Defensive Coding by Example

Kick the Tires, Pump the Brakes, Check Your Blind Spots, and Merge Ahead!

Nancy Brucken & Donna E. Levy
Inventiv Health
Overview

- Motivation
- Expect the unexpected
- Analysis data set requirements
- Out of range and empty data sets
- Checkpoint code
- Missingness
- Efficiency, automation versus manual
- Good programming practices
- Programming no no’s
Driver’s Education

Introduction
Motivation

- Over the years, collected experiences
  - Some good
  - Some not so good

- Periodically always good to have a reminder as to why we do what we do
  - Is there an alternative method?
  - Is there a better way?
The Only Way?

- Presenting concepts
  - May be other methods
  - May be better methods

- Simply providing alternatives

- Many options
  - Better way
    - Why
  - More efficient
  - Robust

Expect the unexpected
Introduction and Objectives

- As presenting
  - Think about what you have seen
  - Think about what you have done
  - Think about what we have done
  - Is there another quality alternative?
Expect the Unexpected

Defending Your Code Against Data
Expect the Unexpected

- Avoid being flattened by oncoming data
  - Immature, incomplete, incorrect data

- But how do we know what to expect?
  - Will not predict everything
    » Code will not be bullet proof
  - But we know the data will not be perfect
    » Common data issues
    » Can be proactive to protect for many issues
Analysis Data Set Requirements

- Instructions for the programmer
- Descriptions for regulatory review
- Describe the final product, including:
  - Variables and attributes
  - Derivations
  - Analytical procedures and options
- Algorithms to produce the final product
Analysis Data Set Requirements con’t

- But developed by humans
  - If only we were perfect

- Data structure a learning process for all team members
  - Statistics and Programming

- Evolving, breathing, living document
  - Evolving datasets
Analysis Data Set Requirements con’t

- Need for critical thinking
  - Just because it says it…..
  - Ask questions
  - Do not push errors downstream
    - More difficult to catch
    - More expensive to fix
    - If they are caught
- Do not just follow the map
  - Contribute to the path taken
Example 1: Analysis Data Set Requirements

- Set to 1 if RACE='WHITE';
- set to 2 if RACE='BLACK OR AFRICAN AMERICAN';
- set to 3 if RACE='ASIAN';
- set to 4 if RACE='AMERICAN INDIAN OR ALASKA NATIVE';
- set to 5 if RACE='NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER'.
Example 1: Analysis Data Set Requirements con’t

<table>
<thead>
<tr>
<th>USUBJID</th>
<th>RACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-003</td>
<td>white</td>
</tr>
<tr>
<td>01-004</td>
<td>BLACK OR AFRICAN AMERICAN</td>
</tr>
<tr>
<td>01-005</td>
<td>OTHER</td>
</tr>
</tbody>
</table>
Example 1: Analysis Data Set Requirements con’t

Following the requirements, you might get this:

data example1_0;
set old;
if race='WHITE' then racen = 1;
else if race='BLACK OR AFRICAN AMERICAN' then racen = 2;
else if race='ASIAN' then racen = 3;
else if race='AMERICAN INDIAN OR ALASKAN NATIVE' then racen = 4;
else if race='NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER' then racen = 5;
run;

<table>
<thead>
<tr>
<th>USUBJID</th>
<th>RACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-003</td>
<td>white</td>
</tr>
<tr>
<td>01-004</td>
<td>BLACK OR AFRICAN AMERICAN</td>
</tr>
<tr>
<td>01-005</td>
<td>OTHER</td>
</tr>
</tbody>
</table>
Example 1: Analysis Data Set Requirements con’t

<table>
<thead>
<tr>
<th>USUBJID</th>
<th>RACE</th>
<th>RACEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-003</td>
<td>white</td>
<td></td>
</tr>
<tr>
<td>01-004</td>
<td>BLACK OR AFRICAN AMERICAN</td>
<td>2</td>
</tr>
<tr>
<td>01-005</td>
<td>OTHER</td>
<td></td>
</tr>
</tbody>
</table>
Example 1: Analysis Data Set Requirements con’t

