



# Cody's Data Cleaning Techniques Using SAS®

*Second Edition*

Ron Cody

SAS® Press Series





From *Cody's Data Cleaning Techniques Using SAS®*, Second Edition.  
Full book available for purchase [here](#).

## Table of Contents

---

List of Programs	ix
Preface	xv
Acknowledgments	xvii

### 1

#### Checking Values of Character Variables

---

Introduction	1
Using PROC FREQ to List Values	1
Description of the Raw Data File PATIENTS.TXT	2
Using a DATA Step to Check for Invalid Values	7
Describing the VERIFY, TRIM, MISSING, and NOTDIGIT Functions	9
Using PROC PRINT with a WHERE Statement to List Invalid Values	13
Using Formats to Check for Invalid Values	15
Using Informats to Remove Invalid Values	18

### 2

#### Checking Values of Numeric Variables

---

Introduction	23
Using PROC MEANS, PROC TABULATE, and PROC UNIVARIATE to Look for Outliers	24
Using an ODS SELECT Statement to List Extreme Values	34
Using PROC UNIVARIATE Options to List More Extreme Observations	35
Using PROC UNIVARIATE to Look for Highest and Lowest Values by Percentage	37
Using PROC RANK to Look for Highest and Lowest Values by Percentage	43
Presenting a Program to List the Highest and Lowest Ten Values	47
Presenting a Macro to List the Highest and Lowest "n" Values	50
Using PROC PRINT with a WHERE Statement to List Invalid Data Values	52
Using a DATA Step to Check for Out-of-Range Values	54
Identifying Invalid Values versus Missing Values	55

Listing Invalid (Character) Values in the Error Report	57
Creating a Macro for Range Checking	60
Checking Ranges for Several Variables	62
Using Formats to Check for Invalid Values	66
Using Informats to Filter Invalid Values	68
Checking a Range Using an Algorithm Based on Standard Deviation	71
Detecting Outliers Based on a Trimmed Mean and Standard Deviation	73
Presenting a Macro Based on Trimmed Statistics	76
Using the TRIM Option of PROC UNIVARIATE and ODS to Compute Trimmed Statistics	80
Checking a Range Based on the Interquartile Range	86

### 3

## Checking for Missing Values

---

Introduction	91
Inspecting the SAS Log	91
Using PROC MEANS and PROC FREQ to Count Missing Values	93
Using DATA Step Approaches to Identify and Count Missing Values	96
Searching for a Specific Numeric Value	100
Creating a Macro to Search for Specific Numeric Values	102

### 4

## Working with Dates

---

Introduction	105
Checking Ranges for Dates (Using a DATA Step)	106
Checking Ranges for Dates (Using PROC PRINT)	107
Checking for Invalid Dates	108
Working with Dates in Nonstandard Form	111
Creating a SAS Date When the Day of the Month Is Missing	113
Suspending Error Checking for Known Invalid Dates	114

**5****Looking for Duplicates and "n" Observations per Subject**

---

Introduction	117
Eliminating Duplicates by Using PROC SORT	117
Detecting Duplicates by Using DATA Step Approaches	123
Using PROC FREQ to Detect Duplicate ID's	126
Selecting Patients with Duplicate Observations by Using a Macro List and SQL	129
Identifying Subjects with "n" Observations Each (DATA Step Approach)	130
Identifying Subjects with "n" Observations Each (Using PROC FREQ)	132

**6****Working with Multiple Files**

---

Introduction	135
Checking for an ID in Each of Two Files	135
Checking for an ID in Each of "n" Files	138
A Macro for ID Checking	140
More Complicated Multi-File Rules	143
Checking That the Dates Are in the Proper Order	147

**7****Double Entry and Verification (PROC COMPARE)**

---

Introduction	149
Conducting a Simple Comparison of Two Data Sets	150
Using PROC COMPARE with Two Data Sets That Have an Unequal Number of Observations	159
Comparing Two Data Sets When Some Variables Are Not in Both Data Sets	161

**8****Some PROC SQL Solutions to Data Cleaning**

---

Introduction	165
A Quick Review of PROC SQL	166
Checking for Invalid Character Values	166
Checking for Outliers	168

Checking a Range Using an Algorithm Based on the Standard Deviation	169
Checking for Missing Values	170
Range Checking for Dates	172
Checking for Duplicates	173
Identifying Subjects with "n" Observations Each	174
Checking for an ID in Each of Two Files	174
More Complicated Multi-File Rules	176

## 9

### **Correcting Errors**

---

Introduction	181
Hardcoding Corrections	181
Describing Named Input	182
Reviewing the UPDATE Statement	184

## 10

### **Creating Integrity Constraints and Audit Trails**

---

Introducing SAS Integrity Constraints	187
Demonstrating General Integrity Constraints	188
Deleting an Integrity Constraint Using PROC DATASETS	193
Creating an Audit Trail Data Set	193
Demonstrating an Integrity Constraint Involving More than One Variable	200
Demonstrating a Referential Constraint	202
Attempting to Delete a Primary Key When a Foreign Key Still Exists	205
Attempting to Add a Name to the Child Data Set	207
Demonstrating the Cascade Feature of a Referential Constraint	208
Demonstrating the SET NULL Feature of a Referential Constraint	210
Demonstrating How to Delete a Referential Constraint	211

**11****DataFlux and dfPower Studio**

---

Introduction	213
Examples	215

**Appendix****Listing of Raw Data Files and SAS Programs**

---

Programs and Raw Data Files Used in This Book	217
Description of the Raw Data File PATIENTS.TXT	217
Layout for the Data File PATIENTS.TXT	218
Listing of Raw Data File PATIENTS.TXT	218
Program to Create the SAS Data Set PATIENTS	219
Listing of Raw Data File PATIENTS2.TXT	220
Program to Create the SAS Data Set PATIENTS2	221
Program to Create the SAS Data Set AE (Adverse Events)	221
Program to Create the SAS Data Set LAB_TEST	222
Listings of the Data Cleaning Macros Used in This Book	222

**Index**

239

From *Cody's Data Cleaning Techniques Using SAS®, Second Edition* by Ron Cody. Copyright © 2008, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



From *Cody's Data Cleaning Techniques Using SAS®*, Second Edition. Full book available for purchase [here](#).

