1 Checking Values of Character Variables

Introduction

There are some basic operations that need to be routinely performed when dealing with character data values. You may have a character variable that can take on only certain allowable values, such as 'M' and 'F' for gender. You may also have a character variable that can take on numerous values but the values must fit a certain form, such as a single letter followed by two or three digits. This chapter shows you several ways that you can use SAS software to perform validity checks on character variables.

Using PROC FREQ to List Values

This section demonstrates how to use PROC FREQ to check for invalid values of a character variable. In order to test the programs you develop, use the raw data file PATIENTS.TXT, listed in the Appendix. You can use this data file and, in later sections, a SAS data set created from this raw data file for many of the examples in this text.

Description of the Raw Data File PATIENTS.TXT

The raw data file PATIENTS.TXT contains both character and numeric variables from a typical clinical trial. A number of data errors were included in the file so that you can test the data cleaning programs that are developed in this text. The programs in this book assume that the file PATIENTS.TXT is located in a directory (folder) called C:\CLEANING. This is the directory that is used throughout this text as the location for data files, SAS data sets, SAS programs, and SAS macros. You can modify the INFILE and LIBNAME statements to fit your own operating environment.

Here is the layout for the data file PATIENTS.TXT.

Variable Name	Description	Starting Column	Length	Variable Type	Valid Values
PATNO	Patient Number	1	3	Character	Numerals only
GENDER	Gender	4	1	Character	'M' or 'F'
VISIT	Visit Date	5	10	MMDDYY10.	Any valid date
HR	Heart Rate	15	3	Numeric	Between 40 and 100
SBP	Systolic Blood Pressure	18	3	Numeric	Between 80 and 200
DBP	Diastolic Blood Pressure	21	3	Numeric	Between 60 and 120
DX	Diagnosis Code	24	3	Character	1 to 3 digit numeral
AE	Adverse Event	27	1	Character	'0' or '1'

There are several character variables that should have a limited number of valid values. For this exercise, you expect values of GENDER to be 'F' or 'M', values of DX the numerals 1 through 999, and values of AE (adverse events) to be '0' or '1'. A very simple approach to identifying invalid character values in this file is to use PROC FREQ to list all the unique values of these variables. Of course, once invalid values are identified using this technique, other means will have to be employed to locate specific records (or patient numbers) corresponding to the invalid values.

Use the program PATIENTS.SAS (shown next) to create the SAS data set PATIENTS from the raw data file PATIENTS.TXT (which can be downloaded from the SAS Web site or found listed in the Appendix). This program is followed with the appropriate PROC FREQ statements to list the unique values (and their frequencies) for the variables GENDER, DX, and AE.

Program 1-1 Writing a Program to Create the Data Set PATIENTS

```
*-----------------------------------------------------------*
|PROGRAM NAME: PATIENTS.SAS IN C:\CLEANING                  |
|PURPOSE: TO CREATE A SAS DATA SET CALLED PATIENTS          |
*-----------------------------------------------------------*;
LIBNAME CLEAN "C:\CLEANING";

DATA CLEAN.PATIENTS;
   INFILE "C:\CLEANING\PATIENTS.TXT" PAD; /* Pad short records
                                             with blanks  */

   INPUT @1  PATNO    $3. @4  GENDER   $1.
         @5  VISIT    MMDDYY10.
         @15 HR       3.
         @18 SBP      3.
         @21 DBP      3.
         @24 DX       $3.
         @27 AE       $1.;

   LABEL PATNO   = "Patient Number"
         GENDER  = "Gender"
         VISIT   = "Visit Date"
         HR      = "Heart Rate"
         SBP     = "Systolic Blood Pressure"
         DBP     = "Diastolic Blood Pressure"
         DX      = "Diagnosis Code"
         AE      = "Adverse Event?";
    FORMAT VISIT MMDDYY10.;
RUN;
```

The DATA step is straightforward. Notice the PAD option in the INFILE statement. This will seem foreign to most mainframe users and is probably no longer necessary on other platforms. The PAD option pads all records (adds blanks to the end of short records) to the default logical record length or a length specified by another INFILE option, LRECL. It prevents the possibility of skipping to the next record (line) of data when a short line is encountered.