- Improved SAS coding:

```sas
data example 1_1;
  set old;
  length racet $ 50;
  racet = upcase(race);
  if racet='WHITE' then racen = 1;
  else if racet='BLACK OR AFRICAN AMERICAN' then racen = 2;
  else if racet='ASIAN' then racen = 3;
  else if racet='AMERICAN INDIAN OR ALASKAN NATIVE' then racen = 4;
  else if racet='NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER'
    then racen = 5;
run;
```
Example 1: Analysis Data Set Requirements con’t

<table>
<thead>
<tr>
<th>USUBJID</th>
<th>RACE</th>
<th>RACEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-003</td>
<td>white</td>
<td>1</td>
</tr>
<tr>
<td>01-004</td>
<td>BLACK OR AFRICAN AMERICAN</td>
<td>2</td>
</tr>
<tr>
<td>01-005</td>
<td>OTHER</td>
<td></td>
</tr>
</tbody>
</table>
Example 1: Analysis Data Set Requirements con’t

- More Improved SAS coding:

```sas
data example 1_1;
  set old;
  length racet $ 50;
  racet = upcase(race);
  if racet='WHITE' then racen = 1;
  else if racet='BLACK OR AFRICAN AMERICAN' then racen = 2;
  else if racet='ASIAN' then racen = 3;
  else if racet='AMERICAN INDIAN OR ALASKAN NATIVE' then racen = 4;
  else if racet='NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER'
    then racen = 5;
  else put 'WAR' 'NING: Unexpected RACE value- Patient=' usubjid ' race=' race;
run;
```
Example 1: Analysis Data Set Requirements con’t

- More Improved SAS coding:

```sas
data example 1_1;
  set old;
  length racet $ 50;
  racet = upcase(race);
  if racet='WHITE' then racen = 1;
  else if racet='BLACK OR AFRICAN AMERICAN' then racen = 2;
  else if racet='ASIAN' then racen = 3;
  else if racet='AMERICAN INDIAN OR ALASKAN NATIVE' then racen = 4;
  else if racet='NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER' then racen = 5;
  else put 'WAR' 'NING: Unexpected RACE value- Patient=' usubjid ', race=' race;
run;
```

In your log, you would see:

```
WARNING: Unexpected RACE value- Patient= 01-005, race= OTHER
```

(If there are no unexpected values, the PUT statement will not trigger log checker warnings)
Example 1: Analysis Data Set Requirements con’t

<table>
<thead>
<tr>
<th>USUBJID</th>
<th>RACE</th>
<th>RACEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-003</td>
<td>white</td>
<td>1</td>
</tr>
<tr>
<td>01-004</td>
<td>BLACK OR AFRICAN AMERICAN</td>
<td>2</td>
</tr>
<tr>
<td>01-005</td>
<td>OTHER</td>
<td></td>
</tr>
</tbody>
</table>

Plus warning in log for patient 01-005 indicating that the patient did not fall into a category
Example 1: Analysis Data Set Requirements con’t

- Did we mention how important it is to check your logs?
Example 1: Analysis Data Set Requirements con’t

- Can we improve the code further?

- On occasion
  - Leading or trailing blanks

- SAS TRIM, LEFT, STRIP and COMPRESS function
  - TRIM: To remove trailing blank spaces
  - LEFT: Left-justify a text string
  - STRIP: Equivalent to TRIM and LEFT
  - COMPRESS: Removes all blanks, including in between words
Example 1: Analysis Data Set Requirements con’t

data example 1_3;
  set old;
  length racet $ 50;
  racet = upcase(trim(left(race)));  *** or racet = upcase(strip(race));
  if racet='WHITE' then racen = 1;
  else if racet='BLACK OR AFRICAN AMERICAN' then racen = 2;
  else if racet='ASIAN' then racen = 3;
  else if racet='AMERICAN INDIAN OR ALASKAN NATIVE' then racen = 4;
  else if racet='NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER' then racen = 5;
  else put 'WAR NING: Unexpected RACE value- Patient ‘ usubjid ‘, race’
  race;
run;
Example 2: Translating Analysis Dataset Requirements

