STA 311 Statistical Computing and Data Management

Instructor: Cheng Peng, Ph.D.
Department of Mathematics
West Chester University
West Chester, PA 19383

Office: 25 University Avenue, RM 111
Phone: 610-436-2369
Email: cpeng@wcupa.edu

WCU WEST CHESTER UNIVERSITY

Topics to Be Covered

- Review of Do Block & DO Loops
- Concepts of SAS Arrays
- PROC TRANSPOSE
- Temporary Arrays
- Restructuring Tables

DO Block & DO Loop

Designate a group of statements to be executed as a unit using a DO block. The following are general syntaxes:

```
DO;
   SAS Statements;
END;
```

```
DO var=1 TO x;
   SAS Statements;
END;
```

```
DO UNITL ( condition);
   SAS Statements;
END;
```

```
DO WHILE ( condition);
   SAS Statements;
END;
```

The difference between the two DO loops is that DO UNTIL statement tests At the bottom of the of the loop and DO WHILE statements tests at the top.

DO Block & DO Loop

DO loops can be used to create an ordered sequence of numbers.

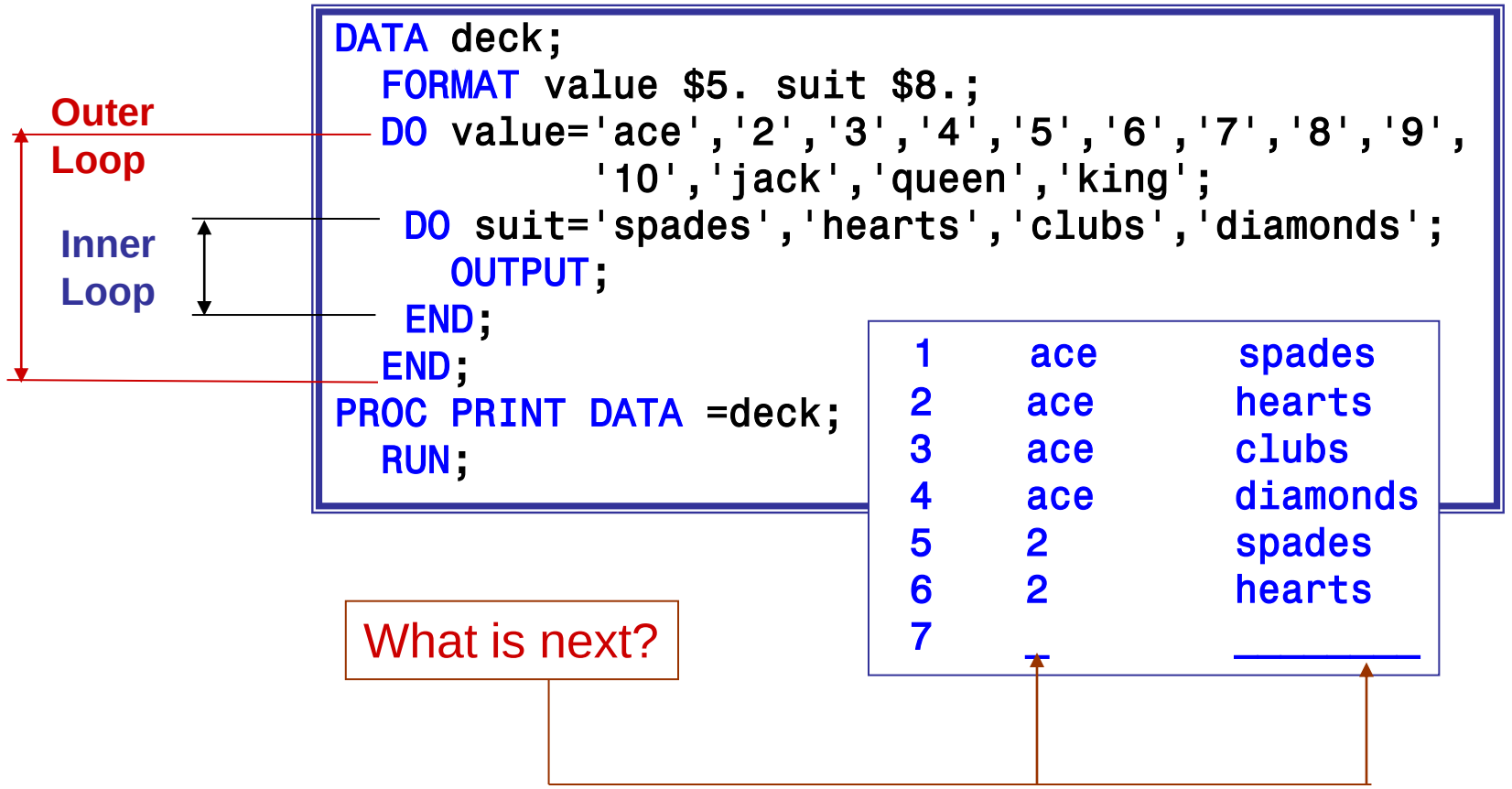
```
DATA example;
  DO q=1 TO 5;
    qtimes2=q*2;
    qsquared=q**2;
    OUTPUT;
  END;
PROC PRINT DATA =example;
  RUN;
```

