# SAS Workshop - Reading data into SAS

A Michelle Edwards, Ph.D., MLIS

August 28, 2019

## Table of Contents

## Entering data into Excel

### Best Practices for data entry

When we start to enter data into Excel or any other program, we tend to enter it in a way that makes the most sense for data entry. We want it to be quick, easy to understand, and efficient. However, for statistical analysis purposes, we need to ensure that our data is entered in a way to makes the most sense for the analysis we are about to conduct. I tell my students that one of the most time consuming parts of any data analysis will be the time it takes to clean your data and get it ready for the analysis.

Here are a few recommendations or best practices to ensure your data is ready to be analyzed. Let's start with the variable names or the column names if you wish:

1. Variable names should be set to a maximum length of 32 characters - even that is VERY long
2. ALWAYS start variable names with a letter
3. Numbers can be used anywhere in the variable name AFTER the first character
4. Do NOT use blanks or spaces
5. ONLY use underscores "_" in a variable name
6. Use lowercase for your variable name

These best practices or naming recommendations also pertain to SAS dataset names.

### Knowing your data

Before bringing your data into any statistical package, you should take a look at it - or at least be comfortable with it. Chances are it has been entered using Excel. Open your Excel program and browse the data. Here is a snippet of a sample dataset.

*Sample Data Table*

| | mpg | cyl | disp | hp | drat | wt | qsec | vs | am | gear | carb |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Mazda RX4 | 21.0 | 6 | 160 | 110 | 3.90 | 2.620 | 16.46 | 0 | 1 | 4 | 4 |
| Mazda RX4 Wag | 21.0 | 6 | 160 | 110 | 3.90 | 2.875 | 17.02 | 0 | 1 | 4 | 4 |
| Datsun 710 | 22.8 | 4 | 108 | 93 | 3.85 | 2.320 | 18.61 | 1 | 1 | 4 | 1 |
| Hornet 4 Drive | 21.4 | 6 | 258 | 110 | 3.08 | 3.215 | 19.44 | 1 | 0 | 3 | 1 |
| Hornet Sportabout | 18.7 | 8 | 360 | 175 | 3.15 | 3.440 | 17.02 | 0 | 0 | 3 | 2 |

1. Is your data in the right columns?
2. Are there values that look out of line? Maybe a typo?
3. How many observations do you have? Does it match up to what was collected and entered?
4. Are there missing data? Should there be?

## Different ways to read data into SAS

## 1. Copy and Paste

One of the most popular ways to get data into the SAS program is to copy and paste from Excel directly into the SAS editor . This is fine if you have a smaller dataset. Now - small is a subjective term and really up to the user to "define". If you have a dataset that has a million records or so - the Copy and Paste method may not be the best way to approach getting your data into SAS. Personally, I think more than 20 records is too big - but again that decision is up to you - personal preference.

To bring data into SAS, we start our SAS code or syntax with the word **DATA** - this will let SAS know that we will be working with data. I want to take a little side step here and talk about Excel for a minute. How we use Excel can be used as an analogy and guide to the way we will bring our data into SAS.

### *Excel Sideline*

**What is the first thing you do when you start to enter data into Excel?** Type names in the columns to help guide you for data entry. These are usually long and descriptive to start.

**What is the second thing you do when you enter data into Excel?** Start entering your data. Matching data to the column headings.

**Generally speaking, what tends to be the last thing you do when entering data into Excel?** Save our work. Save the file and give it a name and save it somewhere on our computer.

### *Back to SAS*

**1. DATA statement**

SAS forces us to save the dataset name first! The DATA command tells SAS that we are about to work with data, but it really is creating a new SAS dataset and the name is ***first*** in our example. We save our data in SAS with the DATA step as opposed to doing it last as we tend to do when entering our data in Excel.

**DATA *first*** ;

Note that each line of SAS syntax or code will end with a **;**

*A couple of notes, the name of the SAS dataset can be anything you would like. The only caveat is that the name MUST match the Best Practices listed above. Please take a moment to review these*

## 2. INPUT statement

The second step in SAS with the Copy and Paste method is to identify the variables in our dataset. Aha! This matches what we tend to do in Excel. Only catch - we need to use variable names - please review the Best Practices listed above. To let SAS know what is coming up in our data, we use an **INPUT** statement. For the example data we would use:

**INPUT *make mpg cyl disp hp drat wt qsec vs am gear carb* ;**

Notice how each variable name matches a column in our example dataset.

