Paper 2683-2018

SAS® Arrays and Macros Make Processing Claims with Multiple Conditions Easier

Shavonne J. Standifer, Truman Medical Center; Stephanie R. Thompson, Datamum;

ABSTRACT

The ability to translate medical claims data into actionable insights is an important step in deriving knowledge useful to make inferences about the healthcare system. A single patient can have numerous diagnosis codes per encounter. Developing methods to efficiently process this information is necessary in order to provide meaningful health statistics to providers that help them to better understand the needs of their populations with multiple conditions. SAS® Array statements and the Macro facility help to streamline this process in an efficient way. This paper discusses, through examples, how Base SAS® techniques, such as RETAIN, FIRST, LAST, DO LOOPS, and the LAG function can work together with SAS Arrays and Macros to transform a claims data file into population level summaries.

INTRODUCTION

Having multiple chronic conditions is associated with substantial health care costs. Approximately 71% of total health care spending in the United States is associated with care for Americans with more than one chronic condition. As a person's number of chronic conditions increases, so does his or her risk for dying prematurely, being hospitalized, and having a greater risk of poor day-to-day functioning. Identifying if a patient has multiple chronic conditions can be challenging because it's routine for patients to have several encounters over a period of time. Each encounter has its own set of encounter types, diagnosis, and procedures, so this makes the task of transforming a claims data file into actionable insights difficult.

This paper helps SAS programmers who are tasked with the job of cleaning, summarizing, and presenting longitudinal electronic medical claims data through summary tables by providing useful techniques that they can use within base SAS. There are four business cases presented, each with examples geared toward helping you understand some of the ways that SAS makes processing claims with multiple conditions easier.

Below is a fictitious claims data file that is in a structure that's common for healthcare claims. The claims file has multiple encounters and multiple diagnosis codes per patient id over a period of time.

Consider the claims data file below:

```
DATA Sample_1;

INFILE '\\....\claims1.txt';

INPUT
       patient_id $11.
       admit_date 8
       discharge_date 8
       encounter_type $10.`
       diag1 $8.
       diag2 $8.
       Diag3 $8.

RUN;
```

Result:

patient_id	admit_date	discharge_date	encounter_type	diag1	diag2	diag3
1001	08/05/2016	08/07/2016	Inpatient	I10	E0800	R009
1001	10/05/2016	10/05/2016	Emergency Room	I10	R009	R05
1002	08/05/2016	08/07/2016	Inpatient	E0821	R42	
1003	04/21/2017	04/21/2017	Emergency Room	E08331	i120	D513
1003	05/11/2017	05/14/2017	Inpatient	I120	D513	
1004	05/11/2017	05/11/2017	Emergency Room	E0822		
1005	05/27/2017	05/27/2017	Emergency Room	R0602	R42	

Business Case 1: Find the total number of chronic conditions per patient id

The first step in solving this business case is to identify the types of chronic conditions that are present at each encounter. Based on the chronic condition codes that you specify, you would create a subset that creates new variables that indicate if a chronic condition was present at an encounter. Through the data step we can use SAS Arrays and Do Loops to create this subset which, as a result, creates a view that's easier to understand and useful for both reporting and other data manipulation techniques.

Before we dive into the code examples, below is a brief overview of SAS Array statements and Do Group processing.

ARRAY STATEMENTS

Arrays are temporary groupings of SAS variables that have the same data type and are arranged in a particular order. Array statements allow you to do more with less code and only exist for the duration of the data step. Because arrays allow you to process variables in groups, they assist with processing claims data in both an effective and efficient way.

ARRAY REFERENCES

Once an array has been defined in the array statement, it creates an array reference. An array reference is a reference to an element that is to be processed in an array. An array reference may be used within the data step in almost any place that other SAS variables may be used.

DO GROUP PROCESSING

DO GROUP processing also allow you to execute statements as a group. When coupled with a SAS array, do group processing simplify the data manipulation process even further.

DO GROUP

A DO GROUP is a sequence of statements that starts with a simple DO statement and that ends with a corresponding END statement.

DO LOOP

DO LOOPS are executed (usually repeatedly) according to directions that are specified in the DO statement.