END statement important.

OBS	Q	QTIMES2	QSQUARED
1	1	2	1
2	2	4	4
3	3	6	9
4	4	8	16
5	5	10	25

No INPUT or DATALINES used!

DO Block & DO Loop



Concepts of SAS ARRAY

Suppose you have 105 variables (X1-X100, A, B, D, and E) in SAS dataset OLD, and a value of 999 is used to represent missing data. It is common practice in some other database systems to use values such as 99, 999, etc., to represent missing values. Suppose further that you want to substitute a SAS System missing value (.) for the values of 999. How to effectively accomplish this task?

```
DATA Hard_Way;
   SET OLD;
   IF X1 = 999 THEN X1 =.;
   IF X2 = 999 THEN X2 =.;
     .. .. .. .. .. .. ..
   IF X100 = 999 THEN X100 =.;
   IF A    = 999 THEN A =.;
   IF B    = 999 THEN B =.;
   IF C    = 999 THEN C =.;
   IF D    = 999 THEN D =.;
   IF E    = 999 THEN E =.;
RUN;
```

Concepts of SAS ARRAY

An array is a temporary holding site for a collection of **variables** upon which the same operations will be performed. It is often to find DO loops to be used jointly with ARRAY manipulation.

```
DATA EASYWAY;
    SET OLD;
    ARRAY MyArray[105] X1-X100 A B C D E;
    DO I = 1 TO 105;
       IF MyArray[I] = 999 THEN MyArray[I] = .;
    END;
    DROP I;
RUN;
```

Concepts of SAS ARRAY

If the number of variables is NOT specified in the definition of ARRAY, you can use (*) instead. The SAS system will count the number of variables.

In the DO loop you need to use a system function DIM() to return the length of the one-dimensional ARRAY.

```
ARRAY MyArray[ * ] X1 – X100 A B C D E;
```

```
DO I = 1 TO DIM(MyArray);
```

The array name is followed by either a pair of parentheses ( ), braces { }, or square brackets [ ]. By specifying a value inside the bracket, we can assign the same number of variables to the array. We use square brackets [ ] in this course.

Concepts of SAS ARRAY

The internal variable _NUMERIC_ can be used to refer to all the numeric variables in a SAS dataset (either in a DATA step or PROC step). The terms _CHRACTER_ and _ALL_ are also available and represent all character variables and all variables respectively.

```
/* CAUTION: The following SAS program is NOT Working!!*/
DATA NEW;
    SET OLD;
    ARRAY XXX[*] _NUMERIC_;
    DO I = 1 TO DIM (XXX);
       IF XXX[I] = 999 THEN XXX[I] = .;
    END;
    DROP I;
RUN;
```

```
DATA NEW;
    SET OLD;
    ARRAY $ YYY[*] _CHARACTER_;
    DO I = 1 TO DIM (YYY);
       IF YYY[I] = 'NA' THEN YYY[I] = ' ';
    END;
    DROP I;
RUN;
```

The DIM function is especially useful here because you don't have to count the number of numeric variables. Caution: Don't give your array the same name as a variable in your dataset!!

