Paper 90-2018

Let's Talk About Variable Attributes

Elizabeth Guerrero Angel, University of California, Davis

ABSTRACT

Variables can come in all shapes and sizes. Most data we receive are from external data sources, such as extracting data from a complex data entry system, reading in Excel spreadsheets, or importing text files with millions of observations. When combining data from various sources we often have to deal with different variable formats or names that are not functional. This presentation with cover novel solutions to these complex problems, in an easy to understand way, using PUT, INPUT, SUBSTR, and MDY functions. We will demonstrate how these frequently used functions can be used to change formats and fix dates, and can be combined with more complex code to and quickly rename long lists of variables. Variable characteristics in SAS® can seem like a pesky problem that won't go away, but we will help you navigate the world of variable attributes. The programming is demonstrated at a beginner's level.

INTRODUCTION

In contrast to what we all learn in school, most data we receive is not in a workable format, and it is not easy to combine and analyze right off the bat. In particular, when reading data into SAS from different sources, which can range from outside collaborators, government sources such as registries, and varying internal databases, things can get messy. Have you ever tried to combine two data sets that should have overlapping variables, only to run into an error message? Or reading in an Excel spreadsheet, only to have funky looking numbers in lieu of what should be dates? Solutions to these problems can range from simple statements in a DATA step, to working around and manipulating the data to get the correct results. In this paper, you will learn some useful tips for handling issues with the variable formats and attributes, ranging in complexity from easy to technical. Examples used throughout come from a completely fictional list of names, date of birth, and sex for 24 fictional children.

STATEMENTS AND FUNCTIONS IN THE DATA STEP

The first round of techniques to look at are the easiest, typically involving a statement within a DATA step. Consider the following: upon importing an Excel spreadsheet. Below is the code to import the simple spreadsheet consisting of a first name (Fname), a date of birth (DOB), and sex (Sex) using the IMPORT procedure:

```
proc import dbms=xlsx
    file= 'C:\Users\Desktop\list.xlsx'
    out=thelist
    replace;
run;
```

Checking the log, Display 1 shows that the spreadsheet was successfully imported:

```
NOTE: The import data set has 24 observations and 3 variables.
NOTE: WORK.THELIST data set was successfully created.
NOTE: PROCEDURE IMPORT used (Total process time):
      real time             0.03 seconds
      cpu time              0.01 seconds
```

Display 1. Log for output of PROC IMPORT.

Using the CONTENTS procedure, we can check the data, and make sure the variables have imported correctly:

```
proc contents data=thelist; run;
```

Below is a part of the output produced, Output 1:

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat	Label
2	DOB	Char	14	$14.	$14.	DOB
1	Fname	Char	8	$8.	$8.	Fname
3	Sex	Char	1	$1.	$1.	Sex

Output 1. PROC CONTENTS from spreadsheet import.

In most cases, variables such as DOB are not in the cleanest form, so reading it in will not automatically result in the expected DATE format once imported into SAS. Looking at both the DOB and SEX variables, different functions in SAS can be used to obtain the proper formatting for these issues.

For the simplicity's sake, the data set only has 24 observations. It is easy to see why our date variable did not automatically import into SAS as a date; not all of the dates are complete. This one observations whose DOB did not fit into our typical date formatting threw everything off, and now the rest of our dates are in a strange numeric format:

```
proc print data=thelist; run;
```

Below is the output for the PROC PRINT in Output 2:

Obs	Fname	DOB	Sex
1	Anthony	41186	1
2	Bailey	40795	1
3	Carson	41661	1
4	David	43028	1
5	Emily	41075	2
6	Felicity	42938	2
7	Gordon	42593	1
8	Hailey	March 99, 2013	F
9	Isaiah	42149	1
10	John	41619	1
11	Kelly	42331	2
12	Lucia	42049	2
13	Martha	41883	2
14	Nancy	42465	2
15	Oliver	42960	1
16	Patty	42913	2
17	Ronda	40971	2
18	Sean	42128	1
19	Taylor	41997	2
20	Urias	41864	1
21	Veronica	42436	2
22	Wanda	42740	2
23	Yessica	42052	2
24	Zachary	43030	1

Output 2. PROC PRINT output of imported data.

SOME USEFUL FUNCTIONS

When dealing with import issues from EXCEL, some of the solutions may be specific to EXCEL documents as will be demonstrated here.

The first thing is to convert all the dates in a funny number to a proper date. One important thing to note is that Excel dates are not the same as SAS dates. When stored in SAS, a date is actually a number, the number of days from January 1, 1960. Excel similarly stores dates as a number, but it is calculated by taking the number of days since January 1, 1900. The first thing we will need to do is identify those

observations with a number value, and convert them from character to numeric using the INPUT function (conversely, should you want to convert from numeric to character, the PUT function would be used). We will identify these using the LENGTH function, but this may not always be the best method, depending on the complexity of the field, and use the FORMAT statement before the SET statement to set the format for our new variable:

```
data thelist2;
   format dob_correct date9.;
   set thelist;
   if length(DOB) = 5 then DOB_Correct = input(DOB, 8.) - 21916;
run;
```

