Paper 140-2013

How to Use ARRAYs and DO Loops: Do I DO OVER or Do I DO i?

Jennifer L Waller, Georgia Regents University, Augusta, GA

ABSTRACT

Do you tend to copy DATA step code over and over and change the variable name? Do you want to learn how to take those hundreds of lines of code that do the same operation and reduce them to something more efficient? Then come learn about ARRAY statements and DO loops, powerful and efficient data manipulation tools. This workshop covers when ARRAY statements and DO loops can and should be used, how to set up an ARRAY statement with and without specifying the number of array elements, and what type of DO loop is most appropriate to use within the constraints of the task you want to perform. Additionally, you will learn how to restructure your data set using ARRAY statements and DO loops.

INTRODUCTION

Data preparation can take up to 90-95% of the time dedicated to a statistical analysis for a consulting project. Rather than making sure statistical assumptions are correct, running the procedures to actually analyze the data, and examining the results, much of the time spent on a project is spent preparing the data for analysis. Often, when preparing a data set for analysis the raw data needs to be manipulated in some way; for example, new variables need to be created, specific questionnaire items need to be reversed, and/or scores need to be calculated. The list can go on and on. What makes the task of preparing a data set for analysis tedious is that many times the same operation needs to be performed on a long list of variables (e.g. questionnaire items). For a beginning SAS® programmer, the most likely approach taken to writing the necessary SAS code to write the same code over and over, once for each variable. For example, if there is a 100-item questionnaire and 10 items need to be reversed, the code to reverse these 10 items results in 10 lines of code, one line for each questionnaire item to reverse. Needless to say, there ends up being a lot of copying and pasting of the same code and then changing the code for each variable of interest.

How can a beginning SAS programmer write less SAS code for this type of data preparation that is also more efficient? One way is to use SAS ARRAYs and DO loops.

SAS ARRAYS

A SAS ARRAY is a set of variables of the same type that you want to perform the same operation on. The set of variables is then referenced in the DATA step by the array name. The variables in the array are called the “elements” of the array.

Arrays can be used to do all sorts of things. To list just a few, an array can be used to

1. Set up a list of items of a questionnaire that need to be reversed
2. Change values of several variables, e.g. change a value of “Not Applicable” to missing for score calculation purposes
3. Create a set of new variables from an existing set of variables, e.g. dichotomizing ordinal or continuous variables.

For example, assume we have collected data on the Centers for Epidemiologic Studies Depression (CES-D) scale, a 20-item questionnaire. Each questionnaire item is measured on an ordinal 0 to 3 scale. An overall CESD-D score needs to be calculated and consists of the sum of the 20 questionnaire items. However, 4 questionnaire items were asked such that the responses to the items need to be reversed; that is, 0 needs to become a 3, 1 needs to become a 2, 2 needs to become a 1, and 3 needs to become a 0. The four items that need to be reversed are items cesd4, cesd8, cesd12, and cesd16. An example of the data is given in Figure 1.

Obs	ID	CESD1	CESD2	CESD3	CESD4	CESD5	CESD6	CESD7	CESD8	CESD9	CESD10
1	1101	2	3	2	.	3	2	2	3	3	2
2	1102	0	2	3	0	2	2	2	1	0	0
3	1103	3	0	2	3	2	1	2	3	1	2
4	1104	1	0	0	2	3	3	2	3	3	2
5	1105	3	2	2	.	3	.	3	3	.	2

Obs	CESD11	CESD12	CESD13	CESD14	CESD15	CESD16	CESD17	CESD18	CESD19	CESD20
1	1	3	3	2	3	3	0	1	3	0
2	2	2	2	3	2	3	3	2	1	1
3	1	3	2	2	3	3	1	1	0	2
4	1	2	2	2	0	3	2	2	2	2
5	2	3	3	3	3	3	0	0	2	0

Figure 1: Raw CES-D Data

You might use the following SAS code to reverse the four items resulting in the output in Figure 2.

```
data cesd;
  set in.cesd1;
  cesd4=3-cesd4;
  cesd8=3-cesd8;
  cesd12=3-cesd12;
  cesd16=3-cesd16;
```