Concepts of SAS ARRAY

Rescale all numeric variables

```
DATA melons;
  INPUT location $ ltpink pink salmon red;
  nomelons=50-(ltpink+pink+salmon+red);
  ltpink  =ltpink/50;
  pink    =pink/50;
  salmon  =salmon/50;
  red     =red/50;
  nomelons=nomelons/50;
  DATALINES;
North 3 14 16 8
East 8 23 9 2
South 0 4 10 19
;
```

The HARD way

```
DATA melons;
  INPUT location $ ltpink pink salmon red;
  nomelons=50-(ltpink+pink+salmon+red);
  ARRAY colors (5) ltpink pink salmon red nomelons;
  DO i=1 TO 5;
    colors(i)=colors(i)/50;
  END;
DATALINES;
North 3 14 16 8
East 8 23 9 2
South 0 4 10 19
;
RUN;

PROC PRINT DATA =melons;
  RUN;
```

Group response variables into a color array.

An observations is output only after the DO loop is completed. The DO loop is repeated for each observation.

The EASY way

Concepts of SAS ARRAY

Input Data →

```
North 3 14 16  8
East  8 23  9  2
South 0  4 10 19
```

```
nomelons=50-(ltpink+pink+salmon+red);
 ltpink  =ltpink/50;
 pink    =pink/50;
 salmon  =salmon/50;
 red     =red/50;
 nomelons=nomelons/50;
```

OBS	LOCATION	LTPINK	PINK	SALMON	RED	NOMELONS	I
1	North	0.06	0.28	0.32	0.16	0.18	6
2	East	0.16	0.46	0.18	0.04	0.16	6
3	South	0.00	0.08	0.20	0.38	0.34	6

Note that grouping variables into a color array did nothing to variable names. The DO loop index variable, I, was added in the dataset. Note that I is always 6, since at the beginning of the sixth execution, the value of I is 6, exceeding the specified range of 1 TO 5, SAS stops processing the loop.

Concepts of SAS ARRAY

```
DATA melons;
  INPUT location $ red1 red2 red3 red4;
        red5=50-(sum(of red1-red4));
  ARRAY colors (5) red1-red5;
  DO i=1 TO 5;
    colors(i)=colors(i)/50;
  END;
  DATALINES;
```

One dash needed if it variable names are numbered.

Two dashes needed if variable names not numbered.

```
DATA melons;
  INPUT location $ ltpink pink salmon red;
        nomelons=50-(sum(of ltpink--red));
  ARRAY colors (5) ltpink--nomelons;
  DO i=1 TO 5;
    colors(i)=colors(i)/50;
  END;
 DATALINES;
```

Reshape Data Using SAS ARRAY

Wide Table

Name	hw1	hw2	hw3	hw4	hw5
Amy	6	8	4	8	7	8
Bob	3	5	5	7	5	5
Carol	7	8	8	7	8	9
Dave	10	9	9	8	10	9
Eve	6	6	8	6	9	7

Data in efficient input format.

TRANSPOSE

Data in format needed for data analysis.

Long Table

OBS	NAME	Week	Grade
1	Amy	HW1	6
2	Amy	HW2	8
3	Amy	HW3	4
4	Amy	HW4	8
5	Amy	HW5	7
6	Amy	HW6	8
(lines deleted)			
25	Eve	HW1	6
26	Eve	HW2	6
27	Eve	HW3	8
28	Eve	HW4	6
29	Eve	HW5	9
30	Eve	HW6	7