Next, you want to use PROC FREQ to list all the unique values for your character variables. To simplify the output from PROC FREQ, use the NOCUM (no cumulative statistics) and NOPERCENT (no percentages) TABLES options because you only want frequency counts for each of the unique character values. (Note, sometimes the percent and cumulative statistics can be useful — the choice is yours.) The PROC statements are shown in Program 1-2.

Program 1-2 Using PROC FREQ to List All the Unique Values for Character Variables

```
PROC FREQ DATA=CLEAN.PATIENTS;
   TITLE "Frequency Counts for Selected Character Variables";
   TABLES GENDER DX AE / NOCUM NOPERCENT;
RUN;
```

Here is the output from running Program 1-2.

```
Frequency Counts for Selected Character Variables

The FREQ Procedure

      Gender

GENDER    Frequency
-------------------
2                1
F               12
M               14
X                1
f                2

Frequency Missing = 1


Diagnosis Code

DX    Frequency
---------------
1           7
2           2
3           3
4           3
5           3
6           1
7           2
X           2

Frequency Missing = 8



Adverse Event?

AE    Frequency
---------------
0          19
1          10
A           1

Frequency Missing = 1
```

Let's focus in on the frequency listing for the variable GENDER. If valid values for GENDER are 'F', 'M', and missing, this output would point out several data errors. The values '2' and 'X' both occur once. Depending on the situation, the lowercase value 'f' may or may not be considered an error. If lowercase values were entered into the file by mistake, but the value (aside from the case) was correct, you could change all lowercase values to uppercase with the UPCASE function. More on that later. The invalid DX code of 'X' and the adverse event of 'A' are also easily identified. At this point, it is necessary to run additional programs to identify the location of these errors. Running PROC FREQ is still a useful first step in identifying errors of these types, and it is also useful as a last step, after the data have been cleaned, to ensure that all the errors have been identified and corrected.

Using a DATA Step to Check for Invalid Values

Your next task is to use a DATA step to identify invalid data values and to determine where they occur in the raw data file (by listing the patient number).

This time, DATA step processing is used to identify invalid character values for selected variables. As before, you will check GENDER, DX, and AE. Several different methods are used to identify these values.

First, you can write a simple DATA step that reports invalid data values by using PUT statements in a DATA _NULL_ step. Here is the program.

Program 1-3 Using a DATA _NULL_ Step to Detect Invalid Character Data

```
DATA _NULL_;
   INFILE "C:\CLEANING\PATIENTS.TXT" PAD;
   FILE PRINT; ***Send output to the Output window;
   TITLE "Listing of Invalid Patient Numbers and Data Values";
   ***Note: We will only input those variables of interest;
   INPUT @1  PATNO    $3.
         @4  GENDER   $1.
         @24 DX       $3.
         @27 AE       $1.;
   ***Check GENDER;
   IF GENDER NOT IN ('F' 'M' ' ') THEN PUT PATNO= GENDER=;
   ***Check DX;
   IF VERIFY(DX,' 0123456789') NE 0 THEN PUT PATNO= DX=;
   ***Check AE;
   IF AE NOT IN ('0' '1' ' ') THEN PUT PATNO= AE=;
RUN;
```

Before discussing the output, let's spend a moment looking over the program. First, notice the use of the DATA _NULL_ statement. Because the only purpose of this program is to identify invalid data values, there is no need to create a SAS data set. The FILE PRINT statement causes the results of any subsequent PUT statements to be sent to the Output window (or output device). Without this statement, the results of the PUT statements would be sent to the SAS Log. GENDER and AE are checked by using the IN operator. The statement

```
IF X IN ('A' 'B' 'C') THEN . . .;
```

is equivalent to

```
IF X = 'A' OR X = 'B' OR X = 'C' THEN . . .;
```

That is, if X is equal to any of the values in the list following the IN operator, the expression is evaluated as true. You want an error message printed when the value of GENDER is not one of the acceptable values ('F', 'M', or missing). Therefore, place a NOT in front of the whole expression, triggering the error report for invalid values of GENDER or AE. You can separate the values in the list by spaces or commas.