- The objective of the program is to set a baseline metabolic syndrome flag to ‘Y’ if three or more of the five component symptoms were present.
Example 2: Translating Analysis Dataset Requirements

- Here is the programmer code

```c
if (bmetsyn1="Y" and bmetsyn2="Y" and bmetsyn3="Y") then bmetflg= "Y";
if (bmetsyn1="Y" and bmetsyn2="Y" and bmetsyn4="Y") then bmetflg= "Y";
if (bmetsyn1="Y" and bmetsyn2="Y" and bmetsyn5="Y") then bmetflg= "Y";
if (bmetsyn1="Y" and bmetsyn3="Y" and bmetsyn4="Y") then bmetflg= "Y";
```

- Do you see any issues with the code?
- If so, can you make improvements?
Example 2: Translating Analysis Dataset Requirements con’t

- **Issues with the code**
  - Only enumerated the actual combinations found in the data at the time the program was written
    - 16 possible combinations
    - But only 4 combinations were coded

- New version of data
  - Many cases that should have been flagged were unfortunately missed
Example 2: Translating Analysis Dataset Requirements con’t

- Better option

```plaintext
if (bmetsyn1='Y') + (bmetsyn2='Y') + (bmetsyn3='Y') +
  (bmetsyn4='Y') + (bmetsyn5='Y') >= 3
then bmetflg = 'Y';
```
Example 2: Translating Analysis Dataset Requirements con’t

- **Better option**

```plaintext
if (bmetsyn1='Y') + (bmetsyn2='Y') + (bmetsyn3='Y') +
(bmetsyn4='Y') + (bmetsyn5='Y') >= 3
then bmetflg = 'Y';
```

- Boolean operator: True=1 or False=0
- (bmetsyn1='Y') = 1, IF bmetsyn1="Y", Boolean operator is true
- (bmetsyn1='Y') = 0, IF bmetsyn1 NE “Y”, Boolean operator is false
Example 2: Translating Analysis Dataset Requirements con’t

if (bmetsyn1='Y') + (bmetsyn2='Y') + (bmetsyn3='Y') +
   (bmetsyn4='Y') + (bmetsyn5='Y') \geq 3
then bmetflg = 'Y';

- (bmetsyn1='Y') = 1, IF bmetsyn1="Y"
- (bmetsyn1='Y') = 0, IF bmetsyn1 NE “Y”

- However be aware of the impact of missingness on your Boolean operator
  - (bmetsyn1 NE ’N’) may not be equivalent to (bmetsyn1 = ’Y’)
Example 2: Translating Analysis Dataset Requirements con’t

Better option

```plaintext
if (bmetsyn1='Y') + (bmetsyn2='Y') + (bmetsyn3='Y') +
    (bmetsyn4='Y') + (bmetsyn5='Y') >= 3
then bmetflg = 'Y';
```

** bmetflg is Yes if 3 or more of the 5 bmetsynX variables are TRUE=1;
Running Out of Gas

Out of Range Values
**Empty data set – Do not slam on the brakes**

- Maybe there are no records that meet the criteria for subset of data
  - Check the data
  - Check the code
- Proceed with caution
- Similar checks for empty output
Example 3: Empty Data Set

Should the output data set be empty?

data example_2_1;
  length renalc $ 3.;
  set myel;
  if renal=0 then renalc="No";
  else if renal=1 then renalc="Yes";
run;

proc print data=example_2_1 n noobs;
  var renal renalc;
  where renalc="NO";
run;
Example 3: Empty Data Set con’t