Reshape Data Using SAS ARRAY

```
DATA grades1;
  INPUT name $ hw1-hw6;
  DATALINES;
 Amy    6  8  4  8  7  8
 Bob    3  5  5  7  5  5
 Carol  7  8  8  7  8  9
 Dave  10  9  9  8 10  9
 Eve    6  6  8  6  9  7
;
DATA grades2;
  SET grades1;
  week=1;  grade=hw1;  OUTPUT;
  week=2;  grade=hw2;  OUTPUT;
  week=3;  grade=hw3;  OUTPUT;
  week=4;  grade=hw4;  OUTPUT;
  week=5;  grade=hw5;  OUTPUT;
  week=6;  grade=hw6;  OUTPUT;
  keep name week grade;
  RUN;
PROC PRINT DATA =grades2;
  RUN;
```

Brute Force

OBS	NAME	WEEK	GRADE
1	Amy	1	6
2	Amy	2	8
3	Amy	3	4
4	Amy	4	8
5	Amy	5	7
6	Amy	6	8
7	Bob	1	3
8	Bob	2	5
9	Bob	3	5
10	Bob	4	7
11	Bob	5	5
12	Bob	6	5
13	Carol	1	7
.....			

Reshape Data Using SAS ARRAY

```
DATA grades1;
  INPUT name $ hw1-hw6;
  DATALINES;
 Amy    6  8  4  8  7  8
 Bob    3  5  5  7  5  5
 Carol  7  8  8  7  8  9
 Dave  10  9  9  8 10  9
 Eve    6  6  8  6  9  7
;
DATA grades2;
  SET grades1;
  ARRAY homework (6) hw1-hw6;
  DO i=1 TO 6;
    week=i;
    grade=homework(i);
    OUTPUT;
  END;
KEEP name week grade;
RUN;
```

Array/DO Loop

OBS	NAME	WEEK	GRADE
1	Amy	1	6
2	Amy	2	8
3	Amy	3	4
4	Amy	4	8
5	Amy	5	7
6	Amy	6	8
7	Bob	1	3
8	Bob	2	5
9	Bob	3	5
10	Bob	4	7
11	Bob	5	5
12	Bob	6	5
13	Carol	1	7
.....			

PROC TRANSPOSE

```
DATA grades1;
  INPUT name $ hw1-hw6;
  DATALINES;
 Amy    6  8  4  8  7  8
 Bob    3  5  5  7  5  5
 Carol  7  8  8  7  8  9
 Dave  10  9  9  8 10  9
 Eve    6  6  8  6  9  7
;
RPOC SORT DATA =grades1;
  BY name;
  RUN;
PROC TRANSPOSE DATA =grades1
               OUT=grades2;
  BY name;
  VAR hw1-hw6;
  RUN;
PROC PRINT DATA =grades2;
  RUN;
```

OBS	NAME	_NAME_	COL1
1	Amy	HW1	6
2	Amy	HW2	8
3	Amy	HW3	4
4	Amy	HW4	8
5	Amy	HW5	7
6	Amy	HW6	8
(lines deleted)			
25	Eve	HW1	6
26	Eve	HW2	6
27	Eve	HW3	8
28	Eve	HW4	6
29	Eve	HW5	9
30	Eve	HW6	7

All variables in the var statement are now placed in new rows.

RETAIN:
"Remembering" Values from Previous Observations

```
DATA NOGOOD;
   SUBJECT = SUBJECT + 1;
   INPUT SCORE1 SCORE2;
DATALINES;
 3 4
 5 6
 7 8
;

PROC PRINT DATA=NOGOOD;
   TITLE1 'Incorrect Program';
RUN;
```

```
DATA BETTER;
   RETAIN SUBJECT 0;
   SUBJECT = SUBJECT + 1;
   INPUT SCORE1 SCORE2;
DATALINES;
3 4
5 6
7 8
;
PROC PRINT DATA=BETTER;
   TITLE1 'Correct Program';
RUN;
```