Obs	ID	CESD1	CESD2	CESD3	CESD 4	CESD 5	CESD 6	CESD 7	CESD 8	CESD 9	CESD10	CESD11
1	1101	2	3	2	.	3	2	2	0	3	2	1
2	1102	0	2	3	3	2	2	2	2	0	0	2
3	1103	3	0	2	0	2	1	2	0	1	2	1
4	1104	1	0	0	1	3	3	2	0	3	2	1
5	1105	3	2	2	.	3	.	3	0	.	2	2

Obs	CESD12	CESD13	CESD14	CESD15	CESD16	CESD17	CESD18	CESD19	CESD20
1	0	3	2	3	0	0	1	3	0
2	1	2	3	2	0	3	2	1	1
3	0	2	2	3	0	1	1	0	2
4	1	2	2	0	0	2	2	2	2
5	0	3	3	3	0	0	0	2	0

Figure 2: CES-D Data with Items 4, 8, 12, and 16 Reversed.

Notice that the code to reverse each of the four items is essentially the same with the only difference being the variable name of the item needing to be reversed. Copying code that performs the same operation for a small number of variables is not that big of a problem. However, what if the same operation had to be performed on a 100 variables? It would be very inefficient, and I know I would have an increased likelihood of coding errors, to copy the code 100 times and change the variable name in each line. The solution is to use a SAS ARRAY.

INDEXED ARRAY SYNTAX

There are two types of arrays that can be specified in SAS. The first is what I call an indexed array and the second is a non-indexed array. All arrays are set up and accessed only within a DATA step. The syntax for an indexed array is as follows:

```
ARRAY  arrayname {n} [$] [length] list_of_array_elements;
```

where

ARRAY	is a SAS keyword that specifies that an array is being defined
arrayname	a valid SAS name that is **not** a variable name in the data set.
{n}	the index used to give the number of elements in the array, optional
[$]	used to specify if the elements in the array are character variables, the default type is numeric
[length]	used to define the length of new variables being created in the array, optional
list_of_array_elements	a list of variables of the same type (all numeric or all character) to be included in the array

An indexed array is one in which the number of elements, {n}, is specified when the array is defined. A non-indexed array is one in which the number of elements is not specified and SAS determines the number of elements based on the number of variables listed in the array. You can always use an indexed array, however you can only sometimes, depending on the situation, use a non-indexed array.

Remember that the arrayname must be a valid SAS name that is not a variable name in the data set. One tip I can give you to help distinguish an array name from a variable name is to start the arrayname with the letter “a”.

EXAMPLE OF AN INDEXED ARRAY

Going back to the example of reversing the CES-D items, the SAS code that would be required to define an indexed array containing the 4 CES-D items that need to be reversed is

```
data cesd;
  set in.cesd1;
  array aireverse {4} cesd4 cesd8 cesd12 cesd18 ;
```

In defining this array we first specify the SAS keyword ARRAY with

aireverse	the arrayname used to reference the array in future SAS code
{4}	there are 4 elements that will be in the array
[$]	not needed as all variables in the array are numeric
[length]	not needed
cesd4 cesd8 cesd12 cesd18	is the list of the variables that specify the 4 array elements.

NON-INDEXED ARRAY SYNTAX

In addition to the indexed array, SAS also provides the option of using a non-indexed array. Here you don't specify the number of elements in the array, {n}. Rather, during the creation of the array, SAS determines the number of elements of the array based on the set of variables listed. The syntax for a non-indexed array is as follows:

```
ARRAY  arrayname [$] [length] list_of_array_elements;
```

where

ARRAY	is a SAS keyword that specifies that an array is being defined
arrayname	a valid SAS name that is **not** a variable name in the data set.
[$]	used to specify if the elements in the array are character variables, the default type is numeric
[length]	used to define the length of new variables being created in the array, optional
list_of_array_elements	a list of variables of the same type (all numeric or all character) to be included in the array

EXAMPLE OF A NON-INDEXED ARRAY

Again, using the CES-D item reversal example, the SAS code that would be to define a non-indexed array containing the 4 CES-D items that need to be reversed is

```
data cesd;
  set in.cesd1;
  array areverse cesd4 cesd8 cesd12 cesd18;
```

In defining this array we first specify the SAS keyword ARRAY with

areverse	the arrayname used to reference the array in future SAS code
cesd4 cesd8 cesd12 cesd18	is the list of the variables that specify the 4 array elements.

