SESUG Paper 177-2019

The Power of PROC SQL's SELECT DISTINCT INTO

HoaiNam Tran, National Agricultural Statistics Service;

Mai Anh Ngo, Catalyst Clinical Research, LLC.

ABSTRACT

Many SAS® programmers are familiar with PROC SQL procedures that join database tables or SAS datasets. However, they may not be familiar with using PROC SQL combined with SAS data steps and procedures to improve programming efficiency. Specifically, using the DISTINCT and SEPARATED BY arguments and SELECT and INTO clauses in PROC SQL as well as the macro SCAN function, DO loop, and array in SAS data steps and PROC REPORT procedure can provide programming flexibility, simplify SAS code, and minimize typographical errors. In this paper, we provide examples that illustrate how to use these specific PROC SQL arguments and clauses and SAS procedures to create automated data-driven programs that can be readily tailored to different applications.

INTRODUCTION

Data-driven program is a program that receives instructions from the incoming data rather than the code itself. Code in this kind of program is efficient, compact, easy to maintain and tailor, and, most importantly, the code is reusable and is robust in accommodating changes to input.

The DISTINCT and SEPARATED BY arguments and SELECT and INTO clauses in PROC SQL are great tools in writing compact and efficient code in data-driven programming. Used together, this procedure creates a macro variable that contains a list of distinct values in alphabetical order that are separated by a delimiter from the column that get selected as well as the count of number of distinct rows SQLOBS - an automatic macro variable. This list of variables can then be used as array-elements in an array definition and the count obtained above can be used as the upper limit in a DO loop to manipulate data automatically.

In this paper, we will give a quick overview of PROC SQL syntax and go over 4 examples of using PROC SQL's SELECT DISTINCT, INTO: and SEPARATED BY to accomplish data-driven programming and increase programming efficiency. The first example illustrates how to dynamically initialize missing values to zero as well as order the variables in a desired fashion; the second example illustrates how to dynamically flag records; the third example illustrates how to dynamically rename variables; and the fourth example illustrates how to dynamically create a report.

QUICK OVERVIEW OF PROC SQL

Here are the basic Proc SQL options, clauses, and arguments that use throughout the paper:

```
   PROC SQL NOPRINT;
      SELECT DISTINCT _Name_ INTO: varlist SEPARATED BY " "
      FROM DataIn;
   QUIT;

%LET N_vars = &SQLOBS;
```

SELECT DISTINCT INTO IN PROC SQL

Let us go over the basics of each of the PROC SQL options, clauses, arguments, and SAS functions that are used throughout the paper.

SELECT DISTINCT

SELECT DISTINCT procedure eliminates duplicate rows from the column _NAME_ and selects only unique values. Distinct option also put the values in the alphabetical order.

INTO: AND SEPARATED BY

The INTO: clause and SEPARATED BY argument from SELECT DISTINCT in the PROC SQL procedure creates a macro variable which is a string of characters that contains all distinct values of the rows from the column that gets selected, in the alphabetical order that separated by a specified delimiter. The name of the macro variable is the name specified right after the colon in 'INTO:'. In our examples, a space is specified as delimiter as this macro variable would later be used to specify the list of variable names in an array since variables in an array must be separated by a space to follow SAS array syntax.

SQLOBS

SQLOBS is an automatic macro variable. It has the value of the number of rows that were processed by an SQL procedure. In our example, SQLOBS tells us the distinct number of rows were processed since SELECT DISTINCT option was specified.

ARRAY

An array is a list of variables. An array can include either all character or all numeric variables and can potentially include hundreds of variables. SAS arrays are often used together with a simple do loop in a data step to simplify and automate program code.

%SCAN

The %SCAN function can be used to extract words or a string of characters from the value of a macro variable. The general syntax of the %SCAN function is as follow: %SCAN(argument, n, <delimiters>). The argument can consist of constant text, macro variable references, macro functions, or macro calls; and n is an integer or a text expression that yields an integer, which specifies the position of the word to return. Delimiters option specifies an optional list of one or more characters that separate "words" or text expressions that yield one or more characters. In our examples, the specified delimiter is space (' ').