# 1 Checking Values of Character Variables

---

Introduction	1
Using PROC FREQ to List Values	1
Description of the Raw Data File PATIENTS.TXT	2
Using a DATA Step to Check for Invalid Values	7
Describing the VERIFY, TRIM, MISSING, and NOTDIGIT Functions	9
Using PROC PRINT with a WHERE Statement to List Invalid Values	13
Using Formats to Check for Invalid Values	15
Using Informats to Remove Invalid Values	18

## 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 pattern, 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.

You can download all the programs and data files used in this book from the SAS Web site: <http://support.sas.com/publishing>. Click the link for SAS Press Companion Sites and select *Cody's Data Cleaning Techniques Using SAS, Second Edition*. Finally, click the link for Example Code and Data and you can download a text file containing all of the programs, macros, and text files used in this book.

## 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. Programs, data files, SAS data sets, and macros used in this book are stored in the folder C:\BOOKS\CLEAN. For example, the file PATIENTS.TXT is located in a folder (directory) called C:\BOOKS\CLEAN. You will need to 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) containing 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:\BOOKS\CLEAN          |
|PURPOSE: To create a SAS data set called PATIENTS    |
*-----*;
```

```
libname clean "c:\books\clean";

data clean.patients;
    infile "c:\books\clean\patients.txt" trunccover /* take care of problems
                                                    with short records */;

    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 TRUNCOVER option in the INFILE statement. This will seem foreign to most mainframe users. If you do not use this option and you have short records, SAS will, by default, go to the next record to read data. The TRUNCOVER option prevents this from happening. The TRUNCOVER option is also useful when you are using list input (delimited data values). In this case, if you have more variables on the INPUT statement than there are in a single record on the data file, SAS will supply a missing value for all the remaining variables. One final note about INFILE options: If you have long record lengths (greater than 256 on PCs and UNIX platforms) you need to use the LRECL= option to change the default logical record length.

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**

```
title "Frequency Counts for Selected Character Variables";
proc freq data=clean.patients;
    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

*(continued)*

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.

For those users who like shortcuts, here is another way to have PROC FREQ select the same set of variables in the example above, without having to list them all.

### Program 1-3 Using the Keyword `_CHARACTER_` in the TABLES Statement

```
title "Frequency Counts for Selected Character Variables";
proc freq data=clean.patients(drop=Patno);
    tables _character_ / nocum nopercnt;
run;
```

The keyword `_CHARACTER_` in this example is equivalent to naming all the character variables in the `CLEAN.PATIENTS` data set. Since you don't want the variable `Patno` included in this list, you use the `DROP=` data set option to remove it from the list.

## 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 1-4 Using a DATA \_NULL\_ Step to Detect Invalid Character Data

```
title "Listing of invalid patient numbers and data values";
data _null_;
    set clean.patients;
    file print; ***send output to the output window;
    ***check Gender;
    if Gender not in ('F' 'M' ' ' ' ') then put Patno= Gender=;
    ***check Dx;
    if verify(trim(Dx),'0123456789') and not missing(Dx)
        then put Patno= Dx=;
    /*****
    SAS 9 alternative:
    if notdigit(trim(Dx)) and not missing(Dx)
        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 and print them out, there is no need to create a SAS data set. The reserved data set name \_NULL\_ tells SAS not to create a data set. This is a major efficiency technique. In this program, you avoid using all the resources to create a data set when one isn't needed.

## 8 Cody's Data Cleaning Techniques Using SAS, Second Edition

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. An equivalent statement to the one above is:

```
if X in ('A','B','C') then . . . ;
```

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.

If you know from the start that you may have both upper- and lowercase values in your raw data file, you could use the \$UPCASE informat to convert all lowercase values to uppercase. For example, to read all Gender values in uppercase, you could use:

```
@4 Gender $upcase1.
```

to replace the line that reads Gender values in Program 1-1.

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 as a Dx value. The next section describes the VERIFY function along with several other functions.

## Describing the VERIFY, TRIM, MISSING, and NOTDIGIT Functions

The verify function takes the form:

```
verify(character_variable,verify_string)
```

where *verify\_string* is a character value (either the name of a character variable or a series of 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 some examples of the VERIFY function.

