```
/*********************************************
    Week 11:  SAS Loops and Reshaping Data
     Author:  Cheng Peng
        Date: 04/11/2021

   Topics: 1. Review of Do Block & DO Loops
           2. Concepts of SAS Arrays
           3. PROC TRANSPOSE
           4. Temporary Arrays
           5. Restructuring Tables
*********************************************/
OPTIONS PS = 65 LS = 78 NONUMBER NODATE;

/** Topic 1. More examples on DO-Loop

We have done examples of DO-block, DO-WHILE, and DO-UNTIL.
The following is an example of a DO block that uses the
explicit list of numeric or character values.         */

* Example 1;
DATA SALES;
INPUT units_sold warranty $ Laptopmodel $;
DATALINES;
21 1 AT3810
34 0 AT3600
12 1 AT3600
11 0 AT3810
;
RUN;

DATA revenue;
SET sales;
     IF Laptopmodel= "AT3600" THEN DO;
           IF warranty = 1 THEN revenue = units_sold*(1199.99 + 39);
           IF warranty = 0 THEN revenue = units_sold*1199.99;
     END;
RUN;
```

```
PROC PRINT DATA = revenue;
RUN;


* Example 2;
DATA DECK;
  FORMAT value $5. suit $8.;
  DO value='ace','2','3','4','5','6','7','8','9',
           '10','jack','queen','king';
    DO suit='spades','hearts','clubs','diamonds';
        OUTPUT;
    END;
 END;
RUN;

PROC PRINT DATA = DECK;
RUN;


/***********************************************************
          Topic 2: SAS ARRAY: Basics and Structure
************************************************************
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.                          **/


/** Working SAS data set: family income  **/
DATA faminc;
  INPUT famid faminc1-faminc12 ; /* short cut of patterned variable names */
DATALINES;
1 3281 3413 3114 2500 2700 3500 3114 3319 3514 1282 2434 2818
2 4042 3084 3108 3150 3800 3100 1531 2914 3819 4124 4274 4471
3 6015 6123 6113 6100 6100 6200 6186 6132 3123 4231 6039 6215
;
RUN;
```

```
/** Task: replacing each income < 3000 with a miss value.
          An inefficient approach - intuitive and understandable. **/
DATA recode_manual;
  SET faminc;
  IF faminc1 < 3000  THEN  faminc1=.;
  IF faminc2 < 3000  THEN  faminc2=.;
  IF faminc3 < 3000  THEN  faminc3=.;
  IF faminc4 < 3000  THEN  faminc4=.;
  IF faminc5 < 3000  THEN  faminc5=.;
  IF faminc6 < 3000  THEN  faminc6=.;
  IF faminc7 < 3000  THEN  faminc7=.;
  IF faminc8 < 3000  THEN  faminc8=.;
  IF faminc9 < 3000  THEN  faminc9=.;
  IF faminc10 < 3000 THEN  faminc10=.;
  IF faminc11 < 3000 THEN  faminc11=.;
  IF faminc12 < 3000 THEN  faminc12=.;
RUN;

/* NOTE: heading option specifies horizontal (H) column headings */
PROC PRINT DATA = recode_manual NOOBS HEADING = H;
TITLE "Processing with a regular data step";
RUN;


/* What if you 1000 variables in a dataset, it is inefficient, if not infeasible,
   to use the above data step to process the replacement in the data set. SAS array
   can be used to simplify the process!                                         */

DATA recode_array;
  SET faminc;
  ARRAY Afaminc[12] faminc1-faminc12;
      DO i = 1 TO 12;                /* loop index */
        IF Afaminc[i] < 3000 THEN Afaminc[i] = . ;
      END;
  DROP i;
RUN;

PROC PRINT DATA = recode_array NOOBS HEADING = H;
```

```
TITLE "Processing with an ARRAY in a data step";
RUN;


/*************************************************************
          Topic 3. Types of Automatic ARRAYS

*************************************************************;

/* Working Data Set */
DATA Sample_data;
INPUT x1 x2 x3 x4 $ x5 $;
DATALINES;
1 2 3 AA BB
2 3 4 AB CC
3 4 5 AC DD
4 5 6 AD EE
5 6 7 AE FF
6 7 8 AF GG
;
RUN;

/* Example 1. automatic numeric ARRAYs */
DATA NUM_ARRAY;
SET Sample_data;
* ARRAY NUMARRAY[*] x1-x3;      /* explicit list of variables             */
ARRAY NUMARRAY[*] _NUMERIC_;    /* implicit list of all numeric variables */
   DO i = 1 TO DIM(NUMARRAY);
      IF NUMARRAY[i] > 3 THEN NUMARRAY[i] =.;
   END;
DIMARRAY = DIM(NUMARRAY);
DROP I;
RUN;

PROC PRINT DATA = NUM_ARRAY;
RUN;

/* Example 2. automatic character ARRAYs */
```

```
DATA CHAR_ARRAY;
SET Sample_data;
ARRAY  CHARARRAY[*] _CHARACTER_;   /* implicit of character variables */
  DO i = 1 TO dim(CHARARRAY);
     CHARARRAY[i] = SUBSTR(CHARARRAY[i],1,1);  /* extract the first character */
  END;
DROP i;
RUN;

PROC PRINT DATA = CHAR_ARRAY;
RUN;

/** Example 3: temporary ARRAY:  _TEMPORARY_.
