Introduction and Reading Data

December 6, 2023

1 An Introduction to SAS

1.1 Some beginnning notes

- Something about me
  - Name: Yutong Xie
  - Email: yxiefin@gmail.com
- My materials are mostly from the following 3 sources. They all deserve the credit if you learnt anything from this course.
  - **The Little SAS Book** by Lora D. Delwiche and Susan J. Slaughter. This is absolutely the best book in the world for SAS programming.
  - **The Book of R: A First Course in Programming and Statistics** by Tilman M. Davies. There are just so many R resources on the internet. I like this book because it is very fundamental.
  - **Google** Both SAS and R have well maintained communities. We can ask questions and find answers to questions that are new to we but old to the internet. People who programs in these languages are nice most of the times. I also program in Stata. There are only a few people answering a lot of the questions in that community and they are mean.

1.2 The Language

- Just think programing as a kind of language that a machine can understand.
- We talk in the same language, so we can tell the machine what to do.
  - Language has rules. Same with SAS.
  - The most important rule is: ***EVERY SAS STATEMENT ENDS WITH A SEMICOLON***
  - Basically SAS will listen to what we tell it to do until it sees a semicolon, and then do it.

1.3 A good program is a green program

- SAS is much more flexible than a lot other languages. As a result, it is possible to write a piece that is very difficult to understand.
- ***Commenting is very Very very Very very Very very important. We cannot comment more.***
- Comments are not executed by SAS. They are just notes for people to read.
- How to comment in SAS?
  - /*…*/ or * … ;

- Let's write our first comment and take the chance to get to know the SAS Studio!
  - https://welcome.oda.sas.com/

[ ]:
```
/* This is my first SAS program! */
* This is my first SAS program! ;
```

1.4 Small Steps Forward

- I have to say a machine is powerful but not very smart. For the machine to do what we want it to do, we have to give the machine corret, detailed, and unambigious instructions.
- Think about which one of the following is easier for a 5-year-old to understand
  - Build a pyramid with these 15 cubic block
  - Lay 5 blocks right next to each other in a straight line; Lay 4 blocks on top of the 5 we have laid down; Lay 3 blocks on top of the 4 we h ve laid down; Lay 2 blocks on top of the 3 we have laid down; Lay 1 blocks on top of the 2 we have laid down; We have a pyramid.
- Writing programs should be done in small steps.
  - Think about what we want to achieve.
  - Know the data we are handling.
  - Think about how we can move from the data we **have** to the data we **want**, and lawet the *small* steps.
  - Use the right tools to execute each step.
  - Check for and correct any errors or mistakes.
  - Try to improve the program to make it faster, shorter, and easier to understand.
  - Package useful code for future uses.

1.5 SAS Data Sets

- SAS needs data to perform the actions it is capable of.
  - In other words, SAS operates on data .
- Here is an example
- Let's put this data in our SAS Studio!

[5]:
```
/* Copy and paste the following code in wer SAS Studio and Click Submit. */
DATA Height_and_Weight;
INPUT ID Name $ Height Weight;
DATALINES;
53 Susie 42 41
54 Charlie 46 55
55 Calvin 40 35
56 Lucy 46 52
;
RUN;

PROC PRINT DATA = Height_and_Weight;
RUN;
```

```
<IPython.core.display.HTML object>
```

1.6 Read the Log!

- When a program is executed, a log is generated in the log window. ALWAYS read log! It contains useful information.
- There are three types of log messages, colored blue, green and red.
  - NOTE: blue, general (good) information, useful, number of obs.
  - WARNING: green, not an error but SAS informs we that we may have a problem, although it does not stop processing, still creates a data set
  - ERROR: red, an error in the code, SAS cannot process the data step, it stops! If we are running the data step to replace an older version of a data set, it has NOT been replaced!
    - SAS will stop at ERROR. This is pretty important. This means only the steps before the error are executed.
    - When there is not an error, there can still be mistakes in the code. As a result, we have to check the output.

1.7 Data Types

- Each column should have one data type.
  - Numeric or Character
    - Each column can have only one data type.

- Why the difference?
  - They are created differently.
    - Numerics are just numbers: 1, 2, 100.
    - Characters need quotes: "1", "2", "100".
  - We have to perform different operations.
    - For example, we can add two numeric columns together but we cannot sum two characters.
- Let's check the datatypes of the columns in the data set we just loaded!

1.8 DATA and PROC

- SAS programs are constructed from two basic building blocks: ***DATA*** steps and ***PROC*** steps.
  - DATA steps start with DATA and PROC steps start with PROC.
  - A typical program starts with a DATA step to create or modify a SAS data set and then passes the data to a PROC step for processing.* In other words,
    - Data steps are used for actions on rows (eg. create a new variable from another variable).
      - Data steps are usually executed row-by-row and line-by-line. This is a pretty important feature and shapes a lot of the logics in SAS. More later.
    - Proc steps are used for actions on columns (eg. calculate a mean of a variable)
- Let's do this:
  - Create a SAS dataset using the previously loaded dataset and name the new data "BMI".
  - Create a new column in the data called BMI. Name the column BMI and BMI is calculated as weight divided the square of height.
  - Print the results.

