Harvesting Information - Data Cleanup and Analysis Techniques

Gary F. Plazyk, CDP
A.T. Kearney, Inc.

Introduction

What will you get out of this paper?

An overview of how someone else uses the SAS® System.

A methodology and some time-tested techniques for analyzing data, whether from a client, or from another organization within your own company.

I'll emphasize SIMPLE techniques for extracting VALUABLE information QUICKLY. But don't let the simplicity of this approach fool you; we're going to use some basic SAS procedures as tools to help us turn boring numbers into exciting information. Just try some of these techniques and you'll find out how powerful they are.

Your background:

I'd like to talk primarily to the new SAS users here at SUGI. You've probably taken the introductory SAS courses. You may even have some background in another programming language - either on a mainframe computer using COBOL or FORTRAN or PL/I, or on a PC using BASIC or Pascal or C. You may have learned how to use SAS to read files, write reports, print checks, or do complicated statistical analysis. But what isn't mentioned very often is that SAS lets you LOOK at your data and see its overall structure. That's what makes SAS such a powerful tool. I'm not going to go into detailed statistics or measurements of error - just the meat and potatoes of data examination and cleanup.

My background:

I work for A.T. Kearney, a major management consulting firm. We do a significant amount of analysis of client data in our consulting assignments.

The data files we work with may range from a few hundred records uploaded from a PC spreadsheet, through 20,000 - 50,000 record vendor summary files, to 100,000 - 1,500,000+ record files of detailed transaction and shipment data. Most of our analysis is done using SAS.

We get data from clients in a variety of forms. SAS allows us to read almost every form of data we receive, and analyze it many different ways. SAS also lets us respond quickly to requests for additional analysis.

We have developed a methodology for looking at unknown data and for cleaning it up. It works well for us and I hope some of the techniques we use in our work will be helpful to you.

Data analysis strategy:

This is a summary of the strategy we will be examining in this paper:

Know your data! Carefully review your input records.

Read raw data into a permanent data set, creating corrected and derived values in standard form.

Do a preliminary analysis, using PROCs CONTENTS, CHART, MEANS, PRINT, and PLOT.

Perform data cleanup.

Perform additional analysis.

Data Analysis Techniques

I'm going to begin with some analysis techniques you can use right away, and then come back to the DATA step to look at some detailed cleanup techniques. So, let's assume we have our data already in a SAS data set.

Use PROC CONTENTS to document your processing:

What's the first thing we should do when we read our data in? Run a PROC CONTENTS after creating every permanent data set:

PROC CONTENTS DATA=SAVE.XXX;
RUN;

This will:

List all variables in alphabetic order, showing lengths, labels, informats, and formats.

Help track the count of records in the dataset.

Help spot typos (misspelled variables don't have labels? if you use labels on each variable).

Help spot temporary variables you meant to drop.

Help spot wrong length variables (a character variable can be 200 bytes long if no length was assigned to it).

If you use the POSITION option:

PROC CONTENTS DATA=SAVE.XXX POSITION;
RUN;
you will get a second listing of your variables in the order which they are stored in your SAS data set records. This is handy if you want to use the short form of the SAS data variable list (VAR1--VARX).

Use PROC CHART to get the "big picture":

Now we get to one of the real workhorses of the SAS procedures: PROC CHART. You may have dismissed PROC CHART because it doesn't produce presentation graphics. Don't underestimate the value of a picture - even mere printer graphics can be a powerful tool!

Have you ever looked at a color picture in a magazine with a magnifying glass? All you see is individual color dots. Data is often like that - just single records having variables with values. PROC CHART is sort of like a magnifying glass in reverse; it lets you move back and see the big picture in summary form (which is what most managers want to see, anyway).

I use PROC CHART for summaries of most of the variables in my SAS data sets. In general, I recommend doing an HBAR chart on all the values in a test sample of a dataset, and on all the character variables having a "reasonable" number of values for the full dataset. Use the MISSING DISCRETE options for character, date, and numeric code variables, and the MISSING LEVELS=40 options for numeric range variables. (I use LEVELS=40 because it nicely fills a listing page).

**HBAR**

```
HBAR VAR C1 all "reasonable" character vars
      VAR C2 all numeric code variables
      . all dates
      /
      /
      /
      / MISSING DISCRETE;
```

**HBAR**

```
HBAR VAR N1 all numeric range variables
      VAR N2
      .
      .
      /
      /
      / MISSING LEVELS=40;
```

The HBAR charts of character values will highlight bad or unexpected values, such as illegal codes.

The HBAR charts of numerics will help you spot whether there are any extreme values or outliers, and what their approximate magnitudes are.