One great thing about non-indexed arrays is that they allow for less typing, but give the same functionality in the use of an array.

SAS DO LOOPS

So we have now defined our array, but now we have to use it to manipulate the data. We use a DO loop to perform the data manipulations on the array(s). Within a DATA step, a DO loop is used to specify a set of SAS statements or operations that are to be performed as a unit during an iteration of the loop. It is important to note that operations performed within a DO loop are performed **within** an observation. Another thing that you need to be aware of is that every DO loop has a corresponding END statement. If you don't END your DO loop, you will get a SAS Error message in your log indicating that a corresponding END statement was not found for the DO statement.

There are four different types of DO loops available in SAS.

1. DO index=, an iterative, or indexed, DO loop used to perform the operations in the DO loop at a specified start and ending index value for an array
2. DO OVER loop used to perform the operations in the DO loop over ALL elements in the array

3. DO UNTIL (logical condition) loop used to perform the operations in the DO loop until the logical condition is satisfied
4. DO WHILE (logical condition) loop used to perform the operations in the DO loop while the logical condition is satisfied

Many times, DO loops are used in conjunction with a SAS array, with the basic idea being that the operations in the DO loop will be performed over all the elements in the array. It should be noted that within a single DO loop multiple arrays can be referenced and operations on different arrays can be performed as long as the arrays have the same number of elements in them.

ITERATIVE DO LOOP DEFINITION AND SYNTAX

An iterative DO loop executes the statements between a DO statement and an END statement repetitively based on the value of the specified starting and stopping values of an index. The syntax for an iterative DO loop begins with the SAS keyword DO and is given by

DO indexvariable = startingvalue TO stoppingvalue <BY increment>;

or

DO indexvariable = startingvalue, nextvalue, …., endingvalue;

where

indexvariable	a valid SAS variable name, e.g. i
startingvalue	a valid starting value, for an indexed array this should be greater than or equal to 1 but less than the number of elements in the array, can be a character value if not used in conjunction with an array
endingvalue	a valid ending value, for an indexed array this should be less than or equal to the total number of elements in the array, can be character if not used in conjunction with an array
<BY increment>	can specify for numeric starting and ending values how to increment the array, optional, e.g. by 2 to do every other element in the array.

Note, if the DO loop is being used in conjunction with an array, the way to reference the array element in the DO loop is to reference the array name and specify the index variable used in the DO statement in square brackets, arrayname[indexvariable]. An advantage of an iterative DO loop used in conjunction with an array is that the DO loop can be used to specify a subset of the array elements to perform the operation on.

EXAMPLE OF ITERATIVE DO LOOP WITH INDEXED ARRAY

Going back to the CES-D example, the indexed array "**aireverse**" has been defined. The DO loop is now defined and the operation that needs to be performed included between the DO and END statements.

```
data cesd;
  set in.cesd1;
  array aireverse {4} cesd4 cesd8 cesd12 cesd16;
  do i=1 to 4;
    aireverse[i]=3-aireverse[i];
  end;
```

The above SAS code specifies the indexed array "**aireverse**" that contains 4 elements with the list of elements being cesd4, cesd8, cesd12 and cesd16. The iterative DO loop is specified as

indexvariable	i
startingvalue	1
endingvalue	4
<BY increment>	not specified so the loop increments by 1 each time

Within the DO loop the following operation is performed on the array "**aireverse**" where

aireverse[i]	specifies the arrayname and the indexvariable to use for each iteration of the loop

Specifically, Table 1 gives the value of the index variable and the value of the array element being used for every iteration of the loop.

