SESUG Paper 229-2018

From CSV to SAS®: Dealing with Empty Fields and Repeated Observations

Smith, Kelly D.; Earls, Paul G., Central Piedmont Community College

ABSTRACT

Sometimes, higher education students need to register for classes before official transcripts from prior education institutions have arrived at their new institution. If a student needs to enroll in a class with prerequisites and official transcripts have not been evaluated, the student can obtain an override from an advisor, using unofficial documentation, in order to complete the registration process. At Central Piedmont Community College (CPCC) in North Carolina, information about student overrides is pulled from the institutional database as a CSV file. In a recent effort to standardize and improve student placement data maintained by Institutional Research, it was decided to transform the override CSV files into SAS data sets.

This paper summarizes the multi-step process involved in the data set creation. Producing the data set required the modification of student identification and date information into standard formats, the appropriate addition of student identification and date information to observations with missing values, and the creation of a condensed variable from multiple observations. Particular attention is paid to the use of IF THEN processing, the use of RETAIN and DROP in a DATA step, and the use of ARRAY to collapse a multiple-value variable. In addition, the use of internal check mechanisms within the SAS code is presented as a method for data validation. The creation of a SAS data set from institutional data is a beneficial step that maximizes the information's potential for later use.

INTRODUCTION

Accurate information on academic placement levels of beginning students is a necessary component of student outcome analysis for the Voluntary Framework of Accountability and other report frameworks. Information about student overrides is extracted from the CPCC student learning management system (LMS). When LMS data is pulled into SAS, the data may not be immediately useful for a number of reasons: formatting changes may need to be made, missing values may need to be addressed, and multiple-value variables may need to be collapsed. Within the data set program, internal checks should be included to verify data accuracy and usability following variable modification. Specific sections of SAS programming related to variable modification and internal check mechanisms are highlighted and discussed.

VARIABLE STANDARDIZATION

Override information is obtained through the use of an Informer report, exported as a comma separated values (CSV) file. The CSV file contains seven character variables, defined as in Table 1.

Name	Example	Definition
ADCPersonID	1234567	CPCC student identification number
ADCStuOvrCourse	CHM-110	Course prefix and number for override
ADCStuOvrReason	Pre-Requisite	Basis for override permission
ADCStuOvrTerm	2015FA	Academic term for override
ADCStuOvrStartDate	6/14/2013	Start date of override permission
ADCStuOvrEndDate	12/5/2013	End date of override permission
Comments	Transcript verification	Explanation of basis for override

Table 1. Variables in CSV

The CSV data is pulled into SAS through the INFILE procedure (Smith, Frye, & Earls, 2017). Three character variables present with formatting issues: ADCPersonID, ADCStuOvrStartDate, and ADCStuOvrEndDate. The two date variables need two digit day and month values (MM/DD/YYYY) while ADCPersonID values are frequently missing leading zeroes.

DATE FORMATTING

Not all date values need changing, and some date values only need one zero added, so it is important to design the process to analyze the initial date value. The final code uses temporary, interim variables (stdt, stdt2) as part of the process:

```
/* Making StartDate MM/DD/YYYY */

IF SUBSTR(ADCStuOvrStartDate,3,1) NE '/'
    THEN stdt=CATS('0',ADCStuOvrStartDate);
    ELSE stdt=ADCStuOvrStartDate;

IF SUBSTR(stdt,6,1) NE '/'
    THEN stdt2=CATS(SUBSTR(stdt,1,3),'0',SUBSTR(stdt,4));
   ELSE stdt2=stdt;

ADCStuOvrStartDate=stdt2;
IF ADCStuOvrStartDate EQ '00' THEN ADCStuOvrStartDate=' ';
```

The stepwise pattern of the code is important: if the day portion of the date is addressed prior to the month portion, the final result could still be inaccurate. The first step uses the SUBSTR procedure to check for the presence of a forward slash in position three. If the forward slash is not present, a zero is inserted in front of the month value with the use of the CATS procedure. If a forward slash is present, no change is made.

The second step of the process checks for the presence of a forward slash at position six. If a forward slash is not present, CATS and SUBSTR are used to add a zero in front of the day value. Otherwise, no changes are made. The final line of code removes zeroes from values that were initially blank. This is a necessary step for later processing. The same stepwise process is applied to standardize ADCStuOvrEndDate.

