Lesson 3: Working with SAS Code

Summary

Main Points

Understanding the Structure of SAS Programs

- SAS compiles and executes each DATA or PROC step independently based on step boundaries, such as a RUN statement, a QUIT statement, or the beginning of a new DATA or PROC step.
- The Enhanced Editor illustrates the separation between steps with a horizontal line.

Understanding the Syntax of SAS Programs

- SAS has specific syntax; however, SAS statements are free-format. Each statement must end with a semicolon.
- Although SAS does not require it, you can follow some simple formatting guidelines to make your SAS code easier to read and use.

Adding Comments to SAS Programs

- You can use comments in your code to make it easier for you and others to read and interpret it. A comment is text in your program that SAS ignores during processing. You might use comments to document the purpose of the program, explain segments of the program, or mark SAS code as non-executing text. Comments can be especially useful for testing your code.
- You can use a block comment, which can appear anywhere a single blank can appear and can contain semicolons or unmatched quotation marks. Block comments can also span several lines. You should avoid placing block comments in the first or second columns.
- You can create a comment by beginning any SAS statement with an asterisk, which indicates to SAS that all of the following text up to the next semicolon is a comment. Comments in this form are complete statements, and they can't contain semicolons or unmatched quotation marks.

Diagnosing and Correcting Syntax Errors

- A syntax error is code that doesn't conform to the rules of the SAS language. Common syntax errors include misspelled keywords, unmatched quotation marks, missing semicolons, and invalid operators. The Enhanced Editor highlights potential errors in your SAS code by displaying it in red text.
- SAS code can also contain logic errors.

- When you submit a program, SAS attempts to interpret any syntax errors that it encounters. If it can't interpret the error, SAS stops processing the step and writes an error message in the log. You must correct your code to fix errors that appear in the log.

Diagnosing and Correcting Unbalanced Quotation Marks

- Unbalanced quotation marks are a syntax error that might not generate a log message. Instead, the main symptom might be a message such as "PROC PRINT running" in the title bar of an active programming window.
- To correct unbalanced quotation marks, you must cancel the submitted statement, correct the error, and resubmit the code. You can use the **Break** tool to cancel submitted statements.

Sample Code

Unformatted Code

```
data work.newsalesemps;length First_Name $ 12
          Last_Name $ 18 Job_Title $ 25; infile
'newemployees.csv' dlm=',';
input First_Name $ Last_Name $ Job_Title $
Salary; run;
title 'New Sales Employees';
proc print data=work.newsalesemps; run;
proc means data=work.newsalesemps; class Job_Title;var
Salary; run;
title;
```

Formatted Code

```
data work.newsalesemps;
   length First_Name $ 12
          Last_Name $ 18 Job_Title $ 25;
   infile 'newemployees.csv' dlm=',';
   input First_Name $ Last_Name $
         Job_Title $ Salary;
run;

title 'New Sales Employees';
proc print data=work.newsalesemps;
run;

proc means data=work.newsalesemps;
   class Job_Title;
   var Salary;
run;

title;
```

Comments in Code

```
  /*
This is a comment.
This title statement won't be processed.
title 'New Sales Employees';
  */

*******************************
This is a comment;
proc print data=work.newsalesemps;
run;
*This is another comment;
```

Lesson 5: Creating SAS Data Sets

Summary

Main Points

Creating A SAS Data Set from a SAS Data Set

```
DATA output-SAS-data-set-name;
    SET input-SAS-data-set-name;
RUN;
```

- To create a SAS data set, you start with input data, which could be a SAS data set, a Microsoft Excel worksheet, or a raw data file. This lesson focuses on creating a SAS data set from a SAS data set. You can use a DATA step to read input data and create a new SAS data set.
- A basic DATA step that contains only a SET statement creates an exact copy of the input SAS data set. This type of DATA step is a good building block, because a DATA step can include many other kinds of statements.
- A DATA step can reference temporary or permanent libraries. Remember that you use a LIBNAME statement to assign a libref to a SAS library.
- A SAS data set name must follow the rules for SAS names.

Understanding DATA Step Processing

- SAS processes the DATA step in two phases: the compilation phase and the execution phase.
- During the compilation phase, SAS scans the step for syntax errors. Then, SAS creates the program data vector (PDV) and the descriptor portion of the output data set. The PDV contains two temporary variables: **_N_**, which tracks the iterations of the DATA step, and **_ERROR_,** which signals the occurrence of an error that is caused by the data during execution.
- During the execution phase, SAS reads and processes the input data set, and creates the data portion of the output data set.

Subsetting Observations in the DATA Step

```
WHERE where-expression;
```

- By default, the SET statement reads all observations from the input data set, but you can use the WHERE statement to control which observations SAS writes to the output data set.

- A where-expression consists of operands and operators. There are several categories of operators including comparison, arithmetic, and logical operators. There are also several special WHERE operators that can only be used in WHERE statements.

