Paper 17-2010

Building Match Code Using SAS®

David Li, Prime Therapeutics, Eagan, MN

Abstract

Business analysis frequently entails joining data from different sources. Oftentimes, the linkage must be made based on a customer's name and address data. A SAS name and address match code can accomplish this task and move the project forward without interruption.

This paper highlights three concepts: (1) modular design, (2) looped sequential processing, and (3) the parsing of data elements. SAS Macros can compactly implement the basic match code. They organize the code modules and perform a sequence of tasks against a single input record at a time. Macros also manipulate the parsing and transforming of text strings into the final match code. The programming techniques necessary to move a data element through the various stages are discussed in detail.

This paper details the process of parsing a sub element from a specific variable of a data set, converting a data set variable into a macro variable, and storing a macro variable value as a record within a data set, and provides other useful tips as well.

Introduction

Business analysis is employed in many different sectors to forecast customer behavior. In the retail sector, merchants attempt to predict which customers will make another purchase. Telecom carriers want to know which subscribers will churn (cancel service). In healthcare, pharmacy benefit managers (PBMs) want to identify which members will use the mail channel to fill their prescriptions. All of these businesses need a great deal of information about their customers to build the best predictive models.

Analytics cannot take place without data. Grouping all relevant information about a customer is an important data management activity. The bulk of the information for analysis usually comes from internal transaction data. However, some external data are very predictive. The insurance industry's use of credit information to predict driver behavior is a prime example. Name and address matching is frequently utilized to link information from diverse sources. Once disparate customer information has been joined, it can be analyzed using a universal customer identifier.

Similar examples of the need for comprehensive customer data are readily found in banking, nonprofits, and other industries. Accurate customer data is needed not only for analytics, but also for operational efficiency in customer service centers, the creation of customized marketing messages, modeling of customer lifetime values, and mailing of private health information. Finally, the combined data set yields an information value greater than the sum of its parts. High value analytic processes such as predictive modeling, forecasting, and healthcare benefit design are all ravenous consumers of data.

The Basic Concept

Match code creates a standard view for different renderings of the same information. Its algorithm must account for different name and address formats and deal with transcribing errors. The probability of a correct match is increased by focusing on the initial of the last name, the numeric part of the address, and the 5-digit zip code.

The general matching process is described as follows:

1. Count the number of records in an input file.
2. Loop through the input file and process each record individually.
3. Convert specific variable values in the record into macro variables.
4. Modify the macro variables in accordance with the match code logic.
5. Combine components into one match code per record.
6. Save the match code results to the output file.
7. Repeat the sequence for all input files.
8. Join the different data files using the match code.

The match code building process starts with two or more differing customer data files. Each of these files contains some non-overlapping information that we wish to combine to create a new view of the customer. To match the files based on name and address, last name (LastName), address (Address), and zip code (ZipCd) are used to build the

match code. These three data elements allow accurate matching at the household level. Additional data fields may be needed to link the generated match code to other data tables.

A sample input file may look like the data table below.

LastName	ZipCD	Address	UniqueID
LANE	55408	3241 E CALHOUN CIRCLE	001
SMITH	55666	12584 PARK AVE	002

A data preparation step is needed here, so that the letters are all capitalized and special characters removed. It is considerably easier to prepare the data as a separate step than to try and anticipate all possible variations.

Building Macro Modules

To implement the match code algorithm, macro modules are first sketched out using pseudo code. From an initial rough outline, details are iteratively developed to refine the coding tasks. The design of the main macro module greatly influences the fit of the other components. This code starts by determining how many times to call a yet unknown process, "process each record," to create the match code. Then it stores the returned values.

```
%macro MatchBld(dsin, dsout);
      Count the number of records (N).
      Create a do loop to process the input file N times.
          Process each record.
          Create a temporary table, once, to hold the processed outputs.
          Insert data into the temporary table one record at a time.
      End do loop.
      Create the output data set containing the final match code.
%mend;
```

Once the programming logics have been worked out in the pseudo code, SAS coding can begin. The %MatchBld (Match code Build) macro obtains two pieces of information from its macro parameters. The input data set name (dsin or data set in) tells the program where to find the name and address data for processing. The output data set name (dsout or data set out) tells it where to write the results of the newly created match code.

