

PDA Data Collection: Data Collection issues... SAS® Data Step to the Rescue!

Mira Shapiro, Independent Consultant, Bethesda, MD

ABSTRACT

With the increased use of handheld devices for data collection, some interesting data management issues have emerged. This paper will describe the challenges we overcame in order to prepare and analyze data that was presented to us at the end of a pilot study designed to measure the effectiveness of a PDA-based health education program.

We were presented with data that was collected on PDAs that were given to the participants at the beginning of a cancer screening education pilot study and then collected at the end of the study. The individual files were uploaded to our server for analysis. We were faced with a number of challenges, such as the presence of multiple coding schemes for the same question, the option for a participant to use the PDA's "back button" to change answers to survey questions, the need to extract the subject number from the filename, and the fact that some responses to a given question were accepted as text strings, while others were accepted as numeric. By combining a series of *data step* programming techniques and using *proc sort*, we were able to overcome multiple challenges and prepare the data for analysis.

This paper is appropriate for novice to intermediate Data Step programmers.

INTRODUCTION

Often statisticians and statistical programmers are presented with data that was collected prior to their involvement with a project. The project described here is one that I completed as a graduate student at the University of Massachusetts Biostatistics Center. We were presented with data that was collected from Personal Digital Assistants (PDAs) that were distributed to participants in a health education study. The participants were contacted multiple times per day, presented with some educational material concerning prostate cancer screening, and then asked to answer a series of questions.

This discussion is focused on the challenges presented by this pilot study's data and on how we prepared the data for analysis. What first seemed to be daunting challenges were easily conquered in a few hours through use of the SAS® *Data Step*, *Proc Format*, and *Proc Sort*.

CHALLENGES

The challenges we faced, which were evident as soon as we received the data, were:

1. Each subject's responses were stored in a separate file with no subject identifier other than in the filename.
2. Multiple days and sessions per day were indicated with a date and time format that did not conform to any of the SAS-supplied formats.
3. The data contained the following idiosyncrasies:
 - a. Multiple item numbers really referred to the same question and needed to be analyzed as such.

b. Responses to a given question sometimes were coded as text strings and sometimes as numeric.

c. Respondents had the option of using the PDA's "back button" to change their response. Use of the back button was reflected in the data as the line "BACK BUTTON" between the original and revised responses.

As we proceeded through our examination of each of these issues, we realized that by using some simple "tried and true" SAS techniques, we would be able to create a clean dataset for analysis in a much shorter timeframe than we expected.

DATA SAMPLE

Following are examples of a filename and its data file. These examples are included to facilitate the reader's understanding of the sample code and of the descriptions of the challenges we encountered and the techniques we used to solve them.

Filename:

```
x:\depts\biosta\statistical consulting\edu\input data\subject67.csv
```

Data file:

Session	DateCreated	Questionnaire	Question	Answer	
1	2009-07-11T09:28:31	QuestionnaireMon	Item1	yes	
1	2009-07-11T09:28:39	QuestionnaireMon	Item2	BACK BUTTON	
1	2009-07-11T09:28:51	QuestionnaireMon	Item1	No	
1	2009-07-11T09:29:09	QuestionnaireMon	Item2	Cook/eat/drink	
1	2009-07-11T09:29:21	QuestionnaireMon	Item3	Probably yes	
1	2009-07-11T09:29:36	QuestionnaireMon	Item4	Family	
1	2009-07-11T09:29:59	QuestionnaireMon	Item6		2
1	2009-07-11T09:30:05	QuestionnaireMon	Item7		2
1	2009-07-11T09:30:16	QuestionnaireMon	Item8		3
1	2009-07-11T09:30:27	QuestionnaireMon	Item9		4
1	2009-07-11T09:30:33	QuestionnaireMon	Item10		4
1	2009-07-14T09:30:38	QuestionnaireMon	Item539		5
...					
1	2009-07-14T09:30:46	QuestionnaireMon	Item540	BACK BUTTON	
1	2009-07-14T09:30:47	QuestionnaireMon	Item541	BACK BUTTON	
1	2009-07-14T09:30:48	QuestionnaireMon	Item542		2

SOLUTIONS

Since the entire SAS program is too lengthy to be included, the description of each solution will be followed by an excerpt from the program that illustrates the techniques applied.