Suppose you have a variable called ID that is stored in five bytes and is supposed to contain only the letters X, Y, Z, and digits 0 through 5. For example, valid values for ID would be X1234 or 34Z5X. You could use the VERIFY function to see if the variable ID contained any characters other than X, Y, Z and the digits 0 through 5 like this:

```
Position = verify(ID, 'XYZ012345');
```

Suppose you had an ID value of X12B44. The value of Position in the line above would be 4, the position of the first invalid character in ID (the letter B). If no invalid characters are found, the VERIFY function returns a 0. Therefore, you can write an expression like the following to list invalid values of ID:

```
if verify(ID, 'XYZ012345') then put "Invalid value of ID:" ID;
```

This may look strange to you. You might prefer the statement:

```
if verify(ID, 'XYZ012345') gt 0 then put "Invalid value of ID:" ID;
```

However, these two statements are equivalent. Any numerical value in SAS other than 0 or missing is considered TRUE. You usually think of true and false values as 1 or 0—and that is what SAS returns to you when it evaluates an expression. However, it is often convenient to use values other than 1 to represent TRUE. When SAS evaluates the VERIFY function in either of the two statements above, it returns a 4 (the position of the first invalid character in the ID). Since 4 is neither 0 or missing, SAS interprets it as TRUE and the PUT statement is executed.

There is one more potential problem when using the VERIFY function. Suppose you had an ID equal to 'X123'. What would the expression

```
verify(ID, 'XYZ012345')
```

return? You might think the answer is 0 since you only see valid characters in the ID (X, 1, 2, and 3). However, the expression above returns a 5! Why? Because that is the position of the first trailing blank. Since ID is stored in 5 bytes, any ID with fewer than 5 characters will contain trailing blanks—and blanks, even though they are sometimes hard to see, are still considered characters to be tested by the VERIFY function.



To avoid problems with trailing blanks, you can use the TRIM function to remove any trailing blanks before the VERIFY function operates. Therefore, the expression

```
verify(trim(ID), 'XYZ012345')
```

will return a 0 for all valid values of ID, even if they are shorter than 5 characters.

There is one more problem to solve. That is, the expression above will return a 1 for a missing value of ID. (Think of character missing values as blanks). The MISSING function is a useful way to test for missing values. It returns a value of TRUE if its argument contains a missing value and a value of FALSE otherwise. And, this function can take character or numeric arguments! The MISSING function has become one of this author's favorites. It makes your SAS programs much more readable. For example, take the line in Program 1-4 that uses the MISSING function:

```
if verify(trim(Dx), '0123456789') and not missing(Dx)
    then put Patno= Dx=;
```

Without the MISSING function, this line would read:

```
if verify(trim(Dx), '0123456789') and Dx ne ' '
    then put Patno= Dx=;
```

If you start using the MISSING function in your SAS programs, you will begin to see statements like the one above as clumsy or even ugly.

You are now ready to understand the VERIFY function that checked for invalid Dx codes. The verify string contained the characters (numerals) 0 through 9. Thus, if the Dx code contains any character other than 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. Output from Program 1-4 is shown below:

```
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.

If you have SAS 9 or higher, you can use the NOTDIGIT function.

```
notdigit(character_value)
```

is equivalent to

```
verify(character_value, '0123456789')
```

That is, the NOTDIGIT function returns the first position in *character\_value* that is not a digit. The NOTDIGIT function treats trailing blanks the same way that the VERIFY function does, so if you have character strings of varying lengths, you may want to use the TRIM function to remove trailing blanks.

Using the NOTDIGIT function, you could replace the VERIFY function in Program 1-4 like this:

```
if notdigit(trim(Dx)) and not missing(Dx)
  then put Patno= Dx=;
```

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 statement:

```
if notdigit(trim(Patno)) 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 1-5.

### Program 1-5 Using PROC PRINT to List Invalid Character Values

```
title "Listing of invalid gender values";
proc print data=clean.patients;
  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 who 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. However, the WHERE statement does require that all variables already exist in the data set being processed. 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 1-5 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 1-6.

#### **Program 1-6 Using PROC PRINT to List Invalid Character Data for Several Variables**

```

title "Listing of invalid character values";
proc print data=clean.patients;
  where Gender not in ('M' 'F' ' ') or
         notdigit(trim(Dx)) and not missing(Dx) 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 1-4. 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.

## 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 1-7 uses formats to convert all invalid data values to a single value.

### Program 1-7 Using a User-Defined Format and PROC FREQ to List Invalid Data Values

```
proc format;
  value $gender 'F','M' = 'Valid'
               ' '      = 'Missing'
               other    = 'Miscoded';
```

## 16 Cody's Data Cleaning Techniques Using SAS, Second Edition

```
value $ae '0','1' = 'Valid'
          ' '      = 'Missing'
          other    = 'Miscoded';

run;

title "Using formats to identify invalid values";
proc freq data=clean.patients;
  format Gender $gender.
           AE    $ae.;
  tables Gender AE/ nocum nopercnt 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'.

You may choose to combine 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. (Important note: When you use the MISSING TABLES option with PROC FREQ and you are outputting percentages, the percentages are computed by dividing the number of a particular value by the total number of observations, missing or non-missing.) 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

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 1-8 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 $ae '0','1' = 'Valid'
           ' '      = 'Missing'
           other    = 'Miscoded';
run;

title "Listing of invalid patient numbers and data values";
data _null_;
  set clean.patients(keep=Patno Gender AE);
  file print; ***Send output to the output window;
```

## 18 Cody's Data Cleaning Techniques Using SAS, Second Edition

```
if put(Gender,$gender.) = 'Miscoded' then put Patno= Gender=;  
if put(AE,$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 1-8, the first argument of the PUT function is a character variable you want to test and the second argument is the corresponding character format. The result of the PUT function for any invalid data values would be the value 'Miscoded'.

Here is the output from Program 1-8.