The SQL Procedure's SELECT INTO statement is used to produce a count of how many records are in the input data set. It converts the count from a descriptive statistic into a value held by a macro variable, NCNT. The NOPRINT option is chosen to prevent the default output from being displayed.

NCNT tells the Do Loop how many times to call the macro that will "process each record." The LastName and Address variables are modified separately and then stored together in an output data set. We will cover these processes in more depth later.

The match code is designed to process each record sequentially. In each sequence, it transforms three (3) different input variables into a single match code variable. The %ObsLp (Observation Loop) macro is used to "process each record." It routes the variables through additional macro modules that modify the different input variables into match code components. The modified results are returned to the main macro, %MatchBld, for final assembly.

A temporary data table is created when %MatchBld's Do Loop is executed. To create the table only once, the macro statement %IF, %THEN and %ELSE is applied to branch off the first pass-through, where the table creation takes place. After the first pass, the INSERT INTO SQL statement saves the modified LastName and Street Address values. It also converts the macro variables &LNCSTNT and &ADDRCSTNT back into variable values to be stored as a data file record.

```
%macro MatchBld(dsin, dsout);
      proc sql noprint;
            select count(*) into:NCNT from &dsin;
      quit;
      %do i=1 %to &NCNT;
      %ObsLp(&dsin, &i);
            %if &I=1 %then %do;
                  proc sql noprint;
                        create table MODWORD (LNCSTNT char(32), ADDRCSTNT char(200));
                        insert into MODWORD values("&LNCSTNT", "&ADDRCSTNT");
                  quit;
            %end;
            %else %if &I>1 %then %do;
```

```
                proc sql noprint;
                      insert into MODWORD values("&LNCSTNT", "&ADDRCSTNT");
                quit;
          %end;
    %end;
                  .
                  .  (Additional data step statements)
                  .
%mend;
```

Looped Sequential Processing

Only two of the three input variables, LastName and Address, need to be modified for the match code. The 5-digit zip code, ZipCd, is incorporated into the match code as is, for a specific name, address, and zip code combination. Therefore, when creating macro variables from data set variables, our code design calls for LastName and Address to be the first step in the looped sequential processing. These transformation tasks are coordinated by the looping macro, %ObsLp.

Creating Macro Variables

Since this match code algorithm is written in SAS Macro, it needs to be able to transition seamlessly between SAS data set variables and SAS macro variables. It creates macro variables from the data set and other inputs through four common macro variable creation methods:

1. Passing parameters into a macro module.
2. Applying SELECT INTO SQL statement.
3. Applying CALL SYMPUT statement.
4. Applying %LET statement.

In %MatchBld, we have already encountered methods #1 and #2. The looping macro, %ObsLp, reads in the ith record each time it is called from the input data set. It then uses the CALL SYMPUT statement to create macro variables &LN and &ADDR1 from the data set record. It calls two more macros, %LNMOD (Last Name Modified) and %ADDRMOD (Address Modified), to process the newly created macro name and address variables. The macro variables are then passed as parameters to the new macros.

```
%macro ObsLp(dsin, i);
data _NULL_;
      set &dsin (firstobs=&i obs=&i);
      call symput('LN', LN);
      call symput('ADDR1', ADDR1);
run;
            %LNMOD(&LN);
            %ADDRMOD(&ADDR1, ADDROUT);
%mend;
```

Since the 5-digit ZipCd is used as is, there is no need to create a transforming macro for it.

Parsing Data Elements

The actual match code building activities are performed in the two macros, %LNMOD and % ADDRMOD. A generic phonetic algorithm generates the LastName match code component. It uses the first letter of the last name, eliminates the second duplicate consonant letter, and converts the 2nd through 6th characters into a predefined numeric equivalent. Then the address variable is transformed at the word level.

In order to affect these changes, LastName is decomposed into individual letters while Address is decomposed into individual words. Parsing out sub elements of LastName and Address and then reconstitute the results is the final piece of the puzzle in building match code.

Working with the LastName Variable

The %LNMOD macro dissects the word (text string) and eliminates the duplicate consonant. It counts the word length and uses the count to iterate through a Do Loop and parse out each letter. Each letter is inserted into a temporary SAS data set “letterlist” as a record. Thus, if the last name is “SMITH,” then the data set “letterlist” would contain five records, as shown in the table below.

