SESUG Paper 253-2019

DO loops :INTO efficient programming: DATA STEP do loop + PROC SQL in boilerplate programming

Christopher Hargett, U.S. Department of Commerce

ABSTRACT

Boilerplate SAS programs are real time savers, but frequently input datasets have different variables and values not accounted for in the code. Going through a whole program to manually update RENAME or SUM statements is laborious. This paper will introduce a technique to dynamically modify boilerplate programs based on input dataset. Combining a DO loop in a DATA STEP and the :INTO keyword in PROC SQL results in a flexible method of creating macro variable values that can be used in a wide array of applications. Using the COUNT function with a DO loop in the DATA STEP to create a SAS data set consisting of programming language, and the INTO clause in PROC SQL allows the programming language to be dynamically generated and applied efficiently in multiple scenarios in a longer program. This paper builds on concepts covered by Kelly Schlessman, "PROC SQL – GET "INTO:" IT!," SESGUG Paper 267-2018.

INTRODUCTION

A fundamental concept of macro variables is that they work based on text substitution. Creating a SAS data set with the variable names and then using PROC SQL to create macro variables results in a remarkably flexible and intuitive way to streamline programming.
The general structure of automating the creation of macro variables is:

1. Create a list of variables to be manipulated
2. Use the list to create a SAS dataset with the desired code
3. Transfer the code into a macro variable using PROC SQL
4. Apply the macro variables in boilerplate programming language.

This paper uses the following sets of syntax:

1. Creating a macro variable using the LET statement:

```
%LET macro-variable-name = text-or-text-value;
```

2. Using the COUNTW function, SCAN function, CATS function, and DO I = 1 TO N statement in the DATA STEP:

```
DATA new-data-set-name;
        DO I = 1 TO COUNTW(<string><, chars><,modifiers>);
                VARIABLES = SCAN(string, count<,charlist><,modifiers>);
                RENAME_VARS  = CATS(item-1 <, ..., item-n>);
                OUTPUT;
        END;
RUN;
```

3. Using PROC SQL to create macro variables:

```
PROC SQL;
        SELECT macro-specification
        INTO :macro-variable-name SEPARATED BY "character"
        FROM source-data
QUIT;
```

4. Applying the macro variables in standard programming language:

DATA *data-set-name* (RENAME = (&*macro-variable-name1*));
SET *data-set-name*;
&macro-variable-name2;
&macro-variable-name3;
RUN;

CREATING THE VARIABLE LIST

In our dataset, we have a list of variables that we need to compare to each other, based on some (hypothetical) criteria. Those variables need to be renamed, summed, and a ratio of the variables compared to the first variable listed needs to be calculated.

We can create a list the variables we want to manipulate in a macro variable, and have SAS create a data set with the variables in it:

```
%let listofvar = cow fat water bact heat past;
```

CREATING THE SAS DATA SET

Using the **COUNTW** and **SCAN** functions in a data step, we can easily create a data set with the list of variables in it:

```
data varlist;
      do I = 1 to countw("&listofvars");
      variables = (scan("&listofvars", I));
      output;
      end;
run;
```

The code produces the following table, *varlist*:

Obs	I	variables
1	1	cow
2	2	fat
3	3	water
4	4	bact
5	5	heat
6	6	past

Table 1 - Result of Data Step with Do Loop

MODIFYING THE DATA STEP

By adding the **CATS** function, we can easily modify the above data step to create new words or equations. The three actions we want to take are:

1. Append "*_new*" on the end of each of the variables in our list.
2. Create the equation *var1 = var1_new, var2 = var2_new*, ect, to use in a rename statement.
3. Create the equation *ratio1 = (var1 – var1)/var1, ratio2 = (var2 – var1)/var1*, etc, to use in a data step.

