SAS® GLOBAL FORUM 2018

USERS PROGRAM

Considerations of Analysis of Healthcare Claims Data

Bradley M. Casselman, M.S.

The University of Alabama

April 8 – 11 | Denver, CO

#SASGF

Considerations of Analysis of Healthcare Claims Data

Bradley M. Casselman, M.S.

The University of Alabama

ABSTRACT

- Healthcare related data is estimated to grow exponentially over the next few years, especially with the growing adaptation of electronic medical records and billing software. The growth in available data will yield many opportunities for healthcare analytics as analysts, researchers, and administrators look to leverage this information. One needs to consider the operational constructs of the data in order to process and analyze it in an efficient manner. Healthcare claims data is the transactional level of data that is the core from which most analysis of data results. Each claim can contain hundreds of variables about the course of care. Claims include diagnosis and procedure data as well as variables outlining information about the recipient of services (Enrollment), providers of services (Provider), and ancillary variables that outline a number of administrative fields. Key to understanding this data are the inherent structures as to how this data is stored and accessed. There are a number of considerations that need to be addressed in order to effectively process and manage the data to get it into an analytical form. This session will discuss utilizing common SAS procedures and data step functionality to address the various challenges specific to healthcare data. Topics will include SQL, transposing data, efficient joins, data structures, and considerations stemming from the claims adjudication process.

KEY CONSIDERATIONS

- For the analysis of healthcare claims, the analyst needs to consider a number of data management tasks PRIOR to undertaking a thorough analysis of the data or they risk generation of results that may be either misleading or wrong. Key to these considerations are:

1. Ensuring that the analyst has taken claim adjustments into consideration when going to analyze the data. Claims are often adjusted post-hoc and are usually contained in either a separate adjustment file or may be added to the original claims file via a table update. Knowing the structure of this data is key to any analysis of claims as an improper handling of adjustments can lead to incorrect, missing, or duplicate data.
2. Ensuring that there is a proper knowledge of claims and ancillary file layouts. Healthcare data at the claims level is often stored in disparate tables to gain storage efficiencies. However, that setup is usually not conducive to analysis of the data. The analyst must be able to understand all appropriate data structures not only relating to the claims themselves, but also to any number of ancillary information files, including, but not limited to: Provider files, Enrollment files, Procedure files, Diagnosis files, and any number of trailer records related to inpatient stays. Often techniques such as transposing files or complex SQL joins are necessary to get the data into an analysis ready state.
3. Once the data is in an appropriate analytical form, it is often extremely large and complex. Analysts need to precede their analysis by careful data management techniques to create efficiencies in their data access. For instance, strategic use of macro variables can often help the analyst create a more efficient data pull than pulling multiple joins which not only leads to efficiency, but also reduces the risk of coding error.

STRATEGIES

1) Claim Adjustments

First, let us consider the question of handling adjustments to claims. An adjustment occurs when any change is made to the initial billing of the healthcare claim. This can occur for many reasons. Some common situations that initiate claim adjustments are incorrect data entered on the claim by the provider of service, duplicate claims submitted by the provider, changes in deductibles, co-pays, or coinsurance, disallowance of a claim by the insurer, or a number of other reasons. This can create a number of issues, however, with the analysis of claims data in that the analyst must ensure that the most recent version of a claim is included for analysis. Otherwise, the analyst risks incorrect or incomplete data in their results.

Commonly, adjustments are handled in 3 different ways by database administrators (DBA's). First, they may append adjustments to the original claims table via an update. Typically, when this is done, a variable is created that indicates either a) a version number to indicate the iteration of the claim update or b) the DBA may include a date field that accompanies the revised claim as it is appended to the table. One can see where this strategy may cause immediate issues if not handled correctly as the table in this form will contain duplicated claim lines across the various versions that are loaded to the claim table. Another strategy is to c) keep the initial claims submission only in the claims table and to then accumulate adjustments in a separate file on a periodic (usually monthly) basis. In this situation, the analyst must approach the data management differently as they will need to "apply" the separate adjustment file to the original claims files and replace the now obsolete claims with the newer versions that exist on the adjustment file.

