Data Management using SAS Arrays

- Lavanya Vangala

When do we need Arrays?

- ✓ repetitive actions on a group of variables
- ✓ computing new variables
- ✓ reshaping data

Repetitive action: An Example

We have been given a SAS data set where a value of 999 was entered whenever there was a missing numeric value.

General coding Practice (Manual Recoding with If – Then):

```
data recode_data;
  set actual_data;
  if a =999 then a1=.;
  if b = 999 then a2=.;
  if c = 999 then a3=.;
  if d = 999 then a4=.;
  if e = 999 then a5=.;
  if f = 999 then a6=.;
Run;
```

“Same code on multiple variables”

What is An Array?

- Array is similar to Vector/matrix notation in Mathematics

Structure:

✓ Group of variables in a data step

✓ Elements must be of same data type

✓ No need of same length

✓ No need to be related

✓ Can be multi dimensional

ARRAY Statement

ARRAY arrayname {n} [$] [length] array_elements;

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 (A numeric constant/Numeric SAS expression /(*)
[$]	- 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
array_elements	- a list of variables of the same type (all numeric or all character) to be included in the array

SAS DO Loops and Why?

- ✓ SAS Array provides a different way to reference a group of variables
- ✓ Reference to the Array elements can be done using DO loops.
- ✓ DO loop should have a corresponding END statement (Caution !)
- ✓ within a single DO loop multiple arrays can be referenced and operations on different arrays can be performed.

Using Arrays – Recoding variables

```
data recode_data;
  set actual_data;
  Array recode(6) a b c d e f;
  Do I = 1 to 6;
    If recode(i) = 999 then recode(I) = .;
  End;
  Drop I;
Run;
```

For Recoding hundreds of variables arrays are certainly helpful..☺

Note :– Array elements do not need to be named consecutively

Extension to All variables

SAS also has a few code words to put all character and/or numeric variables into arrays for you. These can save a lot of typing and is useful for cleaning up missing values in your variables.

```
data recode_data;
  set actual_data;
  Array numeric_recodes(*) _numeric_;
  Array character_recodes(*) $ _character_;
  Do _n_= 1 to dim(numeric_recodes);
    If numeric_recodes(_n_) = 999 then numeric_recodes(_n_) = .;
  End;
  Do _n_= 1 to dim(character_recodes);
    If character_recodes(_n_) = 'N/A' then character_recodes(_n_) = '';
  End;
Run;
```

Note: * - Is used when the number of elements in the array isn't known

Assigning Initial values

- ✓ Initial values can be assigned to Array members
- ✓ Values must always be specified at the end of ARRAY statement
- ✓ Character values need to be in Quotes

Examples:

Array a {10} a1-a10 (5 5 5 5 5 5 5 5 5 5);

Array a{10} (10*5);

Array new {4} $ test1 – test4 (‘English’ ‘Math’ ‘Sci’ ‘Hist’);

All variables that have been initialized will be retained.

Referencing Array Members

To refer an array member place the name of the array and the number of the member we want to refer

Example:

Array new {4} $ test1 – test4 (‘English’ ‘Math’ ‘Sci’ ‘Hist’);

New (2) returns ‘Math’

Array Functions

Three SAS functions that pertain specifically to arrays

Function	Definition
Dim(arrayname)	Returns the number of array elements
Lbound(arrayname)	Returns the lowest subscript
Hbound(arrayname)	Returns the highest subscript

Example: Do I = 1 to dim(Arrayname);

Dim function is useful when we do not know the number of elements referenced in an array.

Specifying ARRAY Bounds

Array YR(5) YR2003 – YR2007

represents 5 variables YR2003, YR2004, YR2005, YR2006, YR2007

(but we have to remember that YR(1) represents YR2003 and so on)

However, we can also specify the lower and upper bounds by entering the lower bound, a colon and then the upper bound. That is, the above statement could be written as:

ARRAY YR(2003:2007) YR2003 – YR2007;

Another application where specifying array bounds is extremely useful is when we are counting from 0 instead of 1. For example, we may be measuring pulse rate (HR) at times 0,1,2,3:

Array HR(0:3) HR0-HR3;

Temporary Arrays

Temporary arrays - There are no variables saved in the array, only the values, and the values are thrown away at the end of the data step. It is defined by the key word _TEMPORARY_

There are some benefits to using temporary arrays:

- **They save a lot of space because the values are not kept for every observation**
- **They process faster than using regular arrays with variables**

Temporary arrays are useful if we need the values only for a calculation within the data step.