The resulting code looks like this:

```
data varlist;
      do I = 1 to countw("&listofvars");
      variables = (scan("&listofvars", I));
      new_vars = cats(scan("&listofvars", I),"_new");
      rename_vars = (cats(scan("&listofvars", I),"=",new_vars));
      ratio = cats(cats(scan("&listofvars", I),"_ratio")," = (",
            cats(scan("&listofvars", I),"_new")," -",
            cats(scan("&listofvars", 1),"_new"), ") /",
            cats(scan("&listofvars", 1),"_new")) ;
      output;
      end;
run;
```

The code now produces the following table, *varlist*:

Obs	I	variables	new_vars	rename_vars	ratio
1	1	cow	cow_new	cow=cow_new	cow_ratio= (cow_new-cow_new) /cow_new
2	2	fat	fat_new	fat=fat_new	fat_ratio= (fat_new-cow_new) /cow_new
3	3	water	water_new	water=water_new	water_ratio= (water_new-cow_new) /cow_new
4	4	bact	bact_new	bact=bact_new	bact_ratio= (bact_new-cow_new) /cow_new
5	5	heat	heat_new	heat=heat_new	heat_ratio= (heat_new-cow_new) /cow_new
6	6	past	past_new	past=past_new	past_ratio= (past_new-cow_new) /cow_new

Table 2 - Result of Complex Data Step with Do Loop

CREATING MACRO VARIABLES IN PROC SQL

Once the desired code is created in the DATA step, PROC SQL easily transfers the data into macro variables. In each case, we used a different “separated by” character to facilitate the use of the macro variables later in the program; the SUMVARS macro variable will be used in a SUM(OF: *var1, var2, var-n)*, which requires a comma (,) separator between the variables, the RENAME_VARS macro will be used in a rename option, which needs no characters between the sets of variables, and the CALCULATION macro will have a semi-colon (;) between the calculations, to have each calculation executed independently.

It is important to remember that the macro variables are assigned in the order of the columns in the SELECT statement. In this case, the values from the column “variables” will go into the macro variable “sumvars,” “rename_vars” will go into macro variable “rename_vars,” and “cmplex3” will go into macro variable “calculation.”

```
proc sql;
      select variables, rename_vars, cmplex3
      into :sumvars SEPARATED BY ",",
            :rename_vars SEPARATED BY " ",
            :calculation SEPARATED BY ";   "
      from varlist;
quit;
```

The macro variables thus have the following values:

SUMVARS cow,fat,water,bact,heat,past

RENAME_VARS cow=cow_new fat=fat_new water=water_new bact=bact_new heat=heat_new past=past_new

CALCULATION cow_diff= (cow_num-cow_num) /cow_num; fat_diff= (fat_num-cow_num) /cow_num; water_diff= (water_num-cow_num) /cow_num; bact_diff= (bact_num-cow_num)

/cow_num; heat_diff= (heat_num-cow_num) /cow_num; past_diff= (past_num-cow_num) /cow_num

APPLYING THE MACRO VARIABLES IN BOILERPLATE LANGUAGE

The boilerplate SAS code is simple:

```
data test (rename=(&rename_vars));
      set test.cost2;
      sum_test = sum(&sumvars);
      &calculation;
run;
```

When the macro variables are resolved, the result is:

```
data test (rename=(cow=cow_new fat=fat_new water=water_new
           bact=bact_new heat=heat_new past=past_new));
      set test.cost2;
      sum_test = sum(of cow,fat,water,bact,heat,past);
      cow_diff= (cow_num-cow_num) /cow_num;  fat_diff= (fat_num-
      cow_num)/cow_num;  water_diff= (water_num-cow_num)
      /cow_num;  bact_diff= (bact_num-cow_num) /cow_num;
      heat_diff= (heat_num-cow_num) /cow_num;  past_diff=
      (past_num-cow_num) /cow_num;
run;
```

DRAWBACKS OF THIS TECHNIQUE

1. If you are creating macro variables in macro language, you must use a **GLOBAL** statement to allow the macro variable to be accessed.
2. Macro variables have a maximum length of 65,534 characters. The value in the macro variable will be truncated if any longer, which will cause incorrect calculations.
3. If you use a %PUT &*MACRO_VARIABLE* statement in your code, you may get an error when you use the semi-colon separator because SAS will interpret the semi colon as the end of the %PUT statement.

CONCLUSION

By combining a SAS data step with the INTO keyword in PROC SQL, we can create macro variables with both simple and complex values variables and easily apply them in boilerplate programming.

CONTACT INFORMATION

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

Christopher Hargett
United States Department of Commerce
(202) 482-4161
Christopher.Hargett@trade.gov