Introductory Computing for Statistics

Lecture 2: Data Management

Xiao Li
Rutgers University

October 16, 2017

Lecture 1 Reminders

- SAS interface: editor, output, and log;
- DATA and PROC steps (also OPTIONS);
- Three INPUT methods: list, column, and formatted;
  - Note on formatted method:
- PROC print statement;

Lecture 1 Reminders

- SAS interface: editor, output, and log;
- DATA and PROC steps (also OPTIONS);
- Three INPUT methods: list, column, and formatted;
    - Note on formatted method:
- PROC print statement;

Today's topics

- Input data les from external resources.
  - FILENAME statement
  - INFILE statement
- Write simple custom reports to external les.
  - PUT statement
  - FILE statement

Today's topics

- Input data les from external resources.
  - FILENAME statement
  - INFILE statement
- Write simple custom reports to external les.
  - PUT statement
  - FILE statement
- Reconstructing SAS dataset.
  - Copy an existing dataset
  - Modify variables
  - Get a subset of a dataset
  - Combine datasets.

Today's topics

- Input data les from external resources.
  - FILENAME statement
  - INFILE statement
- Write simple custom reports to external les.
  - PUT statement
  - FILE statement
- Reconstructing SAS dataset.
  - Copy an existing dataset
  - Modify variables
  - Get a subset of a dataset
  - Combine datasets.

INFILE statement general form

General form of simple data

```
DATA dataset;
     INFILE '<file path and filename>';
     INPUT variables;
RUN;
```

Example:

- Windows:

  ```
  INFILE 'c:\MyDir\mydata.dat';
  ```

- UNIC:

  ```
  INFILE '/home/MyDir/mydata.dat';
  ```

General form with FILENAME statement

General form of simple data with FILENAME statement

```
FILENAME datain ‘file path and filename’;
     DATA dataset;
     INFILE datain;
     INPUT variables;
RUN;
```

FILENAME gives an alias (datain) to an external data file. SAS statements can then use the alias to refer to the external file. If only a filename is given, then SAS will search for it in the default directory. You can assign an alias to a file which may or may not exist, but if the alias is going to be used in the INFILE statement, then this file must exist, otherwise, SAS log will report an error message.

General form with FILENAME statement

General form of simple data with FILENAME statement

```
FILENAME datain ‘file path and filename’;
     DATA dataset;
     INFILE datain;
     INPUT variables;
RUN;
```

FILENAME gives an alias (datain) to an external data file. SAS statements can then use the alias to refer to the external file. If only a filename is given, then SAS will search for it in the default directory. You can assign an alias to a file which may or may not exist, but if the alias is going to be used in the INFILE statement, then this file must exist, otherwise, SAS log will report an error message.

INFILE statement

INFILE statement tells SAS where to read the data.
The INFILE statement has options:

- DELIMITER (DLM) Is useful for files whose fields are delimited with special characters (eg. comma, semicolon) instead of spaces.
- FIRSTOBS Tells SAS to begin reading at a given line.
- OBS Tells SAS at which line to stop reading.
- MISSOVER Tells SAS not to go to the second line for the remaining variables when there are missing values.

DELIMITER-INFILE

Example 2.1 Age data

In order to create a SAS dataset from this age data, we use the following SAS code:

```
/*Example 2.1*/
FILENAME roster 'stat390/age.dat';
DATA age;
     INFILE roster delimiter=',';
     INPUT name $ age;
RUN;

PROC PRINT data=age;
RUN;
```

FIRSTOBS and OBS-INFILE

Example 2.2 Bank accounts data

The following lines tell SAS to skip the first three lines and to start reading in data from the fourth line.

```
FILENAME bankacct 'stat390/bankaccts.dat';
DATA bankacct;
     INFILE bankacct firstobs=4 obs=6;
     INPUT bank $12. balance type $;
RUN;

PROC PRINT DATA=bankacct;
RUN;
```

PUT statement

Example 2.3 Write to LOC

The PUT statement is used to write information to the LOG window.

```
DATA cash;
     INPUT bank $ acctnum money;
     PUT bank acctnum money;
     DATALINES;
     CHASE 1536253 50.32
     PNCBANK 189273462 1563.82
     FLEET 287363 20000.00;
RUN;
```

PUT statement cont.