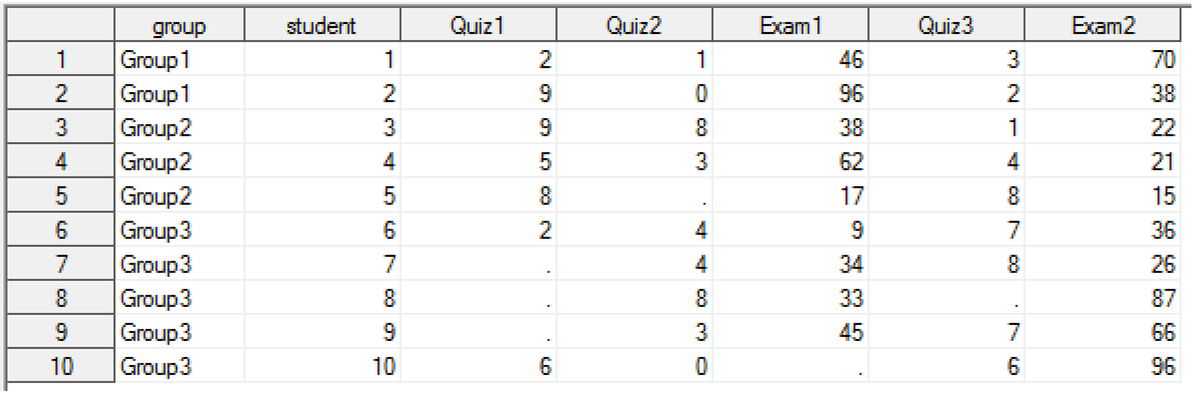
%EVAL

The %EVAL function can be used to evaluate the numeric value stored in a macro variable. As macro variable by design is always a character variable even if the information stored in it is numeric value, the %EVAL becomes a handy tool to be used later in the code any time after the macro variable is created to extract the numeric value from the macro variable to be used in calculations, conditional statements or do loops.

Examples of USING PROC SQL's SELECT DISTINTC INTO

Table 1 presents the example data that will be used throughout this paper. This is a dataset of individual student's test scores with missing data. Column group identifies which group the student is in and column student is the student ID number.

Table 1. Dataset of Test Scores with Missing Data



	group	student	Quiz1	Quiz2	Exam1	Quiz3	Exam2
1	Group1	1	2	1	46	3	70
2	Group1	2	9	0	96	2	38
3	Group2	3	9	8	38	1	22
4	Group2	4	5	3	62	4	21
5	Group2	5	8	.	17	8	15
6	Group3	6	2	4	9	7	36
7	Group3	7	.	4	34	8	26
8	Group3	8	.	8	33	.	87
9	Group3	9	.	3	45	7	66
10	Group3	10	6	0	.	6	96

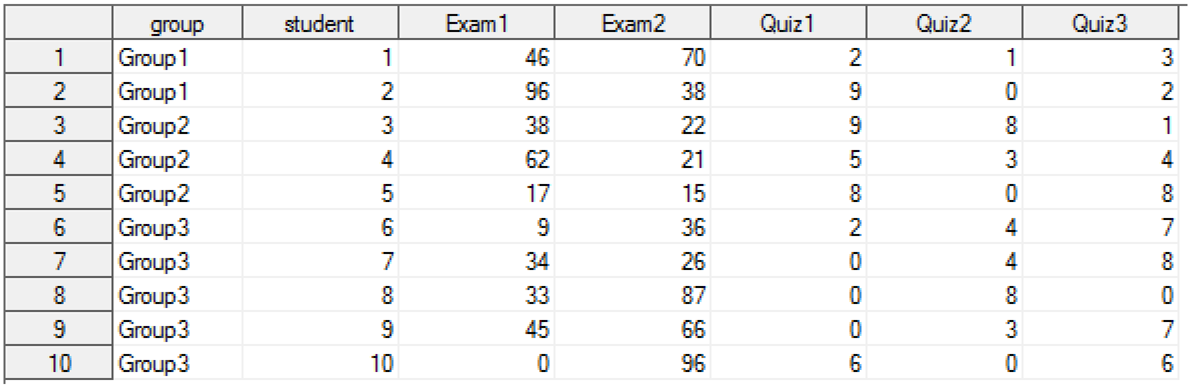
EXAMPLE 1 – USING SELECT DISTINCT INTO: AND SAS ARRAY TO INITIALIZE/RESET VARIABLE'S VALUES