Improved code

- Here is a possible code improvement using the UPCASE option:

```sas
proc print data=example_2_1 n noobs;
  var renal renalc;
  where upcase(renalc)=”NO”;
run;
```
Example 3: Empty Data Set con’t

Improved code

- Here is a possible code improvement using the UPCASE option:

```sas
proc print data=example_2_1 n noobs;
  var renal renalc;
  where upcase(renalc)=”NO”;
run;
```

- As coding your data set
- Check your log
  - Review record counts closely
- Investigate any unexpected changes in counts
  - Too few or too many
  - Merge or join data sets
    » Multiple records per patient
  - Unexpected results many-to-many merge
Example 3: Empty Data Set con’t

- Did we mention how important it is to check your logs?
Example 4: Varying Lengths

- Know your data set
  - Look at your data
  - PROC FREQ

Nothing more dangerous than developing data set without understanding

- Input data
- How computed variables relate to each other
- Objective of what you are doing

Add a little defensive coding

- Improve the quality of code
- Improves the quality of data set
Example 5: Validated Therefore Validated?

If a data set has been validated, does it mean that the data set is validated?

- Input data set says “NO”
- Requirements say “No” instead of “NO”
- Both primary and validation use “No”
  - Miss issue with requirement
Example 5: Validated Therefore Validated?

If a data set has been validated, does it mean that the data set is validated?

- Data set says “NO”
- Requirements say “No” instead of “NO”
- Both primary and validation use “No”
  - Miss issue with requirement

Solution?

- Perform review of output data set
  - Truncation
  - Populated as expected
  - Eliminate any obvious issues
  - Known ranges
  - Excessive missingness
Handling Missing Values
Example 6: Do You See Any Issues?

if (date2 – date1 + 1) < 10 then catvar=”<10 days”;
else if (date2 – date1 + 1) < 30 then catvar=”<30 days”;}
Example 6: Do You See Any Issues?

```plaintext
if (date2 – date1 + 1) < 10 then catvar=’<10 days’;
else if (date2 – date1 + 1) < 30 then catvar=’<30 days’;
```

- What if a date is missing?
  - Mis-categorization is likely
Example 6 (cont.)

- Check to see if dates are missing
  - NMISS function returns number of missing values

```sas
if nmiss(date1, date2) ge 1 then catvar="Missing/Unknown";
else if nmiss(date1, date2)=0 then do;
  if (date2 – date1 + 1) < 10 then catvar="<10 days";
  else if (date2 – date1 + 1) < 30 then catvar="<30 days";
  end;
end;
```
Example 7: Missing Dates

- Here’s the code:

```sas
data mydata;
set olddata;
if datvar < trt01date then phase= “Pre”;
else if datvar >= trt01date then phase= “Post”;
run;
```

And the resulting data:

<table>
<thead>
<tr>
<th>Patient ID</th>
<th>TRT01DATE</th>
<th>DatVar</th>
<th>Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12DEC2000</td>
<td>01JAN2000</td>
<td>Pre</td>
</tr>
<tr>
<td>1</td>
<td>12DEC2000</td>
<td>01MAR2001</td>
<td>Post</td>
</tr>
<tr>
<td>1</td>
<td>12DEC2000</td>
<td>.</td>
<td>Pre</td>
</tr>
<tr>
<td>1</td>
<td>12DEC2000</td>
<td>17JUL2001</td>
<td>Post</td>
</tr>
<tr>
<td>2</td>
<td>.</td>
<td>21MAR2002</td>
<td>Post</td>
</tr>
<tr>
<td>2</td>
<td>.</td>
<td>01MAY2002</td>
<td>Post</td>
</tr>
</tbody>
</table>
Example 7 (cont.): Missing Dates

- Improved code:

```sas
data mydata;
set olddata;
if not missing(trt01date) and not missing(datvar) then do;
  if datvar < trt01date then phase = "Pre";
  else if datvar >= trt01date then phase = "Post";
end;
run;
```

