Paper 95

A Novel Approach to Code Generation for Data Validation and Editing using Excel and SAS

Mai Nguyen, Katherine Mason, Shane Trahan
RTI International, 3040 Cornwallis Rd, Research Triangle Park, NC 27709

ABSTRACT

A common task in the data processing of hardcopy surveys is data validation and editing. For complicated and/or lengthy surveys, this task can be very labor-intensive and error-prone. The challenge was to find a better way to improve the process of specifying validation/editing rules and implementing these rules in a more efficient manner. Validations may include things such as:

- Detecting skip pattern violations
- Detecting missing critical items
- Detecting logical inconsistencies between responses within a questionnaire

Often these types of validations and data cleaning/editing involve writing SAS code. Our team has developed a unique approach which uses Microsoft Excel to generate most of SAS code from the validation and edit rules specified in the spreadsheet. This technique provides multiple benefits:

- Codes are easily edited and updated automatically when the rules change
- Repeating patterns can be quickly observed and thus similar rules can be implemented quickly using a "copy and paste" technique
- There is better quality control because the rules and generated SAS code are kept together side-by-side in the spreadsheet

In this paper, we'll provide the implementation details of the technique and demonstrate the technique with some code samples. We'll also provide the pros and cons of this technique and possible enhancements for the future.

INTRODUCTION

As opposed to computerized surveys, hardcopy or paper surveys often require more robust QA processes to ensure data quality. Data validation can be implemented directly in computerized data-entry application or included as a separate step after the data has been entered. A drawback with the former approach is that hard-check errors need to be resolved before the remaining data can be entered. With the latter approach, data can be entered as is and validation can be performed in subsequence stages. Validations that are commonly needed are:

1. Skip-pattern validations – these validations are needed to detect if respondent provided data to questions that should have been skipped.
2. Logical inconsistency validations – These validations are usually needed to detect inconsistent responses to different questions in the questionnaire. For example, the sum of male and female residents in a nursing home should sum to the total number of residents.
3. Missing critical item validations – Some surveys require responses from certain questions in the questionnaire to ensure sufficient data were collected for analyses. Therefore, missing critical item validations are needed to flag the missing data so project staff can re-contact respondents to collect data for these critical items.

Another situation commonly encountered in both paper and computerized surveys is that corrections are needed to responses that were already entered into the system. Instead of re-opening each case, data corrections are usually logged and data edits will be applied in batch at a later stage.

Below we include examples of questions in our survey where these types of validations are needed. Sample SAS code is provided around these examples for illustration in later sections.

EXAMPLE 1 – QUESTION WITH SKIP LOGIC

Skip logic will direct respondents through different paths in the survey based on previously asked questions. Listed below in Figure 1 is an example of skip logic. Typically, this type of logic is frequently employed on questions that are Multiple Choice or Rating Scale type of questions.

[OTHOWN] 3. Is this adult day services center owned by any other type of organization?

Yes
No, not part of another organization (**skip to question 4**)

3a. **For each item (a - f) below**, please indicate whether or not this type of organization owns this center.

		Yes	No
[OWNHOSP]	a. Hospital		
[OWNSNF]	b. Nursing home or skilled nursing facility		
[OWNHHA]	c. Home health agency		
[OWNHOS]	d. Hospice agency		
[OWNADSC]	e. Assisted living or similar residential care community (e.g. adult care or personal care residence)		
[OWNOTH]	f. Other		

[MAXPART] 4. What is the maximum number of participants allowed at this adult day services center at this location? This may be called the allowable daily capacity and is usually determined by law or by fire code, but may also be a program decision.

____ Maximum number of participants allowed

Figure 1 – Skip Logic Example

In the figure above, OTHOWN is the gateway question and should a respondent check No, then question 3a can be ignored. Each questionnaire received must validate this type of response to ensure data quality and reliability.

EXAMPLE 2 – QUESTION REQUIRING LOGICAL INCONSISTENCY VALIDATION

Many questions have numerical equivalents that lend themselves to quality control checks. Without such checks, the data collected could be considered invalid. For example, a question may ask for a respondent's age and then later on in the survey we ask for a birth date. These two types of data can be compared to determine if the values being requested do in fact match

[TOTPART] 5. What is the total number of participants currently enrolled at this center at this location? Include respite care participants.