Letterlist
S
M
I
T
H

The INSERT INTO statement from PROC SQL is used to create an individual record of the letters from the &WORDIN (Word In) macro variable. When calling a function within the macro environment, %SYSFUNC is invoked to execute the standard SAS functions “LENGTH” and “SUBSTR.” A SAS data step with a RETAIN statement handles the identification of duplicate letters.

The final macro creation method, %LET, can be found in the macro %LNMOD.

```
%MACRO LNMOD(WORDIN);
      %global LNCSTNT;
      %let WORD_LENGTH=%sysfunc(length(&WORDIN));
      %do I=1 %to &WORD_LENGTH;
            %let LETTER=%sysfunc(substr(&WORDIN, &I, 1));
            %if &I=1 %then %do;
                  proc sql noprint;
                        create table letterlist (LETTER CHAR(32));
                        insert into letterlist values("&LETTER");
                  quit;
            %end;
            %else %if &I>1 %then %do;
                  proc sql noprint;
                        insert into letterlist values("&LETTER");
                  quit;
            %end;
      %end;
      data letterlist;
            set letterlist;
            retain LAST_LETTER;
            if _N_ = 1 then do;
                  LAST_LETTER=LETTER;
                  MOD_LETTER=LETTER;
            end;
            else if _N_>1 then do;
                  if upcase(LAST_LETTER)=upcase(LETTER) then MOD_LETTER="";
                  if upcase(LAST_LETTER) ne upcase(LETTER) then MOD_LETTER=LETTER;
                  LAST_LETTER=LETTER;
            end;
            NAME=MOD_LETTER;
            if NAME ne "";
            keep NAME;
      run;
      %WordBld(letterlist);
      %let LNCSTNT=&XVARNM;
%MEND;
```

This data set is then processed to reconstitute the letters into a single modified word through the word built macro, %WordBld (Word Build). This technique calls for using CALL SYMPUT to convert each letter’s values, (represented in the data set “letterlist”), into a series of macro variables. The modified last name is realized from variable concatenation and compression (removing the blanks).

Building the LastName Match Code Component

The %WordBld macro is a very useful macro routine. It elegantly creates a series of macro variables from the values of a data set and counts the number of records at the same time. This new arrangement of information allows a quick Do Loop to concatenate the letters into a single word text string.

```
%MACRO WordBld(dsin);
      %GLOBAL XVarnm;
      %let XVarNM =;
      %let N = 0;
```

```
      DATA _NULL_;
            SET &dsin;
            if _N_ > 0 then do;
                  call symput("X_"||LEFT(_N_), NAME);
                  call symput("N", _N_);
            end;
      RUN;
      %if &N > 0 %then %do;
            %do i = 1 %to &N;
                  %let XVarNm = &XVarNm. &&X_&i.;
            %end;
            %let XVarNm = %sysfunc(compress(&XVARNM));
      %end;
%MEND;
```

Working with Address Variable

After the last name has been processed, the looping macro %ObsLp sends the address value to macro %ADDRMOD for similar processing. Using the same concept as employed in the %LNMOD macro, the address value variable is parsed into a new temporary data set.

The macro %Decompose (Decompose) decomposes an Address sentence (text string) into its component words. The insight that made this macro possible is that we can determine the number of words in a sentence by counting the number of spaces between words. By counting the spaces and adding one (1), we arrive at the number of words in an address text string. Once the word count is established, the CREATE TABLE statement from PROC SQL is used to create a SAS data set that holds each word of the Address sentence as an observation.

```
%MACRO DECOMPOSE(LISTIN, DSOUT);
%LET I = 1;
%LET cdh = 0;
%if %sysfunc(compress(&listin)) = "" %then %let wordcount = 0;
%else %do;
      %let wordcount = %eval(%length(%sysfunc(trim(&listin))) -
                        %length(%sysfunc(compress(&listin))) + 1);
%end;
%if &wordcount = 0 %then %do;
      PROC SQL NOPRINT;
            CREATE TABLE &DSOUT. (CATEGORY CHAR(100));
      QUIT;
      %let cdh=1;
%END;
%else %DO %UNTIL (&cdh = 1);
    %LET WORD = %SCAN(&LISTIN, &I);
      %if &i>&wordcount %then %let cdh=1;
      %ELSE %DO;
            %IF &I=1 %THEN %DO;
                  PROC SQL NOPRINT;
                        CREATE TABLE &DSOUT. (CATEGORY CHAR(100));
                        INSERT INTO &DSOUT. VALUES("&WORD.");
                  QUIT;
            %END;
            %ELSE %IF &I. > 1 %THEN %DO;
                  PROC SQL NOPRINT;
                        INSERT INTO &DSOUT. VALUES("&WORD.");
                  QUIT;
            %END;
      %LET I = %EVAL(&I + 1);
      %END;
%END;
%MEND;
```