Often in data analysis, an analyst would like to initialize missing values to zero (or a desired value) as well as order the key variables in the alphabetical order. In our example, we would like to keep group and student variables in the first two columns and put the rest of the variables in the alphabetical order. It's possible to complete the task by using the retain statement and manually write a series of 'If … then …' statements as shown below:

```
data Scores_no_missing;
retain group student Exam1 Exam2 Quiz1 Quiz2 Quiz3;
  set Scores;
  if Exam1 = .  then Exam1 = 0;
  if Exam2 = .  then Exam2 = 0;
  if Quiz1 = .  then Quiz1 = 0;
  if Quiz2 = .  then Quiz2 = 0;
  if Quiz3 = .  then Quiz3 = 0;
run;
```

This will create the new dataset from the example dataset where all missing scores are replaced with zero and variables are listed in the desired fashion as shown in the table 2 below.

Table 2. Scores Dataset Without Missing Data



	group	student	Exam1	Exam2	Quiz1	Quiz2	Quiz3
1	Group1	1	46	70	2	1	3
2	Group1	2	96	38	9	0	2
3	Group2	3	38	22	9	8	1
4	Group2	4	62	21	5	3	4
5	Group2	5	17	15	8	0	8
6	Group3	6	9	36	2	4	7
7	Group3	7	34	26	0	4	8
8	Group3	8	33	87	0	8	0
9	Group3	9	45	66	0	3	7
10	Group3	10	0	96	6	0	6

However, manual coding is tedious, time-consuming, therefore, it potentially introduces typographical errors during the process. Quite often, variable names from input datasets are not the same. Specifically, the numbers of tests, quizzes, homework, exams, etc., and their conventional names are varied, depending on the courses and/or instructors. This “manual” program with the “retain” and its series of “If …. then…” statements cannot be easily tailored and maintained in this setting. This creates a demand for data-driven and dynamic programming in accommodating changes to input.

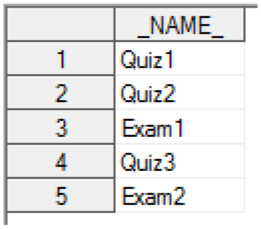
PROC SQL’s SELECT DISTINCT, INTO:, and SEPARATED BY can be used to store the list of variables, in the alphabetical order, in a specified macro variable. As a byproduct of this step, SAS’s automatic macro variable SQLOBS gives the number of distinct values (rows) selected, which can be used as the upper limit in the do loop. Therefore, it naturally makes sense to use the combination of arrays and do loops in SAS data step to iterate through the list of variables to search for missing values and replace them with zero. The programming steps are broken down as follows:

First, PROC TRANSPOSE is used to obtain a dataset that contains all the variable names from the Scores dataset:

```
PROC TRANSPOSE DATA=SCORES (OBS=0 DROP=student group)
   OUT=VARNAMES(KEEP = _NAME_ );
RUN;
```

Below is the output of Variable Names dataset.