Iteration	i =	areverse[i]	Value of areverse[i]	Operation Performed	Increment i by 1
One	1	areverse[1]	cesd4	cesd4=3-cesd4	i=1+1=2
Two	2	areverse[2]	cesd8	cesd8=3-cesd8	i=2+1=3
Three	3	areverse[3]	cesd12	cesd12=3-cesd12	i=3+1=4
Four	4	areverse[4]	cesd16	cesd16=3-cesd16	i=4+1=5 so exit the loop

Table 1: Iterative Step Through of DO Loop Processing of Array "aireverse"

The resulting output is exactly the same as that presented in Figure 2.

DO OVER LOOP DEFINITON AND SYNTAX

The DO OVER loop is one of the most useful DO loops that I have discovered in SAS. It can be used with an array when indexing of the array is not needed. For example, if the operation needs to be performed on all elements there is no reason to define the array with an index and no need to created an iterative DO loop. The key to using this type of DO loop is that the operations between the DO and END statements will be performed over ALL elements in the array. An advantage is that the number of array elements does not need to be known. The syntax for the DO OVER loop is

```
DO OVER arrayname;
```

where

arrayname	valid non-indexed arrayname that has been previously defined

When using an arrayname in the operations to perform within the DO loop, one nice feature of the DO OVER loop is that no index is needed when referencing the array. One just needs to reference the arrayname like one would a variable name.

EXAMPLE OF DO OVER LOOP WITH A NON-INDEXED ARRAY

Again, using the CES-D example, the non-indexed array "**areverse**" has been defined. The DO OVER loop is now defined and the operation that needs to be performed included between the DO and END statements.

```
data cesd;
  set in.cesd1;
  array areverse cesd4 cesd8 cesd12 cesd16;
  do over areverse;
    areverse=3-areverse;
  end;
```

The above SAS code specifies the non-indexed array "**areverse**". SAS will determine that the array "**areverse**" contains 4 elements: **cesd4, cesd8, cesd12 and cesd16**. The non-iterative DO OVER loop is specified as

DO OVER areverse;

where

areverse the arrayname to perform the operation on each element of the array

REFERENCING MULTIPLE ARRAYS IN A SINGLE DO LOOP

One advantage of using ARRAYs and DO loops is that operations on multiple arrays can be done within a single DO loop; however there are things that must be kept in mind. The first is that the arrays should all contain the same number of elements. The second is that if operations are being done within a DO loop on two or more arrays simultaneously (i.e. an operation using one array is being used to create new variables in a second array), the order of the array elements needs to be the same in each array. In other words, if we are creating newvar1 in newarray from var1 in oldarray then newvar1 and var1 need to be in the same position in the list of array elements. Here is an example.

```
              .      .       .       .
              ▼      ▼       ▼       ▼
array aold cesd4  cesd8   cesd12  cesd16;
array arev rcesd4 rcesd12 rcesd16 rcesd8;
do over aold;
  arev=3-aold;
end;
```

Looking at the above code, the DO loop will correctly assign the reversed value of the first element of the array **aold**, which is reference variable **cesd4**, to the first element of the value of the first element of the array **arev**, **rcesd4**. However, during the second pass through the loop, and hence the arrays, the value of the second element of the array **aold**, which is referencing the variable **cesd8**, will incorrectly assign the reversed value of **cesd8** to the second element of the array **arev**, which is the variable **rcesd12**. Similarly, an incorrect assignment will be made for the third and fourth elements of the array **arev**.

When using multiple arrays within a DO loop, it is imperative to make sure that the elements are in the correct order in each array.

```
              .      .       .       .
              ▼      ▼       ▼       ▼
array aold cesd4  cesd8  cesd12  cesd16;
array arev rcesd4 rcesd8 rcesd12 rcesd16;
do over aold;
  arev=3-aold;
end;
```

EXAMPLES USING NON-INDEXED ARRAYS AND DO OVER LOOPS

As stated before, ARRAYs and DO loops can be used for a variety of data manipulation needs from assigning or reassigning values, renaming variables, creating new variables from existing variables, simulating data, creating randomization schemes, and changing the data structure. The following are examples of how to manipulate data for some of these applications.