There are several alternative ways that the gender checking statement can be written. The method above uses the IN operator.

A straightforward alternative to the IN operator is

```
IF NOT (GENDER EQ 'F' OR GENDER EQ 'M' OR GENDER = ' ') THEN
PUT PATNO= GENDER=;
```

Another possibility is

```
IF GENDER NE 'F' AND GENDER NE 'M' AND GENDER NE ' ' THEN
PUT PATNO= GENDER=;
```

While all of these statements checking for GENDER and AE produce the same result, the IN operator is probably the easiest to write, especially if there are a large number of possible values to check. Always be sure to consider whether you want to identify missing values as invalid or not. In the statements above, you are allowing missing values as valid codes. If you want to flag missing values as errors, do not include a missing value in the list of valid codes.

If you want to allow lowercase M's and F's as valid values, you can add the single line

```
GENDER = UPCASE(GENDER);
```

immediately before the line that checks for invalid gender codes. As you can probably guess, the UPCASE function changes all lowercase letters to uppercase letters.

A statement similar to the gender checking statement is used to test the adverse events.

There are so many valid values for DX (any numeral from 1 to 999), that the approach you used for GENDER and AE would be inefficient (and wear you out typing) if you used it to check for invalid DX codes. The VERIFY function is one of the many possible ways you can check to see if there is a value other than the numerals 0 to 9 or blank as a DX value. The VERIFY function has the following form:

```
VERIFY(character_variable,verify_string)
```

where the verify string is either a character variable or a series of character values placed in single or double quotes. The VERIFY function returns the first position in the character_variable that contains a character that is not in the verify_string. If the character_variable does not contain any invalid values, the VERIFY function returns a 0. To make this clearer, let's look at the following statement that uses the VERIFY function to check for invalid GENDER values:

```
IF VERIFY (GENDER,'FM ') NE 0 THEN PUT PATNO= GENDER=;
```

Notice that you included a blank in the verify_string so that missing values will be considered valid. If GENDER has a value other than an 'F', 'M', or missing, the VERIFY function returns the position of the invalid character in the string. But, because the length of GENDER is 1, any invalid value for GENDER returns a 1.

You are now ready to understand the VERIFY function that checked for invalid DX codes. The verify string contained a blank plus the characters (numerals) 0 through 9. Thus, if the DX code contains any character other than a blank or a 0 through 9, it returns the position of this offending character, which would have to be a 1, 2, or 3 (DX is three bytes in length), and the error message would be printed.

Although the function

```
VERIFY(DX,' 0123456789')
```

returns a 0 if there are no invalid characters in the DX code, it should be pointed out that DX codes with embedded blanks will not be identified as invalid with this statement. If you want to ensure that only the character representations of the numbers 1 to 999 are considered valid, the following statements can be used:

```
X_DX = INPUT(DX,3.);
IF X_DX EQ . AND DX NE ' ' THEN PUT PATNO= DX=;
```

The first line above creates a numeric variable (X_DX) from the character DX value. The INPUT function can be thought of in a similar manner to an INPUT statement. It says to pretend you are reading a variable (DX) from a data file according to the INFORMAT 3. except you are actually "reading" the value from a character variable. The result of this process is to be assigned to the variable X_DX. In other words, the INPUT function performs a character-to-numeric conversion. If there is an invalid DX code (containing a letter or embedded blank, for example), the INPUT function sends an error message to the SAS Log and returns a missing value. In the second line, you test if the numeric equivalent of the DX code is missing and the original DX is not missing, putting out an error message when this condition is true. (Note, because the original character value was three bytes, you don't have to test if X_DX is greater than 999, because this is the largest number you can write with three digits.) Any invalid DX code will then cause the error message to be printed.