Caution - * can't be used in a temporary Array

Temporary Array - Example

A firm has 5 types of employees and the salary rise could be 2% for grade A , 3% for grade B, 5% for grade C, 6% for grade D and 8% for grade E. To find their new salaries the coding might be :

```
if type_employee = 'A' then new_sal= salary*1.02;
else if type_employee eq 'B' then new_sal= salary*1.03;
else if type_employee eq 'C' then new_sal= salary*1.05;
else if type_employee eq 'D' then new_sal= salary*1.06;
else if type_employee eq 'E' then new_sal= salary*1.08;
```

Using _temporary_ the above code can be re-written as :

```
array newrate {5} _temporary_ (1.02  1.03  1.05  1.06  1.08);
if type_employee ne '' then
 new_sal = salary* newrate{rank(type_employee)-64};
```

Arrays – more examples

Arrays can also be used inside of functions. For example:

Total = sum(a{1}, a{2}, a{3}, a{4});

or

Total=sum(of a{*});

Computing new variables

New variables can be generated using existing variables with the help of Arrays:

Array nitem [60] nitem1 – nitem60;

Array price [60] price1 – price60;

Array tax [60] tax1 – tax60;

Do I = 1 to 60;

If nitem[i] gt 0 then tax[i] = nitem[i] * price[i] * .06 ;

End;

This example demonstrates the use of array references on both the right and left sides of an equation. It also shows how to create new variables with an array (the tax variables). These will be new numeric variables.

Counting Missing Values in the list of variables

If we wish to check the number of missing values in each variable from a group of variables the following can be used.

```
count=0;
array miss_var {10} a1 – a10;
do i = 1 to 10 while (miss_var{i} ne ‘ ’);
count+1;
end;
```

Reshaping Data

Suppose we have a SAS data set called score with variables id, score1, score2 and score3.

ID	Score1	Score2	Score3
01	150	350	
02	120	200	300
03	200		
04		400	

And if we wish to restructure the data set as:

Reshaping Data (Cont.)

ID	Score
01	150
01	350
02	120
02	200
02	300
03	200
04	400

Now it is easy to use proc freq on a single variable: Arrays helps us in this:

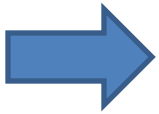
```
Data new;
  Set score;
  Array res(3) score1 – score3;
  Do I = 1 to 3;
    Score = res(i);
    If score ne . Then output;
  End;
  Keep id score;
Run;
```

Creating multiple observations from a single observation

Another Situation

- A SAS data set has ID variable and 3 variables s1, s2, s3 which represents a score at times 1, 2,3.

ID	S1	S2	S3
01	3	6	7
02	4	7	8
03	5	9	4



Data set

ID	Time	Score
01	1	3
01	2	6
01	3	7
02	1	4
02	2	7
02	3	8
03	1	5
03	2	9
03	3	4

Cont.…

Array Statement

```
Data manyobs;
  Set obs;
  Array s(3);
  Do i = 1 to 3;
    Score = s(i);
    Output;
  End;
  keep id t score;
Run;
```

Observation:

Note that ARRAY statement does not have variable list.

Array S(3);

Is equivalent to

Array s(3) s1- s3;

Creating single observation from Multiple observations

Example

Subject	topicA	count
1	0	1
1	1	2
1	2	3
2	4	1
2	5	2
2	6	3
3	7	1
3	3	2
3	2	3

SAS Code

```
data wide_real;
  set count_real;
  array AtopicA(6) topicA_1-topicA_6;
  retain topicA_1-topicA_6;
  by person;
  If first.person then do;
    do i = 1 to 6;
      AtopicA[i] = .;
    end;
  end;
  AtopicA(count) = topicA;
  if last.person then output;
 run;
```

Cont.....

Result

Subject	topicA_1	topicA_2	topicA_3
1	0	1	2
2	4	5	6
3	7	3	2

Proc Format and Array

Proc Format and Array would provide us the flexible of searching data.

Any changes in search criteria can be made with minimal editing in Proc format

Example

```
proc format;
value $search
‘291’-‘29299’,‘303’-‘30493’,‘305’-‘30593’= ‘1’;
run;
data req_out;
set  all;
array diag(5) $ pdx dx2-dx5;
do i = 1 to 5;
if put(diag(i),$search.) = ‘1’ then do;
output;leave;end;end;drop i;
```

General Errors while working with Arrays

- ✓ **INVALID INDEX RANGE**
- ✓ **FUNCTION NAME AS AN ARRAY NAME**
- ✓ **ARRAY REFERENCED IN MULTIPLE DATA STEPS, BUT DEFINED IN ONLY ONE**

Conclusions

General Uses

- The combination of arrays and Do loops rend power to the programming
- Extremely useful for iteration of similar steps for group of variables
- Variables referred in an array need not be related

Limitations:

- Only be used in DATA step not useful in PROC step
- Can only be referred to variables but not data sets or values of a variable.

Thank You & Regards

Lavanya Vangala
Email: lavanya_vangala@gmail.com