EXAMPLE 1 – ASSIGNING OR REASSIGNING VALUES

Suppose data have been entered such that for a series of items in a list an individual had to mark whether the item applied to them or not. In order to make data entry faster, the person entering the data only entered a 1 if the item was checked and otherwise left the item as missing. We want to change all missing values to 0.

```
data list;
  set in.list1;
  array aquest q1-q20;
  do over aquest;
    if aquest=. then aquest=0;
  end;
run;
```

There are a couple things to note. The first is that because we want to perform the operation on all variables in the array, we can create a non-indexed array and use a DO OVER loop. Secondly, we can specify the list of array elements with the "-" in between the first variable name, q1, and last variable name, q20, because the variables are named with the same root name followed by a number that is indicated sequentially from 1 to 20.

EXAMPLE 2 – "RENAMING" VARIABLES

Another use of arrays and DO loops for data manipulation is the ability to "rename" variables. The word "rename" is in quotes for a reason. We are not really renaming the variables, but creating a new variable that contains the exact information as the existing or old variable. Below is the SAS code used to "rename" a group of variables. You should know that there are other ways to actually rename variables.

```
array aquest q1-q20;
array anquest nq1-nq20;
do over aquest;
  anquest=aquest;
  if anquest=. Then anquest=0;
end;
```

Note that here we "rename" the variables in array **aquest**, q1-q20, to nq1-nq20 in array **anquest**, and then assign the new variables a value of 0 if the value was missing. If we do not drop the original variables, q1-q20, we have preserved the raw data values within our data set as well as created new variables that we can use for analysis with no missing data.

EXAMPLE 3 – CREATION OF NEW VARIABLES USING MULTIPLE ARRAYS

Arrays and DO loops can also be used to create new variables using multiple arrays. For example, an investigator collects data on height in meters and weight in kilograms at 5 different time points and needs the body mass index (BMI) determined for each weekly measure. To utilize arrays and DO loops to calculate the BMI you must first set up three arrays. These can be indexed or non-indexed arrays. If you are using non-indexed arrays you need to be careful to order the variables for height and the variables for weight in the same order in their respective arrays. The SAS code that might be used is

```
array awtkg wtkg1-wtkg5;
array ahtm htm1-htm5;
array abmi bmi1-bmi5;
do over awtkg;
  abmi=awtkg/(ahtm**2);
end;
```

Notice in the above code that the order of the variables in each array is the same, **var1-var5**. As well, notice that you only need to specify one arrayname in the DO OVER as each array contains the same number of elements. If you wanted to use indexed arrays and iterative DO loops to perform the same operations above, the SAS code is

```
array awtkg {5} wtkg1-wtkg5;
array ahtm  {5} htm1-htm5;
array abmi  {5} bmi1-bmi5;
do i=1 to 5;
  abmi[i]=awtkg[i]/(ahtm[i]**2);
end;
```

EXAMPLE 4 – USING INDEXED ARRAYS AND ITERATIVE DO LOOPS TO CHANGE THE DATA STRUCTURE

The final example we will examine demonstrates the solution to a common problem of changing the data structure from a short wide data set to a long narrow data set; from one record per ID to multiple records per ID. The use of arrays and DO loops for this problem is especially useful when an investigator has collected repeated measures of several variables yet entered one observation containing all measures for each individual. Using the height and weight data from Example 3, the raw data are given in Figure 3 as an example of a short wide data set. Here there are 5 measurement times with height and weight collected at each time point. Only 3 observations are shown.

