THE PROBLEM

In large, long-term research studies, validating and correcting data can become extremely cumbersome and resource-intensive. Indeed, ensuring that data are correct may well take more time than analyzing those same data once they are “clean”. To improve the efficiency of data cleaning, researchers have tried to detect errors as early as possible, by checking the data as they are entered, using screen-based data entry. While screen-based data entry is a useful and appropriate tool, it generally cannot provide all of the checking and correction needed to guarantee clean data. This paper describes an automated data validation and correction system which solves several of the problems of screen-based data cleaning.

LIMITATIONS OF SCREEN-BASED DATA CLEANING

Cleaning data is a two-stage process. First errors must be detected and then they must be corrected. Detecting errors using a screen-based system presents both procedural and functional limitations. Correcting errors with such a system presents problems of documentation.

For research data in particular, screen-based checking presents a procedural problem; checking data as they are entered requires that the appropriate checks be specified before data entry begins. But research, by its very nature, deals with phenomena that are not completely understood. Realistically, researchers are not able to specify before a project begins which data checks will be needed. The range of valid values may be unknown for some variables, and codes for unexpected responses may need to be added during the project.

Another problem stems from limits on the functionality of canned screen-based data entry programs. Generally, such programs support range checks over continuous ranges (1 ≤ X ≤ 5) and, in some cases, over non-continuous ranges (1 ≤ X ≤ 5 or X EQ 9). Other kinds of checking are usually not available. These include cross-variable checking, validating skip patterns, and overriding range checks for outlier data.

CROSS-VARIABLE CHECKING is needed when the value of one variable affects the valid values for another variable. For instance, a person cannot be diagnosed with a disease before he is born, so a reasonable cross-variable check might be (DOB ≤ DATE_DX).

SKIP PATTERNS embody another kind of inter-variable dependency, in which the value of one (or more) variables determines whether or not a second variable (or, more usually, a set of variables) should have missing data. A respondent with children, for example, may be required to answer some questions about those children; an answer of “no children” means that the following questions should be blank. In general, skip patterns can be said to describe combinations of values which are logically impossible.

A way of OVERRIDDING RANGE CHECKS is needed in order to increase the effectiveness of those range checks: to use range checks to detect implausible values, as well as impossible ones. If a user cannot override a range check when outlier data is encountered, he must specify ranges wide enough to accommodate any conceivable value, thus losing much of the power and purpose of the check. Ideally, a data cleaning system would notify the user of data exceeding a range check, but could ignore the check for that case once it had been confirmed to be accurate.

One possible solution to the functional limitations of canned programs is to write a specialized application which hard-codes checks such as those described. Two obvious disadvantages to this approach are the amount of code to be written and the likelihood of semantic coding errors. In addition, such systems are inflexible to changes, which, as already noted, are a fact of life in research databases. Finally, custom-built systems bury the checks deep in code which, while readable to the programmer, is inaccessible to the personnel who must specify the checks. An important feedback loop between the specification and the implementation is thus lost.

Whether canned or custom-built, most screen-based data entry systems used for data cleaning do not provide sufficient documentation. Used by itself, screen-based cleaning leaves no audit trail of which records have been corrected, much less of who did the correction, when they did it, or what the old value was. For this reason, a paper documentation system is sometimes used to augment screen-based cleaning. Such paper systems help, but they too have shortcomings. First, they cannot insure that an update was actually made -- only that someone intended the change to be made. Perhaps more importantly, paper-based correction tracking systems restrict later access to the documentation; papers can be filed in only one order, yet the correction information may need to be accessed by case, variable, or date.
A few screen-based data entry systems do provide an audit trail. One such implementation requires reserving space for details about potential corrections, whether or not a correction was actually made. This can result in a significant increase in file size overhead, more than doubling the size of the file needed to store the information. The ADVANCS system, by contrast, stores correction information only when data are actually corrected. Furthermore, information about the corrections is stored separately from the corrected data records themselves, and therefore, does not have to be accessed during data analysis.