____ Number of participants

26. Of the participants currently enrolled at this adult day services center, how many are…

[MALENUM] _____ Male

[FEMNUM] _____ Female

Figure 2 – Inconsistency Validation Example

From the example above, question 5 asks for a number representing the total participants. Later on in the survey, we have repeated this question in a different way. In this case, the variables MALENUM and FEMNUM should sum to equal TOTPART asked earlier in the questionnaire instrument.

DEFINE VALIDATION SPECIFICATIONS IN EXCEL

Traditionally, validation specifications are detailed in a document and programs are written to implement the validations. Our approach combines validation specifications and snippets of generated SAS code that implement the validations by using an Excel spreadsheet. We provide 2 examples of specifications, one for a skip pattern validation and the other a logical inconsistency validation in the following sections.

SKIP PATTERN VALIDATION SPECIFICATION

Using Excel, we divide the coding into more manageable sections or specifications where conditions, actions and various code elements are each in their own columns of the spreadsheet. Then we use Excel functions to create needed text to be copied and pasted into a SAS validation program. Utilizing Excel gives flexibility to any variable name changes that could arise during development and increases efficiency of the code development process.

This specification consists of 2 primary elements:

1. Conditions to be checked – what conditions are to be evaluated during the validation process
    i. Gate condition – Checks if the follow-on questions should be skipped
    ii. Subordinate condition – Checks if there are responses when the questions should be skipped
2. Actions to be taken when a violation is detected – when a violation is detected, perform an action suitable for the data being evaluated. Data is expensive to collect and should be salvaged where possible.
    i. Set flag
    ii. Indicate the validation that fails
    iii. Increment failed validation counter

Specification for Example 1 above is shown in the Excel spreadsheet in Figure 1A below.

	A	B	C	D	E	I
1						
2	**Gate Condition**		**Subordinate Condition**		**Action**	**SAS Code**
3						
4	OTHOWN = 2					if (OTHOWN = 2) and (
5			OWNHOSP	NOT IN (., 0)		OWNHOSP NOT IN (., 0)
6		or	OWNSNF	NOT IN (., 0)		or OWNSNF NOT IN (., 0)
7		or	OWNHHA	NOT IN (., 0)		or OWNHHA NOT IN (., 0)
8		or	OWNHOSP	NOT IN (., 0)		or OWNHOSP NOT IN (., 0)
9		or	OWNRCF	NOT IN (., 0)		or OWNRCF NOT IN (., 0)
10		or	OWNOTH	NOT IN (., 0)		or OWNOTH NOT IN (., 0)
11					OTHOWN_ERR = 1; SKIP_ERR = 1; SKIP_ERR_COUNT = SKIP_ERR_COUNT + 1;	) then do OTHOWN_ERR = 1; SKIP_ERR = 1; SKIP_ERR_COUNT = SKIP_ERR_COUNT + 1; end;
12						

Figure 1A – Skip Logic Question Specification

LOGICAL INCONSISTENCY VALIDATION SPECIFICATION

Similar to the skip pattern validation specification, we divide our specification into more management pieces and each piece is entered into a separate column in the spreadsheet.

A logical inconsistency specification consists of 3 distinct elements:

1. Name of the validation
2. Conditions to be checked. The conditions for this validation often involve some computations and comparison(s) of the results of the computations
   i. LHS – Left-hand-side value of the comparison
   ii. RHS – Right-hand-side value of the comparison
   iii. Comparison operator
3. Actions to be taken when a violation is detected
   i. Set flag
   ii. Indicate the validation that fails

Specification for Example 2 is shown in the Excel spreadsheet in Figure 2A below.

	A	B	C	D	E	F	J
1							
2	**Val Name**	**LHS**	**Cond**	**RHS**		**Action**	SAS Code
3							
4	CHK_GENDNUM	(MALENUM +FEMNUM)	<	(0.90 * TOTPART)			if ((MALENUM +FEMNUM) < (0.90 * TOTPART))
5		(MALENUM +FEMNUM)	>	(1.10 * TOTPART)	OR		OR ((MALENUM +FEMNUM) > (1.10 * TOTPART))
6						CHK_GENDNUM_1 = 1; VAL_FAILED = 1;	then do CHK_GENDNUM_1 = 1; VAL_FAILED = 1; end;

Figure 2A – Inconsistency Validation Specification

The specifications in Excel listed above can be easily changed and managed. In addition, having the items organized in Excel allow these specifications to be shared among developers and stored in a centralized location for use later on or reuse in similar types of instruments. Using Excel also allows more input from non-developers since Excel is a common tool to many non-developers.