- When we are done
  - Congratulations, we have created wer first SAS program!
  - Think about it. What did we do? What does the program do for us?
    * We created and modified data using DATA step.
    * We print the data using PROC step.

[6]:
```
DATA BMI;
    set Height_and_Weight;
    BMI = weight / height**2;
RUN;

PROC PRINT data = BMI;
RUN;
```

```
<IPython.core.display.HTML object>
```

1.9 SAS Data Libraries

- SAS data sets are stored and accessed via SAS data libraries.
- There is a WORK library by default created and refreshed every time SAS is initiated.
  - So the data here will be temporary.
- We can define other libraries using the LIBNAME statement.
  - So the data here will be permanent.
- We can create data in a library using the DATA step.
  - When loading data from a library, we need to use the dot "."
- Let's do this:
  - On SAS Studio, create a folder called "Class_Output".
    * Click on "Server Files and Folders"
    * -> Right click on "Files (Home)"
    * -> Hover over "New" and click on "Folder"
    * -> Enter "Class_Output" in the box after Name.
    * -> Click "Save".
  - Create a library called "Class_Export" that's linked to the "Class_Export" folder we just created.
    * We have to know the path to the folder in this one.
      · It should look like /home/u'wer user ID'/. In the code below, 63703618 is my user ID, so the path is "/home/u63703618/Class_Export".
  - Create a copy of the BMI data in this folder.

[ ]:
```
LIBNAME Class_Export "/home/u63703618/Class_Export";
DATA Class_Export.BMI;
    set BMI;
run;
```

2 Reading Data into SAS

2.1 The Goal

- In this section, we just want to read data into SAS. There can be many complexities, but SAS can handle most of it.
- I will also go over some important concepts and backstage operations that helps we with understanding SAS codes overall.

2.2 The easy

- If a data set is a well maintained CSV file, excel file, or even dta file, we can just use PROC IMPORT.

[ ]:
```
PROC IMPORT DATAFILE = 'filename' OUT = dataset DBMS = identifier REPLACE;
```

- There are quite a few useful statements under PROC IMPORT that we can introduce, but instead of listing everything here. Let's try to introduce those as we go. For now, I just want to point out 3
  - **GUESSINGROWS = n | MAX;** This is needed when we have a large file. SAS will just read a limited number of rows if we do not specify this. We may end up with an incomplete import.
  - **GETNAMES = YES;** This is needed if wer excel file has column titles.
  - **SHEET = "Sheet1";** This is needed if we have multiple sheets to read.

[ ]:
```
PROC IMPORT DATAFILE = 'filename' OUT = dataset DBMS = identifier REPLACE;
    GUESSINGROWS = MAX;
    GETNAMES = YES;
    SHEET = "Sheet1";
run;
```

- If the data set is already a sas data file
  - Usually with extension name .sas7bdat
- We can simply use the SET statement.

[15]:
```
DATA CARS; /* Create a sas data called CARS*/
    SET SASHELP.CARS; /* Reading observations from a data also called CARS but␣
 ↪from a place called SASHELP (more on this later).  */
RUN;

PROC PRINT DATA = CARS (OBS = 5); /* Printing the first 5 observations of the␣
 ↪new created data CARS */
RUN;
```

```
<IPython.core.display.HTML object>
```

2.3 The hard

- When we have to type in the data like we did in the last session or when we read **raw data**, we need to use the input and/or datalines.

- Recall the code we used:

[ ]:
```
DATA Height_and_Weight;
INPUT ID Name $ Height Weight;
DATALINES;
53 Susie 42 41
54 Charlie 46 55
55 Calvin 40 35
56 Lucy 46 52
;
RUN;
```

- This data is delimited by ***spaces***. To read space-delimited values, we can use this simple data step.
  - If the values in our raw data file are all separated by at least one space, then using **list input** (also called **free formatted** or **space-delimited input**) to read the data may be appropriate.
    - This means using the ***INPUT*** statement
  - By default, you must read all the data in a record—no skipping over unwanted values.
  - Any missing data must be indicated with a period.
  - Character data must be simple—no embedded spaces, and no values greater than 8 characters in length.
- If the data file contains dates or other values that need special treatment, then list input would not be appropriate.
- This may sound like a lot of restrictions, but a surprising number of data files can be read using list input.

2.4 Complexity 1: what if a character is longer than 8?

- What happens if you run this? Give it a try.

