SAS Programming

Style and Conventions

12/7/2021

SAS Programming and Conventions

First rule of programming: PROGRAMS LIVE

e.g.,

- Rerun/reuse next year
- Extended/updated
- Framework for future programs
- Delivered to client

➔ Write clean, clear, concise code.

- ➔ (Corollary: being clever is not necessarily being smart)
- ➔ (Corollary: 'Quick and Dirty' may be dirty, but not necessarily quick)

The Basics

Always use methodical indentation

Comment code (an art – not too much, not too little)

One SAS statement / line

Use blank lines to create logical separation between sections of code

SAS Log

SAS Log should be clean, free from all warnings.

You never know for sure if the log is trying to tell you something . . .

Especially if rerunning something after a period of time;

and . . .

We often deliver code, which should reflect well on Insight.

SAS Log

Avoid all conversion warnings

Character to Numeric

Numeric to Character

- Know your data types
  - Use the PUT to avoid 'Numeric to Character' warnings
  - Use the INPUT function for 'Character to Numeric' warnings

[example interactive program]

SAS Log

Eliminate 'Missing Values Generated' Warnings.

Use Conditional Statements

[example interactive program]

SAS Log

Handle 'Invalid Data' Warnings

Use Conditional Statements

[example interactive program]

Keep Datasets Clean

Unclean datasets can be an insidious source of errors.

- Drop temporary variables
- Keep only variables that are needed
- Ensure the proper data types (character vs. numeric)
- Ensure variables have the proper length
- Format dates (permanent formats)
- Label all variables

Keep Datasets Clean

Example:

```
Data a(drop=i);
  array _X X1-X10
  array _XSquare XSquare1-XSquare10;
  set b;
  do i=1 to 10;
    Xsquare(i)=X(i)**2;
  end;
run;
```

Compile-Time Statements

Should be placed at the top of the DATA step (convention)

- Label
- Retain
- Length – worthless if after first reference
- Drop/keep
- Rename

PROGRAM STRUCTURE

Define the entire physical environment at the top of the program

- Options
- Input data files and libraries
- Output data libraries and listing files
- Code libraries
- Key macro variables

➔ Easier to maintain, debug and relocate.

[example program structure]

Constants

Define constants:

Unclear:

```
NewAmount=OldAmount*1.0343
```

Clear:

```
%let InflationRate=.0343;  /* EASY TO MAINTAIN IF AT TOP OF PROGRAM */
. . .
NewAmount=OldAmount*(1+&InflationRate)
```

TITLE Statements

For larger programs with many tables of output:

- Define overall TITLE statements at the top of the program

  e.g., TITLE1, maybe TITLE2

- Individual reports use lower TITLE statements, e.g.,

```
TITLE '2020 Final Report';  /* AT TOP OF PROGRAM */
…
PROC PRINT DATA=A;
        TITLE2 'By Gender';
PROC PRINT DATA=B;
        TITLE2 'By Race';
```

Bad Merge

If you see this message:

"MERGE statement has more than one data set with repeats of BY values"

FIX IT – 'nuff said'

Limit Number of Steps

For performance: each step consists of a pass through one or more datasets → increased I/O.

For readability: Generally, one well written DATA step is more readable than multiple steps accomplishing the same thing.

Limit Number of Steps (cont.)

Avoid (if all datasets not needed):

```
Data A;
  set B;
  …..
Run;

Data C;
  set A;
  …..
Run;
```

Avoid Plethora of Datasets

Avoid – unless you really need to retain the original A:

```
Proc sort data=A out=B;
  by X;
Run;
```

Better:

```
Proc sort data=A;
  by X;
Run;
```

Avoid Temporary Variables

Unnecessary temporary variables can reduce program clarity.

For example: BMI calculation (Kilograms/Meters-Squared)

Avoid

```
Data A;
 set B;
 tempM=HeightFt*.305;
 tempMsq=tempM**2;
 tWgt=WeightLb*.454;
 BMI=tWgt/tempMsq;
```

Better

```
Data A;
 set B;
 BMI=(WeightLb*.454)/(HeightFT*.305)**2;
```

Summary

Follow simple conventions:

- Write clean, clear, concise program.
- Write code to be understood.
- Do not be afraid to rewrite or refine unclear code.
- Does not take any more effort to write good code than 'quick and dirty'.
- Well written, clean, clear, and concise programs will be easier to debug and maintain.