LEADING ZEROES

The student identification number at CPCC is formatted as a seven digit character variable. In the process of extracting data from the student LMS, it is not uncommon for leading zeroes to be dropped. For instance, 0035482 may become 35482. The processing logic used to identify and fix problematic student identification numbers is similar to the logic used to fix date values:

```
/* Adding Leading Zeroes to ADCPersonID */

IF ADCPersonID EQ ' ' THEN ADCPersonID=' ';
    ELSE IF SUBSTR(ADCPersonID,2,1) EQ ' '
          THEN ADCPersonID=CATS('000000',ADCPersonID);
    ELSE IF (SUBSTR(ADCPersonID,3,1)) EQ ' '
          THEN ADCPersonID=CATS('00000',ADCPersonID);
    ELSE IF (SUBSTR(ADCPersonID,4,1)) EQ ' '
          THEN ADCPersonID=CATS('0000',ADCPersonID);
    ELSE IF (SUBSTR(ADCPersonID,5,1)) EQ ' '
          THEN ADCPersonID=CATS('000',ADCPersonID);
    ELSE IF (SUBSTR(ADCPersonID,6,1)) EQ ' '
          THEN ADCPersonID=CATS('00',ADCPersonID);
    ELSE IF (SUBSTR(ADCPersonID,7,1)) EQ ' '
```

```
        THEN ADCPersonID=CATS('0',ADCPersonID);
    ELSE ADCPersonID=ADCPersonID;
```

SUBSTR is used to determine if there is a missing character at specific locations. Based on the location of the missing character, CATS is used to add the correct number of leading zeros to the value of ADCPersonID.

Another approach to the problem is to convert ADCPersonID into a numeric variable with the INPUT procedure and then back to a character variable with the PUT procedure, using the Z7 informat to add leading zeroes:

```
PersonID=INPUT(ADCPersonID,7.);
ADCPersonID=PUT(PersonID, Z7.);
```

For our particular situation, the first method is preferred since we also need to deal with missing values of ADCPersonID.

MISSING VALUES

The LMS at CPCC is a Unidata System: data values can be exported out of the LMS under the conditions of repeat/don't repeat and explode/don't explode. The standard export parameters CPCC has chosen are explode and don't repeat. Data for the Comments variable is exported from the LMS in stacked portions of a set length, creating an exploded multiple-value variable. Data for the ADCPersonID variable is not multi-value so the don't repeat parameter will create missing values in the exported data. Table 2 demonstrates how missing values are generated for ADCPersonID based on Comments.

ADCPersonID	Comments
1234567	Permission granted to take Accounting II
(missing value)	based on RSCC transcript showing student
(missing value)	passed Accounting I in Fall Quarter, 2012
(missing value)	with grade of "B"

Table 2. Creation of Missing Values for ADCPersonID

ADCPersonID, ADCStuOvrCourse, ADCStuOvrTerm, ADCStuOvrStartDate, and ADCStuOvrEndDate have short values and often present with missing values due to the export conditions. To fill in missing values, RETAIN and DROP are used:

```
DATA Over002;
    RETAIN PreviousID PreviousCourse PreviousTerm PreviousStart
                PreviousEnd;
    DROP   PreviousID PreviousCourse PreviousTerm PreviousStart
                PreviousEnd;
SET Over001;

IF (ADCStuOvrCourse EQ ' ')
    THEN ADCStuOvrCourse=PreviousCourse;
    ELSE PreviousCourse=ADCStuOvrCourse;

IF (ADCPersonID EQ ' ' AND ADCStuOvrTerm EQ ' ')
    THEN ADCStuOvrTerm=PreviousTerm;
    ELSE PreviousTerm=ADCStuOvrTerm;
```

```
IF (ADCPersonID EQ ' ' AND ADCStuOvrStartDate EQ ' ')
    THEN ADCStuOvrStartDate=PreviousStart;
    ELSE PreviousStart=ADCStuOvrStartDate;

IF (ADCPersonID EQ ' ' AND ADCStuOvrEndDate EQ ' ')
    THEN ADCStuOvrEndDate=PreviousEnd;
    ELSE PreviousEnd=ADCStuOvrEndDate;

IF ADCPersonID EQ ' ' THEN ADCPersonID=PreviousID;
          ELSE PreviousID=ADCPersonID;
RUN;
```

In this situation, the RETAIN statement is being used to carry values from one data step iteration to the next. For each iteration of the data step, five temporary variables are assigned to the program data vector (PDV) prior to the next observation being brought in from data set Over001. The values for these temporary variables are retained from the previous data step iteration. The DROP statement ensures the temporary variables assigned by RETAIN are not included in the new data set Over002. Without DROP, the variables would be written to the Over002 (Fine, 2011). DROP does not have to immediately follow RETAIN, but placement further down in the code may cause issues with PROC statements that precede the step boundary.