These situations will require the analyst to take approaches in data management reflective of the method employed by the Database Administrator. These situations are easily handled using simple data management approaches in SAS® by using either DATA step processing or, in the case of a separate adjustment file, by the efficient use of PROC SQL to join in the correct data and to output an updated claims file. For situations (a) and (b) above, it is often useful to use LAST. processing with the data step to filter out any prior versions of the claim. These tend to be the simplest situations for the analyst, however, one that DBA's do not often employ as the storage on the main claim table can get onerous and the fact that the DBA must continually update the original file as new adjustments are received through the claims processing process. To address situation (C) above, the analyst must prepare the adjustment file to merge with the original claims file. First, one must identify claims that exist on both tables. Next, those "original" versions need to be dropped from the claims table, and finally, the replacement "adjusted" claims from the adjustment table should be added to the claims table. Example coding for these situations can be found in *Figure 1* below.

Considerations of Analysis of Healthcare Claims Data

Bradley M. Casselman, M.S.

The University of Alabama

STRATEGIES CONTINUED

2) Handling of Ancillary Data

Additional to the information that is pertinent to the claim, most of the information about the provider of record and the beneficiary of the service are usually contained in separate files. Most healthcare data handlers refer to these as Provider and Enrollment tables. These tables usually contain essential information relative to the analysis of the overall healthcare record such as provider or beneficiary address, provider tax ID, enrollment start and/or end dates, and dates of birth. This data typically is joined in to the claim record as well for ease of analysis, but here to, the analyst must employ similar data management techniques. For instance, under the Affordable Care Act, beneficiaries can enroll and dis-enroll sometimes multiple times per year if they have a qualifying event. As such, the analyst may only want to consider claims submitted during this period. However, similar to the other ancillary data, the provider enrollment data may be stored vertically, with separate entries on multiple lines for each enrollment period and the analyst may have to employ similar tactics by using PROC TRANSPOSE (or an ARRAY) to move the data into a horizontal form that would be conducive to joining into the main claim table. An example of a PROC TRANSPOSE to accommodate multiple enrollment periods is shown in *Figure 2*.

Once a clean record of claims is established, the analyst must now contend with assembling a variety of information about the claim. However, it is typical that only basic information about the claim is actually stored on the claim file. The analyst then must review a number of ancillary files that are related to the health record. For instance, while procedure and diagnosis code information may be included on the claim file, the descriptions of those codes are usually kept in a separate "lookup" table. Likewise, on facility claims for instance, there may be a number of ancillary files that contain specific information about a claim, such as diagnosis, occurrence, or value codes. To make these data more complicated, whereas the analyst typically wants to see the data in a one claim line per record format, these files are typically stored vertically, where the claim number repeats each time for the respective code desired to add to the main table. In these cases, transposing the data via PROC TRANSPOSE or the use of an ARRAY may be necessary. Other times, it may be necessary to move the data from horizontal to vertical, depending on the data needs. *Figure 3* shows how to use an array with diagnosis codes to go the other way, horizontal to vertical, if the data demands it.

3) Other Considerations for Efficient Claims Analysis

There are numerous other tips and techniques for assisting with the efficiency of claims analysis once the entire record is prepared for analysis using the strategies above. While these techniques are too numerous to include all for this poster, I do want to mention some of the more common useful techniques, including using PROC SQL to create macro variables to select claims conditionally, storing various SELECT clauses as a macro variable for repeated queries, and use of efficient PROC SQL joins

STRATEGIES CONTINUED

Use of a macro variable is particularly helpful for the analyst, specifically for queries into the data that look for all claims that contain a certain procedure. Procedure codes are claim line items and a particular claim may contain numerous claim lines. Thus, if one used a WHERE clause, they would only receive those claim lines, and not the others on the claim. One strategy around this is to utilize code similar to the adjustment situation (c) above and use multiple joins. However, a better strategy is to use a PROC SQL SELECT INTO clause to create a macro variable of the unique claim numbers as shown in *Figure 4*. Another useful aspect of macros is the ability to store code and execute it dynamically. For instance consider basic reporting on claims data. Many analysts will create reports in PROC SQL, however there may be a large number of created report queries whereas the end user only needs one or a few. One solution is to create a generic SQL report code and store the parameters as macro variables then only call the ones needed. An example is shown in *Figure 5*.

Last, due to the complexity of data, it is imperative that an analyst know all sources of data and how the various tables that contain the necessary fields for reporting and analysis and how the variables in those tables relate together. In healthcare data, this can lead to some very complicated joins when selecting data from multiple tables. *Figure 6* shows an example of simply the FROM clause of a complex join of healthcare claims data from various sources to illustrate the complexity inherent in the selection of data and how the order of the joins to the master table is of high importance. (Note that there is a join to the provider table twice. Once to get information on the practicing provider, and once to get information on the referring provider).