Use some judgement here! You probably don't want to run an HBAR chart on every unique product number sold this year, or on every customer name or address. However, if a code is hierarchical in form, like zip codes or some types of product code, it might make sense to create another variable with the highest code level (like 3-digit zip code is approximately county-sized) and run an HBAR chart on it.

After you do the basic single-variable HBAR charts, do grouped HBAR charts on fields that are related. Use the GROUP= and possibly the NOZEROS options:

**HBAR TRANMODE**

```
/ MISSING DISCRETE GROUP=O_STATE NOZEROS;
```

**HBAR O_STATE**

```
/ MISSING DISCRETE GROUP=TRANMODE NOZEROS;
```

If the variable TRANMODE stood for transportation mode, and O_STATE for origin state, this would show what transportation modes were used for each origin state and what the origin states were for each transportation mode.

You can also do grouped HBAR charts for ranges of values. I normally just specify LEVELS=10 for these to keep the size of the listings manageable:

**HBAR TRANMODE**

```
/ MISSING LEVELS=10 GROUP=O_STATE NOZEROS;
```

You can use the SUMVAR= option to categorize codes by values:

**HBAR TRANMODE**

```
/ MISSING DISCRETE SUMVAR=WEIGHT;
```

Add the DESCENDING options, and you have a very powerful report:

**HBAR TRANMODE**

```
/ MISSING DISCRETE SUMVAR=WEIGHT DESCENDING;
```

gives the total weight shipped by each transportation mode, in descending total weight order.

Use PROC PLOT to spot relationships:

Another procedure that can be used for looking at numeric range variables is PROC PLOT. Comparing two variables can be done by plotting one variable against another. This example compares shipment cost to shipment weight:

**PROC PLOT:**

```
PROC PLOT;
PLOT SHIPCOST*WEIGHT;
```

If one or both variables vary over a wide range, the plot can be compressed by making one or both scales logarithmic:

**PROC PLOT:**

```
PROC PLOT;
PLOT Y*X
/ HAXIS= 0 10 100 1000 YAXIS= 0 10 100 1000;
```

With all of these graphic displays, it's important to remember to think in terms of what they can show you about your own data. A sample of PROC CHART output is not very interesting. BUT if it shows something important to you, say, the costs you spend on each transportation
carrier you use, in order of descending cost, it suddenly comes alive.

Use PROC MEANS to quantify limits:

A good complement to using PROC CHART and PROC PLOT to get a picture of the numeric range values is to use PROC MEANS to nail down some hard numbers. If you use:

```
PROC MEANS N NMISS MIN MAX SUM MEAN STD;
RUN;
```
you will find out:

- the number of observations having values and missing values;
- the value of the smallest and largest value;
- the sum, mean, and standard deviation.

These statistics will help you to get a feel for the quality and completeness of the data, and what the extreme values are.

Use PROC PRINT to look at values:

To get a look at how the values in an observation work together, it's a good idea to print the first 500 to 1000 using PROC PRINT:

```
PROC PRINT DATA=XXX(OBS=500);
RUN;
```

PROC PRINT is also useful for looking at subsets of data that have missing or questionable values, and for checking the results of your data cleanup DATA steps.

Other useful procedures:

The other procedures that you will probably use most frequently are PROC SORT, PROC SUMMARY, and PROC FREQ. Some more adventurous folks might want to play around with PROC TABULATE, although it is almost a language in its own right. PROC FORMAT is one of the most useful tools for data cleanup.

Data Cleanup Techniques

Now let's talk about some data cleanup techniques. Raw data is almost never clean. It often contains unexpected codes or some values out of the range we expected. For it to be useful, we must clean it up.

Prevent creation of bad data:

Naturally, the best way to deal with dirty data is not to generate it in the first place. If you have control over the data input, build in appropriate validation checks to prevent bad data from ever getting into your files.

However, some of us don't have any control of the data we receive. Some data files are worse than others, and the amount of effort you must put into data cleanup depends on what you want to do with the final data. It has been my experience that the data cleanup phase of a project can take from 25% to 80% or more of the total effort.

Examine raw data before reading it in:

Know your data! Most installations have utilities to produce hexadecimal dumps of data files. There are many programs available for looking at data files on tape and disk; some are available through user groups for your computer system. Common ones have names like TAPESNIF, TAPESCAN, OS/DITTO, SPECIALI, DENE (Does Everything But Eat), and of course, EATS. Some editors also allow display and printing of hex values.

SAS lets you print a record dump in a DATA step with the LIST statement.

Review the input formats. Compare your hex dump to a record layout, and verify that the fields are as they are documented to be.