Example 2.4 Write to an external file

A FILENAME, a FILE and a PUT statement are needed for writing to an external file.

```
FILENAME dataout 'stat390/newcash.txt';
DATA cash;
     INPUT bank $ acctnum money;
     FILE dataout;
     PUT @2 bank @13 acctnum @23 money;
     DATALINES;
     CHASE 1536253 50.32
     PNCBANK 189273462 1563.82
     FLEET 287363 20000;
RUN;
```

Note on PUT and FILE statements

- The file specifed in the FILENAME statement may or may not exist originally. If it did not exist, then it will be created; otherwise, the old file will be overwritten.
- FILE statement is the complement of the INFILE statement;
- FILE statement must be preceded by a FILENAME statement, just like the INFILE statement.

- PUT statement should only appear in the DATA step;
- PUT statement is the complement of the INPUT statement and has many of the same formatting commands.
- It is unnecessary to specify the variable type, numerical or character. (Ie. No $ sign required here.)

Note on PUT and FILE statements

- The file specifed in the FILENAME statement may or may not exist originally. If it did not exist, then it will be created; otherwise, the old file will be overwritten.
- FILE statement is the complement of the INFILE statement;
- FILE statement must be preceded by a FILENAME statement, just like the INFILE statement.

- PUT statement should only appear in the DATA step;
- PUT statement is the complement of the INPUT statement and has many of the same formatting commands.
- It is unnecessary to specify the variable type, numerical or character. (Ie. No $ sign required here.)

Reconstructing SAS dataset

Take note of these important facts:

1. An existing SAS data set cannot be changed. Rebuilding a SAS data set is only accomplished by creating a new data set.
2. An existing variable cannot be modified. "Modifying" a variable is achieved by creating a new variable.

We will discuss how to:

- Copy an existing dataset
- Modify variables
  - Create new variables
  - Delete variables
  - Rename variables
- Get a subset of a dataset
- Combine datasets

Reconstructing SAS dataset

Take note of these important facts:

1. An existing SAS data set cannot be changed. Rebuilding a SAS data set is only accomplished by creating a new data set.
2. An existing variable cannot be modified. "Modifying" a variable is achieved by creating a new variable.

We will discuss how to:

- Copy an existing dataset
- Modify variables
  - Create new variables
  - Delete variables
  - Rename variables
- Get a subset of a dataset
- Combine datasets

Copy an existing dataset

General form of SET statement

```
DATA new-dataset;
     SET old-dataset; /* copy from old-dataset */
```

Example 2.5

```
DATA agecopy;
     SET age;
RUN;

PROC PRINT DATA=agecopy;
RUN;
```

Modify variables

General form for modifying variables

```
DATA new-dataset;
     SET old-dataset; /* copy from old-dataset */
     ...; /*statements that defines new variables,
     subset, delete variables, etc.) */
RUN;
```

Create new variables - modify variables

- Sometimes we need a transformation of the original dataset. We need to copy data from the old (original) dataset and then add to this a new variable which is some mathematical function of the old variable(s).
- All the usual mathematical operators are: $+$, $-$, $*$ and $**$ for exponentiation. Some other frequently used functions are: LOG, LOG10, EXP, SQRT, $**2$, MEAN(), SUM(), ABS(), SIN(), COS().

Create new variables - modify variables

Example 2.6 Create a new variable called total by adding up read and write scores for each student.

```
DATA students;
     INPUT id female schtype $ read write;
     DATALINES;
     147 1 pub 87 72    /*suppose 1 indicates female, */
     108 0 pub 74 63   /*       0 indicates male*/
     18 0 pri 50 93 ;
RUN;

DATA students1;
     SET students;
     total=read+write; /* add up read and write */
RUN;

PROC PRINT data=students1; RUN;
```

Create new variables - modify variables

Example 2.7 Create the dataset and add new variables in one data step

```
DATA students2;
     INPUT id female schtype $ read write;
     total = read + write;
     DATALINES;
     147 1 pub 87 72 /*suppose 1 indicates female, */
     108 0 pub 74 63 /*        0 indicates male*/
     18 0 pri 50 93;
RUN;

PROC PRINT data=students2;
RUN;
```

IF statement - create new variables - modify variables

the IF statement allows using different defnitions for different subsets of the dataset. This follows the same format as conditioning in other programming languages.