In the data step below, the DO statement combined with if-then/else statements tell SAS to create new variables for Hypertension and Diabetes if those code values are present in the encounter data.

Consider the code below:

```
Data Claims2;
set work.sample_1;

Diabetes_Flag= 0;
Hypertension_Flag = 0;

array diag {3} diag1-diag3;

do i=1 to 3;

if diag{i} in ('I10','I120')
then Hypertension_Flag_= 1;
else if diag{i} in ('E0800','E0821','E08331','E0822')
then Diabetes_Flag = 1;
end;

tot_all = Hypertension_Flag + Diabetes_Flag;

Run;
```

At the top of the data step, the variables Hypertension and Diabetes are initialized to 0. SAS automatically sets the values of variables to missing. If we do not set the values of the new variables to 0, the program would execute successfully and the SAS log would report no error messages, however, the summing assignment statement, *tot_all* would not process correctly, and would be returned with missing values. For each encounter, the SAS Array processes the diagnosis columns as a group. The Do loop iteratively executes per encounter until there are no more encounters to process. The summing assignment statement adds up the total amount of diagnosis indicators per encounter.

Processing claims data that are structured in this way ensures that all diagnosis codes on an encounter are considered when determining if a disease is prevalent or not.

Results

patient_id	**admit_date**	**discharge_date**	**Diabetes_Flag**	**Hypertension_Flag**	**tot_all**
1001	08/05/2016	08/07/2016	1	0	1
1001	10/05/2016	10/05/2016	0	0	0
1002	08/05/2016	08/07/2016	1	0	1
1003	04/21/2017	04/21/2017	1	0	1
1003	05/11/2017	05/14/2017	0	1	1
1004	05/11/2017	05/11/2017	1	0	1
1005	05/27/2017	05/27/2017	0	0	0

The last step is to drill down a bit more in order to find the total number of chronic diagnosis per patient id. For this to occur, we must sum the total number of diagnosis tags per patient. Below are some useful SAS Procedures that can be utilized to make the process of creating summary tables for analysis easier.

The PROC TABULATE below demonstrates how to create a summary report that displays total aggregates in a tabular format. The class statement in the procedure defines the categorical variable. The Var statement tells SAS which value to aggregate and the table statement instructs SAS on how to display the data.

Title ‘The total number of chronic diagnosis per patient id’;

```
Proc Tabulate data=Work.claims2;
class patient_id;
var tot_all;
table patient_id,tot_all;
Run; Title;
```

Results:

The total number of chronic diagnosis per patient id

	tot_all
	Sum
Patient ID	
1001	1
1002	1
1003	2
1004	1
1005	0

You could also use Proc SQL to get to the desired results as well:

```
Proc Sql;
Create Table Work.claims3_v2 As Select
patient_id,
(sum(tot_all)) as count
From Work.claims2
Group by patient_id;
Quit;
```

Results

patient_id	count
1001	1
1002	1
1003	2
1004	1
1005	0

Let’s say that you want to create an additional view that only showed the patients who had more than one chronic diagnosis. You could use the following code:

```
PROC SQL;
Create Table Work.claims3 As Select
patient_id,
(sum(tot_all)) as all_count
From Work.claims2
```

```
Group by patient_id;
Having (Calculated count) > 1;
QUIT;
```

Results

patient_id	all_count
1003	2

PROC SQL is a handy procedure that can be used in addition to PROC TABULATE to process summary tables with very little code. Deciding which one is best to use depends on the business scenario and the knowledge of the programmer.

The examples above work well in business scenarios where there are a small set of specifically defined diagnosis codes to consider.

Business Case 2: Transform a claims file

Electronic heath data is frequently stored in a way that is efficient for the database, however, these storage methods can sometimes present more difficult problems for the business analyst tasked with turning the data into evidence based insights.