    A temporary array is an array that only exists for the duration
    of the data step where it is defined. A temporary array is useful
    for storing constant values, which are used in calculations. In a
    temporary array there are no corresponding variables to identify
    the array elements                                          **/
DATA TEMP_ARRAY;
SET Sample_data;
   ARRAY numvars[*] _NUMERIC_;                          /* The initial numeric variable */
   ARRAY newvars[*] px1-px3;                            /* The array of new numeric variables (place holder)
                                                           for the store the computed values               */
   ARRAY inival[3] _TEMPORARY_ (1.1, 1.2, 1.3);         /* This temporary ARRAY provides initial values    */
       DO i = 1 TO dim(numvars);                        /* loop index, i, will be dropped from the data    */
        newvars[i] = numvars[i] * inival[i];            /* valculate the values for the new variable       */
       END;
DROP i;
RUN;

PROC PRINT DATA = TEMP_ARRAY;
RUN;



/****************************************************
   Topic 4. Applications of SAS ARRAYs.
****************************************************/
```

```
/** Example 1: numeric array **/
DATA TIMES;
       INPUT TIME1 TIME2 TIME3 TIME4;
       DATALINES;
       22.3      25.3      28.2      30.6
       22.8      27.5      33.3      35.8
       18.5      26.0      29.0      27.9
       22.5      29.3      32.6      33.7
       ;
RUN;

/* Using ARRAY to perform a repetitive task: Some detailed steps on how to access ARRAY and complete
   each record of the new variable SUMTIME. */
DATA TIME_SUM;
SET TIMES;
    ARRAY TMARRAY[4] TIME1-TIME4;  /* Declare a numeric ARRAY to store the four TIME variables for bulk
processing */
      *ARRAY TMARRY[*] TIME1-TIME4;  /* This an alternative definition of ARRAY without specifying the
dimension */
    SUMTIME = 0;                   /* Initialize a new variable, SUMTIME, to start the loop     */
       DO I= 1 TO 4;               /* Use DO-LOOP to access the ARRAY TMARRAY                   */
          SUMTIME = SUMTIME + TMARRAY(I);   /* To complete each record of SUMTIME - DO-LOOP processes each
record of
                                                   data set TIME_SUM wiht new variable SUMTIME.
*/
            * OUTPUT; /* Obs   TIME1    TIME2     TIME3     TIME4     SUMTIME   I
                         1     22.3*     25.3      28.2      30.6       22.3*     1
                         2     22.3*     25.3*     28.2      30.6       47.6*     2
                         3     22.3*     25.3*     28.2*     30.6       75.8*     3
                         4     22.3*     25.3*     28.2*     30.6*     106.4*     4    */
      END;
      OUTPUT;  /* ==>       22.3*     25.3*     28.2*     30.6*     106.4*    5   */
        *DROP I;
RUN;

PROC PRINT DATA = TIME_SUM;
RUN;
```

```
/* Example 2. automatic numerical variable _NUMERIC_:

   Two variable lists, _NUMERIC_, and _CHARACTER_ are especially relevant for arrays.
   As the names imply, these variable lists reflect all of the numeric or character
   variables that are defined in the current DATA step when the variable list is
   specified. You can use these variable lists in the ARRAY statement, as shown here:
                         array my_nums[*] _numeric_;
                         array my_chars[*] _character_;
   These variable lists are especially helpful if you want your array to contain
   all of the numeric or character variables in a SAS data set referenced in a SAS
   a statement that reads the data set.