For long records, or records having many variables, writing your INPUT statement can be a real chore. Create your INPUT statement with a FC spreadsheet program: make a record layout with columns containing variable name, format, label, starting and ending bytes. Output the spreadsheet to a print file on disk, and edit it to create your INPUT statement. No more counting columns!

If you use open formatting, you can work towards something like this:

```
INPUT
  @ 1 VAR1 $5.
  @ 6 VAR2 10.2
  @ 16 VAR3 $5.
;
```

Convert "odd" data to standard form:

When you are forced to read your data from a print file saved on tape, you often encounter negative values printed with trailing minus signs, such as "472-". Read the minus sign column as a separate character, and test for a "-" to apply the unary minus operator:

```
DROP XWEIGHT;
INPUT
  WEIGHT 1-5
  XWEIGHT $6;
  IF XWEIGHT='-' THEN WEIGHT = -WEIGHT;
```
Use SAS date variables:

Make all dates SAS date variables. The date variable is a standard form used by all the SAS procedures. Using values stored in SAS internal date format allow easy sorting by date, direct computation of day of week, day of year, and day of month using SAS formats and functions. Direct computation of elapsed days is possible by simply subtracting one date from another.

Dates can be read directly using formats: MDYYDD, DMDMYDD, YMDYDD.

If dates are stored in strange forms like packed decimal MDYYDD or DMDMYDD or YMDYDD, break them up after reading them in, and convert them to SAS dates using the MDY or DATEJUL functions.

Convert numeric code variables to character:

A numeric value can stand for a specific code or for a measure. A good way to tell how a numeric value is used is to ask yourself if it makes sense to compute its mean. If the mean of a value (such as a weight) makes sense, then it's a numeric range variable. If not (such as a zip code), then it's a numeric code variable. I recommend saving all numeric codes as character strings.

Advantages of saving numeric code variables as character strings include:

- They will usually take less space on SAS data file records (8 bytes numeric vs 1 byte character).
- They won't show up on PROC MEANS (no more average zip codes).
- Some codes require leading zeros (like zip codes).

Convert numeric codes to character with leading zeros on input:

```
DATA TEMP;
  DROP XPLANT;
  FORMAT
    PLANT $3.
  ;
  INPUT
    XPLANT 1-3 ;
  PLANT = PUT(XPLANT,Z3.)
  RUN;
```

Use SAS functions for character cleanup:

First, LOOK at character data on your terminal screen; make sure you have mixed case display enabled.

Use the UPCASE function if strings contain unwanted mixed upper and lower case. Another alternative might be to use your operating system COPY command with an UPCASE option, if one is available. A symptom of mixed case values is apparently out-of-sequence character strings in sorted files:

<table>
<thead>
<tr>
<th>On EBCDIC systems</th>
<th>On ASCII systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>ab</td>
<td>AB</td>
</tr>
<tr>
<td>cd</td>
<td>CD</td>
</tr>
<tr>
<td>AB</td>
<td>ab</td>
</tr>
<tr>
<td>CD</td>
<td>cd</td>
</tr>
</tbody>
</table>

Become familiar with how SAS handles character strings. Clean up character string variables with SAS character functions, especially COMPRESS, TRANSLATE, TRIM, REVERSE, SUBSTR, INDEX, RIGHT, UPCODE, and VERIFY.

For extracting zip codes from a city-state-zip character string:

```
ZIPS = LEFT(REVERSE( 
  SUBSTR(LEFT(REVERSE(CTSTZP}),1,5))); 
```

Be wary about automatic left-justification with character input, and use the $CHAR format with caution.

Use PUT function and formats to create codes:

You can create codes based on other values:

```
IF X=1 THEN CODE = '01';
ELSE IF X=2 THEN CODE = '02';
ELSE IF X=10 THEN CODE = '10';
ELSE      CODE = '??';
```

You can create codes using the PUT function and a SAS format:

```
CODE = PUT(X,Z2.); 
```

You can create codes using a table lookup technique with the PUT function and a user format:

```
CODE = PUT(X,UFMT.); 
```

These techniques were covered in great detail by Neil Howard in her paper "Efficiency Techniques for Improving I/O and Processing Time in the Data Step", presented at SUGI 88.

Tips and Tricks

These techniques may help make producing documentation less of a chore.

Formatting SAS source code:

Use a good editor - if you have a PC-mainframe environment, there are several good program editors available. I create all my source code on my PC and upload it to run on our mainframe.
Begin each source code file with comments containing the program name, a brief description of what it does, a project identification code if you use one, your name, and the date.

Always use labels and formats on all variables; use length specifications if specifying character variable formats. If you are running release 6 of SAS, you might want to use the ATTRIB statement instead of FORMAT and LABEL.

