**SCRIPT: Restructuring SAS Data Sets**

Recorded in: Camtasia 2.8.1 for Mac

Edit in: Camtasia 8.4.1 for Windows

Record: Full Screen

System audio: 15%

|  |  |
| --- | --- |
| **Action on Screen** | **Narration** |
| S1  Intro |  |
| S2 | Up until this module, all the data sets we’ve used so far have shared a common structure. They’ve all had a wide, or person-level, structure.  When you have person-level data, each person has one observation and multiple variables contain data for each measurement.  As a side note, your observational unit is usually a person, but could also be a school, a state, or an air quality meter. It’s the entity from which you are gathering data. |
| S3 | Here’s an example of person-level, or wide, data. This table contains the weight in pounds for eight babies at ages 3 months, 6 months, 9 months, and 12 months.  Notice that each baby has one, and only one, observation, and that a new variable is created for each subsequent measure of weight. |
| S4 | An alternative, and often preferable, data structure is the person-period, or long, data structure. In the person-period data structure each person has multiple observations – one for each measurement occasion. |
| S5 | Here we look at the baby weights again. In the interest of saving space, we’re only looking at the first two babies from the data.  On the left is the person-level data.  On the right is the person-period data. Notice that now each baby has multiple observations – one for each weight measurement. Also, notice that there is a new variable in the person-period data to explicitly denote time.  Finally, I want to teach you a couple of terms that apply to person-period data – time-varying and time-invariant variables. In this data Gender is time-invariant. It remains constant over all 4 measurement occasions for each baby. Not only that, but for all intents and purposes it isn’t *allowed* to change.  Weight, on the other hand, is time-varying. The weight values change over time. And not only do they change, the amount, rate, and/or shape of their change is probably precisely what this researcher is interested in. |
| S6 | Earlier I said that person-period data is often preferable when working with longitudinal or clustered data. A full discussion of the reasons why is beyond the scope of this class. However, here I give 4 bullets on the disadvantages of person-level data, and highly recommend the Singer and Willet textbook for those of you who are interested in a more complete treatment on analyzing longitudinal or clustered data.  Additionally, the purpose of this video is not to teach you to analyze longitudinal data, only to manage it by converting back and forth between the two data structures we’ve been discussing. |
| S7  Demonstration | Before going any further, it may be helpful for you to see a worked example. |
| Open on m7.5\_lecture\_demo  \_Restructuring | At the top of this program we are just creating and viewing and temporary SAS data set that contains the same baby weight data you previously saw. |
| Show DATA step | Here we are using a DATA step to restructure this data. It looks a little bit complicated, but there isn’t anything here you haven’t already seen. First, I’m going to run it just to show you that it works. Then we’ll walk through how to build it together step by step. |
| Show log | As you can see in the log, the new SAS data set we just created has 32 total observations (4 per baby) and 4 variables. |
| Show list report | And the list report looks exactly as we would expect it too. Each baby has four observations. There is an explicit time variable – month. And, the weight variable contains the value of each baby’s weight in pounds at 3, 6, 9, and 12 months of age.  Now let’s take the DATA step apart to show you how we got to our result. |
| Step 1 | In this DATA step we are inputting the temporary SAS data set called baby\_weight using a SET statement.  We are using it to create the new temporary SAS data set called baby\_weight\_long1.  Here we are creating an ARRAY named weights, which contains four elements – the variables weight3, weight6, weight9, and weight12. Here we use the double hyphen syntax to select our variables. The double hyphen tells SAS “all variables that are located between weight3 and weight12 – regardless of what the variable is named.  This is in contrast to the single hyphen form of the variable list, which would tell SAS “all variables that start with ‘weight’ and end with a number, starting at weight3 and ending at weight12 – regardless of their location in the data set. So, that would include weight3, weight4, weight5, etc.  Next, we have a DO loop that will execute once for each “i” when i is between 1 and 4.  Inside the DO group is an assignment statement that creates a new variable called weight and sets it equal to the ith element of the ARRAY called weights.  Let’s take a look at what this program gives us. |
| Show list report | Not quite what we want. Let’s try to understand why. |
| S8 | So, for the first observation (id = 1001) SAS reads the values for all existing variables into the PDV. Then SAS sets the value of i to 1. Because 1 is between 1 and 4, SAS executes the assignment statement inside the DO group. This assignment statement assigns a value to a new variable called weight. The value of weight is equal to the value of the ith element in the ARRAY called weights. In this iteration it’s equal to the first element in the ARRAY because i is equal to 1. And, the first element in the ARRAY is the variable weight3. |
| S9 | So, in this iteration, the value of the new variable weight is 9. |
| S10 | Next, SAS hits the END statement, so it loops back up to the top of the DO group and increments the value of i by 1. Because 2 is between 1 and 4, SAS executes the assignment statement inside the DO group. Remember that the value of weight is equal to the value of the ith element in the ARRAY called weights. In this iteration it’s equal to the second element in the ARRAY because i is equal to 2. And, the second element in the ARRAY is the variable weight6. |
| S11 | So, the value of weight changes to 13. |