Resulting data:

<table>
<thead>
<tr>
<th>Patient ID</th>
<th>TRT01DATE</th>
<th>DatVar</th>
<th>Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12DEC2000</td>
<td>01JAN2000</td>
<td>Pre</td>
</tr>
<tr>
<td>1</td>
<td>12DEC2000</td>
<td>01MAR2001</td>
<td>Post</td>
</tr>
<tr>
<td>1</td>
<td>12DEC2000</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>12DEC2000</td>
<td>17JUL2001</td>
<td>Post</td>
</tr>
<tr>
<td>2</td>
<td>.</td>
<td>21MAR2002</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>.</td>
<td>01MAY2002</td>
<td></td>
</tr>
</tbody>
</table>
Vehicle Maintenance

Defending Your Code Against Code
(or other programmers)
Example 8: Curves, Potholes, Stop Signs, Hills and U-Turns

- Developing code is not linear
  - Many stops and starts
- Develop code in preparation for the rough road ahead
- Develop code knowing someone will use/inherit your code

And other drivers!
Example 8: Curves, Potholes, Stop Signs, Hills and U-Turns

- How can we improve this code?

```sas
proc means sd mean min max;
  var var1;
  output out=SummOut;
run;
```
Example 8 (cont): Curves, Potholes, Stop Signs, Hills and U-Turns

- Explicitly specify the input data set
  - SAS does not require

```sas
*** Summary statistics for age;
proc means data=mydata sd mean min max;
  var age;
  output out=SummOut;
run;
```

- Also comments are extremely helpful
**Checkpoints**

- Intermediate datasets that you know are correct, and can be used to check subsequent calculations every time new data is run through the program
- Do not assume your calculations are correct - check them!
- Check early in the project, and every time new data comes in
- Add checking code throughout the program
Good Gas Mileage

Efficient Coding
Example 9: Requirements Versus the Code

Requirements
- Set Y to 1 if x < 10; set Y to 2 if 10 ≤ x < 20.

Resulting Code
- if x < 10 then y = 1;
- if x < 20 then y = 2;

- Any issues?
- Can you improve?
- If x=5, what does y equal?
Example 9 (cont): Requirements Versus the Code

Requirements

- Set $Y$ to 1 if $x < 10$; set $Y$ to 2 if $10 \leq x < 20$.

Better Code

```
if x < 10 then y = 1;
else if x < 20 then y = 2;
```

*If conditions in your requirements are mutually exclusive, take advantage of that in your code!*
Cruise Control

Automation
Manual Versus Automatic?

- What would you choose?
  - Cost more up front
    » How many times are you going to run the code over the duration of the study
  - Possibly less effort overall
    » Greater efficiency
  - Simplifies the process
  - Decreases validation time

- Not a replacement for a manual check and review
Wallpaper Code

- Repeating the same code over and over
  - Pretty patterns, but horrible to maintain
  - Use macros or BY-Group processing!
    » Less code to maintain
    » Update in one place
Example 10: Wallpaper Code

What would you do to improve this code?

```sas
title1 "Output 1: Var 1 by Var2";
proc freq data=mydata;
  tables var1*var2 / fisher;
  exact;
run;

title1 "Output 2: Var 2 by Var3";
proc freq data=mydata;
  tables var2*var3 / fisher;
  exact;
run;

title1 "Output 3: Var 1 by Var3";
proc freq data=mydata;
  tables var1*var3 / fisher;
  exact;
run;
```
**Example 10 (cont): Wallpaper Code**

Use a macro!!

```sas
%macro freqnum(inds=, num=, v1=, v2=);
    title1 "Output &num: &v1 by &v2";
    proc freq data=&inds;
        tables &v1*&v2 / fisher; *** line A;
        exact;
    run;
%mend freqnum;
```

```
%freqnum(inds=mydata, num=1, v1=var1, v2=var2);
%freqnum(inds=mydata, num=2, v1=var2, v2=var3);
%freqnum(inds=mydata, num=3, v1=var3, v2=var4);
```
Towing Capacity

