Seeing the Forest for the Trees: Part Deux of Defensive Coding by Example

Nancy Brucken and Donna Levy, Syneos Health

June 20, 2018
Seeing the Forest for the Trees: Part Deux of Defensive Coding by Example

- Donna Levy has worked in oncology statistics in academic, consulting and pharmaceutical settings, and has been a SAS user for 20 years. She is a member of the Biostatistical Consulting Group at Syneos Health and an ASA Kentucky Chapter elected official. She has presented SAS and statistical topics at BASUG, NESUG & SCT. She has learned a lot of her SAS skills from colleagues as well as attending local and regional professional meetings. She is also an avid Habs fan (go Habs go!).

- Nancy Brucken has been a SAS programmer for over 25 years, more than 20 of which have been spent in the pharmaceutical industry. She has been a frequent presenter at Regional & Local SAS User Group meetings, and is a proud graduate of Marietta College and a devout Ohio State fan.
Introduction

Identifying Your Tools
Areas of Focus

• Efficient coding
  – Improving code quality and productivity

• Good programming
  – Increasing code readability
  – Improving SAS skills

• Programming no-no’s
  – No need for a description….
Efficient Coding

Make a packing list. Talk to the paratroopers. Walk away.
Efficient Coding

- It is still important to write programs that make good use of both programmer and machine resources
- Minimize passes through datasets
- Modularize and document code
Use of Modular Code

• Make a “packing list”
  – Organize your thoughts and your program before you start coding
  – Break code into logical chunks / modules
  – Use pseudo code for planning
  – Modules can often be reused across multiple programs
  – Step back and see the big picture!
%MACRO RandExp(sigma);
    ((&sigma) * RAND("Exponential"))
%MEND;

Can be reused for time to event simulations for the time and censoring variables

Minimize Passes Through Data

- Create multi-purpose DATA steps
- Only sort data when necessary
- Use WHERE and KEEP statements to remove extra baggage
- Pay attention to placement of WHERE and KEEP statements
- Summarize as early as possible
Make Use of SAS Features

• Take advantage of BY-group processing
  – Many tables/reports can be generated with a single PROC UNIVARIATE (or PROC MEANS) call that includes all variables, run BY treatment arm and any subgroup variables
  – Transpose results as needed
  – Much faster and easier to debug than programming 1 row/column at a time via macro variables
Minimize I/O

• Reading data from disk storage is often one of the slowest parts of a program
• Structure your programs to read the data in once (subsetting to desired records and variables)
Good Programming Concepts

Be prepared: Appropriately pack the needed gear
Keep Your Equipment Current

• Be Prepared
• Use it or lose it
  – Not applicable
• Build your library of tools
  – Keep exposing yourself to new methods
  – May not remember everything
  – Remember a snippet (that is hopefully searchable)
  – Keep those tools sharp
(New versus Old) and (New and Old)

- RAND & RANDGEN
  - Improved statistical properties
  - Including a longer period
  - Improved true randomness when compared to RANUNI (Wicklin, 2013b)

- PROC GENMOD versus GEE
  - Missingness a concern
    - GEE

- GENMOD
  - Recently added goodness of fit test

Thank you for the Default – No thank you

• Nice that SAS has defaults that do not need to be specified
  – But should specify
  – Defaults can change
  – Code can change

Example 3.A (no defaults specified):
PROC GLM;
class var1;
  MODEL outvar = invar1 invar2 / CLM P;
RUN;

Example 3.B (with defaults specified):
PROC GLM DATA=InData ALPHA=0.05
ORDER=FORMATTED;
  CLASS var1;
  MODEL outvar = invar1 invar2 / CLM P;
RUN; CLASS
Items that you Should Over Pack

- Comments are Free
- Have you ever heard someone say “This person has too many comments in their code?”
Items that you Should Over Pack con’t

• Comments are Free
• Comment. Comment. Comment.
  – For your future self
    • Put it down and come back
    • Are you really going to remember that?

  – For whoever inherits your code
  – Big red bow
Items that you Should Over Pack con’t

• Comments are Free
• Did we mention?
  – Comment. Comment. Comment.

• Indent and white space
  – Aid in readability

• Three big red bows
Programming No-Nos

Do not feed the bears
Overwriting Datasets

Wearing the same wet socks over and over

- We spoke about this in our last paper but important enough to say it again
- Rename datasets as you go
  - Your future self will thank you
    - Debugging
- Naming datasets in a meaningful way
  - WORK6 versus WORK06
  - WORK6 versus AVISTUM01
Overwriting Datasets

Wearing the same wet socks over and over

- We spoke about this in our last paper but important enough to say it again
- Rename datasets as you go
  - Your future self will thank you
    - Debugging
- Naming datasets in a meaningful way
  - WORK6 versus WORK06
  - WORK6 versus AVISTUM01
Appending Changes

No No No No

• The assigned statistician was taught
  – When you are working on a program that is fully functioning
  – An update is needed
  – You make the update at the end of the code

• Why?
  – Not messing with code that is working.