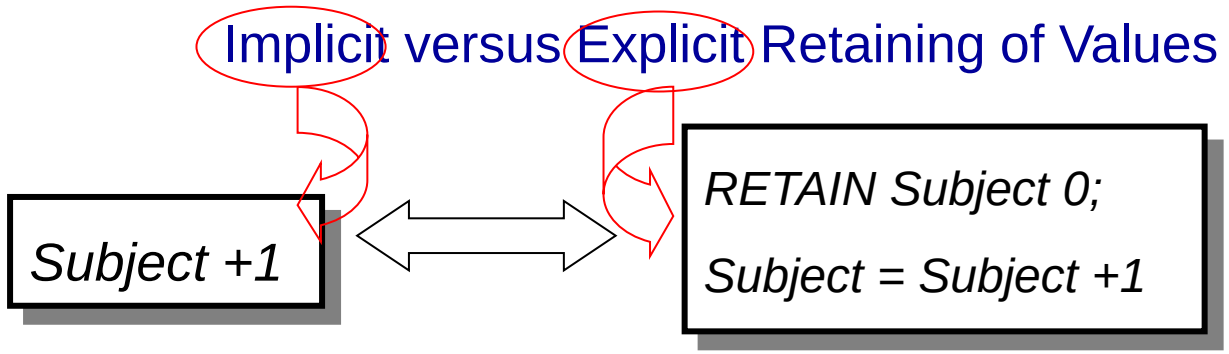
Incorrect Program

Obs	SUBJECT	SCORE1	SCORE2
1	.	3	4
2	.	5	6
3	.	7	8

Correct Program

Obs	SUBJECT	SCORE1	SCORE2
1	1	3	4
2	2	5	6
3	3	7	8

RETAIN: "Remembering" Values from Previous Observations




```
DATA BEST;
   SUBJECT + 1;
   INPUT SCORE1 SCORE2;
DATALINES;
3 4
5 6
7 8
;
PROC PRINT DATA=BEST;
   TITLE1 'Correct Program';
RUN;
```

Correct Program

Obs	SUBJECT	SCORE1	SCORE2
1	1	3	4
2	2	5	6
3	3	7	8

Temporary ARRAY

A temporary array does not actually refer to a list of variables at all! Instead, we can declare an array to be temporary and use the array elements in their subscripted form in the DATA step.

No real variables are created when you use a temporary array. You can also provide initial values for each of the array elements.

The most compelling reason to use temporary arrays is for efficiency. Also, you do not have to bother dropping useless variables

Renaming and Dropping

```
DATA grades2;
  SET grades2(drop=_name_ rename=(col1=grade));
  BY name;
  RETAIN week;
  IF first.name=1 THEN week=0;
    week=week+1;
  RUN;
PROC PRINT DATA = grades2;
  RUN;
```

SAS Trick: Use of retain and IF-THEN to get a repeating sequence of week numbers.

OBS	NAME	WEEK	GRADE
1	Amy	1	6
2	Amy	2	8
3	Amy	3	4
4	Amy	4	8
5	Amy	5	7
6	Amy	6	8
7	Bob	1	3
8	Bob	2	5
9	Bob	3	5
10	Bob	4	7
11	Bob	5	5
12	Bob	6	5
13	Carol	1	7
.....			

Making a Wide Table from a Long Table

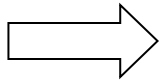
```
DATA OLD;
 INPUT SUBJECT TIME X;
 DATALINES;
 1 1 5
 1 2 6
 1 3 7
 2 1 8
 2 2 9
 2 3 4
 ;
RUN;
```

Restructuring a SAS Dataset: creating a single observation from multiple observations:

KEYWORDS: ARRAY, REATIN, SET, PROC SORT, FIRST.Byvar, LAST.Byvar

Old Dataset

Subject	Time	X
1	1	5
1	2	6
1	3	7
2	1	8
2	2	9
2	3	4



Desired New Dataset

Subject	X1	X2	X3
1	5	6	7
2	8	9	4

Making a Wide Table from a Long Table

```
PROC SORT DATA=OLD;
   BY SUBJECT TIME;
RUN;
DATA NEW;
   SET OLD;
   BY SUBJECT;
   ARRAY XX[*] X1-X3;
   RETAIN X1-X3;
   IF FIRST.SUBJECT = 1 THEN DO I = 1 TO 3;
      XX[I] = .;
      END;
   XX[TIME] = X;
   IF LAST.SUBJECT = 1 THEN OUTPUT;
   KEEP SUBJECT X1-X3;
RUN;

PROC PRINT DATA = NEW;
TITLE "make a wide table from a long table";
RUN;
```

Old Dataset

Subject	Time	X
1	1	5
1	2	6
1	3	7
2	1	8
2	2	9
2	3	4

Define an array XX[3] and RETAIN the value of all three variables: X1-X3.