Consider the claims data file below:

```
DATA Sample_2;

    INFILE '\\....\claims2.txt';
    INPUT
        patient_id          8
        encounter_num       8
        type              $ 9
        admit_date          8
        discharge_date      8
        diagnosis         $ 7 ;

RUN;
```

Results:

patient_id	encounter_num	type	admit_date	discharge_date	diagnosis
1001	309	Inpatient	08/05/2016	08/07/2016	I10
1001	309	Inpatient	08/05/2016	08/07/2016	E0800
1001	309	Inpatient	08/05/2016	08/07/2016	R009
1001	310	ER	10/05/2016	10/05/2016	I10
1001	310	ER	10/05/2016	10/05/2016	R009
1001	310	ER	10/05/2016	10/05/2016	R05
1002	311	Inpatient	08/05/2016	08/07/2016	E0821
1002	311	Inpatient	08/05/2016	08/07/2016	R42
1003	312	ER	04/21/2017	04/21/2017	E08331
1003	312	ER	04/21/2017	04/21/2017	I120
1003	312	ER	04/21/2017	04/21/2017	D513
1003	313	Inpatient	05/11/2017	05/14/2017	I120

1003	313	Inpatient	05/11/2017	05/14/2017	D513
1004	314	ER	05/11/2017	05/11/2017	E0822
1005	315	ER	05/27/2017	05/27/2017	R0602
1005	315	ER	05/27/2017	05/27/2017	R42

The dataset is displayed in a longer vertical view that includes several rows of encounter data per patient id. We need to reshape the data into a format that shows horizontally so that we can use SAS arrays to manipulate the data file and create a new subset. PROC TRANSPOSE is one of the most common ways to reshape data in SAS because once you understand the options, it's easy to use and adapt.

Below shows how to transform the dataset above using PROC TRANSPOSE:

```
Proc Transpose data=work.sample_2
out=sample_2_results
name=transposed
prefix=diag;
by patient_id encounter_num;
var diagnosis;
run;
```

The out= option names the resulting dataset. The name= option assigns a name to the variable in the dataset that is being transposed. The prefix= option assigns the string of variables that are transposed. The By statement allows you to transpose data within the combination of the BY variables. The var statement identifies the actual data variable to be transposed.

Results

patient_id	**encounter_num**	**transposed**	**diag1**	**diag2**	**diag3**
1001	309	diagnosis	I10	E0800	R009
1001	310	diagnosis	I10	R009	R05
1002	311	diagnosis	E0821	R42	
1003	312	diagnosis	E08331	I120	D513
1003	313	diagnosis	I120	D513	
1004	314	diagnosis	E0822		
1005	315	diagnosis	R0602	R42	

Proc transpose reshaped the dataset into a shorter view that works quite handy with our Arrays and do loops.

MACRO PROCESSING

Macro processing makes identifying claims with multiple chronic conditions easier. The macro facility is a code generator that allows you to store values and use them with other operations throughout the data step. Macro processing can be helpful in creating insightful summaries from your claims file with less code. Macros are stored text that contains SAS language and Macro language. Macro variables hold the value of text strings and have an ampersand(&) before the name.

Through the data step in business case 1, we added specific chronic conditions into an array, in order to return values that indicated if a diagnosis was present or not. For the purposes of the paper, we only had a short list of chronic conditions of which we wanted to receive indicators, but what if we had hundreds of codes of which we wanted to consider? In this scenario it would be more efficient to utilize the SAS

Macro facility to read in a list of chronic condition code values and store the strings within a macro variable.

Below is an alternative version to answer the same question as business case 1 using macros:

```
Data Work.dx_report;

Hypertension : $CHAR8.
Diabetes : $CHAR8.;
Run;

PROC SQL noprint;
select quote (trim(Hypertension))
into :Hyp_dx separated by ','
from work.dx_report
where (trim(Hypertension)) ne '';
%put Hyp_dx = &Hyp_dx;

select quote (trim(Diabetes))
into :Diab_dx separated by ','
from work.dx_report
where (trim(Diabetes)) ne '';
%put Diab_dx = &Diab_dx;

Quit;
```

First, the program reads in the values of Hypertension and Diabetes from an external file and stores those values in the macro variables &Hyp_dx and &Diab_dx. The %put tells SAS to write each diagnosis code value to the log. Next, through a data step you can create a SAS Array that creates a new variable to indicate if a diagnosis was present on an encounter.

```
Data Work.dx_report_munip;
Set Work.sample_1;
array icd{3} diag1-diag3;
do i=1 to 3;
if icd{i} in (&Hyp_dx) then Hyp_dx = 1;
else if icd{i} in (&Diab_dx) then Diab_dx = 1;
end;
drop i diag1 diag2 diag3;
Run;
```