## 3. DATALINES statement

The next step in the Copy and Paste method is to let SAS know that the data is coming up next by using the **DATALINES** statement, followed by the data on the next line(s).

**DATALINES;**

**Mazda RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4**

**Mazda RX4 Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4**

## 4. ;

The last step to reading our data into SAS is to add a **;** at the end of your data on a line all by itself. This lets SAS know that there is no more data coming and that it can start saving the data in the dataset called ***first*** - in this example.

## 5. RUN;

I, personally like to add the **RUN;** statement at the end of the code to close that section of the SAS syntax. It is not required, but if at some point you want to select parts of your code to run - you will need the **RUN;** - so let's add it and make it a good habit.

*Note: This is part of my Computer setup - you will NOT see this!! Please ignore*

```
## SAS found at C:/Program Files/SASHome/SASFoundation/9.4/sas.exe

## sas, saslog, sashtml, sashtml5, and sashtmllog & sashtml5log engines

##    are now ready to use.
```

*End of Note*

**SAS code for our example:**

```
Data first;
  input make mpg cyl disp hp drat wt qsec vs am gear carb;
  datalines;
Mazda_RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4
Mazda_RX4_Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4
Datsun_710 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1
Hornet_4_Drive 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1
Hornet_Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2
;
Run;
```

Try this on your own computer. What happens???

- Nothing
- Errors

## How do you troubleshoot errors?

It does not matter what version of SAS you are using - you always have access to the SAS Log. This is key part of running any SAS program. By reviewing your SAS log you can see what SAS has done every step of the way. In SAS Studio, please scroll up until you see the code that you entered into the editor.

So… we have a few errors! The main one that you will see states:

```
NOTE: Invalid data for mpg in line 4 1-9
```

Or something very similar. It might be selecting a different variable name, but in the end, SAS did not read our data properly.

So let’s work through 3 different challenges that we should be aware of:

1. **Numeric vs character variables**

SAS loves numbers or numeric data. That means if you have any variables or columns of data that are letters - also known as characters or strings - we need to let SAS know, by adding a **$** after the variable name.

In our example data, the first column called *make* is the name of the vehicle and is a string variable. To ensure SAS is aware that our first variable is not numeric, we add the **$** .

It can appear immediately after the variable name or you can have a space between the variable name and the **$** . Either way will work.

**INPUT make$ mpg cyl disp hp drat wt qsec vs am gear carb;**

## 2. Spaces in character data values

Notice how the names of the make of cars have spaces in them? SAS sees the space and assumes that the next word should be in the next variable. Take a close look at what is in your LOG. Notice how the value for mpg is set to a . and the value of cyl = 21 for the first line of data?

- Can you explain what has happened here?
- How do you fix this?

## 3. Spaces vs. Tabs

This is where the version of SAS you are using will be important (SAS 9.4 on your own laptop OR SAS Studio via Virtual Box).

Note that when you copy the data from Excel and paste it into your SAS editor that there is a large space between the values and/or the spacing between your values is not even. This is the Excel formatting that is being copied along with the numbers. In Excel we have the nice neat cells - all the same size and shape (unless we've changed it). When we copy the contents of Excel, there are hidden (to the naked eye) formatting of the cells that comes along with your data. The main formatting that comes along is the spacing between your values which is a **TAB**.

What is a TAB? When you are typing in a text editor, Word as an example, and you hit the TAB key, your cursor automatically advances 8 spaces. But it advances as a jump, it does not move ahead 8 spaces one at a time. This "jump" is known as the "TAB", and this is what we are copying when we copy from Excel.

As I noted above, SAS loves numbers and it also loves spaces. If you are using SAS 9.4, when you copy the values from Excel, the formatting is present, but this version of SAS overlooks it. So you do **NOT** need to make any further changes to your SAS code to read the data. Your data may still look uneven, since the formatting is brought over - but you don't need to worry about it.

If you are using SAS Studio though - we need to add a line of coding to let SAS know that there is a TAB between your data values and not spaces. To do this we need to add an **INFILE** statement.