READING ONE FILE PER SUBJECT AND EXTRACTING THE SUBJECT NUMBER FROM THE FILENAME

In order to read in each subject's file and extract the subject number from the filename, all the fully qualified dataset names were listed following a *datalines* statement. Each line was read via an *input* statement into an eighty-character text field, parsed for the subject number, and then used as a filename variable for an *infile dummy*

statement. The *infile dummy* statement allows for the reading of several raw data files within the same *data step*. Processing the records within the file was accomplished by using a *do while* loop that was terminated when the end-of-file condition was reached, whereupon the next record was read.

The following are some of the programming statements that were used to accomplish this task:

```
data psarecoded (keep= subject studyday session date_time day itemno answer_text
answer);
/* define order for kept variables */
length subject studyday session itemno answer 4. answer_text $40. ;
length fn q item $80 dt $19; /* allow long file path name*/
retain studyday session lagsubject lagdate laganswer_text lagitemno;
input fn $char80.; /*read full filename instream, pick off subject number. */
    st=index(fn,'Subject')+8;
    len=index(fn,'.')-st;
    subject=substr(fn,st,len);
infile dummy filevar=fn dlm=',' end=done firstobs=3; *data starts on 3rd row;
do while (not done); * read data from current input file;
    input one dt $ q $ item $ answer_text $;
...
end;*while not(done)..end of this file go to next one;
...
datalines;
x:\depts\biosta\statistical consulting\edu\input data\subject33.csv
x:\depts\biosta\statistical consulting\edu\input data\subject67.csv
```

USING THE LAG FUNCTION TO SIGNAL A NEW SUBJECT AND TO CREATE STUDY DAY AND SEQUENCE NUMBERS

In order to initialize the new day and a new session for each new subject, the *lag* function was used to trap the previous subject number, answer text, and item number. The previous subject number (*lag*) was compared to the current subject number, and if they were not equal, the session and day variables were initialized.

The code is excerpted here:

```
...
lagsubject=lag(subject);
laganswer_text=lag(answer_text);
...
lagitem=lag(itemno);
...
/* create studyday & session sequence numbers */
if subject ne lagsubject
    then do; * intialize day/session for new subject;
        startdate=day - 1;
        studyday=1; session=1;
    end; 'BACK BUTTON'
        then do; * same subject, session # not increased
        else if itemno=1 and lagitemno ne 1 and laganswer_text NE if
            repeated item 1 or
                retured to item1 with backbutton;
        if day=lagdate then session=session + 1; * new session;
        else do; *
            *reset studyday, session;
            studyday=day-startdate;
            session=1;
        end; end;
```

EXTRACTING AND RECODING THE ITEM NUMBERS

Accompanying the data was a long list of item numbers and their respective possible responses, along with a list of which item numbers were equivalent. Using a *select* statement with the *in* operator provided a self-documenting, efficient way to transform the item numbers into a consistent set.

Following is an example of some of the code used to transform the item numbers:

```
/* get item number, recode to correspond to question numbers. */
m=index(item,'m')+1;
l=index(item,' ')-m;
no=substr(item,m,l);
itemno=input(no,4.);
lagitemno=lag(itemno);
select;
    when( 538 le itemno le 556 ) itemno=itemno-536 ;
    when( itemno=23 ) itemno=22;
    when( itemno in (28, 29, 30, 31) ) itemno=23;
    when( itemno in (32, 33 34 35 36 37 38 39) ) itemno=24;
    when( 40 le itemno le 55 ) itemno=25;
    when( 56 le itemno le 87 ) itemno=26;
    when( 88 le itemno le 151 ) itemno=27;
    when( 152 le itemno le 279 ) itemno=28;
    when( 280 le itemno le 535 ) itemno=29;
    when (itemno = 536) itemno=30;
    otherwise;
end; /*select*/
```

USING *FORMATS* AND THE *INPUT* AND *TRIM* FUNCTIONS TO PROCESS TEXT STRINGS AND NUMERIC ENTRIES

In order to transform the answer text field to a consistent answer type, we used the *trim* function to capture the non-blank characters in the field. For responses that began as character representations of numbers, we used the *in* operator to identify them and then used the *input* function to fill the answer field with an integer. For text string responses that needed transformation to numeric answers, we used *formats* based on the item number in conjunction with the *input* function to fill the answer field with an integer value. In order to accomplish this transformation, a *select* statement was used with an item number to choose the appropriate format for the character-to-numeric transformation.