Table 3. Variable Names



	NAME
1	Quiz1
2	Quiz2
3	Exam1
4	Quiz3
5	Exam2

Then PROC SQL is used to create a macro variable VARLIST which contains the list of all these variable names from the VARNAMES dataset:

```
PROC SQL NOPRINT;
    SELECT DISTINCT _NAME_ INTO: VARLIST SEPARATED BY " "
    FROM VARNAMES;
    %LET N_vars = &SQLOBS;
    %PUT N_vars = &N_VARS ;
    %PUT VARLIST = &VARLIST;
QUIT;
```

The value of the macro variable N_vars is 5 and of the macro variable VARLIST is “Exam1 Exam2 Quiz1 Quiz2 Quiz3”. Note that the elements in VARLIST by PROC SQL’s default are in alphabetical order.

We can then pass this information from these two macro variables into the ARRAY and DO loop statements in a SAS Data step to automate the task of initializing missing score to zero:

```
Data Scores_No_Missing;
RETAIN group student &Varlist ;
ARRAY VARS_ARRAY{&N_vars} &Varlist; /*ARRAY VARS_ARRAY {*} &Varlist works*/
SET Scores;

DO Ith_var = 1 TO &N_vars;  /*DO Ith_var =1 TO DIM(VARS_ARRAY) WORKS TOO*/
    IF VARS_ARRAY{ith_var}=. THEN VARS_ARRAY{ith_var}=0;
END;
DROP ith_var;
RUN;
```

As more test scores come in, the dataset will have more columns, e.g. Exam 3, Quiz 4, Quiz 5, etc. The manual coding process requires programmers to have to add more ‘If ... then …’ statement for these new columns. On the other hand, the PROC SQL approach that we just presented requires no further update to the code.

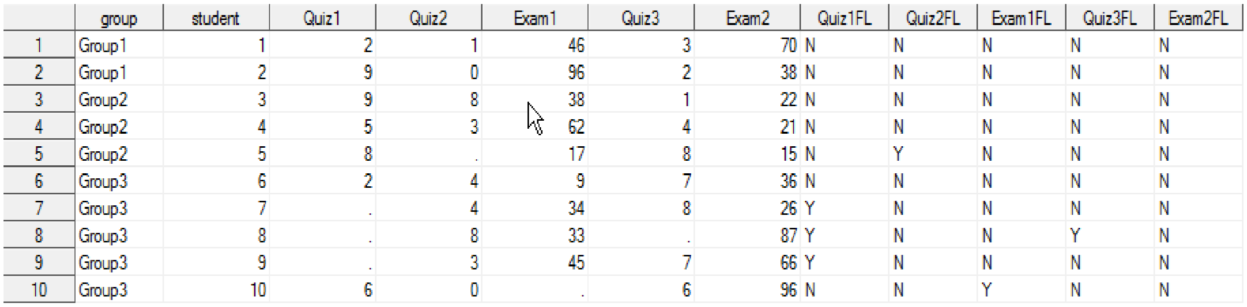
EXAMPLE 2 – USING SELECT DISTINCT INTO TO CREATE FLAGS

Suppose for each test, we want to create a corresponding flag of missing score. For example, for variable Exam1, the corresponding flag named Exam1FL will take a value of ‘Y’ if a student has missing score for Exam1, and ‘N’ if a student has score for Exam1. The example code below achieves this goal without having to manually code each flag variable. Note that in this example, implicit array statement was used instead of explicit array to illustrate another way of doing the task and the array flag statement in the example creates the flag variables with the test names concatenated with the suffix ‘FL’.

```
PROC SQL NOPRINT;
SELECT  _NAME_ , strip(_NAME_)||'FL' INTO: VARNAMES SEPARATED BY " ",
:FLAGNAMES SEPARATED BY " "
    FROM VARNAMES;
    %PUT varnames=&VARNAMES;
    %PUT flagnames=&FLAGNAMES;
QUIT;
Data ALL_MISSING_FLAG;
RETAIN group student &VARNAMES;
ARRAY VARS  &VARNAMES;
ARRAY FLAG $ &FLAGNAMES;
SET SCORES;
DO OVER VARS;
    IF VARS=. THEN FLAG='Y';
    ELSE FLAG='N';
END;
RUN;
```