GENERATE SAS CODE IN EXCEL

We generated snippets of SAS code that implemented the validation specifications in the spreadsheet, side by side with the specifications. Code generation can be easily achieved using:

- String concatenation (&)
- Simple Excel formulas and functions, for example IF, AND, OR…
- Complicated conditions and codes can be broken into smaller chunks stored in different cells which are combined to produce the final code snippet

Once the formulas are created and generated codes verified, they can be “copied and pasted” to other validations of the same category.

The following sections show how we generated SAS code for the skip pattern and logical inconsistency validations.

CODE GENERATION FOR SKIP PATTERN VALIDATION

Figure 1B shows the formulas (column I) behind the validation code shown in figure 1A. Each cell simply concatenates text contents of the cells in columns F, G and H of the same row. String concatenation in Excel is specified with the "&" character. It should be noted that using the "&" is a string concatenation, so numbers will be treated as strings.

	A	B	C	D	E	I
1						
2	**Gate Condition**		**Subordinate Condition**		**Action**	**SAS Code**
3						
4	OTHOWN = 2					=F4&G4&H4
5			OWNHOSP	NOT IN (., 0)		=F5&G5&H5
6		or	OWNSNF	NOT IN (., 0)		=F6&G6&H6
7		or	OWNHHA	NOT IN (., 0)		=F7&G7&H7
8		or	OWNHOSP	NOT IN (., 0)		=F8&G8&H8
9		or	OWNRCF	NOT IN (., 0)		=F9&G9&H9
10		or	OWNOTH	NOT IN (., 0)		=F10&G10&H10
11					OTHOWN_ERR = 1; SKIP_ERR = 1; SKIP_ERR_COUNT = SKIP_ERR_COUNT + 1;	=F11&G11&H11

Figure 1B – Excel Formulas

Formulas for cells in columns F, G, and H are shown in Figure 1C. These functions build SAS code from the specification. Splitting these functions up help with simplification, having all these functions in just one cell could be difficult to read and cumbersome to find potential problems. It is highly advisable to break out the formulas into pieces and then concatenate together.

F	G	H	I
			SAS Code
=IF(ISTEXT(A4),"if (" & A4 & ") and (", "")		=IF(ISTEXT(E4), ") then do " & E4 & " end;", "")	=F4&G4&H4
=IF(ISTEXT(A5),"if (" & A5 & ") and (", "")	=IF(ISTEXT(C5), " &B5 & " " & C5 & " " & D5, "")	=IF(ISTEXT(E5), ") then do " & E5 & " end;", "")	=F5&G5&H5
=IF(ISTEXT(A6),"if (" & A6 & ") and (", "")	=IF(ISTEXT(C6), " &B6 & " " & C6 & " " & D6, "")	=IF(ISTEXT(E6), ") then do " & E6 & " end;", "")	=F6&G6&H6
=IF(ISTEXT(A7),"if (" & A7 & ") and (", "")	=IF(ISTEXT(C7), " &B7 & " " & C7 & " " & D7, "")	=IF(ISTEXT(E7), ") then do " & E7 & " end;", "")	=F7&G7&H7
=IF(ISTEXT(A8),"if (" & A8 & ") and (", "")	=IF(ISTEXT(C8), " &B8 & " " & C8 & " " & D8, "")	=IF(ISTEXT(E8), ") then do " & E8 & " end;", "")	=F8&G8&H8
=IF(ISTEXT(A9),"if (" & A9 & ") and (", "")	=IF(ISTEXT(C9), " &B9 & " " & C9 & " " & D9, "")	=IF(ISTEXT(E9), ") then do " & E9 & " end;", "")	=F9&G9&H9
=IF(ISTEXT(A10),"if (" & A10 & ") and (", "")	=IF(ISTEXT(C10), " &B10 & " " & C10 & " " & D10, "")	=IF(ISTEXT(E10), ") then do " & E10 & " end;", "")	=F10&G10&H10
=IF(ISTEXT(A11),"if (" & A11 & ") and (", "")	=IF(ISTEXT(C11), " &B11 & " " & C11 & " " & D11, "")	=IF(ISTEXT(E11), ") then do " & E11 & " end;", "")	=F11&G11&H11