```
Listing of invalid patient numbers and data values  
Patno=003 gender=X  
Patno=004 AE=A  
Patno=010 gender=f  
Patno=013 gender=2  
Patno=023 gender=f
```

## Using Informats to Remove 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 31 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 1-9 Using a User-Defined Informat to Set Invalid Data Values to Missing**

```

*-----*
| 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:\books\clean";

proc format;
  invalue $gen      'F','M' = _same_
                   other   = ' ';
  invalue $ae       '0','1' = _same_
                   other   = ' ';
run;

data clean.patients_filtered;
  infile "c:\books\clean\patients.txt" pad;
  input @1  Patno      $3.
        @4  Gender    $gen1.
        @27 AE        $ae1.;

  label Patno      = "Patient Number"
        Gender     = "Gender"
        AE         = "adverse event?";
run;

title "Listing of data set PATIENTS_FILTERED";
proc print data=clean.patients_filtered;
  var Patno Gender AE;
run;

```

Notice the **INVALUE** statements in the **PROC FORMAT** above. The keyword **\_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 keyword **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 PATIENTS\_FILTERED

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 't'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 1-9 except the value of `GENDER` for patients 010 and 023 is 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 1-10 reads the SAS data set `CLEAN.PATIENTS` and uses user-defined informats to detect errors.

### **Program 1-10 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;
```

## 22 Cody's Data Cleaning Techniques Using SAS, Second Edition

```
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 1-9 is that the original values of the variables are not lost.

Output from Program 1-10 is shown below:

```
Listing of invalid character values
Error for Gender for patient:003  value is X
Error for AE for patient:004  value is A
Error for Gender for patient:006  value is
Error for Gender for patient:010  value is f
Error for Gender for patient:013  value is 2
Error for AE for patient:013  value is
Error for Gender for patient:023  value is f
```

From *Cody's Data Cleaning Techniques Using SAS®, Second Edition* by Ron Cody. Copyright © 2008, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



From *Cody's Data Cleaning Techniques Using SAS®, Second Edition*.  
Full book available for purchase [here](#).

# Index

## A

- AIC data management 214
- algorithm based on standard deviation,
  - checking ranges with 71–72, 169–170
- `_ALL_` keyword 122–123
- ampersand (&) 41, 102
- AND operator 53
- APPEND procedure 187
  - adding errors to data sets 65
  - adding names to child data sets 207
  - audit trail data 195, 198
  - BASE= option 207
  - DATA= option 207
- ARRAY statement 101
- asterisk (\*) 171
- `_ATDATETIME_` automatic variable 196
- `_ATMESSAGE_` automatic variable 196, 198
- `_ATOPCODE_` automatic variable 196–197
- `_ATRETURNCODE_` automatic variable 196
- `_ATUSERID_` automatic variable 196
- AUDIT statement, DATASETS procedure 195
- audit trails 193–200

## B

- bar charts 33
- BASE= option
  - APPEND procedure 207
  - COMPARE procedure 152
- BETWEEN keyword, WHERE statement (PRINT) 107
- blanks, trailing 10, 12
- Boolean operators 53
- box plots 33
- BOXPLOT procedure 33

- BRIEF option, COMPARE procedure 155, 158
- BY statement 124, 137
- BY variables
  - `_ALL_` keyword 122–123
  - detecting duplicates 125
  - ID variables as 135–138
  - NODUPKEY option, SORT procedure 120

## C

- CALL SYMPUT routine 48
- CALL SYMPUTX routine 48
- Cartesian product 166
- CASCADE feature 203, 208–210
- case conversions 9
- `$CHAR` informat 109–110, 159
- `_CHARACTER_` keyword 6, 94
- character variables
  - checking for invalid dates 110
  - checking for invalid values with DATA step 7–13
  - checking for invalid values with formats 15–18
  - checking for invalid values with SQL procedure 166–168
  - checking for missing values 170
  - checking values with IF statement 13
  - counting missing values 93–96
  - listing invalid values with WHERE statement 13–15
  - listing values with FREQ procedure 1–6
  - removing invalid values with informat 18–22
- Check integrity constraint 188
- child data sets 202, 207–208
- CIMPORT procedure 187
- COMPARE= option, COMPARE procedure 152

COMPARE procedure 149  
 BASE= option 152  
 BRIEF option 155, 158  
 COMPARE= option 152  
 comparing data sets with selected variables 161–163  
 comparing data sets with unequal observations 159–160  
 comparing two data sets 150–159  
 ID statement 152, 159–160  
 LISTBASE option 159–160  
 LISTCOMP option 159–160  
 TRANSPOSE option 156  
 VAR statement 163  
 CONTENTS procedure 190–191, 205  
 converting lowercase to uppercase 9  
 COPY procedure 187  
 corrections  
   *See* error handling  
 COUNT function 173–174  
 counting missing values 93–100  
 CPORT procedure 187  
 CREATE clause, SQL procedure 166

## D

DATA \_NULL\_ step  
 checking for out-of-range values 54–55  
 checking range based on interquartile range 88  
 detecting invalid character data 7–9  
 identifying subjects with  $n$  observations 131  
 listing highest/lowest ten values 47–49  
 MERGE statement 137  
 WHERE statement comparison 14  
 DATA= option, APPEND procedure 207  
 data sets  
   adding errors to 65  
   adding general integrity constraints to 189–191  
   child 202, 207–208  
   comparing with selected variables 161–163

comparing with unequal observations 159–160  
 comparing two 150–159  
 creating 125, 143–144  
 creating audit trails 193–200  
 integrity constraints and 187–191, 202  
 parent 202

## DATA step

checking for invalid values 7–13  
 checking for out-of-range values 54–55  
 checking ranges for dates 106  
 counting missing values 96–100  
 detecting duplicates 123–126  
 identifying missing values 96–100  
 identifying subjects with  $n$  observations 130–132  
 IF statement 13, 106  
 integrity constraints 187  
 listing invalid values 15, 17  
 reading data in 182  
 SQL procedure alternative 165

## DataFlux 213–216

DATASETS procedure 43, 65  
 AUDIT statement 195  
 audit trail data sets 195  
 IC CREATE statement 190–191  
 integrity constraints 188, 190–191, 193, 200–202, 211–212  
 MESSAGE= option 194  
 MSGTYPE=USER option 194  
 NOLIST option 43

## DATE9. format 107

## dates

checking for invalid 108–111  
 checking order of 147–148  
 checking ranges 106–107, 172  
 creating when day of month is missing 113–114  
 printing 105  
 reading 105  
 storing 105  
 suspending error checking for unknown 114–116

- working with nonstandard forms
  - 111–112
- dfPower Studio 213–216
- DISTINCT option, SELECT clause (SQL)
  - 122–123
- DO loops 101, 103
- double entry and verification
  - data sets with selected variables 161–163
  - data sets with unequal observations
    - 159–160
  - defined 149
  - two data sets 150–159
- DOWNLOAD procedure 187
- DROP= data set option 6
- DROP statement 59, 199
- duplicate ID numbers
  - checking with SQL procedure 173
  - detecting 123–129
  - eliminating 117–123
- duplicate observations
  - detecting 123–126
  - eliminating 117
  - identifying subjects with 130–133
  - selecting patients with 129–130

## E

- EDA (exploratory data analysis) 86
- error handling
  - audit trail data and 199
  - describing named input 182–184
  - hardcoding corrections 181–182
  - suspending for unknown dates 114–116