For really compulsive programmers (like the author), there is one final problem with the above approach. A value such as 1.3 would not result in an error message because the number 1.3 is between 1 and 999. There are several ways around this problem. One way is to use the TRANSLATE function to substitute an invalid character for the decimal point before you perform the character-to-numeric conversion.

```
X_DX = INPUT(TRANSLATE(DX,'X','.'),3.);
```

The TRANSLATE function above will convert periods (or decimal points) to X's. If DX originally contained a decimal point, the value of X_DX would be a missing value. In general, the syntax of the TRANSLATE function is

```
TRANSLATE(char_variable,to_string,from_string)
```

where each character in the from_string is translated to the corresponding character in the to_string. For example, to translate the numerals 1 through 5 to the letters A through E for a variable called SCORE, you would write

```
NEW_VAR = TRANSLATE(SCORE,'ABCDE','12345');
```

Another interesting approach is to test to see if the value of X_DX is not an integer. The MOD function is an effective way to do this. If any number modulus 1 is not 0 (the remainder after you divide the number by 1), the number is not an integer. The SAS code using this method is

```
X_DX = INPUT(DX,3.);
IF (X_DX EQ . OR MOD(X_DX,1) NE 0) AND
  DX NE ' ' THEN PUT PATNO= DX=;
```

Here is another point. If you want to avoid filling up your SAS Log with error messages resulting from invalid arguments to the INPUT function, you can use the double question mark (??) modifier before the informat to tell the program to ignore these errors and not to report the errors to the SAS Log. The INPUT function would then look like this:

```
X_DX = INPUT(DX,?? 3.);
```

The ?? informat modifier can also be used with the INPUT statement. Here is the output from running Program 1-3.

```
Listing of Invalid Patient Numbers and Data Values

PATNO=002  DX=X
PATNO=003  GENDER=X
PATNO=004  AE=A
PATNO=010  GENDER=f
PATNO=013  GENDER=2
PATNO=002  DX=X
PATNO=023  GENDER=f
```

Note that patient 002 appears twice in this output. This occurs because there is a duplicate observation for patient 002 (in addition to several other purposely included errors), so that the data set can be used for examples later in this book, such as the detection of duplicate ID's and duplicate observations.

Suppose you want to check for valid patient numbers (PATNO) in a similar manner. However, you want to flag missing values as errors (every patient must have a valid ID). The following statements:

```
ID = INPUT(TRANSLATE(PATNO,'X','.'),?? 3.);
IF ID LT 1 THEN PUT "Invalid ID for PATNO=" PATNO;
```

will work in the same way as your check for invalid DX codes except that missing values will now be listed as errors.

Using PROC PRINT with a WHERE Statement to List Invalid Values

There are several alternative ways to identify the ID's containing invalid data. As with most of the topics in this book, you will see several ways of accomplishing the same task. Why? One reason is that some techniques are better suited to an application. Another reason is to teach some additional SAS programming techniques. Finally, under different circumstances, some techniques may be more efficient than others.

One very easy alternative way to list the subjects with invalid data is to use PROC PRINT followed by a WHERE statement. Just as you used an IF statement in a DATA step in the previous section, you can use a WHERE statement in a similar manner with PROC PRINT and avoid having to write a DATA step altogether. For example, to list the ID's with invalid GENDER values, you could write a program like the one shown in Program 1-4.

Program 1-4 Using PROC PRINT to List Invalid Character Values

```
PROC PRINT DATA=CLEAN.PATIENTS;
   TITLE "LISTING OF INVALID GENDER VALUES";
   WHERE GENDER NOT IN ('M' 'F' ' ');
   ID PATNO;
   VAR GENDER;
RUN;
```

It's easy to forget that WHERE statements can be used within SAS procedures. SAS programmers that have been at it for a long time (like the author) often write a short DATA step first and use PUT statements or create a temporary SAS data set and follow it with a PROC PRINT. The program above is both shorter and more efficient than a DATA step followed by a PROC PRINT. DATA _NULL_ steps, however, tend to be fairly efficient and are a reasonable alternative as well as the more flexible approach.

The output from Program 1-4 follows.

```
LISTING OF INVALID GENDER VALUES

PATNO    GENDER

 003       X
 010       f
 013       2
 023       f
```

This program can be extended to list invalid values for all the character variables. You simply add the other invalid conditions to the WHERE statement as shown in Program 1-5.

