MWSUG 2019 - Paper RF109

10 Cool Things You Can Do in a DATA STEP

Stephanie R. Thompson, Datamum, Cordova, TN

ABSTRACT

A look at some interesting things you can do in a DATA STEP that would be harder to do in other procedures or that are just plain interesting. Different aspects of utilizing the PDV, joins, editing a file without bringing it in to SAS, automatic variables, and a bit of what to do with DATA _NULL_ are some of the highlights. See the possibilities in the DATA STEP!

INTRODUCTION

The DATA STEP in SAS® does not always get the credit it deserves. Many times, the PROCs, MACROS, and output features seem to get the attention. However, there is a lot of power in the DATA STEP. Whenever I teach someone SAS, more often than not they are amazed at what the DATA STEP can do. They say they had no idea.

This paper will provide a short summary of 10 things that you can do in a DATA STEP that are hard to do with other procedures or do as easily. Only an overview of each will be provided.

1. TAKE ADVANTAGE OF THE PDV

The program data vector (PDV) is utilized each time a SAS dataset is created. It is generated during compilation and used during execution. It contains information about the variables in the dataset being created and acts as a temporary storage location for each observation as it is created. The PDV contains all of the variables from the input dataset as well as the variables created in the DATA STEP. Also, variables to be dropped are flagged in the PDV. As the DATA STEP executes, the variables from the dataset on the set statement are initialized to missing at each iteration. Variables that are created in the DATA STEP are not initialized to missing. Understanding how the PDV functions will help you in your programming. This is especially true if you have run in to a situation where you cannot use a variable that you thought existed.

2. CREATIVE JOINS USING DATA SET OPTIONS

DATA STEP options can be helpful in many ways. The in= DATA STEP option is especially useful. It allows you to specify various ways to combine your data in a DATA STEP MERGE. The in= variable is a temporary value that is a 1 if the dataset contains an observation that meets the merge criteria or is 0 if it does not. What follows “in=” can be a single letter or string of letters.

A simple example is below:

```
data newdata;
merge olddata1(in=inq) olddata2(in=ina);
by idno;
if inq and not ina;
run;
```

The dataset newdata will only contain records from olddata1 where there is no matching idno in olddata2. This is not a very interesting example as it can be done in other ways.

Maybe you want to only include records in newdata where the idno exists in at least 3 of the 5 datasets on the merge statement. This can be done as follows:

```
data newdata;
merge olddata1(in=inq) olddata2(in=ina) olddata3(in=inz) olddata4(in=w)
    olddata5(in=s);
by idno;
```

```
if inq + ina + inz + w + s ge 3;
run;
```

The temporary variables can be summed to determine whether or nor the idno is in at least 3 datasets. This would be harder to do in another way.

3. EDITING A FILE IN PLACE

Have you ever had a CSV file that you wanted to read in with SAS that contained embedded commas in some of the fields? Maybe the file also has embedded carriage returns and line feeds as well. Those can all wreak havoc when trying to read in this type of file. You can find and replace these characters if you are able to open the file in Excel or another editor. What can you do if the file is too big to be read in with another tool? Luckily SAS can edit a file in place using a DATA STEP. It is highly recommended you make a copy of the file before you edit it in this way. It may take several tries before you know all of the characters you need to remove.

The details of editing a file in place takes more than can be covered in a Rapid Fire presentation. Since the goal of this paper is to highlight the possibilities of the DATA STEP the details of this will not be provided here. Contact the author for the detailed SAS code.

4. AUTOMATIC VARIABLES

The PDV also contains 2 variables: _N_ and _ERROR_. These only exist in the PDV. _N_ represents the number of times the DATA STEP has executed. _ERROR_ is zero if there are no errors in the DATA STEP or one if there are errors. Each can be used in the data step as well. They are not, however, written to the dataset.

The code below will write an input record containing an error to the log:

```
if _error_=1 then put _infile_;
```

This code assigns the value of _N_ to a variable:

```
num = _N_;
```

5. DATA _NULL_ TO CREATE MACRO VARIABLES

Did you know you can use a DATA STEP and not create a dataset? That is what the DATA _NULL_ statement does. This allows you to execute other code or even generate macro variables based on the dataset on the SET statement. The sample code below creates a pipe delimited list of file names from a directory, creates a dataset of file names from the dataset dirlist, and then creates multiple macro variables with CALL SYMPUTX that are used later in the program in a macro loop. The first is a simple example creating the MACRO variable colluse based on the value of college in the first observation of colls:

```
data _null_;
set colls;
if _n_ = 1;
call symput('colluse', college);
run;
```

A more complex example is next. The second DATA STEP in the code below uses DATA _NULL_ since an output dataset is not needed:

```
filename DIRLIST pipe 'dir "I:\FILES\for dashboard\*.csv" /b ';

data dirlist ;
   infile dirlist lrecl=200 truncover;
```

```
   input file_name $100.;
run;

data _null_;
   set dirlist end=end;
   count+1;
   call symputx('read'||put(count,4.-l),cats('I:\D2L\syllabus
dashboard\',file_name));
   call symputx('dset'||put(count,4.-l),scan(file_name,1,'.'));
   if end then call symputx('max',count);
run;
```

6. USING THE COLON ON A SET STATEMENT

There may be times you have datasets with similar names that you want to all include on a set statement. If there are a large number of them, typing them all out can be tedious and prone to error. Say you have 100 datasets named month1, month2, month3, … month100. This would be a very long set statement indeed. Using a colon can save you a lot of typing.

```
data allmonths;
set mylib.month:;
run;
```

Using a colon after month tells SAS to read in every dataset that begins with “month” from the libname mylib. You need to make sure that you use the correct number of letters prior to the colon to get what you want. For example, if you shortened “month” down to “mo” you would get all of the month datasets in mylib as well as any of these others if they existed there: money1, money2, mortgage, or mom. While this is helpful, you could get more than you bargained for.