  - UPDATE statement and 184–186
- error reports
  - listing invalid values 57–60
  - reading invalid dates 108–109
- \_ERROR\_ variable 56, 115–116
- errors, adding to data sets 65
- ERRORS= system option 109
- %EVAL function 43, 46
- exploratory data analysis (EDA) 86
- extreme observations, listing 34–37

## F

- files, multiple
  - See* multiple files
- filtering invalid values with informats 68–70
- FIRST. temporary variable 123–125, 130
- FIRSTOBS= data set option 48
- foreign keys
  - adding names to child data sets 207–208
  - deleting primary keys 205–206
  - referential constraints and 202–203, 208–212
- FORMAT= option, TABULATE procedure
  - 25
- FORMAT procedure 18
  - invalid values with informats 69
  - INVALUE statement 18–19, 21, 69
- formats
  - checking for invalid values 15–18, 66–68
  - printing dates 105
- FREQ procedure
  - checking invalid values 1–6
  - counting missing values 93–96
  - detecting duplicates 126–129
  - identifying subjects with *n* observations
    - 132–133
  - listing character variable values 1–6
  - listing invalid values 15–16
  - listing variable names 104
  - MISSING option 94
  - TABLES statement 4, 6, 16, 94, 126–127
- FROM clause, SQL procedure 166, 171
- FSEDIT procedure 186
- FULL JOIN operation 174–177
- fuzzy sorts 215

## G

- Gaussian distribution 34
- general integrity constraints
  - adding to data sets 189–191

general integrity constraints (*continued*)  
     defined 187, 202  
     types of 188  
 GROUP BY clause, SQL procedure 166, 173  
 GROUPS= option, RANK procedure 44, 46,  
     73

## H

hardcoding corrections 181–182  
 HAVING clause, SQL procedure 169, 173  
 high values  
     finding by percentage 37–47  
     listing highest ten 35–37, 47–52  
     UNIVARIATE procedure 32, 35–43  
 HISTOGRAM statement, UNIVARIATE  
     procedure 33  
 horizontal bar charts 33

## I

IC CREATE statement, DATASETS  
     procedure 190–191  
 ID checking  
     in each of  $n$  files 138–143  
     in multiple files 135–138, 174–176  
     macro for 140–143  
 ID numbers  
     *See* duplicate ID numbers  
 ID statement  
     COMPARE procedure 152, 159–160  
     UNIVARIATE procedure 38  
 ID variables  
     as BY variable 135–138  
     checking with SQL procedure 175  
 IF statement  
     checking character variable values 13  
     checking date order 148  
     checking ranges for dates 106  
 IN= data set option 127, 135–138, 142  
 IN operator 8  
 INFILE statement 4  
 informats 18  
     ?? modifier 114–116

    checking for invalid dates 108–111  
     filtering invalid values 68–70  
     in INPUT function 18, 21–22  
     in INPUT statement 19  
     in INVALUE statement (FORMAT)  
         18–19  
     reading dates 105  
     removing invalid values 18–22  
 INITIATE option, AUDIT statement  
     (DATASETS) 195  
 INPUT function  
     ?? informat modifier 114–116  
     checking for invalid dates 110  
     checking for missing values 91  
     checking values of numeric variables 59  
     informats in 18, 21–22  
     PUT function comparison 18  
 INPUT statement  
     ?? informat modifier 114–116  
     \_ERROR\_ variable 56  
     informats in 19  
 integrity constraints  
     *See also* general integrity constraints  
     *See also* referential integrity constraints  
     adding user messages 194–195  
     audit trail data sets and 193–200  
     Check 188  
     creating 190  
     data sets and 187–191, 202  
     defined 187–188  
     deleting 193  
     demonstrating 189–190, 202–205  
     involving multiple variables 200–202  
     Not Null 188  
     Primary Key 188  
     reporting violations 197–198  
     types of 187–188  
     Unique 188  
 interquartile range 33, 86–88  
 invalid dates, checking for 108–111  
 invalid values  
     checking with DATA step 7–13  
     checking with formats 15–18, 66–68



- checking with FREQ procedure 1–6
- checking with SQL procedure 166–168
- filtering with informats 68–70
- identifying missing values versus 55–57
- listing in error report 57–60
- listing with DATA step 15, 17
- listing with FREQ procedure 15–16
- listing with PRINT procedure 13–15, 52–54
- listing with WHERE statement 13–15
- looking for outliers 24–34
- removing with informats 18–22
- setting to missing 19

INVALUE statement, FORMAT procedure

- filtering invalid values with informats 69
- informats in 18–19
- UPCASE keyword 21

IS MISSING keyword 167, 170

IS NULL keyword 167

## J

JOIN operations 174–179

## K

KEEP= data set option 48, 64

KEYLABEL statement, TABULATE procedure 26

keypunch machine, verifier 149

## L

LAG function 98

LAG2 function 98

LAST. temporary variable 123–125, 130

LEFT JOIN operation 177–178

LENGTH statement 183

%LET statement 64, 76

LISTBASE option, COMPARE procedure 159–160

LISTCOMP option, COMPARE procedure 159–160

log

- ?? modifier 115
- inspecting missing values 91–93

- reading invalid dates 108–109

low values

- finding by percentage 37–47
- listing lowest ten 35–37, 47–52
- UNIVARIATE procedure 32, 35–43

lowercase, converting to uppercase 9

LRECL= option, INFILE statement 4

## M

%MACRO statement 41

macro variables 41, 102

macros

- automating range checking 60–62
- checking range based on interquartile range 86–88
- checking ranges for several variables 62–66
- defined 41
- detecting outliers based on trimmed statistics 76–80
- ID checking 140–143
- listing highest/lowest percentage 40–41, 44–47
- listing highest/lowest values 50–52
- listing outliers of several variables 82–86
- named parameters 41
- searching for specific numeric 102–104