Program 1-5 Using PROC PRINT to List Invalid Character Data for Several Variables

```
PROC PRINT DATA=CLEAN.PATIENTS;
   TITLE "LISTING OF INVALID CHARACTER VALUES";
   WHERE GENDER NOT IN ('M' 'F' ' ')          OR
         VERIFY(DX,' 0123456789') NE 0        OR
         AE NOT IN ('0' '1' ' ');
   ID PATNO;
   VAR GENDER DX AE;
RUN;
```

The resulting output is shown next.

```
LISTING OF INVALID CHARACTER VALUES

PATNO    GENDER    DX    AE

 002       F       X     0
 003       X       3     1
 004       F       5     A
 010       f       1     0
 013       2       1
 002       F       X     0
 023       f             0
```

Notice that this output is not as informative as the one produced by the DATA _NULL_ step in Program 1-3. It lists all the patient numbers, genders, DX codes, and adverse events even when only one of the variables has an error (patient 002 for example). So, there is a trade-off — the simpler program produces slightly less desirable output. We could get philosophical and extend this concept to life in general, but, that's for some other book.

You can also substitute any of the more complicated logical expressions from the previous section into this WHERE statement if you wish. For example, to perform a more careful check on DX codes, you could modify the WHERE statement as shown here.

```
PROC PRINT DATA=CLEAN.PATIENTS;
   TITLE "LISTING OF INVALID CHARACTER VALUES";
   WHERE GENDER NOT IN ('M' 'F' ' ')                       OR
      (INPUT(DX,3.) EQ . OR MOD(INPUT(DX,3.),1) NE 0)     AND
      DX NE ' '                                            OR
      AE NOT IN ('0' '1' ' ');
   ID PATNO;
   VAR GENDER DX AE;
RUN;
```

Using Formats to Check for Invalid Values

Another way to check for invalid values of a character variable from raw data is to use user-defined formats. There are several possibilities here. One, you can create a format that leaves all valid character values as is and formats all invalid values to a single error code. Let's start out with a program that simply assigns formats to the character variables and uses PROC FREQ to list the number of valid and invalid codes. Following that, you will extend the program by using a DATA step to identify which ID's have invalid values. Program 1-6 uses formats to convert all invalid data values to a single value.

Program 1-6 Using a User-Defined Format and PROC FREQ to List Invalid Data Values

```
PROC FORMAT;
   VALUE $GENDER 'F','M' = 'Valid'
                 ' '     = 'Missing'
                 OTHER   = 'Miscoded';
   VALUE $DX '001' - '999' = 'Valid'  /* See important note below */
             ' '           = 'Missing'
                 OTHER     = 'Miscoded';

   VALUE $AE '0','1' = 'Valid'
             ' '     = 'Missing'
              OTHER  = 'Miscoded';
RUN;

PROC FREQ DATA=CLEAN.PATIENTS;
   TITLE "Using Formats to Identify Invalid Values";
   FORMAT GENDER $GENDER.
          DX     $DX.
          AE     $AE.;
   TABLES GENDER DX AE / NOCUM NOPERCENT MISSING;
RUN;
```

For the variables GENDER and AE, which have specific valid values, you list each of the valid values in the range to the left of the equal sign in the VALUE statement. Format each of these values with the value 'Valid'. For the $DX format, you specify a range of values on the left side of the equal sign.

Important Note: It should be pointed out here, that the range '001' - '999' will behave differently on Windows and UNIX platforms compared to MVS and CMS platforms. You may want to test several values on your platform to be sure the program is performing as you intend. For example, the value '0A1' will be considered 'Valid' on a Windows or a UNIX platform and 'Invalid' on MVS or CMS (as pointed out by two of my reviewers, John Laing and Mike Zdeb). You may want to test for alphabetic values for DX in a short DATA step, prior to running Program 1-6.

