MWSUG 2017 - Paper TT09

An Array of Possibilities: Manipulating Longitudinal Survey Data with Arrays

Lakhpreet "Preeti" Gill, Mathematica Policy Research

ABSTRACT

SAS® arrays are well-suited for handling longitudinal survey data, which are data collected at multiple time points for a sampled population. Depending on the research question and selected sample, the study period for observing respondents may vary based on when they entered and exited the survey. For example, a researcher may only want to add respondents to the study when they reach a certain age, and, in contrast to a cross-sectional study, this will occur at different waves of the survey. In this paper, we will use the Health and Retirement Study to identify differing baseline and follow-up measurements conditioned on age and response status, and learn different methods for leveraging SAS arrays to manipulate the data.

INTRODUCTION

SAS processes data observation by observation, which makes it well-suited for working with longitudinal data. These data typically take a "wide" file structure, with variables being added onto the dataset to record new events (maintaining one observation per unit of analysis) instead of observations being added to the dataset to record new events (creating multiple observations per unit of analysis).

With surveys that are longitudinal, arrays allow the programmer to identify relevant variables that are associated with varying time points (survey "waves"), select the relevant wave variable for a respondent (that could be different for the next respondent), and perform data manipulations (accounting for differences among respondents' waves).

The code in this paper relies on data from the Health and Retirement Study (HRS), which is an ongoing panel study of pre-retirement and retirement adults ages 50 and older. The HRS started in 1992 and interviews respondents every two years. It provides a wealth of information on the health, well-being and financial solvency of an aging population. The publically available survey can be linked to a number of other ancillary datasets, such as restricted access Social Security Disability Insurance (SSDI) claims history or publically available genetic risk scores. Data and resources can be found here: https://hrs.isr.umich.edu/.

Let's create a hypothetical research scenario to guide examples for the paper. In this scenario, a researcher wants to evaluate the effects of being denied SSDI through retirement. Does the applicant chose to take early retirement or wait until reaching the full retirement age ("FRA")? The baseline years will run from Waves 1 (1992) to 7 (2004) and are based on when a respondent is between ages 58 and 60. The analysis years will run from Waves 4 (1998) to 11 (2012) and will be based on when a respondent reaches the FRA. By creating analytic variables for these two measurement time points, this paper will demonstrate how to:

- create and use an array index that varies for respondents
- assign variables and troubleshoot "out of range" cases
- use an index to create and populate time series variables

CREATING AN ARRAY INDEX

The first necessary step to using an array is having an index. A common form of an index is the variable assigned to a start and stop value for a DO LOOP. Data are processed by iterating through the loop, with each iteration instructing SAS on the array element to consider.

Here is a basic example:

```
ARRAY VARLIST (10) VAR1-VAR10;
DO I = 1 TO 10;
    IF VARLIST{I} = . THEN VARLIST{I) = 0;
```

```
END;
```

The array *reference* is **VARLIST**, the array *subscript* that defines the range of elements in the array is **10,** the array *elements* are **VAR1** to **VAR10**, and the *index* is the variable **I,** which is used in the array statement to call an element from the array. The index increments by 1 for each iteration of the 10 loops, then resets to 1 at the next observation.

In this case, a repeated action is performed on all variables that have a missing value. But what we want is an action performed once on only one variable specific to a respondent's baseline or analytic conditions.

DYNAMICALLY GENERATE THE INDEX

The index for processing survey data needs to select the wave variable for a respondent, and so in this case, the index value should be particular to each respondent. Once generated, the index can be used in an iterative DO LOOP, but the primary purpose is to create variables using a simple assignment statement.

Below is an example to help visualize this. We see a wide file in which a respondent's age over the years is captured in enumerated wave variables. Imagine that the data set continues to extend rightward, with a series of wave variables for different measures. We want to use an array to identify the wave in which the measurement conditions for baseline and analysis occur, and then use that wave to identify other analysis variables going forward. So considering a baseline condition of being between ages 58 and 60, Subject 1a reaches baseline in Wave 5 while Subject 3c reaches baseline in Wave 1.