[ ]:
```
DATA Height_and_Weight;
INPUT ID Name $ Height Weight;
DATALINES;
53 Susie 42 41
54 Charlie 46 55
55 Calvin 40 35
56 Lucy 46 52
;
RUN;
```

- We can specify the length before you make the list input. Try to read the following data using similar code and see what goes wrong.

[ ]:
```
53 Susie 42 41
54 Charlie 46 55
55 Calvin 40 35
56 Lucy 46 52
57 Christopher 50 52
```

[ ]:
```
DATA Height_and_Weight;
LENGTH Name $12;
INPUT ID Name $ Height Weight;
DATALINES;
53 Susie 42 41
54 Charlie 46 55
55 Calvin 40 35
56 Lucy 46 52
57 Christopher 50 52
;
RUN;
```

- In SAS programs, the attributes of a variable are set when SAS first encounters that variable. So if a LENGTH statement comes before an INPUT statement, then SAS will use that length.

2.5 The ugly

- What if the data are not in standard format?
  We need to specify the informat!

In other words, we need to tell SAS how each variable is formated in the raw data.

- For example,
  - "$10." Character informat
  - "3." Numeric informat
  - "MMDDYY10." Date informat
- As you can see, they all end with a number and a period.
- Informat tells SAS the format you input; Format tells SAS the format you want the output.
- We can always be very explicit about the data you are loading. That is, regardless if you actually need to, just specify those informats.
  - This is called formatted input.
  - The columns read for each variable are determined by the starting point and the width of the informat.
    - This means the space delimiter is ignored.
    - SAS always starts with the first column.

2.6 Complexity 2: Use Informat

- Try this:

[3]:
```
DATA contest;
INPUT Name $16. Age 2. Type $1. Date $10. Score1 4.1 Score2 4.1 Score3 4.1;
DATALINES;
Alicia Grossman  13 c 10-28-2020 7.8 6.5 7.2
Matthew Lee      9  D 10-30-2020 6.5 5.9 6.8
Elizabeth Garcia 10 C 10-29-2020 8.9 7.9 8.5
Lori Newcombe    6  D 10-30-2020 6.7 4.9
Jose Martinez    7  d 10-31-2020 8.9 9.5 10.0
Brian Williams   11 C 10-29-2020 7.8 8.4 8.5
```

```
;
RUN;

proc print data = contest;
run;
```

```
<IPython.core.display.HTML object>
```

- What's wrong?
- How can you adjust to fix?

[5]:
```
DATA contest;
INPUT Name $16. +1 Age 2. +1 Type $1. +1 Date MMDDYY10. (Score1 Score2 Score3)␣
 ↪(4.1);
DATALINES;
Alicia Grossman  13 c 10-28-2020 7.8 6.5 7.2
Matthew Lee      9  D 10-30-2020 6.5 5.9 6.8
Elizabeth Garcia 10 C 10-29-2020 8.9 7.9 8.5
Lori Newcombe    6  D 10-30-2020 6.7 4.9
Jose Martinez    7  d 10-31-2020 8.9 9.5 10.0
Brian Williams   11 C 10-29-2020 7.8 8.4 8.5
;
RUN;

proc print data = contest;
run;
```

```
<IPython.core.display.HTML object>
```

2.7 Complexity 3: Use the pointer @

- Try this to read this:

[ ]:
```
Yellowstone           ID/MT/WY 1872     4,065,493
Everglades            FL 1934           1,398,800
Yosemite              CA 1864           760,917
Great Smoky Mountains NC/TN 1926        520,269
Wolf Trap Farm        VA 1966           130
```

[ ]:
```
DATA nationalparks;
INPUT ParkName $ 1-22 State $ Year @40 Acreage COMMA9.;
datalines;
Yellowstone           ID/MT/WY 1872     4,065,493
Everglades            FL 1934           1,398,800
Yosemite              CA 1864           760,917
Great Smoky Mountains NC/TN 1926        520,269
Wolf Trap Farm        VA 1966           130
;
run;
```

- What are these?
  - **INPUT ParkName \$ 1-22 State \$ Year @40 Acreage COMMA9.;**
  - **ParkName** First variable is the ParkName
  - **$** ParkName is a character
  - **1-22** ParkName is read from 1 to 22.
  - **State** Second variable is the State
  - **$** State is a character
  - **Year** Third variable is Year
  - **@40** Move the pointer to the 40th character
  - **Acreage** Fourth variable is Acreage
  - **COMMA9.** Informat is COMMA9.; it is a numeric informat.