Table 4 presents the outcome dataset:

Table 4: Creating Flags



	group	student	Quiz1	Quiz2	Exam1	Quiz3	Exam2	Quiz1FL	Quiz2FL	Exam1FL	Quiz3FL	Exam2FL
1	Group1	1	2	1	46	3	70	N	N	N	N	N
2	Group1	2	9	0	96	2	38	N	N	N	N	N
3	Group2	3	9	8	38	1	22	N	N	N	N	N
4	Group2	4	5	3	62	4	21	N	N	N	N	N
5	Group2	5	8	.	17	8	15	N	Y	N	N	N
6	Group3	6	2	4	9	7	36	N	N	N	N	N
7	Group3	7	.	4	34	8	26	Y	N	N	N	N
8	Group3	8	.	8	33	.	87	Y	N	N	Y	N
9	Group3	9	.	3	45	7	66	Y	N	N	N	N
10	Group3	10	6	0	.	6	96	N	N	Y	N	N

The same approach can be applied in a wide variety of settings. For example, in clinical studies, one may need to create flag variables for each scheduled visit to indicate whether a patient has data for that visit. As the study is ongoing, the number of scheduled visits increases over time. Using the same approach above, one can automate the code to create these flags for the number of visits present in the data at each data cut instead of having to manually check and add new visit flags as new data come.

EXAMPLE 3 – USING SELECT DISTINCT INTO: AND %SCAN TO RENAME VARIABLES

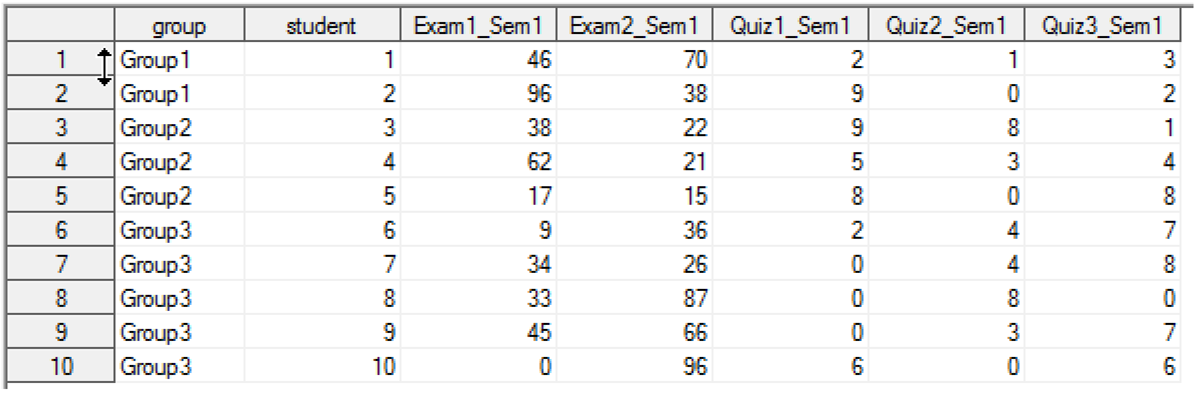
Sometimes, we need to rename a long list of variables by adding a suffix to the end of the variable name. Typing a series of rename statements for all the variables is very tedious and risk making typos. In the example below, we use %SCAN to add the suffix '_Sem1' to the list of variable names stored in &VARNAMES which was obtained using SELECT DISTINCT INTO in PROC SQL:

```
 %MACRO RENAME1(oldvarlist, suffix);
   %LET k=1;
   %LET old = %SCAN(&oldvarlist, &k);
   %DO %WHILE("&old" NE "");
      RENAME &old = &old.&suffix;
       %LET k = %EVAL(&k + 1);
      %LET old = %SCAN(&oldvarlist, &k);
   %END;
%MEND;

PROC DATASETS;
MODIFY Scores_No_Missing;
    %RENAME1(&VARNAMES, _Sem1);
RUN;
```

The output dataset is shown in Table 5 below.