	Wave 1	*Wave 2*	*Wave 3*	*Wave 4*	*Wave 5*	*Wave 6*	*Wave 7 …*
SUBJECT_ID	**RELAGE_1**	**RELAGE_2**	**RELAGE_3**	**RELAGE_4**	**RELAGE_5**	**RELAGE_6**	**RELAGE_7…**
1a	51	53	55	57	*59*	61	63
2b	48	50	52	54	56	58	60
3c	*58*	61	63	66	68	70	.

Raw data example of relative age for three respondents from Wave 1 to Wave 7

Putting this in practice, the following code dynamically generates an index using a DO LOOP to identify the wave in which baseline conditions are met for a valid age and interview status combination:

Code 1

```
ARRAY RELAGE(11) RELAGE_1  - RELAGE_11; ①
ARRAY STATUS(11) RIWSTAT_1 - RIWSTAT_11;

DO K = 1 TO 11; ①
    IF NOT MISSING(B_INDX) THEN LEAVE; ②
   IF 58 <= RELAGE{K}<= 60 AND STATUS{K} = 1 THEN B_INDX= K; ③
END;
```

①The suffix for each variable in the array represents a wave. The index, K, for the DO LOOP therefore maps to the 11 waves of the survey.

② The baseline index variable, B_INDX, is initialized. At the first iteration of the loop, it will be missing. But this may not be true for later iterations, and a LEAVE statement is used to exit the DO LOOP at the first encounter of a true condition to avoid overwriting with a later wave's information that could also be true. (Recall that the survey is biennial, but the baseline condition spans 3 possible years.)

③ The two baseline conditions of age and interview status are checked at each wave, and if true, the variable B_INDX is set to the DO LOOP index, K.

With the index created, records that don't have a value for B_INDX can be dropped. Also, note how the hypothetical states that the baseline years run up until Wave 7. We didn't need to have the range extend to Wave 11, but it can serve as data validation step to catch any cases that fall outside of the expected range for further examination.

Now looking to generate the index for the analysis measurement period, we see the wave range does not start at 1. How do we approach the algorithm? We can let the array cycle through all 11 waves as we did above, or we can simply alter the array subscript and reduce processing time:

Code 2.

```
ARRAY RELAGE(4:11) RELAGE_4  - RELAGE_11; ①
ARRAY STATUS(4:11) RIWSTAT_4 - RIWSTAT_11;

DO J = 4 TO 11; ②
    IF NOT MISSING(A_INDX) THEN LEAVE;
   IF 65 <= RELAGE{J}<= 666 AND STATUS{J} = 1 THEN A_INDX= J;
END;
```

The array subscript ① identifies the elements as, effectively, Waves 4 to 11 and the DO LOOP index, J, ② restricts the variable, A_INDX, from being populated with values less than 4. If, for example, the array index started at 1, the programmer would experience an "out of bounds" error, since the array elements are not identified with that value. We'll look at cases for how to handle this error later in the paper.

USING AN ARRAY INDEX

Now that we have our indexes, B_INDX and A_INDX, assigning analytic variables will be a snap. We'll do three things: ① Assign a series of dummies using binary logic, ② account for missingness, and ③ create a variable that can be used to check the dummy values against the original variable.

Code 3

```
ARRAY MSTAT   (11)RMSTAT_1  - RMSTAT_11;

B_MSTATMAR = MSTAT{B_INDX}  IN (1,2,3); ①
B_MSTATDIV = MSTAT{B_INDX} IN (4,5,6);
B_MSTATWID = MSTAT{B_INDX} = 7;
B_MSTATNEV = MSTAT{B_INDX} = 8;
IF MISSING (MSTAT{B_INDX}) THEN CALL MISSING(OF B_MSTAT:); ②
_MSTATCHK = MSTAT{B_INDX }; ③
```

The code groups the different values for marital status into four analytic dummy variables. A simple assignment statement to the array reference is used and binary logic sets each variable equal to 1 or 0. If the input raw variable is missing, however, all the dummies should be set to missing, and this is accomplished using the CALL MISSING() routine. It's good practice to check all coded variables, and the final variable take the raw information from each respondent's wave, and assigns that to a check variable.