Obs	id	htm1	htm2	htm3	htm4	htm5	wtkg1	wtkg2	wtkg3	wtkg4	wtkg5
1	1	0.9072	1.0080	1.1592	1.3356	1.5372	20.4545	25.9091	29.5455	37.2727	45.4545
2	2	0.7560	0.8316	0.9576	1.1088	1.2348	15.9091	18.1818	21.3636	25.4545	32.2727
3	3	0.8316	0.9324	1.1088	1.2348	1.3608	19.5455	27.2727	32.7273	39.5455	52.2727

Figure 3: Raw Height and Weight Data at 5 Measurement Times.

We want to change the structure of this data set so that each individual has five observations with each observation corresponding to a measurement time. The steps to restructure the data are as follows:

1. Create an array for each measurement that contains the variables corresponding to the measurements at the specific time points.
2. Create an iterative DO loop with an index that uses the first measurement time at the starting index value and the last measurement time as the ending index value.
3. Perform any operations that are needed on the variables, such as creating new variables at each measurement time.
4. Create a measurement time variable. This can be a created as some function of the index variable from the DO loop if you want.
5. Assign each arrayname to a new variable name that is NOT an arrayname, e.g. newvar=arrayname[i];
6. Use an OUTPUT statement **before** ending the DO loop to output the observation to the data set.
7. END the DO loop.

8. Using a DROP or KEEP statement, drop any variables that you don't need or keep only those variables you want to keep.

Using the BMI data in Figure 3, below is example SAS code to transform the short wide data set to a long skinny data set. Because we are creating the BMI for each measurement time from the existing height and weight variables and we are creating a measurement time variable, the new data set should end up having 15 observations and 5 variables.

```
data bmi_longskinny;
  set in.bmi_shortwide;
  array ahtm {5} htm1-htm5;
  array awtkg {5} wtkg1-wtkg5;
  array abmi {5} bmi1-bmi5;
  do i=1 to 5;
    timept=i;                          ** creates the measurement time variable;
    abmi[i]=awtkg[i]/(ahtm[i]**2);     ** calculates the bmi for each time point;
    htm=ahtm[i];                       ** assigns the ith value of array ahtm to htm;
    wtkg=awtkg[i];                     ** assigns the ith value of array awtgk to wtkg;
    bmi=abmi[i];                       ** assigns the ith value of array abmi to bmi;
    output;                            ** outputs the observation to the data set;
  end;
  keep id timept htm wtkg bmi;
run;
```

Figure 4 shows the long skinny data set. Notice that each id has 5 observations corresponding to the 5 measurement times.

Obs	id	timept	htm	wtkg	bmi
1	1	1	0.9072	20.4545	24.8533
2	1	2	1.0080	25.9091	25.4995
3	1	3	1.1592	29.5455	21.9874
4	1	4	1.3356	37.2727	20.8948
5	1	5	1.5372	45.4545	19.2361
6	2	1	0.7560	15.9091	27.8357
7	2	2	0.8316	18.1818	26.2911
8	2	3	0.9576	21.3636	23.2974
9	2	4	1.1088	25.4545	20.7042
10	2	5	1.2348	32.2727	21.1662
11	3	1	0.8316	19.5455	28.2629
12	3	2	0.9324	27.2727	31.3707
13	3	3	1.1088	32.7273	26.6197
14	3	4	1.2348	39.5455	25.9360
15	3	5	1.3608	52.2727	28.2284

Figure 4: Restructured Data Set of 5 Observations Corresponding to 5 Measurement times for Each of Three Individuals

CONCLUSION

The utility of SAS ARRAYs and DO loops are many when having to manipulate data. We have looked at just some of the ways to use arrays and DO loops including creating new variables, performing the same operation over a set a variables more efficiently, “renaming” variables, and changing the data structure. Arrays and DO loops can also be used to simulate data and create randomization schemes. SAS ARRAYs and DO loops are powerful data manipulation tools that help to make your program more efficient and save you lots of time and energy.

CONTACT INFORMATION

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

Jennifer Waller
Georgia Regents University
Department of Biostatistics and Epidemiology, AE-1012
Augusta, GA 30912-4900

jwaller@gru.edu