

SAS® Macro program to “clean” Medicaid claims

Gary W. Moore, Arkansas Foundation for Medical Care, Little Rock, AR

Abstract

To derive many of our agency's reported measures, we need to know how many Medicaid recipients received a given treatment or met a certain criterion. One step in the overall derivation process is retrieving medical claims and then 'cleaning' them based on the claim's status. This poster describes the process we use and discusses how the process is "helped along" with the aid of the macro programming facility of SAS.

Introduction

One step in the overall measure derivation process is retrieving medical claims data via SQL queries and then 'cleaning' them based on the claim's status. Ultimately, we eliminate from consideration claims that were denied outright and claims that were paid but then later voided or refunded.

An individual medical claim is comprised of various fields from many database tables. We uniquely identify an individual claim by the combination of an Internal Claim Number (ICN) and a Recipient Base ID. An overall claim may have many "subparts" identified by a detail number. These detail-level claims further specify a claimant's separate diagnosis codes, medical procedure codes, etc. The important notion here is that claim status can vary at the detail level. In other words, an overall claim may show that one part (detail) of the claim was paid while another detail was denied. The claim's status is indicated as either 1. Paid (A), 2. Partial Refund (P), 3. Denied (C), 4. Full Refund (F), or 5. Voided (V). Initially, a claim's status would be either "A" or "C". Handling those claims with a status of "C" (Denied) is easy — simply eliminate the claim from further consideration. Those claims where the indicated

claim status was "A" (Paid) may subsequently change to "F", "P", or "V".

Previously, we queried the database for all those detail-level claims based on various criteria such as diagnosis codes, procedure codes, etc. The approach our staff used at this point was to determine the claim's total paid amount. A claim was retained if the total paid amount calculated using all the claims by ICN and Recipient Base ID was positive. An easier method was noted if one considered the detail number in conjunction with the claim status indicator.

Process

The macro program cleans (validates) the claims of a data set based on the following data set variables: 1. Internal Claim Number (icn), 2. Recipient Base ID (base_id), 3. Detail Number (detail_num), and 4. Claim Status (claim_status).

The claims in the data set are validated based on their claim_status values at the icn, base_id, and detail_num group level. (For data set reference purposes, assume that a new data set is created as the result of each data step or "proc" operation.) The claim validation process is started by sorting the observations of the claims data set by the icn, base_id, and detail_num variables. The detail_status observations are then transposed based on the 'by' group formed from the icn, base_id, and detail_num variables.

Example "proc transpose" statement:

```
proc transpose prefix=_col data=x out=y;  
  by icn recipient_base_id detail_num;  
  var claim_status;  
run;
```

The following table illustrates observations with 'valid' claims for the 'by' group (x, y, z).

Obs #	icn	base_id	detail_num	_col1	_col2	_col3	_col4	...
1	x	y	z	A	.	.	.	.
2	x	y	z	A	A	.	.	.
3	x	y	z	A	A	A	.	.
4	x	y	z	A	A	A	A	.
5	x	y	z	A	A	A	A	A
6	x	y	z	A	P	.	.	.
7	x	y	z	A	A	P	.	.
etc.								

The overall width of the resulting transposed data set is determined by the number of '_col' variables. The number of '_col' variables is determined by the maximum number of observations for any one icn, base_id, and detail_num group. (Note: Missing character values are illustrated by the 'dot' character.)

The following table illustrates observations with 'invalid' claims for the 'by' group (x, y, z).

Obs #	icn	base_id	detail_num	_col1	_col2	_col3	_col4	...
1	x	y	z	C	.	.	.	.
2	x	y	z	A	C	.	.	.
3	x	y	z	A	V	.	.	.
4	x	y	z	C	V	.	.	.
5	x	y	z	A	A	A	F	.
6	x	y	z	A	A	A	A	V
etc.								

Valid claims must have an "A" in _col1 followed by other '_col' values of "A", "P" (Partial refund), or missing values. At the outset of the validation process the number of '_col' variables is unknown. This is where the ability to dynamically program a data step statement via SAS macro programming really is a blessing. Once the number of '_col' variables is determined the 'valid claim' logic is formed 'on-the-fly' based on this number used as the '%to' value of a macro '%do' loop.

Example of dynamic data step programming:

```

data y;
  set x;
  %do;
    valid_claim = (upcase(_col1) = 'A')
    %do i=2 %to &num_trans_cols;
      AND ((upcase(_col&i) = 'A') OR
        (upcase(_col&i) = 'P') OR
        (upcase(_col&i) = ''))
    %end;;
  %end;
run;

```

The 'validated' claims data set will only have observations for the following variables: icn, base_id, detail_num, and detail_status.

This data set is sorted by the icn, base_id, detail_num, and claim_status variables. This data set is then merged with (via a subsetting 'if' statement) those observations from the

'unclean' data set that have the same 'key' values into an 'almost clean' data set. The 'by' group in the merge data step constitutes the key.

Example 'merging' data step:

```
data z;
  merge x (in=in_1) y (in=in_2);
  by icn base_id detail_num claim_status;
  if (in_1 and in_2);
run;
```

One (possible) result of this data set merge operation is that there will be multiple observations with the same 'key' value. In other words, there could be more than one observation with the exact same icn, base_id, detail_num, and detail_status. This should only happen when

there are multiple observations with claim_status value of "A" at the icn, base_id, and detail_num group level. Therefore, sort this data set with the 'nodupkey' sort option to eliminate all but one of these observations. (We only need one instance of this claim to derive our measurements.) The resulting 'clean' claims data set is returned to the user.

The following tables show the real-world results of cleaning two different types of claims, header-level and detail-level claims. The "HDR_CLAIMS" data set was reduced by 28,100 observations. The "DTL_CLAIMS" data set was reduced by 16,231 observations.

Listing of CLAIM_STATUS Combinations for claims file: HDR_CLAIMS			
Transpose of CLAIM_STATUS by ICN and RECIPIENT_BASE_ID			
_COL1	_COL2	Frequency	Cumulative Frequency
A		136795	136795
A	F	90	136885
A	P	67	136952
A	V	5159	142111
C		17519	159630
C	V	8	159638

Listing of DETAIL_STATUS_CODE Combinations for claims file: DTL_CLAIMS

Transpose of DETAIL_STATUS_CODE by ICN, RECIPIENT_BASE_ID, and DETAIL_NUMBER

_COL1	_COL2	_COL3	_COL4	Frequency	Cumulative Frequency
A				215815	215815
A	A			2853	218668
A	A	F	F	1	218669
A	A	P	P	1	218670
A	F			52	218722
A	P			46	218768
C				12965	231733
C	C			128	231861

Conclusion

The overall savings is three-fold. First, now all the analysts have to do is invoke this SAS autocall macro procedure with the proper keyword parameter settings to clean their claims instead of modifying their existing SAS code to handle new situations. Second, the operations involved are more direct—a few sort procedures, one transposition, and one merging data step. The new process is much simpler than calculating the total paid amount for all the claims and then determining which portions of the individual claims should be removed based on each claim's paid amount. Third, now we have a universally applied, and therefore consistent, method for 'cleaning' our claims.

References

SAS® Language: Reference, Version 6, Third Edition, 1990
 SAS® Macro Language: Reference, First Edition; 1997
 SAS® Procedures Guide, Version 6, Third Edition, 1990
 SAS® Macro Programming Made Easy, Michele M. Burlew, 1998
 Carpenter's Complete Guide to the SAS® Macro Language, Art Carpenter, 1998