Table 5. Adding Suffix Exam and Quiz Variables



	group	student	Exam1_Sem1	Exam2_Sem1	Quiz1_Sem1	Quiz2_Sem1	Quiz3_Sem1
1	Group1	1	46	70	2	1	3
2	Group1	2	96	38	9	0	2
3	Group2	3	38	22	9	8	1
4	Group2	4	62	21	5	3	4
5	Group2	5	17	15	8	0	8
6	Group3	6	9	36	2	4	7
7	Group3	7	34	26	0	4	8
8	Group3	8	33	87	0	8	0
9	Group3	9	45	66	0	3	7
10	Group3	10	0	96	6	0	6

EXAMPLE 4 – USING SELECT DISTINCT INTO: FOR COUNTING AND CREATING A REPORT

Many times, we need to produce summary tables that display the count of certain groups in the column header. Specifically, we want to create a summary table of the number of students with average quiz score greater than or equal to 5, and of the number of students with average quiz score less than 5 in each group. We also want to display the number of students in each group in the column headers of the table. Below are the steps to achieve this goal.

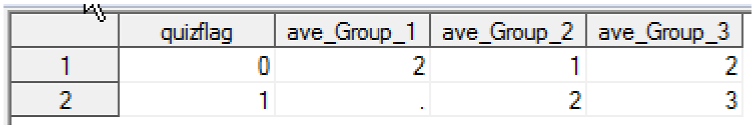
Step 1 - Get the frequency

First, we create a flag variable so that for each student observation, the flag takes value of 1 if the average quiz score is greater than or equal to 5 and zero otherwise. Then we tabulate the frequencies of the created flag variable:

```
DATA all1;
   SET SCORES;
   IF MEAN(quiz1,quiz2,quiz3) >=5 THEN quizflag=1;
   ELSE quizflag=0;
RUN;
PROC FREQ DATA=all1 NOPRINT;
   TABLE quizflag*group/OUT=scoreave;
RUN;
PROC TRANSPOSE DATA=scoreave OUT=scoreave1 PREFIX=ave_ ;
   BY quizflag;
   ID group;
   VAR count;
RUN;
```

Here is the output dataset at the end of Step 1:

Table 6: Frequencies of Students with Flag 1 and 0 By Group



	quizflag	ave_Group_1	ave_Group_2	ave_Group_3
1	0	2	1	2
2	1	.	2	3

Step 2 - Generate macro variables to enable dynamic coding for creating the report

First, we count the number of unique students in each group and store these into macro variables BIGN1, BIGN2, …, BIGNx. By using the hyphen after 'BIGN1', we do not have to specify how many macro variables BIGNx to use but let this step be data-driven:

```
PROC SQL NOPRINT;
   SELECT count(DISTINCT STUDENT)INTO :BIGN1-
   FROM all1
   GROUP BY group
   ORDER BY group;
QUIT;
%let N_groups = &sqlobs;
```

Third, we store all the variables needed for PROC REPORT's column statement into a macro variable called DEFCOL:

```
PROC SQL NOPRINT;
    SELECT name  INTO: DEFCOL SEPARATED BY " "
   FROM SASHELP.VCOLUMN where libname="WORK" and memname="SCOREAVE1" ;
   %PUT &DEFCOL ;
```

```
QUIT;
```

Step 3 – Generating the Report

Finally, we assemble all 3 information pieces generated above in a reporting table with PROC REPORT. The macro variable N_groups is used as the upper limit of the %DO loop statement and for the dynamic footnote that gives the count of the number of student groups in the data. The macro variables BIGN1, BIGN2, and BIGN3, etc. are used to display the count of students in each group in the corresponding column headers. Note that the macro REPORT below is flexible to allow for the generation of reports as there are more groups added to the data:

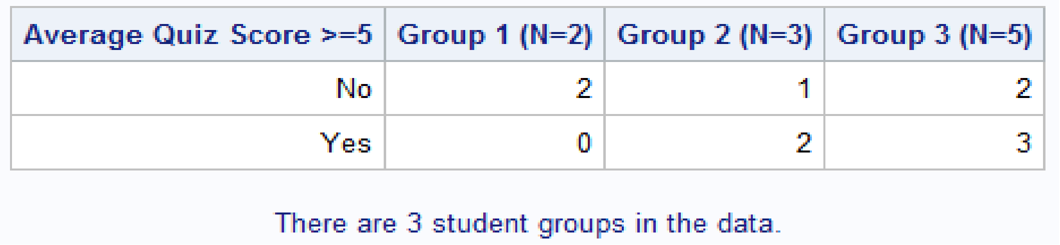
```
%MACRO REPORT;
   PROC FORMAT;
      VALUE yn
       1='Yes'
       0='No'
    ;
    VALUE xmiss
       .='0'
    ;
   RUN;
   ODS LISTING CLOSE;
   ODS RTF FILE="Score Report.rtf";
   PROC REPORT DATA=scoreave1;
   COLUMN &defcol. ;

   DEFINE quizflag/DISPLAY 'Average Quiz Score >=5' FORMAT=yn.;
   %DO i = 1 %TO &N_groups;
   DEFINE ave_group_&i / DISPLAY  "Group &i (N=&&bign&i.)" FORMAT=xmiss.;
   %END;

   FOOTNOTE "There are &N_groups student groups in the data.";
   RUN;
   ODS LISTING;
%MEND;
```

Output of the steps above is provided in Table 7.

Table 7. Test Score Summary Report



Average Quiz Score >=5	Group 1 (N=2)	Group 2 (N=3)	Group 3 (N=5)
No	2	1	2
Yes	0	2	3

There are 3 student groups in the data.

CONCLUSION

In this paper, we use four examples to illustrate how to use SELECT DISTINCT, INTO:, and SEPARATED BY in PROC SQL to meet the need for controlling the program flow in the world of big and frequently changing data. For demonstration purposes, the dataset used in this paper is relatively simple. However, the code can be easily modified to serve the same purpose of data-driven programming on large datasets with many variables in different industries. We hope that this paper inspires you to utilize PROC SQL and other SAS functions and arrays to develop better data-driven programs that are concise and free of typographical errors.

REFERENCES

Sastry Shri. SAS Programming Tips and Tricks. 2012. http://sasbitips.blogspot.com/2012/02/using-proc-sql-automatic-macro.html

Lafler, Kirk. 2018. “Introduction to Data-driven Programming Using SAS®. 2018.” *Proceedings of the Southeast SAS Users Group 2018 Conference*, Available at https://www.lexjansen.com/sesug/2018/SESUG2018_Paper-110_Final_PDF.pdf

SAS® Viya™ SQL Procedure User’s Guide. Available at http://documentation.sas.com/?docsetId=sqlproc&docsetTarget=p0hwg3z33gllron184mzdoqwpe3j.htm&docsetVersion=1.0&locale=en

ACKNOWLEDGMENTS

The authors thank Charles Cook, Peter Quan, and Jeff Beranek for valuable comments

RECOMMENDED READING

- *SAS® Certification Prep Guide Advanced Programming for SAS® 9 Fourth Edition*

CONTACT INFORMATION

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

HoaiNam Tran
National Agricultural Statistics Service
1400 Independence Ave SW
Washington, DC 20250
Work Phone: (202) 720-5556
Email: HoaiNam.Tran@nass.usda.gov

Mai Anh Ngo
Catalyst Clinical Research, LLC
7780 Brier Creek Pkwy STE 310
Raleigh, NC 27617
Email: MaiNgo@catalystcr.com
https://www.linkedin.com/in/maiango/