When using the INPUT function, make sure to identify the proper numeric informat SAS should read your data into. It is more important when using the PUT function, since having too long of an informat could result in unwanted spaces. So, did this work?

```
proc print data=thelist2; run;
```

Here is the PROC PRINT below in Output 3:

Obs	Fname	Sex	DOB	DOB_correct
1	Anthony	1	41186	04OCT2012
2	Bailey	1	40795	09SEP2011
3	Carson	1	41661	22JAN2014
4	David	1	43028	20OCT2017
5	Emily	2	41075	15JUN2012
6	Felicity	2	42938	22JUL2017
7	Gordon	1	42593	11AUG2016
8	Hailey	F	March 99, 2013	.
9	Isaiah	1	42149	25MAY2015
10	John	1	41619	11DEC2013
11	Kelly	2	42331	23NOV2015
12	Lucia	2	42049	14FEB2015
13	Martha	2	41883	01SEP2014
14	Nancy	2	42465	05APR2016
15	Oliver	1	42960	13AUG2017
16	Patty	2	42913	27JUN2017
17	Ronda	2	40971	03MAR2012
18	Sean	1	42128	04MAY2015
19	Taylor	2	41997	24DEC2014
20	Urias	1	41864	13AUG2014
21	Veronica	2	42436	07MAR2016
22	Wanda	2	42740	05JAN2017
23	Yessica	2	42052	17FEB2015
24	Zachary	1	43030	22OCT2017

Output 3. PROC PRINT with DOB corrections.

Now to tackle the problem causer, using a function called SCAN comes in handy:

```
data thelist3;
   set thelist2;
   if length(DOB) > 5 then do;
         month_c = scan(DOB,1);
         day_c = scan(DOB,2);
         year_c = scan(DOB,3);
   end;
 run;
```

The SCAN function uses your input variable (a string of characters), and the word “position”, to return the word in said position. Checking using PROC PRINT, we see that the new variables, month_c, day_c, year_c, are exactly what we want:

```
proc print data=set thelist3;
   where fname = ‘Hailey’;
run;
```

In Output 4 you will find the results of the PROC PRINT:

Obs	Fname	DOB	Sex	month_c	day_c	year_c
8	Hailey	March 99, 2013	F	March	99	2013

Output 4. PROC PRINT after using SCAN.

Now from here it is possible to build the correct date. There are several ways to do this, including using PROC FORMAT and a series of PUT and INPUT statements. In a more straightforward method, a series of IF/THEN statements will also work.

Now, the day variable is still in dire need of fixing. Noting from **Error! Reference source not found.**, the day value is ‘99’. Depending on your data, what it will be used for, and how to handle missing data, you will need to fill in this with a valid day if you want to obtain a final date value. Often, the missing date is filled in with the midpoint of the month, so we will fill it in with 15. Finally, all of the pieces are put together to fill in that final date variable using the MDY function, which stands for “Month, Day, Year”:

```
data thelist3;
   set thelist2;
   if length(DOB)>5 then do;
         month_c = scan (DOB,1);
         day_c = scan(DOB,2);
         year_c = scan(DOB,3);

         if upcase(month_c) = 'JANUARY' then month=1;
         if upcase(month_c) = 'FEBRUARY' then month=2;
         if upcase(month_c) = 'MARCH' then month=3;
         if upcase(month_c) = 'APRIL' then month=4;
         if upcase(month_c) = 'MAY' then month=5;
         if upcase(month_c) = 'JUNE' then month=6;
         if upcase(month_c) = 'JULY' then month=7;
         if upcase(month_c) = 'AUGUST' then month=8;
         if upcase(month_c) = 'SEPTEMBER' then month=9;
         if upcase(month_c) = 'OCTOBER' then month=10;
         if upcase(month_c) = 'NOVEMBER' then month=11;
         if upcase(month_c) = 'DECEMBER' then month=12;

         if day_c='99' then day = 15;
         DOB_Correct = mdy(month,day, put(year_c,4.));
    end;
    drop month_c day_c year_c month day;
  run;

   proc print data=thelist3 (obs=8); var fname dob sex dob_correct; run;
```

When checking if this fixed our problem, observing the PROC PRINT output shows this addressed the issues and now the date of birth is correct:

Obs	Fname	DOB	Sex	dob_correct
1	Anthony	41186	1	04OCT2012
2	Bailey	40795	1	09SEP2011
3	Carson	41661	1	22JAN2014
4	David	43028	1	20OCT2017
5	Emily	41075	2	15JUN2012
6	Felicity	42938	2	22JUL2017
7	Gordon	42593	1	11AUG2016
8	Hailey	March 99, 2013	F	15MAR2013

Output 5. PROC PRINT of Corrected DOB.

OTHER USEFUL FUNCTIONS

Another very useful function is the SUBSTR function, which substrings variables, parsing out certain characters from the variable. For example, in the above data set, the month value of 'MARCH' could be substringed and used in conjunction with the CATS (concatenates character strings while removing spaces from the beginning and end of the strings):

```
data thelist3;
   set thelist2;
   if length(DOB)>5 then do;
         month_c = scan (DOB,1);
         day_c = scan(DOB,2);
         year_c = scan(DOB,3);

         if day_c='99' then day = '15';
         month_c_2 = upcase(substr(month_c,1,3));
         DOB_Correct = input(cats(day,month_c_2,year_c), date9.);
   end;
run;
proc print data=thelist3 (obs=8); var fname dob sex dob_correct; run;
```

Running this code gives the same output as Output 5.