The **INFILE** statement will provide SAS with information about the data in our dataset. The first thing we need to do is to identify the source of our incoming data. In this case, since we are copying and pasting the data from Excel to our SAS editor and the data is sitting in the editor window - the name of our *FILE* will be **DATALINES**. Once SAS sees that the incoming data or the data is contained in a FILE called DATALINES, we can now tell it about the data.

Since there are TABs and not spaces in between our values, we need to tell SAS that there is a delimiter or something separating our data that is not the default space. The SAS code for this is **dlm=" "** or **delimiter= " "** with our delimiter listed in the " ". As an example, if we had a , in between our values, as is the case with a CSV file, then we would type **dlm= “,”"** . Now if we tell SAS we have a TAB and use the TAB key on our keyboard we would have the following

INFILE DATALINES dlm=“________”;

When SAS reads the above statement, it sees 8 spaces in between the " " - it does not read that as a TAB. So, we need to provide a SAS-specific code for a TAB = ’09’x

**INFILE DATALINES dlm=“09”x;**

This line of code, provides SAS the information that it needs to treat the TABs copied from Excel as delimiter or spaces in between our values - and will proceed to read the data correctly.

**NOTE: THIS IS REQURED FOR SAS STUDIO USERS ONLY**

```
Data first;
  /* infile datalines dlm='09'x;  <- to be used for SAS STUDIO users ONLY */
  input make$ mpg cyl disp hp drat wt qsec vs am gear carb;
  datalines;
Mazda_RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4
Mazda_RX4_Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4
Datsun_710 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1
Hornet_4_Drive 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1
Hornet_Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2
;
Run;
```

## 4. **Length of character values**

The last piece of coding we MAY need to add is the length of the character values in our data. By default, SAS will read 8 characters for any string or character variable. If you look at the ***make*** variable in our example dataset, the values are longer than 8 characters. If we leave the program as is, SAS will only read the first 8 characters.

**Can you see what kind of trouble that can lead to?** The first 2 lines of data will have the same make. Imagine this in your own research data - not a good thing! So, we can fix this in 2 ways - the first would be to ensure that your data values are 8 characters or less - may not always be possible. So, let's make the change in SAS.

Before reading the names of the variables that are in our dataset, we are going to warn SAS that we have a variable coming up and it has a length longer than 8, it has a length of 25. To do this we add the following statement after the DATA step:

** LENGTH make $25.; **

LENGTH is warning SAS that we are changing the default length of 8 to something else. $ is telling SAS our variable is a string or character variables. 25 is the new length we are setting. The . is an indication to SAS that we are changing the format of the variable.

Let's try adding this to our complete SAS code:

```
Data first;
  length make $25.;
  /* infile datalines dlm='09'x;  <- to be used for SAS STUDIO users ONLY */
  input make$ mpg cyl disp hp drat wt qsec vs am gear carb;
  datalines;
Mazda_RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4
Mazda_RX4_Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4
Datsun_710 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1
Hornet_4_Drive 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1
Hornet_Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2
;
Run;
```

Try copying the above SAS syntax into your SAS program, make sure you use the INFILE statement if you need to and run the program.

**Is it working?**

**How to you know?**

## Viewing the data in SAS

We just ran a SAS program and it appears that it was successful. Looking at the LOG window there were no errors. If you are running SAS STUDIO, you will now see a window called OUTPUT Data that holds your data. You can scroll around in this window to make sure that the program you just wrote read your data correctly. If you are using SAS 9.4, we cannot see the data. We need to write a bit more code to see our data.

**PROC PRINT data= first;**

**RUN;**

PROC PRINT may be your first PROCedure in SAS. This is how we are going to analyse our data, by using different PROCedures. PROC PRINT is a very basic PROC and one that I find handy for debugging and viewing our data.

```
Proc print data=first;
Run;

 Obs make               mpg cyl disp  hp drat   wt   qsec vs am gear carb

  1  Mazda_RX4         21.0  6   160 110 3.90 2.620 16.46  0  1   4    4
  2  Mazda_RX4_Wag     21.0  6   160 110 3.90 2.875 17.02  0  1   4    4
  3  Datsun_710        22.8  4   108  93 3.85 2.320 18.61  1  1   4    1
  4  Hornet_4_Drive    21.4  6   258 110 3.08 3.215 19.44  1  0   3    1
  5  Hornet_Sportabout 18.7  8   360 175 3.15 3.440 17.02  0  0   3    2
```

## 2. Using the INFILE statement

The second way of bringing data into SAS, is to use an INFILE statement and keeping the data outside of SAS. For example, when you are working with larger files, you do not want to copy and paste hundreds of records into the SAS editor. It is more efficient to keep the data in a readable format and reference it with an INFILE statement.