GOALS AND FEATURES OF THE SYSTEM

The ADVANCS system was written to handle the data-cleaning needs of an nine-year research study involving approximately 2000 variables collected four times for each of 900 cases. A total of at least 2000 checks needed to be made for each of the 3600 records collected.

Given the magnitude of the data cleaning task, the major goal of our system was to automate as much as possible of the code generation and to keep thorough records of which checks and updates had been made. Specifically, the system was to:

1. automate the generation of routine SAS* code (for input, edit checks, missing values coding, and standardized statistical reports)
2. automate the generation of SAS* code to perform updates of erroneous data
3. keep an audit trail of edit checks as they were added or changed. This trail could then be used to reconstruct which data had been tested against which checks.
4. keep an audit trail of corrections to bad records. The structure of this trail was chosen to be both compact and flexible. The compactness was intended to minimize the likelihood of further data-entry errors. Flexibility of access to the information (by case ID, variable, or date) was intended to help resolve questions about corrections quickly and to allow for statistical quality control reporting.

The edit-checking code that was generated automatically needed to handle non-continuous range checks and skip pattern error checking, and to allow range checks to be overridden for specific cases. In addition, ADVANCS needed to separate good from bad records as early as possible, so that the largest possible subset of valid records was always available, on demand, for analysis. In long-term studies, when papers must be published before all of the data are collected and cleaned, the ability to provide subsets of accurate data is extremely important.

IMPLEMENTATION

ADVANCS uses a database composed of three tables to store information about checks and corrections. The RANGE CHECK and SKIP PATTERN tables store information used to detect erroneous data and the CORRECTIONS table stores information about the updates made to the data. SAS* code to perform the checks and corrections is generated automatically from this database using the database management system's report writer.

For logistical reasons pertaining to the research study, the initial implementation of ADVANCS used two database packages: the RANGE CHECK and SKIP PATTERN tables were implemented using EMPRESS, and the CORRECTIONS table was implemented using DataEase. A subsequent version of the system was implemented entirely in DataEase, with some savings in complexity. The particular database management system is of less importance than that it have a versatile report writer.

A. The RANGE CHECK Table

The range check table can be thought of as a specialized data dictionary table. It contains several items describing each variable in the study, whether or not the variable has a range check. These items are needed to generate the SAS* code for input and standardized statistical reports. They are shown above the dotted line in Table 1.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE 1. RANGE CHECK STRUCTURE</td>
<td></td>
</tr>
<tr>
<td>1. variable name (or key uniquely identifying variable, e.g., variable and form)</td>
<td></td>
</tr>
<tr>
<td>2. data type (character, numeric, etc.)</td>
<td></td>
</tr>
<tr>
<td>3. field width</td>
<td></td>
</tr>
<tr>
<td>4. decimal width</td>
<td></td>
</tr>
<tr>
<td>5. check number</td>
<td></td>
</tr>
<tr>
<td>6. valid values</td>
<td></td>
</tr>
<tr>
<td>7. date check was added</td>
<td></td>
</tr>
</tbody>
</table>
Figure 1. Range Check and Derived SAS* code

Range Check Screen

Varname AGE Data type N
Field Width 2 Decimal width 0
Check number 30 Date added 01/28/88
Valid values 20 LE AGE LE 70

Derived SAS* Code

IF (NOT (20 LE AGE LE 70) AND OR30 = 'No') THEN DO;
  RECREJ = 1;
  FILE CORRDB;
  PUT 'AGE, 30'10 ', 'T DAY MMDDYVS ... N'
END;

B. The SKIP PATTERN Table

Skip patterns describe combinations of values across variables which are logically impossible. Typically, these impossibilities stem from the fact that if one question (the index variable) was answered a certain way (the person did not have diabetes, e.g.), certain other questions (set variables) should not have been asked (was he treated with insulin?). In the skip pattern table, one record is entered for each pair of interdependent variables.