When reading in files in other formats, such as text, it is much easier to control the formatting of the data. Below is the text file, thelist.txt, in a fixed width format:

```
Anthony  4-Oct-12 1
Bailey   9-Sep-11 1
Carson  22-Jan-14 1
David   20-Oct-17 1
Emily   15-Jun-12 2
Felicity22-Jul-17 2
Gordon  11-Aug-16 1
Hailey  99-Mar-13 2
Isaiah  25-May-15 1
John    11-Dec-13 1
Kelly   23-Nov-15 2
Lucia   14-Feb-15 2
Martha   1-Sep-14 2
Nancy    5-Apr-16 2
Oliver  13-Aug-17 1
Patty   27-Jun-17 2
Ronda    3-Mar-12 2
Sean     4-May-15 1
```

```
Taylor  24-Dec-14 2
Urias   13-Aug-14 1
Veronica 7-Mar-16 2
Wanda    5-Jan-17 2
Yessica 17-Feb-15 2
Zachary 22-Oct-17 1
```

Display 2.Text file of the list.

When reading in a file such as this, using the INPUT statement wisely can help more easily control the data, and make it easier to fix data issues:

```
data thelist;
   infile "C:\Users\Desktop\thelist.txt";
   input
    @1    FNAME  $8.
    @9    DAY     2.
    @12   MONTH  $3.
    @16   YEAR    2.
   ;
run;
```

Here the variable start points are specified after "@", as well as the desired name, a "$" signifying a character variable, and the length of each in the text file. By reading each part of the date separately, we avoid the inability SAS has to process dates with unknown values.

A mix of these functions and statements can be used to correct data issues, depending on the data source and the "messiness" of the variables.

THE RENAMING CHALLENGE

Renaming variables can often be challenging when doing so for a long list. One method is to use a combination of the procedures: CONTENTS, SQL, and DATASETS. Returning to our data set, thelist3, we will use this as a simple example of using this method. PROC DATASETS allows for easy data set management, which is illustrated in the ease of renaming in this example:

```
proc contents data=thelist3 out=listContents; run;
```

Using the OUT option results in a new data set, listContents. Of interest, is the NAME variable, for which the values are the variable names. Next is creating a data set creating our list of new names. In this case, adding the suffix "_NEW", but it can easily be changed to a prefix, or using SUBSTR to take only part of the old variable:

```
data newnames;
   set listcontents (keep=name);
   newnames=cats(NAME,'_NEW');
run;
proc print data=newnames; run;
```

Here is Output 6 with the results:

Obs	NAME	newnames
1	DOB	DOB_NEW
2	Fname	Fname_NEW
3	Sex	Sex_NEW
4	dob_correct	dob_correct_NEW

Output 6. PROC PRINT of newnames data.

Next is using PROC SQL to create a macro variable, nameslist. This variable is essentially creating the phrase required from a RENAME statement: olvar = newvar, storing it as a text string. In the code below, the NOPRINT option is called, but Output 7 shows the output for demonstration:

```
proc sql noprint;
   select catx("=",name, newnames)
   into :nameslist separated by " "
   from newnames;
quit;
```

```
DOB=DOB_NEW
Fname=Fname_NEW
Sex=Sex_NEW
dob_correct=dob_correct_NEW
```

Output 7. PROC SQL output.

Finally, we invoke PROC DATASETS to rename the variables in the data:

```
proc datasets library=work;
   modify thelist4;
   rename &nameslist;
run;
quit;

proc contents data=thelist4; run;
```

Output 8 below shows the PROC CONTENTS output with the fancy new variable names:

```
Alphabetic List of Variables and Attributes
#    Variable          Type    Len    Format    Informat    Label
3    DOB_NEW           Char     14    $14.      $14.        DOB
2    Fname_NEW         Char      8    $8.       $8.         Fname
4    Sex_NEW           Char      1    $1.       $1.         Sex
1    dob_correct_NEW   Num       8    DATE9.
```

Output 8. Output for PROC CONTENTS with new variable names.

CONCLUSION

Data comes from different sources, in different formats, and will always require checking and cleaning. The power of the combination of various functions and statements in SAS offers the flexibility you need to convert data into something usable. These are examples of how commonly encountered problems are resolved, but it is important to be aware that there are several possible solutions.

It is very important to constantly check output of techniques used, you may encounter other issues with the data that were not apparent at first glance, especially when dealing with very large data sets where it is not possible to simply eyeball the data. Using the tools SAS offers, including PROC FREQ, PROC MEANS, PROC UNIVARIATE, and PROC PRINT, among many others, will help ensure the proper variable formats.

CONTACT INFORMATION

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

Elizabeth E Guerrero Angel, M.S.
University of California, Davis
eeguerrero@ucdavis.edu