Looking to the raw input data and comparing to the output data set, we can see how things are mapped from the raw data to the processed data. Marital information from each respondent's baseline wave (B_INDX) are translated into the analytic dummies, and check variable.

B_INDX	RMSTAT_1	RMSTAT_2	RMSTAT_3	RMSTAT_4
3	.	.	1	1
4	.	2	2	5
7	.	.	.	.

Input 1. Raw input data for three respondents

B_INDX	B_MSTATMAR	B_MSTATDIV	B_MSTATWID	B_MSTATNEV	B_MSTATCHK
3	1	0	0	0	1
4	0	1	0	0	5
7	.	.	.	.	.

Output 1. Processed data for three respondents

CALCULATIONS WITHIN THE ARRAY CALL

If we want to look to the next wave, we can perform a calculation on the index as follows:

Code 4

```
ARRAY STATUS (11) R_RIWSTAT_1 - R_RIWSTAT_11;
B_STAT       = STATUS{B_INDX};
B_STATNXT    = STATUS{B_INDX + 1};①
```

The assignment statement performs a calculation ① on the index to advance to the wave after the baseline wave.

OUT OF RANGE SCENARIOS

If the log returns ERROR: ARRAY SUBSCRIPT OUT OF RANGE, something is not syncing between the index and whatever is specified in the array subscript. Typically, the out of range error is triggered when the index is too small or large for the range specified by the subscript, or the index value is missing.

Conditional logic

Manipulating the array index can take the array out of bounds. Consider the scenario where a respondent's analytic wave occurred in wave 11. If we were to check the status for the next wave, well, there wouldn't be one! The survey only has 11 waves.

Correct by conditioning out those cases:

Code 5

```
IF A_INDX NE 11 THEN A_STATNXT = STATUS{A_INDX + 1};
```

Hard code the index

Sometimes a survey question wasn't in place for the duration of the survey history or is constructed differently at a particular wave. The Activities of Daily Living variable series from the HRS are an example of this. The question from Wave 1 was constructed differently from later waves. One solution is to adopt the Wave 2 values for respondents who reach baseline in Wave 1:

Code 6

```
ARRAY ADLA (2:11) RADLA_2 - RADLA_11 ; ①

IF B_INDX NE 1 THEN DO; ②
```

```
    B_ADLA = ADLA{B_INDX} > 0;
    IF MISSING (ADLA{B_INDX}) THEN B_ADLA = .;
    _ADLABCHK = ADLA{B_INDX};
END;
ELSE IF B_INDX EQ 1 THEN DO; ③
    B_ADLA = ADLA{2} > 0; ④
    IF MISSING (ADLA{2}) THEN B_ADLA = .;
    _ADLAB1CHK = ADLA{2};
END;
```

First, the subscript that restricts the range to from 2 to 11 ① so that it can be tracked with the waves in the index. Next, there are two forms of conditional logic. If the index is greater than one ②, the data are processed as usual. If the index equals one ③, however, the data are processed differently. Because the researcher made a decision to use Wave 2 if a respondent has a baseline in Wave 1, we can hard-code ④ the value of "2" directly into the array statement to assign the Wave 2 value.

ADVANCED USES OF ARRAYS

We have the basics down, so let's look at two advanced examples of using arrays to get a sense of what else is possible. Remember that arrays can serve as a temporary grouping of existing variables, or they can create new variables. We'll employ both tactics in the next examples.

ALIGN SUBSCRIPTS, INDEXES *AND* VARIABLE VALUES

In this example, we want to create dummy variables to control for time period effects based on when the analysis occurred. The variable ANALYSIS_YR captures the year value for a respondent's wave of analysis. To create the dummies, we align the array statement and index to the values of the variable ANALYSIS_YR:

Code 7

```
ARRAY CY (1997:2012) CY_1997 - CY_2012; ①
DO I = 1997 TO 2012; ②
    IF ANALYSIS_YR = I THEN CY{I} = 1; ③
    ELSE CY{I} = 0;
END;
```

The first step creates the variables and sets the subscript to the range of values ① in ANALYSIS_YR. The index is also synced ② to increment through the range of values in ANALYSIS_YR. Because of this, we are able to use an assignment statement that evaluates when ANALYSIS_YR equals the index. When this is true, the index populates the corresponding array element with a 1 ③, otherwise, the array element receives a 0.