ALWAYS run a PROC CONTENTS after creating a permanent data set, and look at it.

I recommend using open formatting on your SAS code; if you are specifying several variables, put one on each line. It may use a little more space, but it allows easy insertion and deletion of text, and makes it easier to move blocks of code around and reuse them with slight modifications:

```
FORMAT
   XXXXXXXX $15.
   YYYYYYYY DOLLAR19.2

LABEL
   XXXXXXXX = 'LABEL FOR XXXXXXXX'
   YYYYYYYY = 'LABEL FOR YYYYYYYY'

INPUT
   XXXXXXXX $1-15
   YYYYYYYY 16-30

HBAR
   XXXXXXXX / MISSING DISCRETE;

HBAR
   YYYYYYYY / MISSING LEVELS=40;
```

Block diagrams for documentation:

Programming in SAS is different from most other computer languages because you must think in terms of steps that process files. Each SAS program may read or write flat data files and SAS files, and produce printed reports. If you are doing a lot of ad-hoc analysis, files and programs sometimes get out of control. I'd like to offer a technique to help keep track of things: block diagrams.

Although block diagrams utilize flow chart symbols, they are not really flow charts because they don't indicate branching logic. But making block diagrams is simple and doesn't take any fancy equipment, so you tend to do it. It's a flexible technique, so you can adapt it to suit your own needs.

Each program box stands for one SAS program, which may consist of many DATA and PROC steps. SAS files are identified by adding an extra arc onto the beginning of the file symbol (representing the header information contained in a SAS file). You can use any half-size flow chart template and large paper; I prefer:

- Berol RapiDesign R-54B flowchart template (half size)
- American Pad & Paper Co. Efficiency Pad 22-202, 17"x11"

This is a sample of a block diagram, illustrating the data analysis and cleanup methodology we discussed earlier. It documents:

- Reading a vendor file VEND into a SAS data set SAVE.RAW with a SAS program READ.
- Merging SAVE.RAW with another SAS file, XREF.LATLON, to add latitude and longitude based on zip code, using a SAS program ADDLL.
- Using PROC FSEDIT to clean up bad locations, referencing a saved screen data set FIXLL.SCREEN, using a SAS program FIXLOCs.
- Producing a map on a CRT display, using a SAS program MAPVEND.
References and resources:
The following are my most-used reference materials and resources. I strongly urge you to become familiar with them.

SAS Institute publications:
I recommend the following publications for examples of SAS source code and techniques:

- SAS Introductory Guide - "must" reading for all new SAS users.
- SAS User’s Guide: Basics (current edition for your installation) - ponderous, fragmented, and just plain heavy, this still is THE definitive source for SAS.
- SAS Companion for your operating system - provides the details to run in your environment.
- SAS Publications Master Index - provides a decent cross-publication index to the most important topics.
- SAS Views - these are the class notes for the SAS courses. They contain many helpful examples.
- SAS Communications - this quarterly magazine often has technical articles that may suggest applications you never thought of.

Machine-readable samples:
- SAS Sample Library - demonstration programs included with the SAS installation tapes.
- Current SAS Usage Notes - although these are not always accessible at every installation, the Usage Notes are distributed with the SAS installation tapes and updated regularly. The Usage Notes document all known outstanding problems in SAS, and offer suggestions for fixing or working around them. They are the first place your SAS site representative should check when you report a problem to him or her.

SUGI Conference Proceedings:
SUGI Conference Proceedings (all the editions that you can get) - the best overview available of how people really use SAS. Fascinating reading.

SUGI Proceedings Master Index - helps you to find topics across all the SUGI Proceedings years.

User group meetings:
Local SAS User Group meetings - give you a chance to discuss your applications with other interested users.

Annual SUGI conference - your most up-to-the-minute source of information on SAS. The invited speakers, tutorials, and the "birds of a feather" sessions (BOFs) are extremely helpful.

Summary
We have looked at several issues related to extracting information from data. The most important were:

- Use PROC CHART, PLOT, MEANS and PRINT for analysis.
- Use appropriate data formats: SAS date for dates, numeric for ranges, and character string for codes.
- Use SAS functions and user formats for data cleanup.
- Use PROC CONTENTS, open formatting, and block diagrams to keep code under control.
- Explore the resources available from SAS Institute and SUGI.

I invite your questions and comments. Please address them to me at:

A.T. Kearney, Inc.
222 South Riverside Plaza
Chicago, IL 60606

Acknowledgements
IBM is the registered trademark of International Business Machine, Inc.

SAS, SAS Communications, and SAS Views are registered trademarks of SAS Institute Inc., Cary, NC, USA.