ADVANCE handles three kinds of skip patterns dependencies, each having different expectations of the values in the set variables when the index variable is true (see Table 3). In a "mandatory block" dependency, all set variables must be present if the index variable was answered yes. A "choice from the universe" requires that at least one set variable be present if the index variable was answered yes. Finally, a "choice from a sample" dependency makes no requirements if the index variable was answered yes. (Note that the difference between a "choice from the universe" and a "choice from a sample" is that the follow-up questions for a sample do not cover every possibility; in Table 3.c, a person working on weekends could legitimately answer "yes" to the index question and not mark any of the set questions as true.) In all cases, the values of all set variables must be missing if the index variable was answered negatively. Figure 2 shows an entry to the skip pattern table and the SAS* code that it (together with the six records for other variables in the set) would generate.

TABLE 2. SKIP PATTERN TABLE

1. "index" variable
2. "set" variable
3. value of the index variable which indicates the index is true or present
4. value of the set variable which is expected if the index variable is true
5. value of the index variable which indicates the index is false or absent
6. value of the set variable which is expected if the index variable is false or absent
7. kind of dependency

TABLE 3. EXAMPLES OF SKIP PATTERNS

a. Mandatory Block:

In your work, do you perform some vigorous activity?

No Yes

How often does this occur?

How long does it last each time?

b. Choice from the Universe:

Do you work outside the home?

No Yes

Do you work on:


c. Choice from a Sample:

Do you work outside the home?

No Yes

Do you work on:

Mon? Tue? Wed?

Thu? Fri?

Figure 2. Skip Pattern and Derived SAS* code

Skip Pattern Screen

Index Var EMPLOYED Set Var WRKMON

True value EQ 'Y' implies NE BLANK
False value NE 'Y' implies EQ BLANK

Dependency type 2

SAS* Code Generated from a Skip Pattern

ARRAY SKIP2 WRKMON WRKTUE WRKWEK WRKTHU WRKFRI WRSAT WRSUN;
N = 0;
DO I = 1 TO DIM (SKIP2);
  IF (SKIP2{I} = NE BLANK) THEN
    N = N + 1;
END;

IF (EMPLOYED EQ 'Y' AND N LT 1) THEN
  PUT <error message>; END;
IF (EMPLOYED NE 'Y' AND N NE 0) THEN
  PUT <error message>; END;

860
In one way, skip patterns are simpler than range checks: since they are logically impossible, they
are never be overridden. There are more complex, however, in that they spot a discrepancy within a
GROUP of variables, without identifying which variable in particular is in error. Since a skip
pattern violation does not guarantee that there is an error with any particular variable, information
about these errors is not put directly in the corrections table from the SAS® program.

C. The CORRECTIONS Table

The corrections table contains a detailed record for
each error detected by the SAS® edit checking
code. For range check errors, several data items
for the corrections table are loaded directly from
the SAS® edit checking program, thus minimizing
the data to be re-entered. The items entered
directly by the SAS® edit checking program are
those above the dotted line in Table 4. For skip
pattern errors all of the data must be entered.
Figure 3 shows an entry from the corrections table
and the SAS® code generated from it.

Figure 3. Correction and Derived SAS® code

<table>
<thead>
<tr>
<th>Correction Screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Varname AGE</td>
</tr>
<tr>
<td>Case number 98</td>
</tr>
<tr>
<td>Old var value 13</td>
</tr>
<tr>
<td>New var value 31</td>
</tr>
<tr>
<td>Override No</td>
</tr>
<tr>
<td>Override Explanation</td>
</tr>
</tbody>
</table>

 Derived SAS® code

```sas
DATA TEMP;
MISSING _.;
CASE = 98;
AGE = 31;
OUTPUT;
AGE = .;
DATA TEMP;
UPDATE REJECT TEMP;
BY CASE;
```

PROCESS FLOW

Figure 4 shows the flow of information through
ADVANCS. First, the entries to the range checks
and skip pattern tables are specified. SAS® code
to input and check the data can then be generated
from these tables. The code to input the data is
executed next, generating a SAS® dataset from
ASCII records.