CONCLUSION

In conclusion, healthcare data is a vast and complex network of data that resides on a great number of interrelated tables. The analyst must be able to not only be able to analyze data in its ready form, but must possess a deep understanding of the data structures, storage mechanisms, and interrelations, but most importantly must have a knowledge of how the claims are processed so that he or she may be able to prepare the data for meaningful analysis. It is often said that analysis is 80% data preparation and 20% analytics, but in this case, it may even be more extreme due to the complex nature existent in healthcare data. That being said, the savvy analyst can utilize the techniques presented here and others to efficiently and confidently produce accurate and meaningful analysis, as long as one takes the proper care in the preparation and management of the claims and ancillary data.

Considerations of Analysis of Healthcare Claims Data

Bradley M. Casselman, M.S.

The University of Alabama

CODE EXAMPLES

```
/*  FIGURE 1  */

/* EXAMPLE 1:  ADJUSTMENTS LISTED IN-LINE VIA AN INDICATOR VARIABLE (VERSION NUMBER) */

proc sort data=claimfile;
  by claim_number claim_version_number;
run;

data newtable; set claimfile;
  by claim_number claim_version_number;
  if last.claim_version_number;
run;

/* EXAMPLE 2:  ADJUSTMENTS LISTED BY A DATE VARIABLE */

proc sort data=claimfile;
  by claim_number claim_submission_date;
run;

data newtable; set claimfile;
  by claim_number claim_submission_date;
  if last.claim_date;
run;
```

```
/* EXAMPLE 3:  ADJUSTEMENTS CONTAINED IN A SEPARATE FILE */

  /* STEP 1 - Identify unique claim numbers that require adjustment */

proc sql;
  create table adjclaims as
  select distinct a.claim_number, 1 as indicator
  from claimfile a inner join adjustmentfile b
    on a.claim_number = b.claim_number;
quit;

  /* STEP 2 - Join back to claim file to mark adjustments with indicator and drop old claims */

proc sql;
  create table claimsfileprep as
  select a.*, b.indicator
  from claimsfile a left join adjclaims b
    on a.claim_number = b.claim_number;
quit;

data claimsfileprepupdated; set claimsfileprep;
  if indicator = 1 then delete;
run;

  /* STEP 3 - Append new adjustments back to the claims file */

proc sql;
  create table adjclaims2 as
  select  a.*
  from adjustmentfile a inner join adjclaims b
    on a.claim_number = b.claim_number;
quit;

data newclaims; set claimsfileprepupdated adjclaims2;
run;
```

Considerations of Analysis of Healthcare Claims Data

Bradley M. Casselman, M.S.

The University of Alabama

CODE EXAMPLES

```
/* FIGURE 2 */

/* Sort data to prepart for transposing */

proc sort data=elig_pd1;
   by id enrollment_start_date;
run;

/* Transpose Provider Enrollment Start Date */

proc transpose data=elig_pd1 out=elig_pd2s;
   by id;
   var enrollment_start_date
run;

/* Transpose Provider Enrollment End Date */

proc transpose data=elig_pd1 out=elig_pd2e;
   by id;
   var enrollment_end_date ;
run;

/* Merge the two together for a whole record */

proc sql;
   create table elig_pd2 as
   select a.id, a.col1 as enrollment_start_date1, b.col1 as enrollment_end_date1, a.col2 as enrollment_start_date2, b.col2 as enrollment_end_date2,
          a.col3 as enrollment_start_date3, b.col3 as enrollment_end_date3
     from work.elig_pd2s a inner join work.elig_pd2e b
       on a.id = b.id
 order by a.id;
quit;

/* Merge back to the main Provider of Service file */

proc sql;
   create table person_b5 as
   select a.*, b.enrollment_start_date1, b.enrollment_end_date1, b.enrollment_start_date2, b.enrollment_end_date2, b.enrollment_start_date3,
          b.enrollment_end_date3
     from work.person_b4 a inner join work.elig_pd2 b
       on a.id = b.id;
quit;
```

```
/* FIGURE 3 */

/* USING AN ARRAY TO TRANSPOSE DIAGNOSIS RECORDS - VERTICAL TO HORIZONTAL */

data test_facsubmit (keep=unq_mem_id diag icd_version lst_dt_svc); set ttt;
 array diags {40} diag_cd_1-diag_cd_40;
 do i=1 to 40;
   if diags{i} ne "~" then do;
     DIAG=diags{i};
   output;
   end;
 end;
 rename member_id = unq_mem_id;
 run;\
```

```
/* FIGURE 4 */

/* USING PROC SQL SELECT INTO: TO CREATE A MACRO VARIABLE TO SELECT ALL CLAIMS */

proc sql;
   select distinct "'"||bene_number||"'" into:  bene_list separated by ","
   from claimstable
   where procedure_code = "A1234";
quit;

proc sql;
   select *
   from claimstable
   where bene_number in (&bene_list.);
quit;
```

Considerations of Analysis of Healthcare Claims Data

Bradley M. Casselman, M.S.