COLLAPSING MULTIPLE-VALUE FIELDS

Some advisors simply list the time and date of the override with their name, while other advisors include specific information regarding the source of the override information basis, and some advisors even include scanned copies of relevant emails. This results in comments with sizes from 1 to more than 100 data lines, creating a multiple-value field that needs to be collapsed to make the data set more useful. The PROC SORT procedure is used to organize the data in preparation for the remaining steps. Following a truncation step to make the comment field a more reasonable size, the ARRAY procedure is used to collapse the multiple records into one.

DATA ORGANIZATION

The PROC SORT procedure prepares the data for further analysis. The BY statement for the PROC SORT is chosen to match the BY statement that is planned for the truncation step which follows:

```
DATA      Comms (KEEP=      ADCPersonID
                            ADCStuOvrCourse
                            ADCStuOvrTerm
                            ADCStuOvrStartDate
                            ADCStuOvrEndDate
                            ADCStuOvrEndDate
                            Comments);
SET Over002;
PROC SORT DATA=Comms;
    BY ADCPersonID ADCStuOvrCourse ADCStuOvrTerm ADCStuOvrStartDate
       ADCStuOvrEndDate;
RUN;
```

TRUNCATING DATA

The original CSV file with override information is maintained within the data warehouse, and is available if needed. Therefore, it is not necessary to retain all the record lines from comments of an extreme size. To make the override data set more useful, we decided to truncate the comment field to 50 records:

```
DATA      TrunComms;
    SET   Comms;
```

```
    BY ADCPersonID ADCStuOvrCourse ADCStuOvrTerm ADCStuOvrStartDate
       ADCStuOvrEndDate;
    IF FIRST.ADCStuOvrStartDate THEN Count=0;
          Count+1;
          IF Count LE 50 THEN OUTPUT;
RUN;
```

The use of the BY statement compartmentalizes the data set into subsets for analysis. Multiple subsets may have the same ADCPersonID, but will be differentiated based on the other variables in the BY statement. At the start of each new ADCStuOvrStartDate, the count variable is reset to zero. As each new observation is read within the subset, the count variable increases by one. If the subset contains less than 50 records, all the records are output to TrunComms. If a subset contains more than 50 records, only the first 50 records are output to TrunComms.

COLLAPSING FIELDS

Once the comment field had been reduced to a more workable size, the individual records are collapsed into an array as the next step in creating a single value variable from a multiple value field. The TrunComms data set is segmented using the same BY statement as was used in the organizational and truncation steps. RETAIN is used to carry over data from one iteration to the next. The ARRAY statement establishes a fifty component (String01-String50) seventy-byte ($70.) character array named Combine.

```
DATA      Comms01;
RETAIN ADCPersonID ADCStuOvrCourse String01-String50;
 ARRAY Combine(50) $70. String01-String50;
    SET   TrunComms;
    BY ADCPersonID ADCStuOvrCourse ADCStuOvrTerm ADCStuOvrStartDate
    ADCStuOvrEndDate;
 IF FIRST.ADCStuOvrStartDate
    THEN DO;
    i=1;
          DO j=1 to 50;
            Combine(j)=' ';
          END;
    END;
    Combine(i)=Comments;
 IF LAST.ADCStuOvrEndDate THEN OUTPUT;
 i+1;
 RUN;
```

A two part DO loop is used to collapse the Comment variable into the Combine array. In the inner loop, missing values are assigned to each component of the array (String01-String50). The outer loop assigns the value of the Comment variable to the matching array component. For example, the value of Comment in record 5 of the subset will be copied into the String05 variable in the Combine array. When the last record in the subset has been read, the data is output to Comms01.

In the following step in the process of collapsing the Comments variable, a marker character (*) replaces missing values that are present in the Combine array. Since an array only persists for the existence of a DATA step, the Combine array must first be re-established:

```
DATA      Comms02;
 ARRAY Combine(50) $70. String01-String50;
 ARRAY Check(50) $70. Star01-Star50;
    SET   Comms01;
 DO i=1 to 50;
    IF Combine(i) EQ ' ' THEN Check(i)='*';
    ELSE Check(i)=Combine(i);
```

```
END;
RUN;
```

A new array, Check, is created to analyze the components of the Combine array (String01-String50). A DO loop is used for the analysis process. If a component of Combine does not have a missing value, the value is copied into the related component (Star01-Star50) of the new Check array. Otherwise, if a component of Combine contains a missing value, the marker character (*) is inserted into the related component of the Check array.