<table>
<thead>
<tr>
<th>TABLE 4. CORRECTIONS TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. variable (or key uniquely identifying variable, e.g., variable and form)</td>
</tr>
<tr>
<td>2. case (or key uniquely identifying the record, e.g., case and date)</td>
</tr>
<tr>
<td>3. date error was detected</td>
</tr>
<tr>
<td>4. old value</td>
</tr>
<tr>
<td>5. flag indicating whether check is to be overridden</td>
</tr>
<tr>
<td>6. new value</td>
</tr>
<tr>
<td>7. comment field explaining reason for override</td>
</tr>
<tr>
<td>8. date error was corrected</td>
</tr>
<tr>
<td>9. who entered correction</td>
</tr>
</tbody>
</table>

Once this uncleaned dataset has been created, the
SAS® code to check the dataset can be executed.
This step adds to the dataset of valid records any
of the current set of records which pass all checks.
The date the record passed the checks is
automatically added to the clean record. Records
which fail any check are added to the dataset of
invalid records and, if the check which was failed is
a range check, a record is generated for the
correction table. In any case, a hardcopy listing of
the error is produced.

After research personnel have compared the
errors with the original data forms, the correction
table can be updated either with new values or with
commands to override the range checks. SAS®
code to implement these changes can then be
generated from the correction table and run
against the dataset of invalid records. Corrected
records are passed through the edit checking
routine again and assigned to the valid or invalid
dataset, as appropriate. This cycle continues until
all records are valid.
Inevitably, the edit checking specifications need to be changed. When this occurs, a new entry is made to the appropriate table (range checks or skip patterns), and the SAS® edit checking code is regenerated. The old check, along with the date of the change, is automatically archived to another table. Since every defunct check has a date on which it became obsolete and every valid record has a date on which it passed the checks, it is possible, if necessary, to reconstruct which records passed which checks.

RESULTS

ADVANCS has been running on a VAX 11-750 with VMS for approximately one year and has recently been converted to run on an Intel 80386 PC system running DOS 3.0.

All of the initial goals of ADVANCS have been met. SAS® code for input, edit checking, updating, and standard statistical tests is now generated automatically. The result is more thorough and consistent checking than had previously been possible. In addition, most data cleaning tasks which were formerly assigned to a programmer are now performed by a technician, and turnaround time to detect errors has been dramatically reduced. Clean data records are separated out early, so the largest possible analysis-ready dataset is always available. Minor changes to the edit checks, such as those required when new protocols are begun, can be implemented easily.

Audit trails of both corrections and edit checks are now available in a single, machine-readable location. The historical information about corrections can be used for quality control reporting as well as to resolve data questions quickly. The information about edit checks serves as an accurate on-line coding book. It also summarizes how the collected data have changed over time: how checks were refined over the course of a study and how variables were added or dropped when protocols changed. Perhaps most importantly, the system has promoted discussion and consensus about coding schemes and interpretation.

Assuring accurate data will remain a challenge for large long-term studies, but systems such as ADVANCS are a useful addition to the researcher's data-cleaning tools.

SAS is a registered trademark of SAS Institute Inc., Cary, NC, USA.

Empress is a registered trademark of Empress Software Inc., Toronto, Ontario, CANADA.

DataEase is a registered trademark of DataEase International, Trumbull, CT, USA.

Acknowledgements:

This project was made possible as part of grant DK-30747 of the National Institutes of Health, Department of Health and Human Services, USA.

Contact Author:
Jessica Bondy
Box B-119
University of Colorado Health Sciences Center
Denver, CO 80262
BITNET: JESS@UCOLMCC
Internet: JESS@FIJI@VAXF.COLORADO.EDU

SAS is a registered trademark of SAS Institute Inc., Cary, NC, USA.

Empress is a registered trademark of Empress Software Inc., Toronto, Ontario, CANADA.

DataEase is a registered trademark of DataEase International, Trumbull, CT, USA.

Acknowledgements:

This project was made possible as part of grant DK-30747 of the National Institutes of Health, Department of Health and Human Services, USA.

Contact Author:
Jessica Bondy
Box B-119
University of Colorado Health Sciences Center
Denver, CO 80262
BITNET: JESS@UCOLMCC
Internet: JESS@FIJI@VAXF.COLORADO.EDU