**SCRIPT: Reading In Other Common Types of Raw Data**

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 | To read in .csv and tab-delimited files in SAS, we use a variation of the same program we just saw. We just need to tell SAS that now the values in the data will be separated by commas or tabs instead of a single space. |
| S2 | In this program, you can see that the only difference is the addition of the “dsd” option to the INFILE statement. |
| S3 | DSD stands for delimiter sensitive data. Adding this option serves three different purposes at once. First, it changes the default delimiter from a space to a comma. Second, if there are two delimiters in a row it assumes there is a missing value between them. Finally, if character values are placed in quotes, the quotes are stripped when read into SAS. |
| S4 | Yet another common way for data to be stored is in a text file with values separated by a tab instead of a single space. |
| S5  (Add white squares over 09 and x until the end of the question). | Using the dlm=, or delimiter=, option in the INFILE statement will allow you to designate almost any symbol as the delimiter separating values in the raw data.  **Question:** How would you tell SAS that the delimiter is a tab? If you can’t guess don’t worry, I never would have guessed either. But, as you can see it’s a pair of single quotes, filled with the number 09, and followed by an x. Apparently this is the hexadecimal equivalent of a tab in the American Standard Code for Information Exchange. But don’t worry about that, just save it somewhere. The raw data sets: comma, comma 2, and tab are posted on the course website if you would like to experiment with them. |
| S6 | Before moving on, recall that in the last demonstration the values for BMI\_Cat were truncated. That’s because SAS creates each variable with a length of 8 bytes by default. The values for BMI\_Cat are longer than 8 characters in the raw data set, so they are truncated in the SAS data set. But, you want to alter your DATA step to specify the correct length for the values in the variable BMI\_Cat. Fittingly you will do this with a LENGTH statement. |
| S7 | Now watch this demonstration, and learn how to use the LENGTH statement. |
| Open editor window on program titled, w3\_lecture\_demo\_Importing Raw Data.  Scroll down to example code with length statement. | Take a look at this data step. It is exactly the same as the data step you previously used to read-in raw height and weight data that was delimited with a single space. This time you have added a LENGTH statement to your program. First, you specify the keyword LENGTH. Then you specify the variable name, a dollar sign if it is a character variable, and then the length. Notice that you can specify multiple variables in one LENGTH statement, and you can set the length to be longer or shorter than 8 bytes. In general, there is no need to ever set the length for numeric variables because SAS is able to fit any numeric variable into 8 bytes, regardless of the number of digits it contains. Let’s submit this data step, and view the data portion of the new SAS data set using the PRINT procedure. |
| Run the data step and the proc print. Open the results viewer window. | You should immediately notice two things here. First, the LENGTH statement did what we wanted it to do. The variable id is now 3 characters in length and the variable BMI\_Cat is 13 characters in length and we are able to see the entire value for each participant’s BMI category.  The second thing you should notice is that BMI\_Cat is now the second variable in your data set, instead of the last variable in your data set.  **Question: Why is BMI\_Cat now the second variable in the data set?** |
| Go back to the code | The answer is in your SAS program. Remember that SAS reads a data step from left to right, top to bottom. Also remember that the descriptor portion of the new SAS data set is created at the end of the compilation phase. Finally, remember that SAS initializes the variable names and attributes in the Program Data Vector the first time it encounters them in the DATA step. In our previous examples, SAS encountered your variables for the first time in the INPUT statement.  **Now where does SAS encounter variables for the first time?**  Now SAS encounters variables for the first time in the LENGTH statement. So, when SAS is initializing the PDV it first creates the variables id and BMI\_Cat as character variables with a modified length that you specified. Then it drops down to the INPUT statement and continues creating variables in the PDV, ignoring id and BMI\_Cat because they have already been created. Although it may seem like it now, these details about what’s going on behind the scenes are important. For example, if you had put the LENGTH statement after the INPUT statement, id and BMI\_Cat would both be 8 bytes. By the time SAS read the LENGTH statement, the vairables id and BMI\_Cat would have already been initialized in the PDV with default lengths of 8 bytes. |