Keep Only What You Need
Improve Your Aerodynamics

- If you do not need it – Drop It!
  - Improves processing time

- Consider the placement of a DROP or KEEP statement or option
  - Does it apply to input or to output dataset?
  - Put it somewhere that is easy to find
    » Top of the step
    » Consistent location
Checking Your Blind Spots

**Good Programming Concepts**
Plan your trip

- Think about what you are doing before you start
- Before each step, know:
  - Where you are- what does the input dataset look like?
  - Where you are going- what does the output look like?
Program Structure

- Program should have a logical flow
Program Structure (cont.)

- Program related variables together
  - Time-to-event code - time to event and censoring
  - Coded and decoded values

- Aim to derive each variable only once
  - If the derivation changes, will you be able to find all of the places in the code that need to be updated??
Readability — Elbow Room

- Which is more readable?

DATA X;
SET Y;
A=X+1;B=Y+1;
PROC SORT;BY A;RUN;

*** Increment A and B;
data addone;
set y;
a = x + 1;
b = y + 1;
run;

proc sort data=addone;
by a;
run;

White space and comments are free!
Read Your Log

- Even if the output looks good, read the log!
- Check for errors, warnings, notes
  - UNINITIALIZED
  - MERGE WITH MULTIPLE
  - LENGTH VARIABLE
  - ZERO RECORDS
- Automated log checkers are a good start
- Always check record counts!
File Organization – Glove Compartment

- Keep related files together
- Store LIBREFs and directory path definitions in one place
  - Initialization program or macro called by all other programs
  - Only need to update in 1 spot when new data arrives in a different location
**Batch Job – Cruise Control**

- Multiple deliveries of multiple programs are expected over time
- Order in which programs are run is important

- True batch job runs each program in an independent SAS session
  - Work datasets erased after each program
  - Macro variables are deleted after each program
Defensive Driving School

- Improve your skills
  - Attend SAS workshops, conferences, user group meetings
  - SAS certification programs
  - Read the “What’s New” documentation with new SAS releases
  - Online resources
    » SAS-L, sasCommunity.org, LinkedIn groups
    » Papers at www.lexjansen.com
- Share your knowledge
Running a Red Light

Programming No-No’s
No Hard-coding

- Data should be traceable
  - Source to analysis results
  - Analysis results to source

Fix it in the source data, not in the program!

If patient=123 and visit=2 then pulse = 72;
Don’t Reuse Data Set Names

- Which is easier to debug?

```
data aevents2;
  set aevents;
  ae = substr(var1);
  num + 1;
run;
```

```
data aevents3;
  set aevents2 demog;
  by subjid;
run;
```

But what if I need to add a step in between?
Meaningful Data Set Names

- Improves readability

```
data aeventct;
  set aevents;
  ae = substr(var1);
  num + 1;
run;

  data aedemog;
    merge aeventct demog;
    by subjid;
run;
```
Meaningful Variable Names

- Improves readability
  - Variable name X1 versus ae1
  - Variable name Y1 versus ae1_base

- Which variable name tells you more?
Conclusions

- As you are developing your code
  - Do not just go into autopilot
  - Think about what you are doing
    » How improve and develop your code as you go?
    » A little care in the beginning
    » Save you time in maintaining your code
    » Avoiding careless errors in the long run
Turn Your Program Into One of These!
Contact Information

Nancy Brucken
inVentiv Health
Ann Arbor, MI  48108
734-887-0255
Nancy.Brucken@inventivhealth.com

Donna Levy
inVentiv Health
Louisville, KY 40205
919-452-6239
Donna.Levy@inventivhealth.com
Spare Tires

Extra Slides
Macros for review

See our paper for general and numeric variable review macros that can be used to help with review