Reading Datasets Into SAS

In this activity, we will learn how to use SAS to read, store and modify data in various formats. The data we receive for analysis will often not be in the format of a SAS dataset (.sas7bdat) – it's more likely to be in Excel (.xls) or Notepad (.txt) formats, among others.

Data: We will be using the Heart Rate data from problem 4.10 (p. 83) in the course textbook. The data has been posted on Blackboard Lab Week 4 in several different formats. Please download these files and place them on your H: drive in a location you can easily remember. My directory path is 'H:\bios600\Data', so any SAS code that I give will correspond to this location. You may choose your own location, but you may not be able to directly use parts of my code.

1 Data Step Structure

In order to read datasets into SAS and modify them appropriately, we make use of the data step type of SAS code referred to previously. Data steps are used to create new SAS datasets and modify them accordingly. The basic setup of a data step is given in Figure 1.

```
DATA your_data ;
  Statement 1 ;
  Statement 2 ;
    ...
RUN ;
```

Figure 1: Basic structure of a data step.

Things to remember about data steps:

- First line starts with SAS Keyword `DATA` (opens data step).
- Second word of first statement is the name of your SAS dataset
  - Dataset in Figure 1 is named "`your_data`"
  - Names can include letters, numbers and '_' (i.e., `your_data2`).
  - Names must start with a letter, and are no more than 32 characters long.
- Last line is SAS Keyword `RUN ;` (closes data step).
- Statements in between first and last lines are instructions for reading and manipulating dataset `your_data`.
  - Not all of these statements will begin with a SAS Keyword (unlike `PROC` code.)
  - Many functions available in SAS for data manipulation.

2 Reading Datasets - Data Step

There are a variety of ways to read datasets into SAS. There is no single preferred method, but certain situations may call for one method over another.

2.1 SAS Variables - Name, Type

SAS Variables have **names** so that we may refer to and use them in operations and/or analysis. Usually, it is up to you to create these names, so you can call variables what you like. Rules for naming SAS variables include the following.

1. Names consist of a combination of letters, numbers, and the underscore (‘_’).
2. Names must start with a letter.
3. Names must be 32 characters in length or less.

There are two **types** of variables that are recognized by SAS.

Numeric	– Values consisting of numbers only – Right-justified in dataset – BMI, Weight, Age, Stage of Cancer (1,2,3,4)
Character	– Values consist of letters and/or numbers – Left-justified in dataset – Race, Gender, Stage of Cancer (‘1’,‘2’,‘3’,‘4’)

2.2 Input Statement

The `INPUT` statement is used when we are reading in data within a data step. Specifically, we’ll use the `INPUT` statement to describe the variables in our dataset **by name** and **by type**.

When declaring our variables, order is important! Whether we are typing in the data manually (§2.3) or reading it from an external file (§2.4), we must list the variables in the correct order of appearance. Distinct variable names must be separated by a space.

We must also declare variable type in the `INPUT` statement. For numeric variables, we simply write their names and add a space. For character variables however, we must include the modifier ‘ `$`’ **directly after** the variable name, so that SAS knows to treat it as a character variable. This must be done for every single character variable! Include a space between the name and the ‘ `$`’.

Examples are given in §2.3 and §2.4 below.

2.3 Method 1 – Manual Entry

For a small dataset, we can simply enter the data by hand within a data step using a `CARDS` statement (appears on line 3 of Figure 2). The SAS Keyword `CARDS` tells SAS, “I am about to enter the data”. Everything that appears after “`CARDS ;`” is considered data, until the first semi-colon (;) is typed.

```
DATA Heart ;
  INPUT Trtmt $ Rate ;
  CARDS ;
    Before  65
    Before  85
     ⋮       ⋮
    Before  60
    After   68
     ⋮       ⋮
    After   72
   ;
RUN ;
```

Figure 2: Read in Heart Rate data from problem 4.10 (p. 83) using a `CARDS` statement.