The output of the %Decompose macro is modified to standardize the street name conventions. If the street name has a numeric component such as “First” or “Second”, these are changed to “1ST” or “2ND.” The common representations for a route are changed to “RT.” All directional destinations and the typical surface road designation (i.e., Street, Road, Ave, etc.) are eliminated. After the transformation, the word build macro, %WordBld, is once again called upon to rebuild the transformed address word.

```
%macro ADDRMOD(ADDR1, dsin);
%global ADDRCSTNT;
%DECOMPOSE(&ADDR1, &dsin);
proc sql;
      create table addrout1 as
      select
      case when upcase(CATEGORY)="ROUTE" then "RT"
            when upcase(CATEGORY)="RR" then "RT"
            when upcase(CATEGORY)="RFD" then "RT"
            when upcase(CATEGORY)="FIRST" then "1ST"
            when upcase(CATEGORY)="SECOND" then "2ND"
            when upcase(CATEGORY)="THIRD" then "3RD"
                  .
                  .  (Additional conversion statements)
                  .
        else CATEGORY
        end as NAME
      from &DSIN
      where upcase(trim(CATEGORY)) not in ("RURAL", "E", "N", "W", "S", "NE", "SW",
"SE", "NW", "AVE", "AVENUE", "BLVD", "HWY", "RD", "ROAD", "ST", "STREET", "TNPK",
"DR", "DRIVE", "TRL", "TRAIL", "CIR", "CIRCLE", "PL", "PLACE", "LANE", "LN", "CT",
"COURT",
. . .,  "PLAZA", "PARKWAY", "PKWAY");
quit;
%WordBld(addrout1);
%let ADDRCSTNT=&XVARNM;
%mend;
```

Putting It Together

Once the component parts have been transformed, they are assembled in the %MatchBld macro. The algorithm is built out of the Base SAS data step and should reflect the specific match code algorithm logic that is being implemented. The basic phonetic transformation rules have been around for quite some time, starting with the US Census conducted in the late 1800's. Many other phonetic schemes have been developed over time.

The result of this match code application is shown below. The 5-digit zip code, the 6-digit modified last name, and 5-digit modified address are concatenated together to form the MatchCode output.

LastName	ZipCD	Address	UniqueID	MatchCode
LANE	55408	3241 E CALHOUN CIRCLE	001	55408L500003241C
SMITH	55666	12584 PARK AVE	002	55666S5300012584

Conclusion

While this paper takes you through the process of building a SAS Macro program that solves a real world problem, the focus of the discussion is on the various techniques employed to manipulate data elements. SAS is one of the most versatile analytical tools and one of the reasons is its ability to assemble and transform data.

In summary, this paper illustrates the techniques utilized to:

1. Process individual records from an input file outside of a data step.
2. Create macro variables that contain the record count for an input file.
3. Extract the value of a specific variable at a specific observation and pass it to a SAS Macro.
4. Parse out the sub element of a variable and make it available for additional transformation.
5. Create a series of macro variables with a single CALL SYMPUT statement.
6. Reconstitute a series of variables into a single text string.

These techniques are demonstrated in the context of building a name and address matching application. I find that analyzing an actual piece of working SAS code is the best way to learn and retain SAS skills, because we then learn in context with readymade examples.

Contact Information

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

David Li
Prime Therapeutics LLC
1305 Corporate Center Drive
Eagan, MN 55121
Phone: (612) 777-4873
Fax: (612) 777-4403
E-mail: dli@primetherapeutics.com
Web: http://www.primetherapeutics.com

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.