Lesson 8: Validating and Cleaning Data

Summary

Main Points

Overview of Validating and Cleaning Data

- Validating data is the process of identifying invalid values in your data. For example, you might have character values stored in a numeric variable. Or you might have invalid date values such as *99NOV1978*. Invalid data values cause data errors when the program runs.
- In order to determine which values in your data are invalid, you must be familiar with the requirements for your data. For example, you might have variables that require unique and non-missing values, or whose values must fall into a specified set or range of values.
- You can use several SAS procedures for detecting invalid data, including PROC PRINT, PROC FREQ, PROC MEANS, and PROC UNIVARIATE.
- After you identify invalid data, you need to clean it. If possible, you should clean the original source of data, such as the raw data file. You can also use integrity constraints, also known as data validation rules, to prevent invalid data from being stored in a SAS data set.
- If you must clean the data after it is in a SAS data set, you can do so interactively using the VIEWTABLE window, or programmatically using the DATA step, PROC SQL, or PROC SORT. You can also clean data using the SAS Dataflux product dfPower Studio.

Examining Data Errors When Reading Raw Data Files

- When SAS encounters a data error in reading a raw data file, it writes messages to the log about the error, assigns a missing value to the variable that the invalid data affected, and continues processing. SAS also prints the values of **_ERROR_** and **_N_** to the log.
- Remember, the variable **_ERROR_** indicates whether or not SAS encountered an error during processing. The variable **_N_** counts the number of times the DATA step iterates.

Validating Data with PROC PRINT and PROC FREQ

```
PROC PRINT DATA=SAS-data-set;
     VAR variable(s);
     WHERE where-expression;
RUN;
```

```
PROC FREQ DATA=SAS-data-set <NLEVELS>;
    TABLES variable(s) </NOPRINT>;
RUN;
```

- You can use a PROC PRINT step with a VAR statement and a WHERE statement to create a report of observations with invalid values. For example, you can create a report that includes observations that have missing values for a particular variable.
- You use a SAS date constant to convert a calendar date to a SAS date value.
- You cannot use PROC PRINT to detect values that are not unique. Instead, you can use PROC FREQ to view a frequency table of the unique values for a variable. You can use the TABLES statement in a PROC FREQ step to specify which frequency tables to produce. You can use the NLEVELS option to display a table that provides the number of distinct values for each variable named in the TABLES statement.
- You can use the NOPRINT option in the TABLES statement to suppress the individual frequency tables.

Validating Data with PROC MEANS and PROC UNIVARIATE

```
PROC MEANS DATA=SAS-data-set <statistics>;
    VAR variable(s);
RUN;
```

```
PROC UNIVARIATE DATA=SAS-data-set NEXTROBS=n;
    VAR variable(s);
RUN;
```

- PROC MEANS and PROC UNIVARIATE are useful for finding data outliers, or data values that fall outside expected ranges.
- You can use PROC MEANS to create summary reports of descriptive statistics such as SUM, MEAN, and RANGE. By default, PROC MEANS prints these summary statistics for all numeric variables: N, MEAN, STD, MIN, and MAX. You can also specify other statistics in a PROC MEAN step, such as NMISS, which is the number of observations with missing values.
- PROC UNIVARIATE also produces summary reports of descriptive statistics. In addition, PROC UNIVARIATE produces tables of extreme observations and missing values.
- The NEXTROBS= option in the PROC UNIVARIATE statement specifies the number of extreme observations that PROC UNIVARIATE lists.

Cleaning Invalid Data Interactively

- Before you can clean your data, you need to obtain the correct values. You can clean data interactively using the VIEWTABLE window. If you're working in the z/OS operating environment, you'll use the FSEDIT window instead.
- In the VIEWTABLE window, you can browse, edit, or create SAS data sets.

Cleaning Data Programmatically Using the DATA Step

variable=expression

UPCASE*(argument)*

IF *expression* **THEN** *statement*;

- If you need to make systematic changes to your data values, it's easier to do so programmatically in the DATA step than in the VIEWTABLE window.
- You can use an assignment statement to convert values of a variable. The assignment statement is one of the few SAS statements that don't begin with a keyword. An assignment statement evaluates an expression and assigns the resulting value to a new or existing variable.
- An expression is a set of instructions that produces a value. An expression is a sequence of operands and operators. Operands are constants or variables. Operators are either symbols that represent an arithmetic calculation, or they're SAS functions.
- The UPCASE function converts all letters in an argument to uppercase.
- By default, an assignment statement in a DATA step applies to every observation in the data set. To apply different assignment statements to different observations, you can use conditional statements. The IF-THEN statement executes a SAS statement for observations that meet specific conditions.

Cleaning Data Programmatically Using PROC SORT

- You can use PROC SORT to remove duplicate observations from your data by specifying the NODUPRECS option in the PROC SORT statement. NODUPRECS checks only consecutive observations, so it's a good idea to sort by all variables when you use NODUPRECS.
- You can use the NODUPKEY option in the PROC SORT statement to remove observations with duplicate BY values.

- You can use the EQUALS option along with the NODUPRECS and NODUPKEY options in the PROC SORT statement to maintain the relative order of the observations within the input data set and the output data set.

Sample Code

Using PROC PRINT to Validate Data

```
proc print data=orion.nonsales;
   var Employee_ID Gender Salary Job_Title Country
       Birth_Date Hire_Date;
   where Employee_ID = . or
         Gender not in ('F','M') or
         Salary not between 24000 and 500000 or
         Job_Title = ' ' or
         Country not in ('AU','US') or
         Birth_Date > Hire_Date or
         Hire_Date < '01JAN1974'd;
run;
```

Using PROC FREQ to Validate Data

```
proc freq data=orion.nonsales nlevels;
   table _all_ / noprint;
run;
```

Using PROC MEANS to Validate Data

```
proc means data=orion.nonsales n nmiss min max;
   var Salary;
run;
```

Using PROC UNIVARIATE to Validate Data

```
proc univariate data=orion.nonsales nextrobs=8;
   var Salary;
run;
```

Cleaning Data Programmatically Using Assignment Statements

```
data work.clean;
   set orion.nonsales;
   Country=upcase(Country);
run;
```

Cleaning Data Programmatically Using IF-THEN/ELSE Statements

```
data work.clean;
   set orion.nonsales;
   Country=upcase(Country);
   if Employee_ID=120106 then Salary=26960;
   else if Employee_ID=120115 then Salary=26500;
   else if Employee_ID=120191 then Salary=24015;
   else if Employee_ID=120107 then
           Hire_Date='21JAN1995'd;
   else if Employee_ID=12011 then
           Hire_Date='01NOV1978'd;
   else if Employee_ID=121011 then
           Hire_Date='01JAN1998'd;
run;
```

Removing Duplicates Using PROC SORT

```
proc sort data=orion.nonsalesdupes out=sorted nodupkey
          equals;
   by Employee_ID;
run;
```