| S12 | Again, SAS hits the END statement and loops back up to the top of the DO group where it increments the value of i by 1. Because 3 is between 1 and 4, SAS executes the assignment statement inside the DO group. In this iteration it’s equal to the third element in the ARRAY because i is equal to 3. And, the third element in the ARRAY is the variable weight9. |
| S13 | So, the value of weight changes to 16. |
| S14 | Again, SAS hits the END statement and loops back up to the top of the DO group where it increments the value of i by 1. Because 4 is between 1 and 4, SAS executes the assignment statement inside the DO group. In this iteration it’s equal to the fourth element in the ARRAY because i is equal to 4. And, the fourth element in the ARRAY is the variable weight12. |
| S15 | So, the value of weight changes to 17. |
| S16 | And again, SAS hits the END statement and loops back up to the top of the DO group where it increments the value of i by 1. Because 5 is not between 1 and 4, SAS exits the DO loop and moves on to the next part of the DATA step. In this case, all that is left in the DATA step is the implicit OUTPUT statement. |
| S17 | So, SAS outputs the contents of the PDV to the new data set called baby\_weight\_long1. |
| S18 | As always, SAS then reads in the next observation (id = 1002) into the PDV. SAS moves through the DATA step in identical fashion to what we just saw. |
| S19 | And outputs the contents of the PDV to the baby\_weight\_long1. |
| S20 | Finally, SAS repeats this entire process for every observation in the data set.  So, what do we need to change about this data set in order to get the structure we want? |
| SAS code  (overlay a screenshot of the data) | First, we want SAS to output the contents of the PDV at the end of each iteration of i for each observation in the data set. We do that with the OUTPUT statement shown here.  Second, we need to create a time variable. We’ll create a variable here called month using an assignment statement and the ARRAY called months – shown here. This ARRAY statement is slightly different than those we’ve already seen. It has the keyword ARRAY, the ARRAY name, and the ARRAY dimension. But, instead of having variables as elements, it has constants – in this case the numeric constants 3, 6, 9, and 12.  Finally, we’ll add the KEEP option to the DATA statement so that our new data set only includes the variables we are interested in.  Here’s one more look at the resulting list report.  In my experience you will more commonly restructure data sets from wide to long, than vice versa. But, in the interest of being thorough I will also demonstrate how to restructure SAS data sets from long to wide. Additionally, I will do so using the TRANSPOSE procedure (an alternative to restructuring using a DATA step). |
| S21 | In the most simplistic terms PROC TRANSPOSE turns columns into rows and rows into columns.  Here is the general form of the TRANSPOSE procedure. You use a DATA statement to identify the input data set. You can use an OUT statement to name a new data set to be created by PROC TRANSPOSE. All other statements shown here are options that provide you with greater control over your output. |
| S22 | Watch this demonstration on using PROC TRANSPOSE. |
| SAS Code | We start by using the SORT procedure to sort our data first by id, and then within id, by month.  Next we’ll take a look at using the TRANSPOSE procedure with no options. |
| Run it | Ok, so take a look at this output for a little bit. All SAS has done is basically turn the data set on it’s side. The id column from above is now the first row. The month column from above is now the second row. And the weight column from above is now the third row. Also, notice that SAS had to create some generic names for all of these “new” columns. Let’s see if we can improve our results by playing with some of the available options. |
| SAS Code | First, let’s add a BY statement. The BY statement tells SAS to group our results by the BY variable – in this case id. That should group each of our variables by id. |
| Run it | And that gets us closer already. We now have one row with the four month values for each id, and a second row with the four weight values for each id. But we just want one row per id that contains the weight values, and the variable names to tell us what month each value is from. |
| SAS code | Next, let’s add the ID statement, and tell SAS to use the values of the month variable to name each column. |
| Run it | Getting closer. Instead of having a row for the variable month, the values of the variable month are now used as the column headers. SAS adds the underscores by default because of the naming rules for SAS variable names. However, we can still clean this up a bit. |
| SAS code | In this final iteration of our PROC TRANSPOSE step, we can add a PREFIX equals option to tell SAS to use the prefix “weight” instead of an underscore in our variable names. Finally, we have no use for the \_NAME\_ variable. Let’s drop it using a DROP equals option after our OUT equals data set name. Notice that the first three lines of this PROC TRANSPOSE step are all contained in a single PROC statement. Let’s run it and take a look at the list report. |
| Run it | There. That is exactly the results we wanted. We have now restructured the baby weight data from wide to long and back to wide. |
| S23 | Now it’s your turn to restructure a data set. |
| S24  References | This completes the video on restructuring data sets. In this video you learned about two basic data structures: person-level (or wide) data, and person-period (or long) data. You also learned two methods for restructuring data in SAS: PROC TRANSPOSE and iterative processing in a DATA step. We used the DATA step to restructure the data from wide to long, and PROC TRANSPOSE to restructure the data from long to wide. But you should know that both methods work for both types of restructuring. PROC TRANSPOSE tends to require less complicated programing, but using a DATA step allows you greater control and flexibility. I encourage you to practice both. |