paired begin and end dates.
Purpose

In this presentation, paired time series data will be transformed from long to wide, resulting in a single record per unique ID variable. The resulting table will contain the number of episodes, the begin and end dates for each episode and the spans (i.e., gaps) between episodes.

TERMS


- **Span (Gap)**: The number of days between each episode. Continuing the above example, there would be 2 episodes and one span of 19 days. This would be the equivalent of a 19-day gap in medical insurance enrollment coverage or a 19-day gap between episodes of care.
**Code Walkthrough**

**PREPROCESSING**

In order to apply this code, the input file must have at least one unique ID variable, a begin date variable and an end date variable. In the preprocessing step, missing/blank date field values are handled by:

- Delete entire record if both dates are missing/blank
- If only one of the two dates are missing/blank, set the missing date equal to the non-missing date (creates a same day period)
- If the end date contains a far future date such as 31Dec9999, it can either be left as is and it will be processed as if it were continuous through the next valid end date (appropriate for enrollment records), it can be handled as if the date is missing/blank creating a same day period, or it can be deleted, depending (appropriate for claims records) on user needs.

**DATE ROLL UP**

In this section, the records are sorted chronologically by the ID variable. Then each begin and end date are evaluated to establish the true periods associated with each unique ID variable.

<table>
<thead>
<tr>
<th>PatientID</th>
<th>LName</th>
<th>FName</th>
<th>DateOfServiceBeg</th>
<th>DateOfServiceEnd</th>
</tr>
</thead>
<tbody>
<tr>
<td>1078JetGeo</td>
<td>Jetson</td>
<td>George</td>
<td>12Jan2017</td>
<td>13Jan2017</td>
</tr>
<tr>
<td>1078JetGeo</td>
<td>Jetson</td>
<td>George</td>
<td>09Feb2017</td>
<td>09Feb2017</td>
</tr>
<tr>
<td>1078JetGeo</td>
<td>Jetson</td>
<td>George</td>
<td>09Feb2017</td>
<td>09Feb2017</td>
</tr>
<tr>
<td>1078JetGeo</td>
<td>Jetson</td>
<td>George</td>
<td>09Feb2017</td>
<td>09Feb2017</td>
</tr>
<tr>
<td>1078JetGeo</td>
<td>Jetson</td>
<td>George</td>
<td>09Feb2017</td>
<td>09Feb2017</td>
</tr>
<tr>
<td>1078JetGeo</td>
<td>Jetson</td>
<td>George</td>
<td>09Feb2017</td>
<td>31Dec9999</td>
</tr>
<tr>
<td>1078JetGeo</td>
<td>Jetson</td>
<td>George</td>
<td>27Mar2017</td>
<td>27Mar2017</td>
</tr>
</tbody>
</table>

Same day period – appropriate for claims

<table>
<thead>
<tr>
<th>DateOfServiceBeg</th>
<th>DateOfServiceEnd</th>
</tr>
</thead>
<tbody>
<tr>
<td>12Jan2017</td>
<td>13Jan2017</td>
</tr>
<tr>
<td>09Feb2017</td>
<td>09Feb2017</td>
</tr>
<tr>
<td>27Mar2017</td>
<td>27Mar2017</td>
</tr>
</tbody>
</table>

Continuous through – appropriate for enrollment

<table>
<thead>
<tr>
<th>DateOfServiceBeg</th>
<th>DateOfServiceEnd</th>
</tr>
</thead>
<tbody>
<tr>
<td>12Jan2017</td>
<td>13Jan2017</td>
</tr>
<tr>
<td>09Feb2017</td>
<td>27Mar2017</td>
</tr>
</tbody>
</table>
Code Walkthrough

**REMOVE DUPLICATE RECORDS & TRANSPOSE**
In this step, duplicate records are removed. The begin and end dates are transposed separately using PROC TRANSPOSE and then merged back together. This populates the episode begin (ep_in) and episode end (ep_out) variables.

**PROCESS THE TRANSPOSED FILE**
In this section, the transposed file is used to populate the episodes variable (episodes) by counting the number of episodes for each unique ID. This step also sets the following macro variables:
- tHigh = the number of episodes for each unique ID variable
- tSpans = the number of spans, or gaps, between each episode

**POPULATE THE SPAN VARIABLES & CREATE OUTPUT FILE**
In this final step, the span variables are populated by counting the number of days between each episode for each unique ID variable. The output data set now contains a count of the number of episodes (episodes), each episode begin (ep_in1 – ep_inn) and end (ep_out1 – ep_outn) dates and the number of days between each episode (span1 – span??) for each unique ID variable.
Conclusion

After rolling up an individual’s experience along dates of service or dates of enrollment, creating episodes of care or continuous enrollment segments respectively, a user can then use the final output file to aggregate information such as medical or pharmacy claims costs that occur within an episode of service or identify diagnosis or procedure codes that occur within an episode of service.