You may choose to lump the missing value with the valid values if that is appropriate, or you may want to keep track of missing values separately as was done here. Finally, any value other than the valid values or a missing value will be formatted as 'Miscoded'. All that is left is to run PROC FREQ to count the number of 'Valid', 'Missing', and 'Miscoded' values. The TABLES option MISSING causes the missing values to be listed in the body of the PROC FREQ output. Here is the output from PROC FREQ.

```
Using Formats to Identify Invalid Values
The FREQ Procedure

       Gender
GENDER      Frequency
---------------------
Missing            1
Miscoded           4
Valid             26

Diagnosis Code

DX          Frequency
---------------------
Missing            8
Valid             21
Miscoded           2

Adverse Event?

AE          Frequency
---------------------
Missing            1
Valid             29
Miscoded           1
```

This output isn't particularly useful. It doesn't tell you which observations (patient numbers) contain missing or invalid values. Let's modify the program by adding a DATA step, so that ID's with invalid character values are listed.

Program 1-7 Using a User-Defined Format and a DATA Step to List Invalid Data Values

```
PROC FORMAT;
   VALUE $GENDER 'F','M' = 'Valid'
                 ' '     = 'Missing'
                 OTHER   = 'Miscoded';
   VALUE $DX '001' - '999' = 'Valid'
             ' '           = 'Missing'
                 OTHER     = 'Miscoded';
   VALUE $AE '0','1' = 'Valid'
             ' '     = 'Missing'
              OTHER  = 'Miscoded';
RUN;
```

```
DATA _NULL_;
   INFILE "C:\CLEANING\PATIENTS.TXT" PAD;
   FILE PRINT; ***Send output to the Output window;
   TITLE "Listing of Invalid Patient Numbers and Data Values";
   ***Note: We will only input those variables of interest;
   INPUT @1  PATNO    $3.
         @4  GENDER   $1.
         @24 DX       $3.
         @27 AE       $1.;

   IF PUT(GENDER,$GENDER.) = 'Miscoded' THEN PUT PATNO= GENDER=;
   IF PUT(DX,$DX.) = 'Miscoded' THEN PUT PATNO= DX=;
   IF PUT(AE,$AE.) = 'Miscoded' THEN PUT PATNO= AE=;
RUN;
```

The "heart" of this program is the PUT function. To review, the PUT function is similar to the INPUT function. It takes the following form:

```
character_variable = PUT(variable,format)
```

where character_variable is a character variable that contains the value of the variable listed as the first argument to the function, formatted by the format listed as the second argument to the function. The result of a PUT function is always a character variable and the function is frequently used to perform numeric-to-character conversions. In Program 1-7, the first argument of the PUT function is a character variable, and the result of the PUT function for any invalid data values would be the value 'Miscoded'.

Here is the output from Program 1-7.

```
Listing of Invalid Patient Numbers and Data Values

PATNO=002  DX=X
PATNO=003  GENDER=X
PATNO=004  AE=A
PATNO=010  GENDER=f
PATNO=013  GENDER=2
PATNO=002  DX=X
PATNO=023  GENDER=f
```

Using Informats to Check for Invalid Values

PROC FORMAT is also used to create informats. Remember that formats are used to control how variables look in output or how they are classified by such procedures as PROC FREQ. Informats modify the value of variables as they are read from the raw data, or they can be used with an INPUT function to create new variables in the DATA step. User-defined informats are created in much the same way as user-defined formats. Instead of a VALUE statement that creates formats, an INVALUE statement is used to create informats. The only difference between the two is that informat names can only be seven characters in length. (Note: For those curious readers, the reason is that informats and formats are both stored in the same catalog and an "@" is placed before informats to distinguish them from formats.) The following is a program that changes invalid values for GENDER and AE to missing values by using a user-defined informat.