- selecting patients with duplicate observations 129–130
- semi-colons and 43

MAX option, MEANS procedure 24

MAXDEC= option, MEANS procedure 24

MDY function 105, 111–114

MEAN summary function 169

MEANS procedure

- checking range based on interquartile range 86, 88
- counting missing values 93–96
- detecting outliers based on 24–25, 71–76
- MAX option 24
- MAXDEC= option 24
- MIN option 24

MEANS procedure (*continued*)

N option 24, 26, 94

NMISS option 24, 94

VAR statement 94

WHERE statement 74

%MEND statement 41

MERGE statement 137, 176

MERGENOBY ERROR system option 137

MERGENOBY NOWARN system option  
137

MERGENOBY system option 137

MERGENOBY WARN system option 137

MESSAGE= option, DATASETS procedure  
194

messages, and integrity constraints 194–195

MIN option, MEANS procedure 24

MISSING function 11, 96

MISSING option

FREQ procedure 94

TABLES statement (FREQ) 16, 94

missing values

checking with INPUT function 91

checking with SQL procedure 170–171

counting 93–100

identifying invalid values versus 55–57

inspecting SAS log 91–93

named input method and 183

removing from listings 110–111

searching for specific numeric 100–104

setting invalid values to 19

MMDDYY10. format 107–109

MONYY informat 113

MPRINT system option 42

MSGTYPE=USER option, DATASETS  
procedure 194

multiple files

checking date order 147–148

checking IDs in 135–138, 174–176

checking IDs in each of "n" 138–143

complicated rules 143–147, 176–179

**N**

\$n. informat 109

## N option

KEYLABEL statement (TABULATE)  
26

MEANS procedure 24, 26, 94

named input method 182–184

named parameters 41

names

adding to child data sets 207–208

obtaining for output objects 34

NEXTROBS= option, UNIVARIATE  
procedure 35–37NEXTRVALS= option, UNIVARIATE  
procedure 35–37

NMISS option

KEYLABEL statement (TABULATE)  
26

MEANS procedure 24, 94

NOBOS= option, SET statement 48

NOCUM option, TABLES statement (FREQ)  
4NODUPKEY option, SORT procedure  
118–120, 137NODUPRECS option, SORT procedure 118,  
120–123

NOLIST option, DATASETS procedure 43

NOPERCENT option, TABLES statement  
(FREQ) 4NOPRINT option, UNIVARIATE procedure  
38, 81

normal distribution 34

normal probability plots 34

Not Null integrity constraint 188

NOT operator 8

NOTDIGIT function 12–13, 59  
identifying missing values 98

\_NULL\_ reserved data set name 7

\_NUMERIC\_ keyword 101

numeric macros, searching for specific  
102–104numeric missing values, searching for specific  
100–104

numeric variables

checking for missing values 170

- checking for out-of-range values 54–55
- checking ranges based on interquartile range 86–88
- checking ranges with algorithm 71–72
- checking values with INPUT function 59
- computing trimmed statistics 80–86
- counting missing values 93–96
- creating range checking macro 60–62
- detecting outliers based on standard deviation 73–76
- detecting outliers based on trimmed statistics 73–80
- filtering invalid values with informat 68–70
- finding highest/lowest values by percentage 37–47
- formats to check invalid values 66–68
- identifying invalid versus missing values 55–57
- listing extreme values 34–37
- listing highest/lowest ten values 47–52
- listing invalid values 52–54, 57–60
- looking for outliers 24–34
- range checking for multiple variables 62–66
- searching for specific 100–102

## O

### observations

- See also* duplicate observations
- comparing data sets with unequal observations 159–160
- listing extreme observations 34–35

- ODS (Output Delivery System) 80–86
- ODS LISTING statement 81
- ODS OUTPUT statement 81–82
- ODS SELECT statement 34–35
- operators 8, 53
- OR operator 53
- ORDER BY clause, SQL procedure 166
- OTHER keyword 19
- out-of-range values
  - checking for 54–55, 66–68

- listing 52–54

### OUT= option

- OUTPUT statement (UNIVARIATE) 38
- SORT procedure 118
- TABLES statement (FREQ) 126–127

### outliers

- box plot example 33
- checking with SQL procedure 168
- detecting based on standard deviation 71–76
- detecting based on trimmed mean 73–76
- detecting based on trimmed statistics 76–80
- listing outliers of several variables 82–86
- looking for in numeric variables 24–34

- Output Delivery System (ODS) 80–86

- OUTPUT destination 81

- output devices 8, 166

- output objects, obtaining names 34

- OUTPUT statement, UNIVARIATE procedure 38

## P

- parameters, named 41

- parent data sets 202

- patients, selecting with duplicate observations 129–130

- PATIENTS.TXT raw data file 2–6

- PCTLPRE= option, OUTPUT statement (UNIVARIATE) 38

- PCTLPTS= option, OUTPUT statement (UNIVARIATE) 38

- PDV (Program Data Vector) 56

- percentage, finding values by 37–47

- PLOT option, UNIVARIATE procedure 26

### primary key

- deleting when foreign key exists 205–206

- referential constraints and 202, 208–212

- Primary Key integrity constraint 188

### PRINT procedure

- checking ranges for dates 107

**PRINT procedure** (*continued*)

- listing invalid values 13–15, 52–54
- viewing audit trail data 193, 195–198
- WHERE statement 13–15, 52–54, 98, 107, 130

printing dates 105

probability plots 34

Program Data Vector (PDV) 56

PUT function 18, 67

**PUT statement**

- checking ranges for dates 106
- formats checking for invalid values 67
- identifying missing values 96
- sending results to output device 8

**Q**

question mark (?) 114–116

QUOTE function 129

**R****range checking**

- automating 60–62
- based on interquartile range 86–88
- checking for out-of-range values 54–55, 66–68
- for dates 106–107, 172
- for multiple variables 62–66
- listing out-of-range values 52–54