Syntax

```
IF ...  THEN ...  (IF ...  THEN) ELSE ...
IF ...  THEN ...  ELSEIF ...  THEN ...  (...)  ELSE ...
```

IF statement - create new variables - modify variables

Example 2.8 Create a new variable called readgrade (≥ 90: A, $[80, 90)$:B, $[70, 80)$: C, $[60 - 70)$:D, < 60:F

```
DATA students3;
     set students;
     if read>=90 then readgrade='A';
     if 80<=read<90 then readgrade='B';
     if 70<=read<80 then readgrade='C';
     if 60<=read<70 then readgrade='D';
     if read<60 then readgrade='F';
RUN;

PROC PRINT data=students3;
RUN;
```

Create new variables - modify variables

Example 2.9 Created total and totalgrade in one DATA step.

```
DATA students4;
     set students;
     total=read+write;
     if total>=120 then totalgrade='PASS';
     else totalgrade='FAIL';
RUN;

PROC PRINT data=students4;
RUN;
```

Delete variables - modify variables

- DROP statement: delete variables following the DROP keyword and keep all the remaining variables.

  ```
  DROP var1 var2,..., varN;
  ```

- KEEP statement: keep variables following the KEEP keyword and delete all the remaining variables.

  ```
  KEEP var1 var2,..., varN;
  ```

Note: DROP and KEEP cannot be used simultaneously in the same DATA step.

DROP and KEEP statements - delete variables - modify variables

Example 2.10 Delete the variable read in students dataset.

```
DATA studentsdrop;
      set students;
     drop read;
RUN;

PROC PRINT data=studentsdrop;
RUN;

DATA studentskeep;
     set students;
     keep id male schtype write;
RUN;

PROC PRINT data=studentskeep; RUN;
```

Renaming variables - modify variables

RENAME statement: Change the variable name from old-name to new-name.

```
RENAME <old-name> = <new-name> ;
```

Example 2.11 Change the name of read to readscore.

```
DATA studentsrename;
     SET studentsl;
     RENAME read=read-score;
RUN;

PROC PRINT data=studentsrename;
RUN;
```

Obtain a subset of a dataset

The IF...THEN...; ELSE ....; statement can also be used to select a subset of a SAS data set.

Example 2.12 Get a subset of students dataset of all female=1.

```
DATA Girls;
     SET students;
     IF female=1;  /*Equivalent to IF female=0 then DELETE;*/
RUN;

PROC PRINT data=Girls;
RUN;
```

Split into several subsets - obtain a subset of a dataset

Example 2.13 Split students dataset into Boys and Girls subsets.

```
DATA Boys Girls; /*Create two datasets */
     SET students;
     IF female=1 THEN OUTPUT Girls;
     ELSE OUTPUT Boys;
RUN;

PROC PRINT DATA=Girls;
RUN;

PROC PRINT DATA=Boys;
RUN;
```

PROC SORT - combine datasets

Before we discuss combining datasets, let's first look at a useful procedure: PROC SORT. PROC SORT tells SAS to sort the data according to the order of one or more variables.

PROC SORT general form

```
PROC SORT DATA=dataset-1 OUT=dataset-2 NODUPKEY;
BY variable-1 descending variable-2 ... ;
```

BY statement - PROC SORT - combine datasets

- The variables listed in the BY statement are called BY variables.
- In a SORT procedure, the dataset is sorted according to the values of the BY variable(s). If there is only one BY variable, then the dataset is ordered according to the values of this variable.
- If there are multiple BY variables, the dataset is first sorted according to the first BY variable, then that list is further sorted by the second variable within the categories of the first variable, etc.
- The optional keyword "descending" tells SAS to sort in a descenting manner rather than the default (ascending).

PROC SORT - combine datasets

- Two options are often used for PROC SORT:
  - "DATA=<dataset-1>" indicates which dataset to SORT.
  - "OUT=<dataset-2>" indicates the name of the sorted dataset
    If name is omitted, the original dataset will be replaced by the sorted dataset.
- The optional "NODUPKEY" option deletes duplicate observations with the same values of BY variables.