Program 1-8 Using a User-Defined Informat to Set Invalid Data Values to Missing

```
*----------------------------------------------------------------*
| PROGRAM NAME: INFORM1.SAS IN C:\CLEANING                       |
| PURPOSE: TO CREATE A SAS DATA SET CALLED PATIENTS2             |
|          AND SET ANY INVALID VALUES FOR GENDER AND AE TO       |
|          MISSING, USING A USER-DEFINED INFORMAT                |
*---------------------------------------------------------------*;
LIBNAME CLEAN "C:\CLEANING";

PROC FORMAT;
   INVALUE $GEN    'F','M' = _SAME_
                   OTHER   = ' ';
   INVALUE $AE    '0','1' = _SAME_
                   OTHER  = ' ';
RUN;

DATA CLEAN.PATIENTS2;
   INFILE "C:\CLEANING\PATIENTS.TXT" PAD;
   INPUT @1  PATNO    $3.
         @4  GENDER   $GEN1.
         @27 AE       $AE1.;

   LABEL PATNO   = "Patient Number"
         GENDER  = "Gender"
         DX      = "Diagnosis Code"
         AE      = "Adverse Event?";
RUN;
```

```
PROC PRINT DATA=CLEAN.PATIENTS2;
   TITLE "Listing of Data Set PATIENTS2";
   VAR PATNO GENDER AE;
RUN;
```

Notice the INVALUE statements in the PROC FORMAT above. The key word _SAME_ is a SAS reserved value that does what its name implies — it leaves any of the values listed in the range specification unchanged. The key word OTHER in the subsequent line refers to any values not matching one of the previous ranges. Notice also, that the informats in the INPUT statement use the user-defined informat name followed by the number of columns to be read, the same method that is used with predefined SAS informats.

Output from the PROC PRINT is shown next.

```
Listing of Data Set PATIENTS2

Obs    PATNO    GENDER    AE

  1     001       M       0
  2     002       F       0
  3     003               1
  4     004       F
  5     XX5       M       0
  6     006               1
  7     007       M       0
  8               M       0
  9     008       F       0
 10     009       M       1
 11     010               0
 12     011       M       1
 13     012       M       0
 14     013
 15     014       M       1
 16     002       F       0
 17     003       M       0
 18     015       F       1
 19     017       F       0
 20     019       M       0
 21     123       M       0
 22     321       F       1
 23     020       F       0
 24     022       M       1
 25     023               0
 26     024       F       0
 27     025       M       1
 28     027       F       0
 29     028       F       0
 30     029       M       1
 31     006       F       0
```

Notice that invalid values for GENDER and AE are now missing values, including the two lowercase 'f's (patient numbers 010 and 023).

Let's add one more feature to this program. By using the keyword UPCASE in the informat specification, you can automatically convert the values being read to uppercase before the ranges are checked. Here are the PROC FORMAT statements, rewritten to use this option.

```
PROC FORMAT;
   INVALUE $GEN (UPCASE)  'F' = 'F'
                          'M' = 'M'
                        OTHER = ' ';
   INVALUE $AE '0','1' = _SAME_
                OTHER  = ' ';
RUN;
```

The UPCASE option is placed in parenthesis following the informat name. Notice some other changes as well. You cannot use the keyword _SAME_ anymore because the value is changed to uppercase for comparison purposes, but the _SAME_ specification would leave the original lowercase value unchanged. By specifying each value individually, the lowercase 'f' (the only lowercase GENDER value) would match the range 'F' and be assigned the value of an uppercase 'F'.

The output of this data set is identical to the output for Program 1-8 except the value of GENDER for patients 010 and 023 are an uppercase 'F'.

If you want to preserve the original value of the variable, you can use a user-defined informat with an INPUT function instead of an INPUT statement. You can use this method to check a raw data file or a SAS data set. Program 1-9 reads the SAS data set CLEAN.PATIENTS and uses user-defined informats to detect errors.

Program 1-9 Using a User-Defined Informat with the INPUT Function

```
PROC FORMAT;
   INVALUE $GENDER 'F','M' = _SAME_
                    OTHER  = 'ERROR';
   INVALUE $AE      '0','1' = _SAME_
                    OTHER   = 'ERROR';
RUN;

DATA _NULL_;
   FILE PRINT;
   SET CLEAN.PATIENTS;
   IF INPUT (GENDER,$GENDER.) = 'ERROR' THEN
      PUT @1 "Error for Gender for Patient:" PATNO" Value is " GENDER;
   IF INPUT (AE,$AE.) = 'ERROR' THEN
      PUT @1 "Error for AE for Patient:" PATNO" Value is " AE;
RUN;
```

The advantage of this program over Program 1-8 is that the original values of the variables are not lost.