- with algorithm based on standard deviation 71–72, 169–170

**RANK procedure**

- GROUPS= option 44, 46, 73
- highest/lowest values by percentage 37, 43–47
- RANKS statement 44, 46
- VAR statement 44

RANKS statement, RANK procedure 44, 46

reading data, with DATA step 182

**referential integrity constraints**

- adding names to child data sets 207–208
- CASCADE feature 203, 208–210
- defined 187–188, 202
- deleting 211–212

deleting primary key when foreign key exists 205–206

demonstrating 202–205

primary key and 202, 208–212

RESTRICT feature 202

SET NULL feature 202, 210–211

REPORT procedure 193, 197–198

RESTRICT feature 202

RIGHT JOIN operation 177

RTSPACE= option, TABLE statement (TABULATE) 25

**S**

\_SAME\_ keyword 19, 21

SAS Component Language (SCL) 188

SAS dates

*See* dates

SAS log

*See* log

SCAN function 142

%SCAN function 142

SCL (SAS Component Language) 188

SELECT clause, SQL procedure 166

asterisk (\*) in 171

DISTINCT option 122–123

QUOTE function 129

semi-colon (;) 43

SET NULL feature 202, 210–211

**SET statement**

- adding names to child data sets 207
- detecting duplicates 124
- example 39, 42
- executing once 72
- NOBS= option 48

**SORT procedure**

eliminating duplicates 117–123

NODUPKEY option 118–120, 137

NODUPRECS option 118, 120–123

OUT= option 118

%SCAN function and 142

sorts, fuzzy 215

SQL procedure 166

as DATA step alternative 165

- checking for duplicates 173
- checking for IDs in multiple files 174–176
- checking for invalid character values 166–168
- checking for missing values 170–171
- checking for outliers 168
- checking ranges based on standard deviation 169–170
- CREATE clause 166
- FROM clause 166, 171
- GROUP BY clause 166, 173
- HAVING clause 169, 173
- identifying subjects with  $n$  observations 174
- integrity constraints 187–188
- JOIN operations 174–179
- multi-file rules 176–179
- ORDER BY clause 166
- ordering clauses 166
- removing duplicate records 122–123
- SELECT clause 122–123, 129, 166, 171
- selecting patients with duplicate observations 129–130
- WHERE clause 166–167, 170, 176, 191
- standard deviation
  - checking ranges 71–72, 169–170
  - computing from standard error 82
  - detecting outliers based on 71–76
- standard error 82
- STD summary function 169
- stem-and-leaf plots 33
- subjects, identifying with  $n$  observations 130–133
- SUM statement 131
- SUSPEND option, AUDIT statement (DATASETS) 195
- SYMPUT CALL routine 48
- SYMPUTX CALL routine 48

## T

- TABLE statement, TABULATE procedure 25

- TABLES statement, FREQ procedure
  - \_CHARACTER\_ keyword 6, 94
  - listing unique values 4
  - MISSING option 16, 94
  - NOCUM option 4
  - NOPERCENT option 4
  - OUT= option 126–127
- TABULATE procedure
  - FORMAT= option 25
  - KEYLABEL statement 26
  - looking for outliers 25–26
  - TABLE statement 25
  - VAR statement 25
- temporary variables 123–125, 130
- TERMINATE option, AUDIT statement (DATASETS) 195
- trailing blanks, removing 10, 12
- TRANSPOSE option, COMPARE procedure 156
- TRIM function 10, 12
  - identifying missing values 98
- TRIM= option, UNIVARIATE procedure 80–82
- trimmed statistics
  - computing 72–76, 80–86
  - detecting outliers based on 73–80
  - macro example 76–80
- TRUNCOVER option, INFILE statement 4
- TYPE= data set option 193, 196
- TYPE=AUDIT data set option 193

## U

- Unique integrity constraint 188
- unique values 94
- UNIVARIATE procedure
  - highest/lowest values by percentage 32, 35–43
  - HISTOGRAM statement 33
  - ID statement 38
  - listing extreme values 35–37
  - looking for outliers 24, 26–33
  - NEXTROBS= option 35–37
  - NEXTRVALS= option 35–37

UNIVARIATE procedure (*continued*)  
 NOPRINT option 38, 81  
 ODS statement support 34  
 OUTPUT statement 38  
 PLOT option 26  
 TRIM= option 80–82  
 unknown dates, checking for 114–116  
 UPCASE function 5, 9  
 \$UPCASE informat 9  
 UPCASE keyword 21  
 UPDATE statement 184–186  
 UPLOAD procedure 187  
 uppercase, converting lowercase to 9  
 user messages, and integrity constraints 194–195

## V

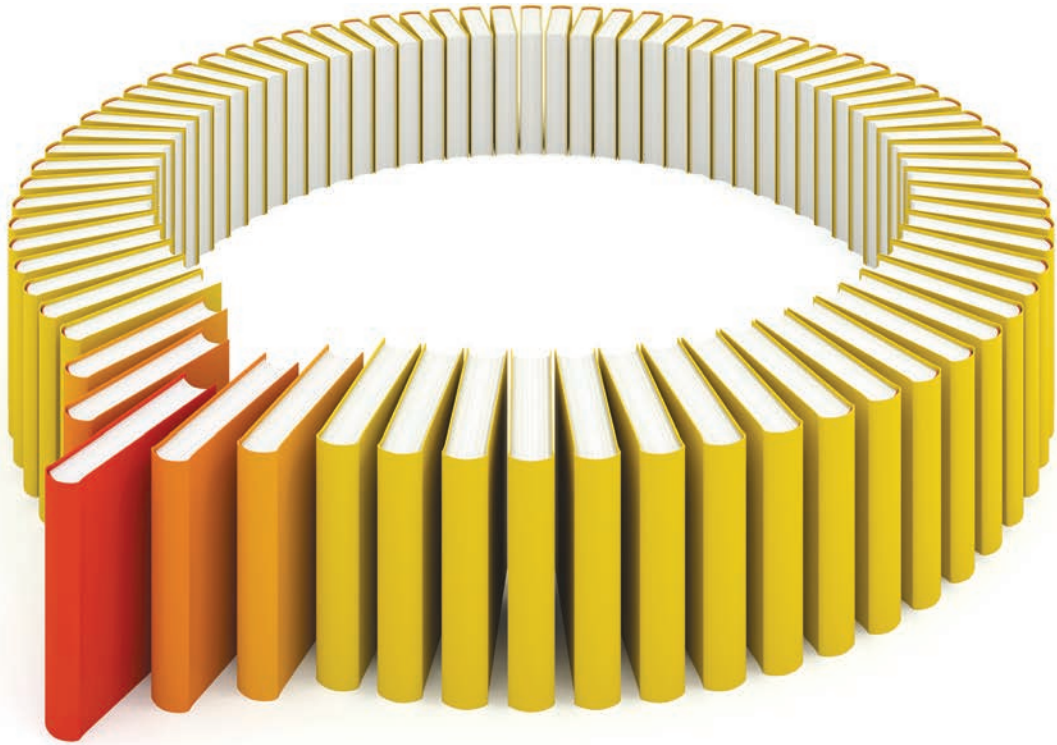
VAR statement  
 COMPARE procedure 163  
 MEANS procedure 94  
 RANK procedure 44  
 TABULATE procedure 25  
 variables  
   *See also* character variables  
   automatic 196–197, 198  
   BY variables 120, 122–123, 125, 135–138  
   comparing data sets with selected variables 161–163  
   \_ERROR\_ 56, 115–116  
   ID variables 135–138, 175  
   integrity constraints and multiple variables 200–202  
   listing variable names 104  
   macro variables 41, 102  
   range checking for multiple 62–66  
   temporary 123–125, 130  
 verifier keypunch machine 149  
 VERIFY function 9–13  
 VNAME function 100–103

## W

WHERE clause, SQL procedure  
   checking for invalid character values 166–167  
   checking for missing values 170  
   integrity constraints and 191  
   multi-file rules 176  
 WHERE= data set option  
   checking values of numeric variables 46, 48  
   detecting duplicates 127  
   multiple files 146  
 WHERE= option, IC CREATE statement (DATASETS) 190–191  
 WHERE statement  
   listing invalid values 13–15  
   MEANS procedure 74  
   PRINT procedure 13–15, 52–54, 98, 107, 130  
 whiskers 33

## Symbols

& (ampersand) 41, 102  
 \* (asterisk) 171  
 ?? informat modifier 114–116  
 ; (semi-colon) 43



# Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

 [support.sas.com/bookstore](https://support.sas.com/bookstore)  
for additional books and resources.

 **sas**  
THE POWER TO KNOW®

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies. © 2013 SAS Institute Inc. All rights reserved. S107969US.0413