Writing Programs in SAS
Data I/O in SAS

Statistics 135

Autumn 2005

Writing SAS Programs

Your **SAS** programs can be written in any text editor, though you will often want to the built in editor in **SAS**. Whatever you do it in, you will want to create an ASCII text file (like you needed with LaTeX).

When writing **SAS** programs, letter case doesn't matter

```
PROC UNIVARIATE;
proc univariate;
PrOc UnIvArIaTe;
```

all do the same thing.

I suggest that you work with a consistent pattern. A couple of possibilities are

- Commands in uppercase, variables in lowercase (I'll try to stick to this one)
- All lowercase
- All uppercase (probably suboptimal)

I also suggest that you comment your code There are two formats for comments

```
* comments

/* comments */       {c style}
```

There is one important thing to remember when writing **SAS** commands. All commands must end with a '`;`' (like in c). This allows for commands to be split across lines, which can help with readability.

Also to help with readability, you might want to put lines between the different `PROC` blocks.

Output Formatting

There are a number of `OPTIONS` that can be set in **SAS** which affect the output formatting. These include

- `CENTER | NOCENTER`: Output is either centered or left justified. (Default = CENTER)
- `DATE | NODATE`: Should today's date appear at the top of each page. (Default = DATE)
- `NUMBER | NONUMBER`: Should the output pages be numbered or not. (Default = NUMBER)
- `LINESIZE =` n: The maximum width of output lines (range = 64 to 256). I suggest that you set this something less than 80 as this will make it easier to plug the output into other documents. (I like 75). You may need to play with this depending on the margins you like to use and your font choice in your document processor. (Default = 109)

- `PAGESIZE = ` n: The maximum number of lines per page of output (range $=$ 15 to 32767). (Default can vary)
- `PAGENO = ` n: Starts numbering output pages at n. (Default is 1)
- `YEARCUTOFF = ` $yyyy$: Specifies the first year in a hundred-year span for interpreting two-digit dates.

So to left justify your output and set the pagesize to 50, you can use the command

```
    OPTIONS NOCENTER PAGESIZE=50;
```

If you wish you can split the command across multiple commands

```
    OPTIONS NOCENTER;
    OPTIONS PAGESIZE=50;
```

Data Entry

SAS can read in data from a wide range of formats, including text files (space, comma, or tab delimited), Excel, Access, DBF, Lotus 1-2-3, etc.

There are two common approaches to inputing data, a `DATA` step or `PROC IMPORT`.

- `DATA` step:

  This can be used for a wide range of text files. Often will be used with space delimited files (the example last class was), but can be used with fixed format files and files with other delimiters.

  The data file being read in should not include the variables names (as you can do in **S**). Instead they are given in the `INPUT` part of the `DATA` step. For example, the file from last time (now named `margarine.dat`) looks like

```
1 167
1 171
1 178
1 175
1 184
1 176

and so on
```

Space delimited files, like the above, can be read in with

```
DATA margarine;
  INFILE 'margarine.dat';
  INPUT brand time;
```

This will create a **SAS** dataset named `margarine` containing two numeric variables `brand` and `time`.

Now suppose that the data set for the example last time (now in a file `margarine.dat`) had brand in columns 1-3 and time was in columns 4-8. This can be read in with

```
DATA margarine;
  INFILE 'margarine.dat';
  INPUT brand 1-3 time 4-8;
```

You can mix variables occurring in fixed columns with others delimited by space as follows

```
DATA margarine;
  INFILE 'margarine.dat';
  INPUT brand 1-3 time;
```

This would get brand from the first 3 columns and would start looking in column 4 to find time, but with no restriction on where it ends.

Being able to state which columns is an artifact when storage was extremely limited or entry was done by punch cards, which are limited to 80 columns. Being able to remove spaces allows you to get more data in a limited space. Today you don't want to do it, as storage usually isn't a problem and trying to read files like this is tough.

To use a different delimiter, use the `DLM` option to `INFILE`. A couple of possibilities are

```
INFILE 'margarine.dat' DLM = ',';    /* comma delimiter */
INFILE 'margarine.dat' DLM = '09'x;  /* tab delimiter */
```

The tab character is character number 9 is ASCII. The `x` indicates that the character is being indicated by a hex code.