7. CREATING OR CHANGING FORMATS

The DATA STEP can be used to assign or change the format of a variable. It is very simple and an example is below:

```
data docs_new;
set docs;
format docdate date9.;
run;
```

8. USE LOGIC TO GENERATE MULTIPLE DATASETS

The DATA STEP can include more than one dataset on the DATA statement. You can use logic and an explicit output to separate one dataset into many. The dataset sales is split into three new datasets in the code below:

```
data north south west;
set sales;
if region = 'N' then output north;
else if region = 'S' then output south;
else if region = 'W' then output west;
run;
```

In the above example, any value of region not explicitly referenced will not be output to any dataset. You need to know your data to effectively use this approach. This method can also be used to find unexpected values in your data as show in the next example:

```
data north south west junk;
set sales;
```

```
if region = 'N' then output north;
else if region = 'S' then output south;
else if region = 'W' then output west;
else output junk;
run;
```

Here junk would contain any value that you may not have expected to be in the data if your company only has 3 regions.

9. DATA _NULL_ TO WRITE A TEXT FILE

DATA _NULL_ can do more than just create macro variables. It can also be used to generate text files. You can generate other types of data files with SAS and this is just one example.

This sample is from a program that created a listing of reports for individual faculty members:

```
data _null_;
set sirs2;

pageref=compress("&outroot.&term.\20&yr.&sem._"||sect_id||"_"||instruct||".
pdf",'
');

if length(crs_pref) = 4 then do;
    test = substr(pageref, 35, length(instruct));
    if instruct eq test then do;
          file "&outroot.&term.\index.txt";
      put @1 sect_id @13 coll $1. @15 instruct $30. @45 pageref $100. @146
      show @147 " ";
    end;
end;
else if length(crs_pref) = 3 then do;
    test = substr(pageref, 34, length(instruct));
    if instruct eq test then do;
      file "&outroot.&term.\index.txt";
      put @1 sect_id @13 coll $1. @15 instruct $30. @45 pageref $100. @146
      show @147 " ";
    end;
end;
else if length(crs_pref) = 2 then do;
    test = substr(pageref, 33, length(instruct));
    if instruct eq test then do;
      file "&outroot.&term.\index.txt";
      put @1 sect_id @13 coll $1. @15 instruct $30. @45 pageref $100. @146
      show @147 " ";
    end;
end;
run;
```

For reference, there were three different course prefix lengths and the conditional logic made sure that the correct instructor was associated with their reports only.

10. WRITE A REPORT EXPLICITLY

DATA _NULL_ can be used to write a report but explicitly by specifying the layout using variables in a dataset. With other output methods, this may not be in use as much. However, if you need something very particular, it can be very useful. This example is from a project that generates a report for a college.

Some of the code has been removed to highlight the code used to write the report. The output is wrapped with ODS statements creating a PDF. In the past, the report went directly to a printer from a VAX.

One way to generate an explicitly defined report layout is shown below:

```
    DATA  _NULL_ ;
    SET PROJPRT ;
    BY DESCENDING GROUP COLL UNIT LEVEL TYPE ;
    FILE PRINT  HEADER=HEADER  N=PS ;

    IF (FIRST.UNIT)  THEN  PUT  _PAGE_ ;
    IF FIRST.LEVEL  THEN  DO ;
    PUT  // LEVEL  $LEVEL.;
    IF (LEVEL EQ "1"  AND  ADMDATA and group ne 9)  THEN
        PUT / @ 1 "ADMISSION"  /
              @ 1 "FALL ENTRY" /
              @ 3 "FRESHMEN"
              @15 AFRESH&YYP2   COMMA8.0
              @23 AFRESH&YYP1   COMMA8.0
              @31 AFRESH&YYC0   COMMA8.0
              @41 AFRESX&YYZ0   COMMA8.0
              @49 AFRESH&YYF1   COMMA8.0
                                 /
              @ 3 "TRANSFER"
              @15 ATRANS&YYP2   COMMA8.0
              @23 ATRANS&YYP1   COMMA8.0
              @31 ATRANS&YYC0   COMMA8.0
              @41 ATRANX&YYZ0   COMMA8.0
              @49 ATRANS&YYF1   COMMA8.0
                                         /
              @18 "-----"
              @26 "-----"
              @34 "-----"
              @44 "-----"
              @52 "-----"
                           /
              @ 3 "TOTAL"
              @15 ATOTAL&YYP2    COMMA8.0
              @23 ATOTAL&YYP1    COMMA8.0
              @31 ATOTAL&YYC0    COMMA8.0
              @41 ATOTAX&YYZ0    COMMA8.0
              @49 ATOTAL&YYF1    COMMA8.0

     [additional code deleted]

    RETURN ;
    HEADER:
    PAGE + 1 ;
    PUT  # 1  @ 1 "*** CONFIDENTIAL ***"
                 @74 "&SYSDATE."
         # 2  @1 "Department"
                @74 "PAGE " PAGE

         # 3  @24 "INSTITUTION"

  [additional code deleted]
```

```
            @36 "&YYC0.1"
            @46 "&YYC0.1"
            @54 "&YYF1.1"
[additional code deleted]

       #13  @20 "---"
            @28 "---"
            @36 "---"
            @46 "---"
            @54 "---"
                     ;
 RETURN ;
RUN ;
```

CONCLUSION

Hopefully this brief summary of some different things you can do with a DATA STEP has been helpful. The DATA STEP is not just for manipulating and generating datasets. It can be used in a variety of ways.

CONTACT INFORMATION

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

Stephanie R. Thompson
Datamum
stephanie@datamum.com
http://www.datamum.com