| S8 | Now you try using the length statement. |
| S9 | I’m probably sent data in excel files more than any other file format. Although the code we have learned so far cannot be used to import an excel file, it’s usually not a problem. Base SAS and SAS Enterprise Guide both have a really easy to use point and click import wizard for getting excel spreadsheets into SAS, which you will learn soon. To my knowledge SAS Studio does not yet have the import wizard. So, if you want to use SAS Studio you will have to import excel files in some other way.  One option is simply to open excel and save the file as a .csv file. Then you can read the data into SAS using the methods we’ve already discussed. |
| S10 | However, another pretty versatile procedure that can be used to directly import excel files is PROC IMPORT. Let’s walk through the IMPORT procedure and see how it works. |
|  | You’re going to start with the PROC statement and the keyword IMPORT. Notice that the PROC statement is typed over 3 lines, but it is all one statement because there is only one semicolon. Here you’re telling SAS that the output that should result from this procedure is a data set called ht\_wt5. The datafile you want to import is located here.  And this dbms= xls specifies that the type of data you are importing is an excel spreadsheet. The REPLACE option tells SAS to replace this file if it already exists. Then the SHEET option tells SAS which sheet within the Excel Workbook you want to import. GETNAMES=yes tells SAS that the first line in the spreadsheet contains the variable names you want to use in your new SAS data set. And finally, MIXED=yes tells says that the spreadsheet includes both numeric and character variables. Let’s run it and view the data portion of ht\_wt5. |
| Open the results viewer window | As you can see, the IMPORT procedure worked. However, you can also import the data using the SAS Import Wizard. To use the Import Wizard you click the File drop-down menu, and then import data. On the first screen of the Import Wizard you will select the type of file you want to import. Here we leave it on Microsoft Excel Workbook, and click next. |
| Click next | On the next screen you will click the browse button, point SAS to the excel workbook you want to import, and then click OK. |
| Point to file and click next | On the next screen you will select the worksheet you want to import. Typically, you can just click next without making any changes here. |
| Click next | Then you are going to provide SAS with a name for the new SAS data set you want to create, as well as the library you want to create it in. In this case you will name it ht\_wt5 and create in the work library just as you did when you typed the IMPORT procedure. And finally, you will click finish. |
| Type ht\_wt5 and click finish. Open the view table window | As you can see, SAS’s Import Wizard is a handy tool for helping you convert data stored in Microsoft Excel spreadsheets into SAS data sets. |
| S11 | Now you try using the Import Wizard. |
| S12 | Ok, so before you can ever analyze data or create any sort of report using SAS, you first have to access the data in SAS. You now know how to build SAS datasets manually by entering the data directly into SAS, and you know how to read in data stored in some of the most common file types. This includes both character and numeric data. Let’s go ahead and add one more little layer of complexity. What about files that include nonstandard data like dates and currency?  As you can see, this excel spreadsheet contains the same height and weight data that we’ve been using, but two new variables have been added. The first is the participant’s date of birth, and the second is the participant’s annual household income. |
| Show SAS code. Add a “2” behind “excel” | In this program you replaced “excel” with “excel 2” in the datafile argument of the PROC IMPORT statement. |
| Submit program and view the data portion of the data set ht\_wt6 | You should notice a couple of this here. First, the variable names. Spaces are allowed in Excel spreadsheets, but not in SAS variable names. By default, when there are spaces in the excel variable names, SAS will replace them with underscores. But, remember what I told you about shorthand variable names to save on typing. Trust me, you will probably get sick of typing “Date\_of\_Birth” and “Annual\_Household\_Income”. You can either go back and change the names in the excel file and import again, or you can rename them in a DATA step, which you will learn how to do in later.  The second thing I want you to notice are the values for the variables date of birth and annual household income. Remember that I told you that to SAS there are only two kinds of variables: Character and Numeric. This is still true. Now use the report generated by the CONTENTS procedure to view the descriptor portion of the data set.  This table shows attributes for all of the variables in your data set. You can see that this is still a numeric variable, but that the format MMDDYY10 has been applied. |
