Paper 057-30

Techniques for Effectively Selecting Groups of Variables

Stuart Pollack, TNS Healthcare, Blue Bell, PA

ABSTRACT

Survey data often contains a large number of variables as each question's response is coded as a separate variable. It is often necessary to analyze groups of variables differently. This entails selecting different groups of variables from a data set containing many variables. When selecting a large number of variables it becomes impractical to write out hundreds of variable names. Different methods for efficiently selecting groups of variables are the topic of this paper. These techniques can be used in an array declaration, with a KEEP or DROP statement or in the VAR statement of any SAS procedure. The examples in this paper were generated using PC SAS version 8.2 on a Windows 2000 platform.

INTRODUCTION

It is often necessary to reference a large group of variables, for example, in a VAR, KEEP, or DROP statement. If the variable names have the same prefix and a sequential numeric suffix the task is easy. However, when there is no discernable pattern in the variable names selecting the group can become more challenging.

Five methods for selecting a group of variables are presented.

1. Selecting a group of variables with the same prefix and a numerically sequential suffix.
2. Selecting a contiguous group of variables.
3. Selecting a group of variables, which have the same prefix.
4 Selecting a group of variables using the CONTENTS procedure OUTPUT statement.
5. Selecting a group of variables using dictionary tables.

To illustrate these techniques a data set is created where the 10 variables called A1 through A10 are assigned a number from 1 through 10, respectively. Then in a second data set a selected group of these variables is read into an array using the indicated variable selection technique to assign variable names to array elements. Finally the array contents are printed to the output window.

EXAMPLE 1: NUMERIC SEQUENCE

Selecting a group of variables that have the same prefix and whose suffixes form a numbered sequence is easy to accomplish. For instance, if the variables are A1, A2, A3,…, AN then they can be referenced as A1 – AN.

The following data step creates a data set, which has one observation and ten variables named A1 through A10, which contain the values 1 to 10, respectively.

```
DATA INPUT1;
INPUT A1 A2 A3 A4 A5 A6 A7 A8 A9 A10;
DATALINES;
1 2 3 4 5 6 7 8 9 10;
RUN;
```

The following data step reads the input1 dataset, assigns the 10 variables A1 through A10 to the ten elements of the one dimensional array X and prints the values of the selected variables, 1 2 3 4 5 6 7 8 9 and 10 to the output window.

```
DATA EXAMPLE1;
SET INPUT1;
FILE PRINT;
ARRAY X[*] A1-A10 ;
DO I = 1 TO 10;
PUT X[I] @;
```

```
END;
RUN;
```

EXAMPLE 2: CONTIGUOUS VARIABLES

If a group of variables is not a numbered list but the variables are contiguous in the data set they can be referred to by the first variable name and the last variable name of the group separated by a double dash. So the variable list A11 Boo6 C14 D2 Z90 U19 R16 M8 A3 E77 can be selected as A11 -- E77.

In the next data step there are 10 variables with values from 1 to 10 assigned to the variables A11 Boo6 C14 D2 Z90 U19 R16 M8 A3 E77, respectively. These variable names have no discernable pattern but they are contiguous.

```
DATA INPUT2;
INPUT A11 BOO6 C14 D2 Z90 U19 R16 M8 A3 E77;
DATALINES;
1 2 3 4 5 6 7 8 9 10;
RUN;
```

In the following data step all the variables can be referenced in the array as A11-- E77 and the output from the data step is again the values 1 through 10.

```
DATA EXAMPLE2;
SET INPUT2;
FILE PRINT;
ARRAY X[*] A11 -- E77 ;
DO I = 1 TO 10;
PUT X[I] @;
END;
RUN;
```

If you are not sure if the variables in your data set are contiguous, you can use PROC CONTENTS with the VARNUM option to print a list of the variable names in the same order as they physically exist in the data file.

EXAMPLE 3: COLON WILDCARD.