String variables can be read into **SAS** but they must be indicated by a `$`. For example, to read in string variables `country` and `government` and numeric variable gpa, use the command

```
INPUT country $ gpa government $ ;
```

It is also possible to include your data as part your **SAS** program. This can be done with the `DATALINES` command in a `DATA` step as follows

```
DATA uspresidents;
  INPUT president $ party $ number;
  DATALINES;
  Adams     F 2
  Lincoln   R 16
  Grant     R 18
  Kennedy   D 35
    ;
RUN;
```

This is something you probably don't want to do often, but it is nice to be able to do this with small data sets occasionally.

- `PROC IMPORT`:

  This approach can be used for non-text files and text files with the variable names in the first row.

  To read in a version of the margarine data, but with the variable names in the first row, the following approach can be used

  ```
  PROC IMPORT DATAFILE = 'margarine2.dat'
    OUT = margarine DBMS = dlm REPLACE;
  RUN;
  ```

  In this set of commands, the space delimited file `margarine2.dat` is read in and a data set `margarine` is created. The option `DBMS = dlm` indicates that a space delimited file is being entered. The `REPLACE` option says to overwrite the data set `margarine` if it exists.

Other possibilities for the `DBMS` option include

- `tab`: tab delimited file
- `cvs`: comma delimited file
- `excel`: Excel file
- `dbf`: dBase 5.0, IV, III+, and III files

This option is not needed if the data file name has the standard file extension (e.g. `*.xls` for Excel, `*.txt` for tab delimited, or `*.mdb` for Microsoft Access).

Data Export

There are two common approaches to exporting data, a `DATA` step or `PROC EXPORT`.

- `DATA` step:

  This is one approach to writing out your data as a text file. An example is the following

```
DATA _NULL_;
  /* loads in SAS datafile with desired variables */
  SET finaldata;
  /* give output file name */
  FILE 'margarineout.dat';
  /* variables to write to file */
  PUT brand time invtime pred z nscores;
RUN;
```

This will write the variables in the the file `margarineout.dat` with each variables separated by a space. Other delimiters can be used with the `DLM` option.

The `DATA` step can be used to create **SAS**binary data files. For example

```
DATA 'margarine2';
  SET finaldata;
RUN;
```

will create a data file `margarine2.sas7bdat`. The quotes in the `DATA` step indicate that this should be a permanent file and not a temporary one (i.e. one that will disappear when **SAS** is quit.

To access a permanent file, commands like the following can be used.

```
DATA test2;
  SET 'margarine2';
RUN;
```

- `PROC EXPORT`:

  The is effectively the opposite of `PROC IMPORT`, taking most of the same options. The main differences `OUTFILE` is used instead of `DATAFILE` and `DATA` is used instead of `OUT`.

  One example is

  ```
  PROC EXPORT OUTFILE = 'margarineout2.txt'
    DATA = finaldata  DBMS = tab;
  RUN;
  ```

  This will create a tab delimited file containing all the variables in the datafile `finaldata`.

  `PROC EXPORT` will create all the file types that `PROC IMPORT` will read in, such as Excel, Lotus 1-2-3, or JMP.

Manipulating Data

In a `DATA` step it is possible to manipulate a data set. For example, last time the program included

```
DATA temp;
  INFILE 'p147.3';
  INPUT brand time;
  invtime = 1/time;
```

This created a new variable `invtime` and added it to the variables read in from the data file `p147.3`.

The are a wide range of functions (over 400) that can be used to alter datafiles. They are in the areas of

- Character
- Date and Time

- Financial
- Macro
- Mathematical
- Probability
- Random numbers
- Sample statistics
- State and ZIP Code

One list of commonly used functions is in section 3.3 of Delwiche and Slaughter.

The following example gives a feeling of what can be done in a **SAS** `DATA` step

```
DATA homegarden;
  INPUT name $ tomato zucchini peas grapes;
  DATALINES;
  Gregor 10  2 40    0
  Molly  15  5 10 1000
  Luther 50 10 15   50
  Susan  20  0  .   20
    ;

  zone = 14;
  type = 'home';
  zucchini = zucchini * 10;
  total = tomato + zucchini + peas + grapes;
  pertom = (tomato / total) * 100;
RUN;
```

So it is possible to create new variables and reassign already created variables. The **SAS** missing data code is `'.'`, as was used for Susan's peas value.