2.8 Complexity 4: More disorganized raw data needs more tools

- If you look at the data we have been reading, observations in each column are usually well aligned or they have consistent relative position each row (For example, the values of two adjacent variables are always separated by exactly 1 space).
- Sometimes you need to read data that just don't line up in nice columns or have predictable lengths. When you have these types of messy files, ordinary list, column, or formatted input simply aren't enough.
  - @'character' column pointer: move the pointer to a particular character.
  - colon modifiers: tells SAS to read a data value until it reaches a space.
  - ampersand modifiers: tells SAS to read a data value until it reaches two or more spaces in a row
- If the data looks like this, Can you try to load it without using these?
- Apparently School and Time should be the column names. Just give it a try using what we have used so far.

[ ]:
```
Bellatorum School CSULA Time 1:40.5
The Kraken School ASU Time 1:45.35
Black Widow School UoA Time 1:33.7
Koicrete School CSUF Time 1:40.25
```

[ ]:
```
DATA contest;
INPUT CanoeName & $13. @'School' School $ @'Time' RaceTime :STIMER8.;
DATALINES;
Bellatorum School CSULA Time 1:40.5
The Kraken School ASU Time 1:45.35
Black Widow School UoA Time 1:33.7
Koicrete School CSUF Time 1:40.25
;
RUN;
```

2.9 Use INFILE and INFILE options

- Here are some important INFILE options. We can load files using the infile option.
  - **FIRSTOBS =**: Specifies the first observation that SAS processes in a SAS data set.
  - **OBS =**: Specifies the last observation that SAS processes in a data set.

  - **MISSOVER**: By default, SAS will go to the next data line to read more data if SAS has reached the end of the data line and there are still more variables in the INPUT statement that have not been assigned values. The MISSOVER option tells SAS not to go to the next line of data when it runs out of data. Instead, assign missing values to any remaining variables. The following data file illustrates where this option may be useful.
  - **TRUNCOVER**: Its applicable to the last variable; if actual value available in the current record of the external file is shorter than the informat specified, it assigns that 'Truncated' value. If it is missing then missing value gets assigned to the variable. DLM: Specify the delimiter
  - **DSD**: Delimiter-Sensitive Data option does three things
    * ignores delimiters in data values enclosed in quotation marks.
    * does not read quotation marks as part of the data value.
    * it treats two delimiters in a row as a missing value. ing value.

- **MISSOVER** and **TRUNCOVER** can get confusing. There are more options that can be more confusing. Here is an article explaining these data set options.
  - https://support.sas.com/resources/papers/proceedings/proceedings/sugi26/p009-26.pdf

2.9.1 MISSOVER

- Try this with and without the MISSOVER option and see the difference.

[ ]:
```
DATA class102;
INFILE DATALINES MISSOVER;
INPUT Name $ Test1 Test2 Test3 Test4 Test5;
datalines;
Nguyen 89 76 91 82
Ramos 67 72 80 76 86
Robbins 76 65 79
;
RUN;
```

2.9.2 Read Delimited Files

- Try this. Anything wrong?

[ ]:
```
DATA music;
INFILE DATALINES;
INPUT BandName :$30. EightPM NinePM TenPM ElevenPM;
datalines;
Lupine Lights,45,63,70,
Awesome Octaves,17,28,44,12
"Stop, Drop, and Rock-N-Roll",34,62,77,91
The Silveyville Jazz Quartet,38,30,42,43
Catalina Converts,56,,65,34
;
RUN;
```

- It seems SAS does not know this file is delimited by commas. Let's try again with the DLM = otpion.

[ ]:
```
DATA music;
INFILE DATALINES DLM = ',';
INPUT BandName :$30. EightPM NinePM TenPM ElevenPM;
datalines;
Lupine Lights,45,63,70,
Awesome Octaves,17,28,44,12
"Stop, Drop, and Rock-N-Roll",34,62,77,91
The Silveyville Jazz Quartet,38,30,42,43
Catalina Converts,56,,65,34
;
RUN;
```

- Anything wrong?
- It seems in the 3rd row, SAS takes the first comma after Stop as a delimiter. Let's try again with the DSD option.

[ ]:
```
DATA music;
INFILE DATALINES DLM = ',' DSD;
INPUT BandName :$30. EightPM NinePM TenPM ElevenPM;
datalines;
Lupine Lights,45,63,70,
Awesome Octaves,17,28,44,12
"Stop, Drop, and Rock-N-Roll",34,62,77,91
The Silveyville Jazz Quartet,38,30,42,43
Catalina Converts,56,,65,34
;
RUN;
```

- Let's try again, but with the MISSOVER option this time.

[ ]:
```
DATA music;
INFILE DATALINES DLM = ',' MISSOVER;
INPUT BandName :$30. EightPM NinePM TenPM ElevenPM;
datalines;
Lupine Lights,45,63,70,
Awesome Octaves,17,28,44,12
"Stop, Drop, and Rock-N-Roll",34,62,77,91
The Silveyville Jazz Quartet,38,30,42,43
Catalina Converts,56,,65,34
;
RUN;
```