PROC SORT - combine datasets

Example 2.14 Sort marine dataset rstly by Family and then by length.

```
DATA marine;
     INPUT Name $ Family $ Length @@;
     Datalines;
     Beluga whale 15 Whale shark 40 Basking shark 30
     Gray whale 50 Mako shark 12 Sperm whale 60
     Dwarf shark .5 Whale shark 40 Humpback . 50
     Blue whale 100 Killer whale 30;
Run;

PROC SORT DATA=marine OUT=seasort NODUPKEY;
     BY Family DESCENDING Length;
Run;

PROC PRINT DATA=seasort;
Run;
```

Two datasets - combine datasets

```
DATA YR1998;
     INPUT Day $ 1-5 Temp Weather $ 10-15;
     DATALINES;
     Jan 1 33 sun
     Jan 2 35 sun
     Jan 3 45 clouds
     Jan 5 28 snow; RUN;

DATA YR1999;
     INPUT Day $ 1-5 Temp;
     DATALINES;
     Jan 1 25
     Jan 2 20
     Jan 3 35
     Jan 6 40
     Jan 8 39; RUN;
```

Combine datasets

There are four ways to combine datasets in SAS

- Concatenating
- Interleaving
- One-to-one merging
- Match merging

Concatenating - combine datasets

Example 2.15 Concatenate two datasets

```
DATA combined1;
     SET YR1998 YR1999;
RUN;

PROC PRINT data=combined1;
RUN;
```

- We use the SET statement (without a BY statement) followed by the dataset names. The number of observations in the final data set is the total number of observations from each of the individual datasets.
- The observations in the first data set listed are read first, followed by the second, and so on.
- The final dataset will have all the variables from individual datasets. Variable with the same name will be listed only once.

Interleaving - combine datasets

Suppose we want to see the weather records for years 1998 and 1999 ordered by date. That is, we want to see all available data (both years) for January 1st (if the record is available), then all the data for January 2nd, etc.

Example 2.16 Interleave two datasets

```
PROC SORT data=YR1998;
     BY Day; RUN;
PROC SORT data=YR1999;
     BY Day; RUN;

DATA combined2;
     SET YR1998 YR1999;
     BY Day; RUN;

PROC PRINT data=combined2; RUN;
```

Interleaving - combine datasets

- Use a SET statement with a BY statement.
- Interleaved data set has the same number of observations as concatenated data set, but the order is different.
- In order to use a BY statement, each individual data set must first be sorted by the same variable.
- When an observation from the first data set has a same By-variable value as another observation from the second data set, the one from the first data set is listed first in the combined data.

One-to-one merge - combine datasets

Suppose we want to merge the first observation from dataset-1 with the first observation from dataset-2, the second with the second, all the way to the end of the dataset with more observations.

Example2.16 Merging one-to one

```
DATA combined3;
     MERGE YR1998 YR1999;
RUN;

PROC PRINT data=combined3;
RUN;
```

One-to-one merge - combine datasets

- Use a MERGE statement without a BY statement.
- The number of observations in the final data set is the same as the number of observations in the data set with the most observations.
- The number of variables is the total number of variables in the datasets minus the number of overlapping variables.
- If two variables from different datasets have the same name, then the values from the second dataset replace the values from the first dataset.

Match merge - combine datasets

Suppose we want to merge the weather records of the same day from 1998 and 1999.

Example 2.17 Matched merging

```
DATA combined4;
     MERGE YR1998 YR1999;
     BY Day;
RUN;

PROC PRINT data=combined4;
RUN;
```

Match merge - combine datasets

- Use a MERGE statement with a BY statement.
- The individual datasets need to be sorted according to the BY variable first.
- Observations with the same value for the BY variable are merged in the same way as in one-to-one merging:
  - The final dataset contains all variables from each of the individual datasets.
  - If a variable appears in both datasets, the value from the second dataset in the MERGE statement replaces the value from the first dataset.
  - There may be missing values in the final dataset. For example, when there are variables that appear only in the first dataset, but for the particular BY variable value, only the second dataset has a corresponding observation(s).