| S13 | SAS actually stores the date information as the number of days since January 1st, 1960. For example, January 2nd, 1960 has a value of positive 1. December 31st, 1959 has a value of negative 1. So, the data value actually stored for participant 001 in the date of birth column is actually 7,810, but because the import procedure recognizes this as a date during the import process, SAS adds the FORMAT MMDDYY10. |
| S14 | Finally, notice that SAS did not automatically apply a format to annual household income, so the dollar signs and commas are not displayed in this report. You will learn more about FORMATS in the next module. |
| S15 | As you previously saw, some data files are stored in fixed columns. You may want you to think of this file as a mini table. Each space can be occupied by a number, letter, or symbol and has a row and column number just like a table or spreadsheet. For instance, the zero in the id 001 occupies row 1 column 1. The T in Tom occupies row 1 column 5, and so on. The important thing to remember is that even empty spaces and symbols (like the forward slashes in the dates) have a row and column number. Once you know this, you can tell SAS where to go get the values for your dataset by specifically stating which columns have the data you want.  This has some advantages over the delimiter method of reading in raw data. First, you don’t have to worry about missing values. If you do not have a value, you can leave the appropriate columns blank. Next, when you write your input statement, you can choose which variables to read in and in what order.  So, how do you tell SAS which columns the values you want are in? |
| S16  Demonstration |  |
| Open the editor window and scroll down to reading in fixed column data. | Here is an example of a SAS code that uses the method called formatted input.  The @ signs are called column pointers. They tell SAS which column the values for each variable start in. Following the variable names are SAS INFORMATS. These are pre-written instructions telling SAS how to treat the values of that variable. For example, you already know that the dollar sign tells SAS to consider that variable to be a character variable.  You may have already guessed that the numbers part of an INFORMAT tells SAS how many columns that variable occupies.  Behind date of birth, you probably noticed the INFORMAT MMDDYY10. This tells SAS that the variable values that begin in column 26 of the raw data are date values that take the form of two-digit month, followed by a forward slash, two-digit day, followed by a forward slash, and four-digit year. Taken together the 8 digits and two forward slashes occupy 10 columns.  Finally, behind the variable income there is the dollar7 INFORMAT. As you probably guessed, this tells SAS that the values you are reading in for the variable income represents currency in dollars and occupies 7 columns.  Finally, all INFORMATS must end with either a period, or a period followed by a number (usually representing decimal places).  Let ‘s submit this code and then take a look at the data. |
| Submit and open the results viewer window. | Everything is there, but notice the appearance of date of birth and income.  **Question: What does 11,185 represent in observation 2?**  The 11,185 in observation 2 is the numeric value, in days since January 1st, 1960, that represents the day August 16th, 1990. This is what unformatted dates look like.  You may be asking, “But didn’t we tell SAS to format the dates, and the income?” No, an *in*format tells SAS what format the values are in in the raw data. It does not tell SAS how to format the data in the new SAS data set. Again, you will learn how to do that in the next module. |
| View SAS code | What if someone is really nice, and they send you the data already in a SAS dataset. How would you use it?  Well let’s look at one final example  All you have to do is use the set statement and point SAS to the directory containing the SAS data set. This includes pointing SAS towards a SAS library. This is good time to point out another best practice. I already told you that I recommend that you never change your raw data. This is so that no matter how much you mess around with your data, you will always have that raw data file to go back to. The same applies with SAS datasets. You may have noticed that in this lesson I kept creating new SAS data sets that were sequentially number, as opposed to writing over the same data set over and over. I highly recommend you also adopt this practice. If you write over a SAS data set, there is no way to get the previous version back. At best you can start over with the raw data and retrace your steps. |
| S17 | You now have a basic understanding of how SAS process a DATA step. You also know everything you need to know to get data into SAS from several common sources. If you have any questions about any of these videos please post them in the appropriate discussion board. |