Notice that the last observation’s data is on a different line from the semi-colon. If they are on the same line, then SAS will delete the data from your last observation! SAS interprets the first semi-colon (;) to be the end of the data, so anything to the left of this semi-colon (;) is ignored.

Some other things to note from Figure 2:

1. Data are highlighted in yellow to emphasize that they are values, instead of SAS coding instructions.
2. Each observation is written on its own line.
3. Nothing else goes in the `CARDS ;` statement.
4. Can also use `DATALINES` instead of `CARDS` .

This is useful for small datasets, but obviously we would not want to use it for datasets with lots of variables or a large number of observations.

2.4 Method 2 – Infile Statement

Within a data step, we can use an `INFILE` statement to directly read data from an external file of another format. The `INFILE` statement usually comes before using the `INPUT` statement to declare our variables. We use it to tell SAS

1. the path or location of our file (H:\bios600\Data\)
2. the filename (heartrate_edit)
3. the file extension (.txt)

Directories composed of these three parts must always be given to SAS in quotes (either single ‘’ or double “”). Furthermore, any quoted text in the Editor will appear in purple font. In Figure 3, we create the dataset Heart.sas7bdat by reading in the data from our text file.

```
DATA Heart ;      *create dataset Heart in Work folder ;
  INFILE ‘H:\bios600\Data\heartrate_edit.txt’ FIRSTOBS=2;
  INPUT Trtmt $ Rate ;      *list var’s in same order as .txt file ;
RUN ;
```

Figure 3: Read in Heart Rate data using an `INFILE` statement.

There are various options available to the `INFILE` statement. For instance, the option `FIRSTOBS`=2 tells SAS to begin reading data on line 2 instead of line 1, allowing me to pass over the variable labels in the .txt file.

Note that the `INFILE` statement will NOT work with SAS datasets in other folders. To access permanent SAS datasets, refer to §4.2. It also will not work for an Excel file (.xls).

3 Importing Files Manually

You can also manually import data files of various extensions using the SAS Import Wizard. The Import Wizard uses a Graphical User Interface (GUI), meaning that you interact with it, rather than typing and running code. There is SAS code being created and run behind the scenes, but we won’t worry about that.

While the Import Wizard is easy to use and very versatile and requires no programming, you will have to redo the process every time you log into SAS. One way around this is to assign the dataset to a permanent SAS library (see §4.2). Also, the Import Wizard names your variables for you. This feature does not always work well and can create some awkward variable names (or blank variables).

The following steps describe how to import an Excel (.xls) dataset into SAS using the interactive SAS Import Wizard.

1. From the File menu in SAS, select 'Import Data . . .'.
2. Check 'Standard Data Source' and select 'Microsoft Excel 97/2000/2002/2003 Workbook' from the pulldown menu.
3. When the window pops up, click Browse and locate your file ('H:\bios600\Data\heartrate_edit.xls'). Click Open and then OK.
4. Select the table you want to work with (Sheet title) from the pulldown menu and hit OK.
5. Select the library where you want to save the file and then name your SAS dataset in the Member: box (Heart).
6. Click Finish.

Similarly, a SAS Export Wizard is available for exporting files manually. This allows you to manage your data in SAS and then export it in another format to give to an investigator who doesn't know SAS.

4 SAS Libraries

So far, we have only discussed temporary SAS datasets, which are deleted once you end your SAS session. However, we can create permanent SAS datasets that will still be available to you the next time you use SAS. To do this, we will use SAS libraries.

SAS Libraries are folders located on our hard drive in which we can save SAS datasets. We basically tell SAS where these folders are and give them specific names, called **librefs**. Permanent SAS datasets must be referenced with both their libref (name of the library where the dataset is located) and their filename, using the form

libref.dataset_name

For example, if I want to use a SAS dataset named '`Heart`' that is located in the library '`MyLibrary`', then I must refer to the dataset as '`MyLibrary.Heart`'. This convention tells SAS, "go get the dataset `Heart` out of the library `MyLibrary`".