A final word of caution: Because this program creates a wide record that is dependant on the number of episodes an individual ID variable has accumulated over time, the number of variables in the final output can become very large, depending on your range of dates and how often episodes occur. For example, a year’s worth of data for an individual with frequent short duration medical claims generates 200 episodes. The resulting file would contain roughly 600 variables per unique ID variable.

REFERENCES


Wicklin, Rick “INTCK and INTNX: Two essential functions for computing intervals between dates in SAS.” Retrieved from https://blogs.sas.com/content/iml/2017/05/15/intck-intnx-intervals-sas.html

**Preprocessing**

```sas
data rollup; set testdata(rename=(PatientID=IDVar DateOfServiceBegin=adm_date DateOfServiceEnd=dis_date));
** Records with missing/blank beginning and ending dates are removed **;
if adm_date eq . and dis_date eq . then delete;
** Create single day date pair when one of the two dates is missing/blank **;
** If just the beginning date is missing/blank set the beginning date equal to the ending date **;
if adm_date eq . and dis_date ne . then adm_date = dis_date;
** If just the ending date is missing/blank set the ending date equal to the beginning date **;
if adm_date ne . and dis_date eq . then dis_date = adm_date;
** If the ending date contains a far future date, replace it with the beginning date.
You may choose to comment out this edit and allow the future date to be processed in the steps
below as if it were continuous through the next valid ending date. ;
*if dis_date eq '31DEC9999'd then dis_date = adm_date;
run;
```

**Begin Date Roll Up**

This section checks each begin date against each ending date and vice versa to get
the true "periods" associated with each unique ID variable.

```sas
**** Sort to get the dates in chronological order ****;
proc sort data=rollup; by IDVar adm_date dis_date;
run;
```
Code split into two columns for readability

Note: The INTCK function counts the full time period between dates such that INTCK('DAYS', '09Jan2017'd, '11Jan2017'd) returns 2. The program is configured so that a returned value of 1 does not initiate a break in the current episode because the dates would be consecutive.
Remove duplicate records

**** create a temporary unique id based on IDVar and dates to remove duplicate records ****;

```sas
data rollup3; set rollup2(rename=(in_date=adm_date out_date=dis_date));

reid=IDVar||adm_date||dis_date;
run;
```

proc sort data=rollup3 nodupkey; by reid; run;

Transpose the rolled up date periods

The purpose of the following section is to give all records for an ID the same period dates. These period dates are then used to check each begin and ending date to see if there were multiple episodes and to calculate the spans between episodes.

**** Populate the ep_in variables by transposing all of the begin dates ****;

```sas
proc transpose data=rollup3 out=wide_ru3l prefix=ep_in;
    by IDVar;
    var adm_date;
run;
```

**** Populate the ep_out variables by transposing all of the ending dates ****;

```sas
proc transpose data=rollup3 out=wide_ru3o prefix=ep_out;
    by IDVar;
    var dis_date;
run;
```

**** Merge the two transposed files ****;

```sas
data wide_ru3; merge wide_ru3l(drop_NAME_) wide_ru3o(drop_NAME_);
    by IDVar;
run;
```

**** Merge final transposed file back with the original dataset ****;

```sas
data rollup4; merge rollup1 wide_ru3;
    by IDVar;
run;
```
Process the transposed file

Use PROC CONTENTS to get a list of all of the different episode dates

```sas
proc contents data=rollup4 noprint out=rollup4cont; run;
```

The resulting file is run through this macro to count the number of episodes and assign values to several macro variables.

```sas
%let tVarName = cp_in;
%let sql noprint;
select count(*)
  into :tNobs from rollup4cont(where=(name="&tVarName"));
quit;
%put number of admits is &tNobs;

%let tHigh = &eval(&tNobs);
%put high = &tHigh;
%let tSpans = &eval(&tNobs - 1);
%put high = &tSpans;
run;
```

Populate the episodes variable by counting the number of episodes for each ID variable

```sas
data rollup5(drop=i); set rollup4;
episodes = 0;
array checki[tHigh] cp_inl - cp_in&tHigh;
array checko[tHigh] cp_outl - cp_out&tHigh;
do i = 1 to &tHigh;
  if adm_date ge checki(i) and dis_date le checko(i) then episodes=i;
end;
run;
```

```sas
proc sort data=rollup5;
  by IDVar adm_date dis_date;
run;
```
Populate the span variables by counting the number of days between each episode for each ID variable.

```sas
data spans1(drop=1); set rollup5;
   by IDVar;
   array span[&tHigh] span1 - span&tHigh;
   array checkepo[&tHigh] ep_out1 - ep_out&tHigh;
   array checkepi[&tHigh] ep_inl - ep_in&tHigh;
   do i = 1 to &tHigh;
      if i lt &tHigh then do;
         span(i)=intok('DAYS',checkepo(i),checkepi(i+1)) - 1;
      end;
   if last.IDVar then output;
run;
```

```sas
proc print data=spans1(obs=100);
   var IDVar episodes ep_inl ep_out1 span1 ep_in2 ep_out2 span2 ep_in3 ep_out3 span3 ep_in4 ep_out4 span4;
run;
```