Again using the INPUT2 data set, say we want to select only the variables that start with the letter A. One way to select these variables is to use the colon wildcard [1]. A colon following a variable name prefix selects any variable name that starts with that prefix regardless of what follows the prefix. So in the following data step only the variables A11 and A3 are selected and the output from the data step is the two values 1 and 9.

```
DATA EXAMPLE3;
SET INPUT2;
FILE PRINT;
ARRAY X[*] A: ;
DO I = 1 TO 2;
PUT X[I] @;
END;
RUN;
```

EXAMPLE 4: PROC CONTENTS OUTPUT STATEMENT

In this example, we will get the column names using PROC CONTENTS, select the variable names we want and then store them in a macro variable. Specifically, we are going to select variables from the INPUT2 dataset that begin with the letter U, B, or E. First, we use the PROC CONTENTS OUTPUT statement to capture all the variable names from the INPUT2 data set.

```
PROC CONTENTS DATA = INPUT2
OUT=VAR_NAMES(KEEP = NAME) NOPRINT;
RUN;
```

Next, we can read in the file of variable names in the parse data step and use the substring function to test if the first letter of the variable's name starts with a U, B, or E.

```
DATA PARSE;
SET VAR_NAMES;
WHERE (SUBSTR(NAME, 1,1) IN ('U' 'B' 'E'));
RUN;
```

The following code constructs a %LET statement using PUT statements that write to a file. For observation one '%LET VAR_LIST =' is written to the file 'BUILD' and on each subsequent observation a variable name is added to this file. When the last observation is reached the value of the automatic variable _N_ (the current or in this case the last observation number) is stored in the macro variable called NUM.

```
DATA _NULL_;
FILE 'C:\BUILD';
SET PARSE END  = FINIS;
IF _N_ = 1  THEN PUT '%LET VAR_LIST = ';
PUT NAME;
IF FINIS THEN  DO;
CALL SYMPUT('NUM' , COMPRESS(_N_));
PUT ';';
PUT 'RUN;';
END;
RUN;
```

Next, the file 'BUILD' is read back into the SAS program using a %INCLUDE statement. The file 'BUILD' will contain the single statement, %LET VAR_LIST = B006 U19 E77.

```
%INCLUDE 'C:\BUILD';
%LET VAR_LIST = B006 U19 E77;
```

When the following data step executes the macro variable VAR_LIST resolves to B006 U19 E77, the macro variable NUM resolves to 3 and the values of these three variables 2,6 and 10 are written to the output window.

```
DATA EXAMPLE4;
SET INPUT2;
FILE PRINT;
ARRAY X[&NUM] &VAR_LIST;
DO I = 1 TO &NUM;
PUT X[I] @;
END;
RUN;
```

EXAMPLE 5: READING DICTIONARY TABLES WITH SQL

This method takes advantage of dictionary tables [2] to get a list of variable names and then stores this list in a macro variable.

In this example, we are going to select only variable names that contain the number '9' from the INPUT2 data set regardless of its position in the variable name. First, we create a table called VAR_NAMES from the INPUT2 data set where name contains '9'. Next, a SELECT statement is used to store the list of names from the table VAR_NAMES into a macro variable called VAR_LIST. In the last select statement we count the number of variables in the table VAR_NAMES and store the value in the macro variable called NUM.

```
PROC SQL NOPRINT;
CREATE TABLE VAR_NAMES AS
SELECT NAME
FROM DICTIONARY.COLUMNS
```

```
WHERE LIBNAME = 'WORK' AND MEMNAME = 'INPUT2'
AND NAME CONTAINS '9';
SELECT COMPRESS(NAME) INTO :VAR_LIST SEPARATED BY " "
FROM VAR_NAMES;
SELECT COUNT(NAME) INTO :NUM SEPARATED BY " "
FROM VAR_NAMES;
QUIT;
```

Since, there are 2 variable names that contain the number ‘9’ the macro variable VAR_LIST resolves to Z90 and U19 and the macro variable NUM resolves to 2. Hence, the following data set writes the values 5 and 6 to the output window.

```
DATA EXAMPLE5;
SET INPUT2;
FILE PRINT;
ARRAY X[&NUM2] &VAR_LIST;
DO I = 1 TO &NUM;
PUT X[I] @;
END;
RUN;
```

CONCLUSION

This paper has explained how to select variable names from a sequentially numbered group, a contiguous group, a group that has the same prefix, as well as groups of variable names which have no desirable patterns. These techniques are useful for many types of applications and can be used in a wide variety of SAS statements. Adding these skills to your programming ability will give you added flexibility in writing efficient code.

REFERENCES

[1] Luo, H., (2001) “That Mysterious Colon (:)”Proceedings of the Twenty Sixth SAS Users Group International Conference, 73-26
[2] Dilorio, F., et. al., (2004) “Dictionary Tables and Views: Essential Tools for Serious Applications” Proceedings of the Twenty Ninth SAS Users Group International Conference, 237-29

AUTHOR CONTACT INFORMATION

Author Name: Stuart Pollack, Ph.D.
E-mail: s_pollack1@yahoo.com