Figure 1C – Detailed Excel Functions Creating SAS Code

Our final result (Figure 1D) is easy to read, understand and is a highly manageable code snippet to use in our validation scripts. With the template now ready, modifications can be done easily and could be managed by non-developers. Codes from column F, G, H are concatenated to produce the complete validation code as shown in Figure 1A.

F	G	H
if (OTHOWN = 2) and (		
	OWNHOSP NOT IN (., 0)	
	or OWNSNF NOT IN (., 0)	
	or OWNHHA NOT IN (., 0)	
	or OWNHOSP NOT IN (., 0)	
	or OWNRCF NOT IN (., 0)	
	or OWNOTH NOT IN (., 0)	
		) then do OTHOWN_ERR = 1; SKIP_ERR = 1; SKIP_ERR_COUNT = SKIP_ERR_COUNT + 1; end;

Figure 1D – Final Excel Output

CODE GENERATION FOR LOGICAL INCONSISTENCY VALIDATION

Similar to the skip pattern validations, logical inconsistency validations can also be scripted in Excel. Figure 2B shows the formulas and resulting SAS code to create validation code for a logical inconsistency validation. Again, using Excel provides a convenient and easy to use framework to develop validation scripts in SAS.

A	B	C	D	E	F	J
Val Name	**LHS**	**Cond**	**RHS**		**Action**	**SAS Code**
CHK_GENDNUM	(MALENUM +FEMNUM)	<	(0.90 * TOTPART)			if ((MALENUM +FEMNUM) < (0.90 * TOTPART))
	(MALENUM +FEMNUM)	>	(1.10 * TOTPART)	OR		OR ((MALENUM +FEMNUM) > (1.10 * TOTPART))
					CHK_GENDNUM_1 = 1; VAL_FAILED = 1;	then do CHK_GENDNUM_1 = 1; VAL_FAILED = 1; end;

Figure 2B – Excel Formulas

Figure 2C outlines the details and gives a behind-the-scenes look at the Excel formulas that build the needed SAS code. The “&” signs merge the strings located in the cell addresses together to create final SAS code.

G	H	I	J
			SAS Code
="if (" & B3 & " " & C3 & " " & D3 & ")"			=G3 &H3 &I3
	=" " & E4 & " (" & B4 & " " & C4 & " " & D4 & ")"		=G4 &H4 &I4
		="then do " &F5 & " end;"	=G5 &H5 &I5

Figure 2C – Detailed Excel Functions Creating SAS Code

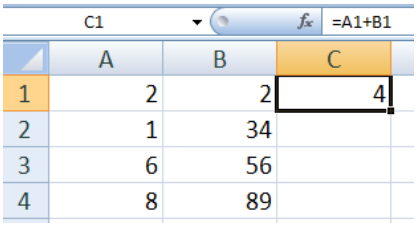
The final result of our Excel functions are in Figure 2D and reflect the validations that are needed to check the results entered for the TOTPART and MALENUM+FEMNUM questions.

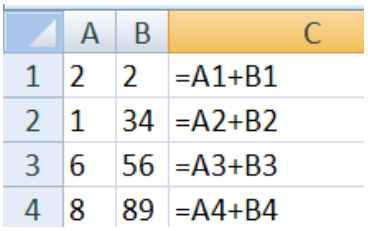
G	H	I	J
			SAS Code
if ((MALENUM +FEMNUM) < (0.90 * TOTPART))			if ((MALENUM +FEMNUM) < (0.90 * TOTPART))
	OR ((MALENUM +FEMNUM) > (1.10 * TOTPART))		OR ((MALENUM +FEMNUM) > (1.10 * TOTPART))
		then do CHK_GENDNUM_1 = 1; VAL_FAILED = 1; end;	then do CHK_GENDNUM_1 = 1; VAL_FAILED = 1; end;

Figure 2D – Final Excel Output

In the two examples above we showed how Excel can be used to create a flexible framework to create validations scripts. The framework we have created allows for easy management and modification of scripts throughout the development process. In addition to easier management of code, the use of Excel also provides functionality to create large amounts of similar scripts simply by using copy and paste or by simply dragging a cells content down a column. Excel is unique in that formulas in cells can be easily copied and transformed to the properties of the current row or column from which it resides.