4.1 Work Library

The Work Library is the only SAS library in which we do not need to worry about this naming convention. Datasets in Work are technically named 'Work.dataset_name' , however if the libref is not included, SAS assumes you mean the Work Library. All datasets in Work are temporary, which means they will be deleted when you close SAS.

4.2 Creating Libraries

By now, you have created the dataset `Heart` in SAS multiple ways, but we have only used it within Work. If we want to use the dataset later, we'll have to re-read it into SAS and modify it all over again. To avoid this, we can create a new SAS Library in which we can save our permanent SAS datasets.

We will use a `LIBNAME` statement to create a SAS Library. The `LIBNAME` statement is free-standing - that is, it is not contained in either a data step or a procedure. We generally include them near the top of our program.

There are three parts to a `LIBNAME` statement: 1) SAS Keyword `LIBNAME` , 2) Library Name (libref), 3) Location on Hard Drive. In SAS, this format looks like

```
LIBNAME libref ‘Folder Path or Directory’ ;
```

Example: To create a library named ‘`MyLibrary`’ out of the folder ‘C:\Elena\SASstuff’, then I run the following code.

```
LIBNAME MyLibrary ‘C:\Elena\SASstuff’ ;
```

Note that the folder ‘C:\Elena\SASstuff’ must already exist on your computer before SAS will make it into a library!

I have created a permanent SAS Library on my H: drive named `bios600` to store my SAS datasets for this class. If you want create the same library to use throughout the semester (and you want it to match mine), first create the necessary folders on your hard drive for the path ‘H:\Bios600\Data\SAS’. Then run the following code in SAS.

```
LIBNAME bios600 ‘H:\Bios600\Data\SAS’ ;
```

If you'd rather name your SAS library something else, or point it to a different folder, then just create the folders and change the above code accordingly.

4.3 Saving To SAS Libraries

Now that we have created a SAS Library, we would like to store datasets in it so that we may use them later. For instance, now that I have read in the dataset `Heart` using one of the ways described in earlier sections (and have finished manipulating it), I would like to save it permanently to my `bios600` library. To do so, I will use the code in Figure 4.

```
DATA bios600.Heart ;   *name of new dataset to be stored in bios600 library;
  SET Heart ;   *old dataset from which new dataset is created ;
RUN ;
```

Figure 4: Creating dataset `bios600.Heart` in `bios600` library from original dataset `Heart` in Work library. Note that `bios600.Heart` is exactly the same as `Heart`.

Note that the name of my new permanent dataset (in library `bios600`) is the same as the name of my old dataset (in Work). This is NOT necessary - I'm free to name my new permanent dataset whatever I want (e.g., `HeartRate`). But the name of the old temporary dataset needs to match the dataset that I have been working with.

4.4 Using Work and SAS Libraries

Any changes you directly make to a permanent SAS dataset will be permanent. For this reason, we often use the Work library and permanent SAS libraries (e.g., `bios600`) together.

Generally, you will want to create a temporary SAS dataset from a permanent one. Then you can just work with (manipulate, analyze) the temporary dataset in your program, so as to avoid making accidental changes to the permanent dataset. For example, if you have the permanent dataset `bios600.Heart`, you can make a new temporary version called `tempHeart` using the following code.

```
DATA tempHeart ;   *name of new temporary dataset saved in Work;
  SET bios600.Heart ;   *old dataset from which new dataset is created ;
RUN ;
```

Figure 5: Creating temporary dataset `tempHeart` in Work library from original permanent dataset `bios600.Heart` in `bios600` library. Note that `tempHeart` is exactly the same as `bios600.Heart`.

Again, we chose the name `tempHeart` arbitrarily. I could have used `Heart2` or `newHrt` just as easily.