Making a Wide Table from a Long Table

```
PROC SORT DATA=OLD;
   BY SUBJECT TIME;
RUN;
DATA NEW;
   SET OLD;
   BY SUBJECT;
   ARRAY XX[*] X1-X3;
   RETAIN X1-X3;
   IF FIRST.SUBJECT = 1 THEN DO I = 1 TO 3;
      XX[I] = .;
      END;
   XX[TIME] = X;
   IF LAST.SUBJECT = 1 THEN OUTPUT;
   KEEP SUBJECT X1-X3;
RUN;

PROC PRINT DATA = NEW;
TITLE "make a wide table from a long
RUN;
```

Old Dataset

Subject	Time	X
1	1	5
1	2	6
1	3	7
2	1	8
2	2	9
2	3	4

Initialize the array to missing (.). Although this initialization is not necessary here, if you are missing an observation for one or more times for a subject, you would retain the values from the previous subject If you did not initialize X1-X3 to missing. This initialization is done each time a new subject is read in.

Making a Wide Table from a Long Table

```
PROC SORT DATA=OLD;
    BY SUBJECT TIME;
RUN;
DATA NEW;
    SET OLD;
    BY SUBJECT;
    ARRAY XX[*] X1-X3;
    RETAIN X1-X3;
    IF FIRST.SUBJECT = 1 THEN DO I = 1 TO 3;
       XX[I] = .;
       END;
    XX[TIME] = X;
    IF LAST.SUBJECT = 1 THEN OUTPUT;
    KEEP SUBJECT X1-X3;
RUN;

PROC PRINT DATA = NEW;
TITLE "make a wide table from a long table";
RUN;
```

Old Dataset

Subject	Time	X
1	1	5
1	2	6
1	3	7
2	1	8
2	2	9
2	3	4

This statement tells the program to set the element of XX with a subscript equal to the current value of TIME, equal to the current value X. For the first observation, TIME = 1 and X =5. Since TIME = 1, XX[TIME] is the same as XX[1] which represents the variable X1 (set equal to 5, the value of X).

The next observation from OLD is then read (Subject =1, TIME =2, X=6, First.subject = 0, and Last.subject = 0.) Values of X1-X3 are Retained in the previous observation (X1 =5, X2 = ., X3 = .) and XX[2], or X2, is set equal to 6, the current value of X in OLD. This happens again for the next observation resulting in X1 = 5, X2 =6, and X3=7. This time, however, since Last.subject = 1 (is true), the observation Is output to NEW. The process then repeated For the next subject!

Making a Wide Table from a Long Table

```
PROC SORT DATA=OLD;
    BY SUBJECT TIME;
RUN;
DATA NEW;
    SET OLD;
    BY SUBJECT;
    ARRAY XX[*] X1-X3;
    RETAIN X1-X3;
    IF FIRST.SUBJECT = 1 THEN DO I = 1 TO 3;
       XX[I] = .;
       END;
    XX[TIME] = X;
    IF LAST.SUBJECT = 1 THEN OUTPUT;
    KEEP SUBJECT X1-X3;
RUN;

PROC PRINT DATA = NEW;
TITLE "make a wide table from a long table";
RUN;
```

Old Dataset

Subject	Time	X
1	1	5
1	2	6
1	3	7
2	1	8
2	2	9
2	3	4

make a wide table from a long table

Obs	SUBJECT	X1	X2	X3
1	1	5	6	7
2	2	8	9	4