Results

patient_id	admit_date	discharge_date	encounter_type	Hyp_dx	Diab_dx
1001	08/05/2016	08/07/2016	Inpatient		1
1001	10/05/2016	10/05/2016	Emergency Room		
1002	08/05/2016	08/07/2016	Inpatient		1
1003	04/21/2017	04/21/2017	Emergency Room		1
1003	05/11/2017	05/14/2017	Inpatient	1	
1004	05/11/2017	05/11/2017	Emergency Room		1

1005	05/27/2017	05/27/2017	Emergency Room		

For business scenarios where you have a long list of diagnosis codes in an external file that are updated regularly, processing claims with macros is the better option.

TIPS AND TRICKS

Many SAS programmers in health science related industries are tasked with having to answer complex business questions about patient histories in very little time in order to provide clinicians, nurses, and physicians with evidence based reporting. This section provides additional techniques that you can use in base SAS to assist in satisfying these types of requests.

Let's review some additional SAS Techniques before we dive into another example.

RETAIN

As mentioned above, by default, at the beginning of a data step, SAS places missing values in the program data vector for all assigned variables. The use of the retain statement tells SAS, that at each iteration of the data step to keep or retain a specified variable instead of setting that value to missing. When used with raw claims data, the retain statement is useful in identifying the first instance where a condition occurred for a specified period of time.

FIRST. / LAST. PROCESSING

A first. variable is a temporary variable that SAS creates to identify the first observation of each BY group.

Business Case 3: Identify the first encounter type per patient id

In order to solve, we must create a subset that's set by patient_id. This means that for each iteration of a patient_id SAS will Retain the value of the first encounter type per paitent_id.

```
Data ret_sample;
set work.sample_1;
by patient_id;
retain encounter_type;
if first.patient_id then do;
first_enc_type = encounter_type;
end;
if first_enc_type ne ' ' then output ret_sample;
Keep patient_id first_enc_type;
run;
```

Results

patient_id	first_enc_type
1001	Inpatient
1002	Inpatient
1003	Emergency Room
1004	Emergency Room
1005	Emergency Room

LAG FUNCTION

The Lag function is an efficient tool in helping you to identify encounters in the claims data that are readmissions. The Lag function looks across observations and analyzes which encounters would be considered a readmission based on the criteria that we specify.

Business Case 4: Identify patients with a hypertension related readmission within 30 days

To solve, we create a new variable that's based on the first discharge date in the claims file. We also create a Gap variable to act as a counter for the number of days between the last admission dates.

```
Data Solutions1;
Set claims1;
By patient_id;
Ref_date = LAG(Discharge_Date);
Format admit_date Ref_date YYMMDD10.;

Gap = Admit_Date - Ref_date;

If First.patient_id then do;
Ref_date =.;
Gap =.;
Tag =.;
Readmissions =.;

End;

If 0 <=Gap<=30 then Tag = 1;
Readmissions + Tag;

if Hyp_dx = 1 and Tag = 1 then output Solutions1;

Run
```

Results

patient_id	admit_date	Ref_date	Gap
1003	2017-05-11	2017-04-21	20

CONCLUSION

This paper outlined multiple ways to approach medical claims data to identify patients with multiple conditions based on diagnosis code and provided techniques that can assist SAS programmers with providing meaningful insights to clinicians.

REFERENCES

Cody, Ronald, Ed.D “Longitudinal Data Techniques: Looking Across Observations.” SUGI 27

Fan,Weifan and Sarfarazi,Maryam. 2014. ”SAS® Solutions to Identifying Hospital Readmissions.” 1622-2014

“2018 ICD-10 CM and GEMs.” https://www.cms.gov/Medicare/Coding/ICD10/2018-ICD-10-CM-and-GEMs.html

CONTACT INFORMATION

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

Shavonne J. Standifer
Truman Medical Center
shavonne.standifer@tmcmed.org

Stephanie R. Thompson
Datamum
Stephanie@Datamum.com