   It is important to emphasize that the variable list refers to all variables of
   that type that are previously defined in the DATA step, not just those that exist
   in a previously referenced data set.                                            */

DATA test;
INPUT A B C D E;
DATALINES;
1 . 1 0 1
0 . . 1 1
1 1 0 1 1
;
RUN;

DATA Replace_Missings;
  SET test;
    /* Use the _NUMERIC_ variable list to associate all of */
    /* the numeric variables with an array.                */
    ARRAY vars[*] _NUMERIC_;
    /* Loop through the array, changing all missing values to 0. */
    DO I = 1 TO dim(vars);
       IF vars[i] = . THEN vars[i] = 0; /* Bulk processing through the DO-LOOP */
    END;
DROP I;
RUN;
```

```
PROC PRINT DATA = Replace_Missings;;
RUN;


/***************************************************
 **************************************************
          Topic 5. Reshape Data Sets
***************************************************
****************************************************/

/**  Example 1.  Reshape the wide table using an ARRAY. Date variables are recorded
                 as formatted numerical variables                              **/
/* This working data set is a typical wide table */

DATA patient_visits;
INPUT patient_ID $ (visit1-visit4) (: mmddyy10.);  /* colon modifier specified format */
FORMAT visit1-visit4 mmddyy10.;
DATALINES;
Joe 01/05/2011 01/15/2011 01/25/2011 02/03/2011
Sam 01/07/2011 01/17/2011 01/27/2011 02/10/2011
Ron 01/09/2011 01/19/2011 01/29/2011 03/15/2011
Bob 01/11/2011 01/21/2011 01/31/2011 02/01/2011
;
RUN;

/**   converting the above wide table to a long table   **/
DATA LONG_TABLE;
SET patient_visits;
    /* Define an array to contain the visits. */
    ARRAY visit[4] visit1-visit4;  /* store the 4 variables VISIT1 - VISIT4 in the ARRAY. */
      *ARRAY visit[4] _NUMERIC_;     /* store the 4 variables VISIT1 - VISIT4 in the ARRAY. */
    /* Loop through the array, assigning each element (visit) */
    /* to the Date_of_Visit variable and then outputting it. */
          DO i=1 TO DIM(visit);       /* DIM() returns the number of variables */
             date_of_visit = visit[i];
             OUTPUT;
          END;
/* Format and drop variables, as desired. */
```

```
FORMAT date_of_visit mmddyy10.;
DROP visit1-visit4 i;
RUN;

PROC PRINT DATA = LONG_TABLE;
RUN;


/** Example 2. long table to wide table: We want to convert the LONG_TABLE to the original
               wide table: PATIENT_VISITS **/

/* Sort data set by Patient_ID and Date_of_Visit. */
PROC SORT DATA = LONG_TABLE;
BY patient_ID date_of_visit;
RUN;

/** Converting LONG_TABLE to the original wide table **/
DATA widetable;
 SET LONG_TABLE;
  BY patient_ID;   /* Sorting the dataset by patient_ID is crucial!!!! */
      /* Define an array for the new visit variables. */
      ARRAY visit[4] VISIT1-VISIT4;   /* place holder */
      /* Retain the variables that are associated with the array. */
      RETAIN visit;   /* retain the value of previous visit until hit the FIRST.var */
      /* Clear the visit variables and the counter for each new BY */
      /* group (Patient_ID).                                       */
      IF FIRST.patient_ID THEN
            DO I = 1 TO DIM(VISIT);
                  *VISIT[I] = .;
                      counter = 0;
                END;
      /* Increment the counter that is used to reference the element */
      /* of array to assign date. */
      counter+ 1;
      /* Assign the date to the proper element of the array. */
      visit[counter] = date_of_visit;
      /* Output one observation per BY group (Patient_ID). */
      IF LAST.patient_ID THEN OUTPUT;   /* important, output a complete records
```

```
                                                  and then prepare the next record   */
      /* Format and drop variables, as desired. */
      FORMAT visit : mmddyy10.;
DROP date_of_visit I counter;
RUN;

PROC PRINT DATA = widetable;
RUN;


/**************************************************/
/** Example 3: tranpose wide table to long table **/
/**************************************************/
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
;
RUN;

/* SORT the dat by NAME since will process the records by names
   and */
PROC SORT DATA =grades1;
  BY name;
  RUN;

PROC TRANSPOSE DATA = grades1
               OUT = grades2;
  BY name;
  VAR hw1-hw6;
  RUN;

  PROC PRINT DATA = grades2;
  RUN;
```

```
DATA LONG_GRADES;
SET grades2;
HW = _NAME_;
GRADE = COL1;
DROP _NAME_ COL1;
RUN;


PROC PRINT DATA = LONG_GRADES;
RUN;

/****************************************************/
/** Example 4: Transform longtable to wide table  **/
/****************************************************/
PROC TRANSPOSE  DATA = long_grades
                OUT = wide_grades (DROP = _NAME_);
    BY NAME ;
    ID HW;
    VAR GRADE;
RUN;

PROC PRINT DATA = wide_grades ;
RUN;

/************************************************************************
                          CONCLUDING REMARK
BOTH PROC TRANSPOSE and ARRAY are equally convenient for reshaping
data with two variables. However, if the data set has more than two
variables, Using PROC TRANSPOSE to TRANSPOSE data can be very tedious.
************************************************************************/
```