• While we understand the logic behind this suggestion, there are multiple issues with this programming technique.

• Remember your future self.
• Remember who will inherit your code.
Appending Changes con’t

No No No No

• Example 4.A (update at the end):
  DATA bad;
  SET x;
  IF age NE . AND age LE 18 THEN
  agecat=”LE 18 years”;
  ELSE IF age GT 18 THEN agecat=”GT 18 years”;
  RUN;
  *** other code ***;
  DATA final;
  SET almost;
  IF age NE . AND age LE 10 THEN
  agecat=”LE 10 years”;

DATA MuchBetter;
  SET x;
  IF age NE . AND age LE 10 THEN
  agecat=”LE 10 years”;
  ELSE IF age NE . AND age LE 18 THEN
  agecat=”LE 18 years”;
  ELSE IF age GT 18 THEN
  agecat=”GT 18 years”;
  RUN;
  .
Appending Changes con’t

No No No No No

• Running the code over and over again
• Debugging over and over again
• Energy and frustration would probably be saved by putting the updated code in the correct and logical place
  – where the original age variable was developed.
• Keep related code together
• This is not a shortcut

  • The safest route to quality programming code. While the code provided is a simple example, the more complex the code, the greater the importance of grouping related variables for the short and long term.
Beta from a Statistician

Watch the weather. Follow the map. Call in the Park Rangers.
Call in the Park Rangers

• Checked your toolbox?
• Googled it?
• Stop circling the trail
• Ask for beta to get over the crux
  – Talk through the problem
  – Sometimes that is enough
Call in the Park Rangers

- Checked your toolbox?
- Googled it?
- Stop circling the trail
- Ask for beta to get over the crux
  - Talk through the problem
  - Sometimes that is enough

Do not forget that sometimes you should walk away (but of course eventually come back)
Beware of Merging Blindly

- The Only SQL I know…merging many to many

```sql
data work.patient_treatment;
  input patientid treatment $ date_treatment :mmddyy10.;
  format date_treatment mmddyy10.;
  datalines;
  1 A 1/5/2018
  1 A 2/5/2018
; run;

data work.treatment_drug;
  input treatment $ drug $;
  datalines;
  A DRUG1
  A DRUG2
; run;
```

```sql
**** Example code 5.A;

data work.merged;
merge work.patient_treatment work.treatment_drug;
  by treatment;
run;
title "With merge";
proc print data=work.merged; run;
```

Output 1 shows the output from example code 5.A

<table>
<thead>
<tr>
<th>Obs</th>
<th>patientid</th>
<th>treatment</th>
<th>date_</th>
<th>drug</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>A</td>
<td>01/05/2018</td>
<td>DRUG1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>A</td>
<td>02/05/2018</td>
<td>DRUG2</td>
</tr>
</tbody>
</table>
Beware of Merging Blindly con’t

• The Only SQL I know…merging many to many

*** Example code 5.B;
title "With Proc SQL";
proc sql;
    select a.patientid, a.treatment,
        a.date_treatment, b.drug
    from work.patient_treatment a,
        work.treatment_drug b
    where a.treatment=b.treatment;
quit;

<table>
<thead>
<tr>
<th>patientid</th>
<th>treatment</th>
<th>date_treatment</th>
<th>drug</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>01/05/2018</td>
<td>DRUG1</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>01/05/2018</td>
<td>DRUG2</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>02/05/2018</td>
<td>DRUG1</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>02/05/2018</td>
<td>DRUG2</td>
</tr>
</tbody>
</table>

Output 2 shows the output from example code 5.B
And from part 1
Conclusions

Conclusions

• We hope you never get lost in the woods
  – Plan your route (pseudo code)
  – Keep your feet on the trail
• Keep your tools sharp
  – Keep expanding your SAS skills and knowledge
• Avoid the No nos
Contact Information

Nancy Brucken
Syneos Health
Nancy.Brucken@syneoshealth.com

Donna Levy
Syneos Health
Donna.Levy@syneoshealth.com
Extra Slides
References


- Wicklin, R. 2013a. Simulating data with SAS. SAS Institute, Cary, NC.

Plan your Hike

• **Be Prepared**
  • Like camping and hiking, we should plan our code before we start writing our code

• Planning helps you
  – Process where you are
  – Process where you are going
  – Get the correct sequence of coding and variables
Plan your Hike con’t

• **Be Prepared**
  - Pseudo code
  - Includes some details
    - What plan to pack?
    - Plan to pack based on identified needs?
    - Plan to pack based on planned hike path
      - Need to take the right path to see the sites
  - Know where you are going and what you want to see
- Programming plan of attack
- Programming methods