The University of Alabama

CODE EXAMPLES

```
/* FIGURE 5 */

/* USING MACROS TO STORE CODE */

%macro report;

proc sql;
  create table &rptname. as
  &selstmt.
  from claimstable
  group by &grpvar.
  order by &ordervar.
quit;

%mend report;

/* REPORT 1 */

%let rptname = prov_sum_npi;
%let selstmt = select distinct RNDR_PROV_NPI,
                  rndr_prov_last_nm,
                  rndr_prov_first_nm,
                  rndr_prov_mid_init,
                  count(distinct HIC_NR) as hic_cnt label='HICN Count',
                  count(distinct CCN) as clm_cnt label='Claim Count',
                  SUM(Tot_Chrg_Amt) as Billed_Amt format=dollar18.2 label='Total Charges',
                  SUM(ALWD_AMT) as Allowed_Amt format=dollar18.2 label='Allowed Amount',
                  SUM(PROV_PD_AMT) as Provider_Paid_Amt format=dollar18.2 label='Paid Amount',
                  SUM(Billed_Units) as Total_units_billed format=comma13.1 label ='Total Billed Units',
                  SUM(allowed_units) as Total_units_allowed format=comma13.1 label ='Total Allowed Units';
%let grpvar = 1,2,3,4;
%let ordervar = 9 desc;

/* REPORT 2 */

%report

%let rptname = npi_yr_sum;
%let selstmt = select distinct
                  year(SVC_FROM_DT) as SVC_YEAR label='Year',
                  RNDR_PROV_NPI,
                  rndr_prov_last_nm,
                  rndr_prov_first_nm,
                  rndr_prov_mid_init,
                  count(distinct HIC_NR) as hic_cnt label='HICN Count',
                  count(distinct CCN) as clm_cnt label='Claim Count',
                  SUM(Tot_Chrg_Amt) as Billed_Amt format=dollar18.2 label='Total Charges',
                  SUM(ALWD_AMT) as Allowed_Amt format=dollar18.2 label='Allowed Amount',
                  SUM(PROV_PD_AMT) as Provider_Paid_Amt format=dollar18.2 label='Paid Amount',
                  SUM(Billed_Units) as Total_units_billed format=comma13.1 label ='Total Billed Units',
                  SUM(allowed_units) as Total_units_allowed format=comma13.1 label ='Total Allowed Units';
%let grpvar = 2,1,3,4,5;
%let ordervar = 1,10 desc;

%report
```

```
/* FIGURE 6 */

/* COMPLEX SQL JOIN ILLUSTRATION */
proc sql;  select /*  */
 FROM mainfile a inner join CLM on clm.CLM_HIC_NUM=a.HICN
 left outer join CLM_INSTNL ON (CLM.GEO_BENE_SK=CLM_INSTNL.GEO_BENE_SK
                                and CLM.CLM_DT_SGNTR_SK=CLM_INSTNL.CLM_DT_SGNTR_SK
                                and CLM.CLM_TYPE_CD=CLM_INSTNL.CLM_TYPE_CD
                                and CLM.CLM_NUM_SK=CLM_INSTNL.CLM_NUM_SK)
 left outer join CLM_BILL_FREQ_CD ON (CLM_INSTNL.CLM_BILL_FAC_TYPE_CD=CLM_BILL_FREQ_CD.CLM_BILL_FAC_TYPE_CD