Output 2 shows an example crosstab of the ANALYSIS_YR values against a subset of the newly created dummy variables:

ANALYSIS_YR	CY_1997	CY_1998	CY_1999	CY_2000
1997	1	0	0	0
1998	0	1	0	0
1999	0	0	1	0
2000	0	0	0	1

Output 2. Crosstab from using a variable value to guide the array subscript and index

CREATE TIME SERIES VARIABLES

Arrays can be one method to create time series variables. In this example we want a series of variables to capture household income between baseline and analysis. We need to use arrays of two different lengths, and will therefore have two different indexes to increment.

Code 8

```
ARRAY HITOT (11)HHITOT_1 - HHITOT_11; ①

ARRAY HHINC_SERIES (6); ②

    J = 1; ②

    DO I = B_INDX TO A_INDX ; ①

          HHINC_SERIES {J} = HITOT {I}; ③

          J + 1; ②

    END;
```

①The array reference HITOT uses the index I which increments from the baseline wave value (B_INDX) to the analysis wave value (A_INDX).

②The array reference HHINC_SERIES creates 6 variables, HHINC_SERIES1 – HHINC_SERIES6. It uses the array index J, which needs to be incremented by one at each iteration of the loop.

③The count of unique values between the start and stop values of I and J are the same, but the elements it corresponds to in an array are different. This is why the assignment statement works.

The input example below shows how the raw data appeared, and the output example shows how this translated to the time series variables.

B_INDX	A_INDX	HHITOT_3	HHITOT_4	HHITOT_5	HHITOT_6
4	6	9,500	10,000	15,000	20,000
3	4	44,000	42,000	40,000	38,000

Input 3. Raw data for household income

B_INDX	A_INDX	HHINC_SERIES1	HHINC_SERIES2	HHINC_SERIES3	HHINC_SERIES4-HHINC_SERIES6
4	6	10,000	15,000	20,000	.
3	4	44,000	42,000		.

Output 3. Processed time series data for household income between baseline and analysis.

CONCLUSION

In this paper, we learned about different ways to create indexes and employ them as:

- A variable (e.g. B_INDX)
- A calculation (e.g. B_INDX + 1)
- A hardcoded value (e.g., Wave = 2)
- A DO LOOP start and stop value (e.g., K = B_INDX to A_INDX)
- A tool for creating and populating time series variables

We also learned how to troubleshoot for out of bounds cases by using conditional logic for the array index, and changing the range for the array subscript. There are many more ways to use arrays for working with longitudinal data, but these key processing steps should set a strong foundation for growing a programmer's took kit.

REFERENCES

Kuligowski, Andrew T. and Mendez, Lisa (2016). "An Introduction to SAS® Arrays". *Proceedings of the SAS Global Forum 2016 Conference.* Cary, NC: SAS Institute, Inc.

Wright, Wendi (2005). "Loop-Do-Loop Around Arrays". *Proceedings of the Northeast SAS User Group 2005 Conference.* Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (2011), "Using Arrays in SAS® Programming". Cary, NC: SAS Institute, Inc. https://support.sas.com/resources/papers/97529_Using_Arrays_in_SAS_Programming.pdf

ACKNOWLEDGMENTS

The author thanks Kerianne Hourihan, Dean Miller, Ken Peckham, and Scott Greiner of Mathematica for their technical review, the Michigan SAS Users Group for giving feedback on a first draft of the presentation, and Jody Schimmel Hyde, Laura Blue and April Wu for guiding research that used the HRS data. The author also thanks Christopher Bost of MDRC for instilling in her a lasting love for SAS.

CONTACT INFORMATION

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

Lakhpreet "Preeti" Gill, Senior Programmer Analyst
Mathematica Policy Research
220 East Huron Street
Ann Arbor, MI
48104
pgill@mathematica-mpr.com
www.mathematica-mpr.com

.