Subsetting Variables in the DATA Step

DROP *variable-list*;

KEEP *variable-list*;

- By default, the SET statement reads all of the variables from the input data set and includes them in the output data set. You can control which variables SAS writes to the output data set by using a DROP statement or a KEEP statement in the DATA step.
- A DROP statement names the variables to exclude from the output data set. A KEEP statement names the variables to include in the output data set.

Assigning Permanent Labels and Formats

LABEL *variable1='label1' variable2='label2'*;

PROC PRINT DATA=*SAS-data-set* LABEL;
RUN;

FORMAT *variable(s) format*;

- You can assign permanent labels and format to the variables in your output data set to control the way that variables names and values appear in reports. Labels and formats don't affect the stored names or values; they are stored in the descriptor portion of the data set.
- You use a LABEL statement to assign a label to a variable. To specify that SAS should display variable labels rather than names in a PROC PRINT report, you use the LABEL option in the PROC PRINT statement.
- You use a FORMAT statement to assign a format to a variable.

Sample Code

```
data work.subset1;
   set orion.sales;
   where Country='AU' and Job_Title contains 'Rep';
   keep First_Name Last_Name Salary Job_Title Hire_Date;
   label Job_Title='Sales Title' Hire_Date='Date Hired';
   format Salary comma8. Hire_Date ddmmyy10.;
run;

proc print data=work.subset1 label;
run;
```

Summary

Main Points

Understanding SAS Libraries

- A SAS library is a collection of one or more SAS files that are recognized by SAS and that are referenced and stored as a unit. You reference a SAS library by a logical name called a libref.
- At the beginning of each SAS session, SAS automatically creates at least two libraries that you can access: **Work**, which is the temporary library, and **Sasuser**, which is a permanent library.
- All SAS data sets have a two-level name that consists of the libref and the data set name. When a data set is stored in the **Work** library, you do not need to include the libref when you reference it.

Defining SAS Libraries

LIBNAME *libref 'SAS-library'***;**

- You use a LIBNAME statement to define a SAS library and assign a libref to it. The LIBNAME statement is a global statement.
- In an interactive SAS session, a libref that you assign with a LIBNAME statement remains assigned until you cancel or change the libref or until you end your SAS session. After you end your SAS session, the contents of a permanent library still exist in their physical location. However, each time you start a new SAS session, you must resubmit a LIBNAME statement to reassign the libref to that permanent library.
- When you assign a libref, you must follow the rules for valid librefs. Also, the LIBNAME statement does not create a new directory. Instead, the LIBNAME statement points to an existing directory.

Viewing the Contents of SAS Libraries

PROC CONTENTS DATA=*libref***.**_ALL_ NODS**;**
RUN;

- You can use a PROC CONTENTS step to view the contents of a SAS library. When you specify the keyword _ALL_ in the PROC CONTENTS statement, the step displays a list of all the SAS files that are in the specified SAS library.

- By default, a PROC CONTENTS report includes the descriptor portion of each data set in the SAS library. You can use the NODS option to suppress the descriptor portions in the report.

Understanding the Structure of SAS Data Sets

- A SAS data set is a table with columns and rows. In SAS, the table is called a data set, a column is called a variable, and a row is called an observation. In each observation, each variable has a specific value.
- The data portion of a SAS data set contains the data values. The descriptor portion of a SAS data set contains information about the attributes of the data set and information about each variable. The information in the descriptor portion is called metadata.
- You can see the data portion of a data set in the VIEWTABLE window. You can view some of the information from descriptor portion in the Properties window for the data set.

Viewing the Descriptor Portion of SAS Data Sets

```
PROC CONTENTS DATA=SAS-data-set;
RUN;
```

- You can use a PROC CONTENTS step to view the descriptor portion of a data set. When you specify a single data set in the PROC CONTENTS step and do not include the NODS option, the step creates a report of the descriptor portion for the specified data set.
- A variable in a SAS data set has a type attribute that can be either character or numeric.
- Character variables are stored with a length of 1 to 32,767 bytes. All numeric variables have a default length of 8 bytes. Numeric values, no matter how many digits they contain, are stored as floating-point numbers in 8 bytes of storage, unless you specify a different length.
- SAS variables have additional attributes such as format, informat, and label, which you learn about in later lessons.

Viewing the Data Portion of SAS Data Sets

```
PROC PRINT DATA=SAS-data-set;
RUN;
```

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

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

- You can use the PRINT procedure to display the data portion of a SAS data set.
- Missing values are represented by blanks for character variables and by periods for numeric variables.
- By default, the PROC PRINT step includes all variables from the data set in the report. You can control which variables SAS includes in the report by using a VAR statement in the PROC PRINT step. The VAR statement also controls the order in which SAS displays the variables in the report.
- BY default, SAS includes the observation numbers in the report. You can suppress the **Obs** column by using the NOOBS option in the PROC PRINT statement.

Sorting Observations in SAS Data Sets

```
PROC SORT DATA=input-SAS-data-set
                <OUT=output-SAS-data-set>;
      BY <DESCENDING> BY-variable(s);
RUN;
```

- You can use the SORT procedure to sort the observations in a data set.
- By default, the SORT procedure replaces the original data set with the sorted data set. You can use the OUT= option to create a new output data set instead of replacing the input data set.
- PROC SORT does not generate printed output.
- By default, the SORT procedure sorts observations in ascending order of the values for the variable or variables listed in the BY statement. You can use the DESCENDING option to sort on the variable or variables in descending order instead.
- SAS treats missing values as the smallest possible values.