                                and CLM_INSTNL.CLM_BILL_CLSFCTN_CD=CLM_BILL_FREQ_CD.CLM_BILL_CLSFCTN_CD
                                and CLM_INSTNL.CLM_BILL_FREQ_CD=CLM_BILL_FREQ_CD.CLM_BILL_FREQ_CD)
   INNER JOIN BENE ON (BENE.BENE_SK=CLM.BENE_SK)
   LEFT OUTER JOIN GEO_ZIP5_CD  C_GEO_BR ON (C_GEO_BR.GEO_SK=BENE.GEO_SK)
   LEFT OUTER JOIN GEO_FIPS_STATE_CD  C_GEO_FIPS_STATE_CD_BR ON (C_GEO_FIPS_STATE_CD_BR.GEO_FIPS_STATE_CD=C_GEO_BR.GEO_FIPS_STATE_CD)
   INNER JOIN CLM_DT_SGNTR ON (CLM_DT_SGNTR.CLM_DT_SGNTR_SK=CLM.CLM_DT_SGNTR_SK)
   LEFT OUTER JOIN CLM_FED_PRVDR_SPCLTY_CD ON (CLM.CLM_BLG_FED_PRVDR_SPCLTY_CD=CLM_FED_PRVDR_SPCLTY_CD.CLM_FED_PRVDR_SPCLTY_CD)
   INNER JOIN CLM_LINE ON (CLM.GEO_BENE_SK=CLM_LINE.GEO_BENE_SK
                                and CLM.CLM_DT_SGNTR_SK=CLM_LINE.CLM_DT_SGNTR_SK
                                and CLM.CLM_TYPE_CD=CLM_LINE.CLM_TYPE_CD
                                and CLM.CLM_NUM_SK=CLM_LINE.CLM_NUM_SK)
   LEFT OUTER JOIN CLM_POS_CD ON (CLM_LINE.CLM_POS_CD=CLM_POS_CD.CLM_POS_CD)
   INNER JOIN PRVDR_NPI  A_PRTY_RPCL_NPI ON (CLM_LINE.PRVDR_RNDRNG_PRVDR_NPI_NUM = A_PRTY_RPCL_NPI.PRVDR_NPI_NUM)
   LEFT OUTER JOIN CLM_LINE_MCS ON (CLM_LINE.GEO_BENE_SK=CLM_LINE_MCS.GEO_BENE_SK
                                and CLM_LINE.CLM_DT_SGNTR_SK=CLM_LINE_MCS.CLM_DT_SGNTR_SK
                                and CLM_LINE.CLM_TYPE_CD=CLM_LINE_MCS.CLM_TYPE_CD
                                and CLM_LINE.CLM_NUM_SK=CLM_LINE_MCS.CLM_NUM_SK
                                and CLM_LINE.CLM_LINE_NUM=CLM_LINE_MCS.CLM_LINE_NUM)
   LEFT OUTER JOIN CLM_FED_PRVDR_SPCLTY_CD  A_CLM_FED_PRVDR_SPCLTY_CD_RP ON (CLM_LINE.CLM_RNDRG_FED_PRVDR_SPCLTY_CD=A_CLM_FED_PRVDR_SPCLTY_CD_RP.CLM_FED_PRVDR_SPCLTY_CD)
   LEFT OUTER JOIN PROD_REV_CTR_CD ON (CLM_LINE.CLM_LINE_REV_CTR_CD=PROD_REV_CTR_CD.PROD_REV_CTR_CD)
   INNER JOIN PRVDR_NPI  A_PRTY_BP_NPI ON (A_PRTY_BP_NPI.PRVDR_NPI_NUM = CLM.PRVDR_BLG_PRVDR_NPI_NUM)
   INNER JOIN PRVDR_NPI  A_PRTY_RF_NPI ON (A_PRTY_RF_NPI.PRVDR_NPI_NUM = CLM.PRVDR_RFRG_PRVDR_NPI_NUM)
   LEFT OUTER JOIN CLM_CNTRCTR_NUM ON (CLM.CLM_CNTRCTR_NUM=CLM_CNTRCTR_NUM.CLM_CNTRCTR_NUM)
   INNER JOIN CLM_TYPE_CD ON (CLM.CLM_TYPE_CD=CLM_TYPE_CD.CLM_TYPE_CD
                                and CLM.CLM_MDCR_PHASE_CD=CLM_TYPE_CD.CLM_MDCR_PHASE_CD)
   LEFT OUTER JOIN CLM_PROD_MTRLZD ON (CLM.GEO_BENE_SK=CLM_PROD_MTRLZD.GEO_BENE_SK
                                and CLM.CLM_DT_SGNTR_SK=CLM_PROD_MTRLZD.CLM_DT_SGNTR_SK
                                and CLM.CLM_TYPE_CD=CLM_PROD_MTRLZD.CLM_TYPE_CD
                                and CLM.CLM_NUM_SK=CLM_PROD_MTRLZD.CLM_NUM_SK)
```

SAS® GLOBAL FORUM 2018

April 8 – 11 | Denver, CO
Colorado Convention Center

#SASGF