For example, using column C to add the values of column A and B we simply enter "=A1+B1" into cell C1. Once the formula has been created, we can then easily copy and paste this formula to the other cells in the C column by using Copy and Paste. Excel is smart enough to change the formulas in the C column to reflect the values in the rows for which it is being inserted into.





This saves significant time for developers who may want to utilize the same basic code over and over again but have to simply change small parts of the code such as variable names.

INTEGRATE GENERATED SAS CODE INTO SAS PROGRAMS

The Excel-generated SAS codes can simply be "copied and pasted" into a DATA step with minimal initialization code. Figure 3 shows examples of the skip-pattern and logical inconsistency validation codes integrated into the DATA step. The PROC SQL following the DATA step selects cases that failed the validations.

```
DATA &out_dataset;

      /** Initialization **/
      SET ds_lib.&in_dataset;

      SKIP_ERR = 0;
      SKIP_ERR_COUNT = 0;
      VAL_FAILED = 0;

       /** SKIP-PATTERN VALIDATION GENERATED CODES **/
       if (OTHOWN = 2) and (
           OWNHOSP NOT IN (., 0)
          or OWNSNF NOT IN (., 0)
          or OWNHHA NOT IN (., 0)
          or OWNHOSP NOT IN (., 0)
          or OWNRCF NOT IN (., 0)
          or OWNOTH NOT IN (., 0)
       ) then do OTHOWN_ERR = 1; SKIP_ERR = 1; SKIP_ERR_COUNT =
SKIP_ERR_COUNT + 1; end;

       * Other skip-pattern validations...;


      /** LOGICAL-INCONSISTENCY VALIDATION GENERATED CODES **/
       if ((MALENUM +FEMNUM) < (0.90 * TOTPART))
              OR ((MALENUM +FEMNUM) > (1.10 * TOTPART))
       then do CHK_GENDNUM_1 = 1; VAL_FAILED = 1; end;

      * Other logical-inconsistency validations...;

RUN;

PROC SQL;

      /** Select cases that failed validation checks **/
      CREATE TABLE ds_lib.FailedValidations AS
      select * from &out_dataset
      where SKIP_ERR = 1 or VAL_FAILED = 1;

QUIT;
```

Figure 3 – Final SAS Validation Program

BENEFITS AND FUTURE DEVELOPMENT

There are several benefits that are immediately attainable with our approach:

- Having specifications and codes side-by-side ensure all validations are implemented. Furthermore, codes can also be more readily validated for correctness
- Once a validation is implemented, other validations of the same category can be quickly created using Excel's copy-and-paste
- Utilizing Excel to automate code generation allows users who are not primary developers to update and modify scripts as needed

Other benefits can be realized by having these workbooks programmed and automated in such a way that scripts can be generated and submitted to SAS engines from within Excel and reporting via ODS can be embedded within the Excel front end. This will further broaden a user base and free up code developers from having to update or modify

code especially for large dynamic data analysis projects. The methodology used here could also automate other types of reporting using SAS ODBC on the backend where changes in the Excel validation scripts can push various metrics reported in SAS to a database backend. From the database, information can be displayed or utilized on intranets or public facing websites. In the end, further broadening the base of users by automating script or code creation will enable higher response time to ad hoc requests and code/variable modifications.

CONTACTS

Mai Nguyen
Programmer Analyst
RTI International
3040 Cornwallis Rd
RTP NC 27709
919-541-8757
mnguyen@rti.org

Katherine Mason
Programmer Analyst
RTI International
3040 Cornwallis Rd
RTP NC 27709
919-541-7010
mason@rti.org

Shane Trahan
Programmer Analyst
RTI International
3040 Cornwallis Rd
RTP NC 27709
919-541-5848
srt@rti.org