SAS will read data from a file that is in any text format. The most common format is the CSV or Comma Separated Value file. Other formats include the TAB-delimited and a plain text file. For our purposes we will use the CSV.

*Note: When working with your data in Excel, make the action of saving the CSV version of your file part of your workflow. A copy of your Excel file will be your MASTER datafile, and your CSV file will be a copy of your WORKING datafile. You make changes in Excel, save it and save a second copy as a CSV. Then all changes are made one time. No need to remember to copy and paste back into SAS, or to edit the data you already have in SAS. Keep the MASTER Excel file where you make your edits, save it as a CSV, which is the file you read into SAS to perform all of your statistical analyses. This is all part of an effective and efficient Research Data Management lifecycle.

So let's start by saving your Excel file as a CSV file. This is accomplished by using the File - SaveAs option in your Excel program. Note that when you use the CSV option, it will only save one worksheet at a time. This is fine!

We start our SAS program the same way we did earlier:

**DATA first;**

Our next statement will be the INFILE statement. You need to tell SAS where the data is before you can tell SAS what is inside the file.

**INFILE "C:\Users\edwardsm\Documents\Workshops\SAS\cars.csv" dlm="," firstobs=2;**

Now we've done a few things with our **INFILE** statement. First, we've told SAS where the data is located - in this case: C:\Users\edwardsm\Documents\Workshops\SAS. Then we've told SAS the name of the file: cars.csv.

Remember SAS loves numbers and spaces, and here we are using a file that has a comma as a delimiter or separating our values. So we add the **dlm=","** once we've told SAS about where the file is located.

Remember that when we enter data into Excel, our first row in the file contains our column or variable names. Rather than removing it from our Excel or CSV file, we can let SAS know that it needs to start reading the data from the second row by using the SAS option of **firstobs=2** at the end of the **INFILE** statement.

## More INFILE statement options

### 1. MISSOVER

When we have missing data, the ideal way to work with this is to replace the empty cell in Excel with a . However, there are times when we forget or maybe our datafile is just TOO big to seek out all those empty cells and replace them with . The **MISSOVER** option can be helpful in these situations. The **MISSOVER** option will use your **INPUT** statement - remember this is where you list all your variables - as a guide to reading the data. For example if you a datafile with 12 variables (as our example dataset does), SAS will ensure that it reads 12 values in each row of data. You should still check your data after SAS has read it - to ensure that everything was read correctly!! But this option will prevent SAS from dropping to the next line of data before it has read the 12 values to match the 12 variables in the **INPUT** statement.

### 2. DSD (delimiter-senstive data)

A second **INFILE** option that you should be aware of is the **DSD** option. In our example, we are reading the data from a CSV file. A file that is uing the comma to separate our data - so rather than a space there is a comma between each of the values in our datafile. Now, what happens if you have string or character data and there is a comma in your value. For example you may be reading in a variable that contains the individual's favourite colour. You are expecting people to provide you with 1 colour only, but a few of your respondents provided you with 2 colours, separated with a comma - blue, red. Since we are reading in a CSV file and we told SAS that we have a delimiter of a comma **dlm=","** SAS will read blue into one variable,. then move onto to read red and place it as the value for the next variable. It sees that comma and says AHA I have 2 variables here - and NOT 2 values for 1 variable. To avoid this problem we add the **DSD** option to our **INFILE** statement. With this option SAS is now aware of any variables that are string (and usually contained in quotes), it will ignore the comma and place the values into the one variable.

Once we have completed our **INFILE** statement, we now tell SAS what is inside the file by using the **INPUT** statement. We use the same conventions we discussed earlier. Please see review the INPUT and Troubleshooting sections above.