5 Modifying SAS Datasets

Once we have read in our dataset, we usually need to manipulate it a bit. This manipulation is broadly referred to as data management, and takes place within a data step. Among other things, data management can include

- looking for errors (e.g., negative ages)
- reshaping a dataset (horizontal vs. vertical format)
- deleting certain subjects (remove subjects with BMI$<$ 17)

- creating new variables based on old ones (average blood pressure across visits 1-3)
- creating character variables out of numeric (BMI category from BMI)
- deleting variables
- renaming variables

5.1 SAS Functions

To do most data management tasks, we can use SAS functions. SAS functions work within data steps only (i.e., only when the dataset is still under construction). These are operations within SAS that can perform a task more easily and efficiently than if you wrote out the code to perform the steps of that function.

For instance, suppose we have a dataset named `SBP` of 20 subjects who had their systolic blood pressure (SBP) taken at three visits.

Subject	SBP1	SBP2	SBP3
1	120	125	124
2	130	129	130
3	114	124	119
4	135	136	129
⋮	⋮	⋮	⋮
20	110	108	108

Table 1: Example dataset `SBP`

Let's say we want to create a copy of this dataset named `SBP2`, but we are interested in each person's average systolic blood pressure. Both Figures 6 and 7 achieve this, but the code in Figure 7 is actually more efficient, in terms of SAS computing time.

```
DATA SBP2 ;
  SET sbp;
  meanSBP = ( SBP1 + SBP2 + SBP3 )/3 ;
RUN ;
```

Figure 6: New dataset with manually created variable meanSBP.

```
DATA SBP2 ;
  SET sbp;
  meanSBP = mean( SBP1, SBP2, SBP3 ) ;
RUN ;
```

Figure 7: New dataset with SAS-function-created variable meanSBP.

Table 2 shows what the dataset `SBP2` looks like based on either of the above sets of code. In this very simple example, it might not matter which one you choose. But there are times when you need to modify a dataset in a way that only a SAS function can do correctly.

Subject	SBP1	SBP2	SBP3	meanSBP
1	120	125	124	123
2	130	129	130	129.66667
3	114	124	119	119
4	135	136	129	133.33333
⋮	⋮	⋮	⋮	⋮
20	110	108	108	108.66667

Table 2: New dataset `SBP2`

SAS functions work across rows. The general format of a SAS function is

variable=function-name(argument1, argument2,...);

Types of SAS functions:

- Arithmetic (absolute value, square root, mean, variance,...)
- Trigonometric (cosine, sine, arc cosine,...)
- Other mathematical and statistical (natural logarithm, exponential,...)
- Pseudo-random number generators
- Character string functions

A table of selected SAS functions and their arguments is posted on Blackboard.

6 Exercises

1. Create a SAS dataset from the file skinfold_edit.xls using manual entry.
   (a) Create a new variable mm_2 to represent half of each person's measurement.
   (b) Create a new variable mm_plus30 to represent each person's measurement plus 30.
2. Create a SAS dataset from the file cholesterol_edit.txt using the `INFILE` statement.
   (a) Create a new variable chol_10 to represent the average of each person's measurement and the number 10 (use the MEAN function).

   (b) Create a new variable chol_group to represent the product of the variables cholesterol and group.

3. Use the SAS Import Wizard to import the file bph_samp_edit.xls.
   (a) Create a new variable qol_delta to represent the absolute difference between qol_base and qol_3mo (use ABS function).
   (b) Create a new variable qol_sum to represent the sum of qol_base and qol_3mo.
   (c) Create a new variable log_qol to represent the natural log of each person's qol_delta (use LOG function).
4. Save these modified datasets permanently in a SAS Library named `bios600` (can use the path identifier described in §4).

7 References

1. UNC ITS Introduction to SAS http://help.unc.edu/4433
2. UCLA SAS Resources http://www.ats.ucla.edu/stat/sas/
3. UCLA SAS FAQ http://www.ats.ucla.edu/stat/sas/faq/default.htm
4. BIOS511 Notes (Fall 2007)