As a result, the answer field, unless the response was “BACK BUTTON”, was filled with an integer. Here are the *proc format* and program statements just described.

```

proc format;    * integer codes to replace text responses;
value $ny
'Yes'=1
'No'=0;
value $itwo
'Job'=1
'Cook/eat/drink'=2
'Exercise'=3
'Rest/sleep'=4
'TV/entertainment/hobby'=5;
value $ithree
'Self'=1
'Family/friends'=2
'Job'=3
'Responsibilities at home'=4
'TV/entertainment/hobby'=5;
...
value $itwenty
'Definitely not'=1
'Probably not'=2
'I`m not sure'=3
'Probably yes'=4
'Definitely yes'=5;
value $ithirty
'Much less concerned'=1
'A little less concerned'=2
'It had no effect'=3
'A little more concerned'=4
'Much more concerned'=5;

```

```

/* convert answer_text to numeric variable answer, using formats defined at beginning.*/
if trim(answer_text)='BACK BUTTON' then answer=.;
else do;
    * convert everything other than BACK BUTTON to integer codes;
    if trim(answer_text) in ('0','1','2','3','4','5','6','7','8','9')
        then answer=input(trim(answer_text),4.);
    Else do;
        * convert text responses to integers codes;
        select(itemno);
            when(2)  answer=put(trim(answer_text),$itwo.);
            when(3)  answer=put(trim(answer_text),$ithree.);
            when(4)  answer=put(trim(answer_text),$ifour.);
            when(14) answer=put(trim(answer_text),$ifourteen.);
            when(15) answer=put(trim(answer_text),$ififteen.);
            when(17,18,19) answer=put(trim(answer_text),$i789teen.);
            when(20) answer=put(trim(answer_text),$itwenty.);
            when(1,21,22,23,24,25,26,27,28,29)
                answer=put(trim(answer_text),$ny.);
            when(30) answer=put(trim(answer_text),$ithirty.);
            otherwise answer=.;
        end; /*select*/
        if answer=. then put itemno= answer= answer_text=; * list problems in
log.;
        end; /*else do*/
end; /* convert everything other than back button...*/

```

PROCESSING A “BACK BUTTON” ENTRY

If the subject decided to change a response to a question, the application permitted them to hit the back button. In such a data file the initial response would appear, then the words “BACK BUTTON” on the next line, then the amended response on the line thereafter. Initially we considered an approach utilizing the *lag* function. However, we discovered that if there were more than one subject/day/session/item combination, then we could capture the

combination with the latest timestamp in order to get the subject's final response to the question. After sorting the file by subject, day, session, item number and datetime, we then *set* the file by the string of sort variables in a subsequent data step, creating *first* and *last* variables for each combination. We then used the *last* construct to select the latest answer and output it to the dataset. This approach was far simpler than using the *lag* function and was more efficient and less error-prone. The following program statements were used to process the "BACK BUTTON" entries.

```
/* "back button" - if more than one per subject/day/session/item take latest.
   sort on subject/studyday/session/itemno/day_time, select last.itemno*/
proc sort data=psarecoded;
by subject studyday session itemno date_time;
run;
data psa.psaclean ;
set psarecoded;
by subject studyday session itemno date_time;
if last.itemno then output;
```

CONCLUSION

What first seemed like a daunting task became a satisfying exercise in teamwork and SAS programming. My supervisor at the time of the project and I have more years of SAS programming experience than we would like to admit. By putting our heads together and each using some of our favorite "tried and true" SAS programming techniques, we quickly came up with a solution to each of the challenges associated with this data.

One of my favorite SAS constructs is the *Select* statement. In this project its repeated use was the core of our efficient, self-documenting, and simple solution. My colleague's suggestion of using *proc sort* and a *data step* rather than exploiting the *lag* function to process "back button" entries is a great example of how collaboration can result in a simple and elegant solution.

This small project reminded me why so many experienced programmers still find SAS interesting and fun. There is always something new to learn, and those techniques that you've used over the years never go out of style.

ACKNOWLEDGMENTS

Special thanks to Elaine Puleo, PhD, and Eva Goldwater, MS, of the University of Massachusetts Biostatistics Center, Amherst, MA.

CONTACT INFORMATION

The author can be emailed at mira.shapiro@gmail.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.