Now that all components of the Check array contain data from the original Comment variable or a marker character, a new variable (Combo) can be created and transformed into the final data set variable named ADCStuOvrComment:

```
DATA      Comms03 ;
    SET   Comms02;
LENGTH    ADCStuOvrComments $3500.
          Combo $3500.;
Combo=CATS(OF Star01-Star50);
ADCStuOvrComments=COMPRESS(Combo,'*');
RUN;
```

To create the new Combo variable, the CATS function is used to combine all fifty Star variables together. The use of CATS(OF var1 – varX) simplifies the code and also instructs SAS to use the TRIM(LEFT) function on each value being combined (SAS, 2011, p. 545). The final variable, ADCStuOvrComments, is created when the COMPRESS function removes all marker (*) characters from Combo.

INTERNAL CHECK MECHANISMS

Data entry errors can result in unexpected results from a program. A common place for a data entry error that would impact the new override data set is in the start date for override approval. On occasion, the start date is typed into the LMS with the year listed first rather than the month (2017/11/8 rather than 11/8/2017). When SAS applies the date formatting code (page 2) to 2017/11/08 the final value of ADCStuOvrStartDate becomes 02017/11/8. To check for potential errors, the Over002 data set is sorted by ADCStuOvrStartDate (saved as the Check data set) and the first ten observations are output with a PROC PRINT procedure. Problematic start dates rise to the top of the Check data set and can be easily fixed with a manual correction added to the program.

Before the final version of the new override data set is output to the data warehouse, a final check mechanism is run. This final check ensures no student overrides have been lost in the process of collapsing the multiple value field Comments:

```
DATA      Compare00;
    SET   Over002;
PROC SORT NODUPKEY DATA=Compare00; BY ADCPersonID ADCStuOvrCourse
    ADCStuOvrTerm ADCStuOvrStartDate;
RUN;

DATA      CompCheck;
    MERGE Compare00 (IN=A)
          Overrides (IN=B);
    BY ADCPersonID ADCStuOvrCourse ADCStuOvrTerm ADCStuOvrStartDate;
    IF A*B=0 THEN OUTPUT CompCheck;
RUN;

PROC PRINT DATA=CompCheck;
TITLE "Potential Errors in Overrides Dataset - Check ADCPersonIDs";
VAR ADCPersonID ADCStuOvrCourse;
RUN;
```

The Compare00 data set is created by the reduction of the Over002 data set to one observation per override record through the use of PROC SORT NODUPKEY. The Compare00 data set is then merged with the Overrides data set to determine if any records were lost in the process of collapsing Comments into ADCStuOvrComments. If an observation is not in both the Compare00 and Overrides data sets, that observation is output to CompCheck. Any missing overrides are documented with the PROC PRINT procedure.

CONCLUSION

Data is most useful when it does not have to be generated for each use: creating data sets from Informer reports enables analysts to focus on analysis questions rather than the repeated retrieval of static data from a LMS. Creating a data set also offers the opportunity to standardize data formatting, deal with missing values, and simplify unwieldly information. Analysts should always consider including internal check mechanisms to ensure information is not lost as data is formatted or simplified.

REFERENCES

Fine, L. (2011). Ready, SET, RETAIN, and then maybe reset. *Proceedings of the Global Forum 2011.* Available at http://support.sas.com/resources/papers/proceedings11/091-2011.pdf

SAS Institute Inc. (2011*). SAS® 9.2 Language reference: Dictionary* (4th ed.). Cary, NC: SAS Institute Inc.

Smith, K.D., Frye, B. E., & Earls, P. G. (2017). SAS® and the Voluntary Framework of Accountability: A prime example of the use of SAS in education. *SESUG 2017: The Proceedings of the 25th Annual SouthEast SAS Users Group, Cary, North Carolina.* Available at: https://analytics.ncsu.edu/sesug/2017/SESUG2017_Paper-96_Final_PDF.pdf

CONTACT INFORMATION

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

Kelly D. Smith
Central Piedmont Community College
704-330-2722 x3810
kelly.smith@cpcc.edu

Paul G. Earls
Central Piedmont Community College
704-330-6399
paul.earls@cpcc.edu