Defining SAS Libraries for Relational Databases

LIBNAME *libref engine-name <SAS/ACCESS-options>*;

LIBNAME *libref* CLEAR;

- You can use the SAS/ACCESS LIBNAME statement to define a library so that your SAS programs can access tables that are stored in another vendor's relational database.
- SAS/ACCESS uses SAS/ACCESS engines to read from or write to files. An engine is named for a specific database management file, such as Oracle or DB2. Your SAS installation must include the appropriate SAS/ACCESS interfaces for the types of files that you want to access.
- After a database is associated with a libref, you can use a SAS two-level name to specify any table in the database and then work with that table as you would work with a SAS data set.
- You use the CLEAR option in the LIBNAME statement to disassociate a libref that was previously assigned. Disassociating a libref disconnects the database engine from the database and closes any resources that are associated with that libref's connection.

Sample Code

Windows: Replace *my-file-path* with the location where you stored the practice files.
UNIX and **z/OS**: Specify the fully qualified path in your operating environment.

Assigning a Libref and Viewing the Contents of the Library

```
libname orion 'my-file-path';

proc contents data=orion._all_nods;
run;
```

Creating Work.NewSalesemps

```
data work.newsalesemps;
   length First_Name $ 12 Last_Name $ 18 Job_Title $ 25;
   infile 'my-file-path\newemployees.csv' dlm=',';
   input First_Name $ Last_Name $ Job_Title $ Salary;
run;
```

Displaying the Descriptor Portion of a Data Set

```
proc contents data=work.newsalesemps;
run;
```

Displaying the Data Portion of a Data Set

```
proc print data=work.newsalesemps noobs;
   var Last_Name First_Name Salary;
run;
```

Sorting a Data Set by Two Variables and Displaying the Output Data Set

```
proc sort data=work.newsalesemps
          out=work.newsalesemps3;
   by Job_Title descending Salary;
run;

proc print data=work.newsalesemps3;
run;
```

Lesson 6: Creating SAS Data Sets from Microsoft Excel Worksheets

Summary

Main Points

Accessing and Viewing Excel Data in SAS

```
LIBNAME libref 'physical-file-name';
```

- You can use the SAS/ACCESS LIBNAME statement to assign a libref to a Microsoft Excel workbook. Then, SAS treats each worksheet in the workbook as though it is a SAS data set. You can use the Explorer window or the CONTENTS procedure to view the worksheets, or you can reference a worksheet directly in a DATA or PROC step.
- You must have a license for SAS/ACCESS for PC Files to use the SAS/ACCESS LIBNAME statement.
- In SAS, Excel worksheet names contain a dollar sign. To reference an Excel worksheet directly in a DATA or PROC step, you use a SAS name literal because a valid SAS name cannot contain a dollar sign.

Using an Excel Worksheet as Input in the DATA Step

- You can use an Excel worksheet as input data in a DATA step to create a new SAS data set. You use a SET statement with a SAS name literal to read from an Excel worksheet. You can also use other DATA step statements such as WHERE, KEEP, DROP, LABEL, and FORMAT statements with input from an Excel worksheet.

Importing an Excel Worksheet

```
PROC IMPORT OUT= output-data-set
            DATAFILE='input=excel-workbook'
            DBMS=EXCEL REPLACE;
     RANGE='range-name';
RUN;
```

- You can use a PROC IMPORT step to read an Excel worksheet and create a SAS data set from it.
- The Import Wizard is a point-and-click interface that generates PROC IMPORT code for you.

Creating an Excel Workbook in SAS

```
LIBNAME libref 'physical-file-name';

DATA output-excel-worksheet;
     SET input-data-set;
RUN;
```

```
LIBNAME output-libref 'physical-file-name';

PROC COPY IN=input-libref OUT=output-libref;
      SELECT input-data-set1 input-data-set2;
RUN;
```

```
PROC EXPORT DATA= input-data-set
                  OUTFILE='output-excel-workbook'
                  DBMS=EXCEL REPLACE;
RUN;
```

- You can use the DATA step to create an Excel worksheet from a SAS data set if you use the SAS/ACCESS LIBNAME statement to assign a libref to the Excel workbook that contains the worksheet. You use one DATA step for each worksheet that you want to create.
- You can use the SAS/ACCESS LIBNAME statement along with the COPY procedure to create an Excel workbook that contains one or more worksheets from one or more SAS data sets. The COPY procedure creates a worksheet for each SAS data set that is listed in the SELECT statement.
- You can use the EXPORT procedure to create an Excel workbook from a SAS data set.
- The Export Wizard is a point-and-click interface that generates PROC EXPORT code for you.

Sample Code

Windows: Replace *my-file-path* with the location where you stored the practice files.
UNIX and **z/OS**: Specify the fully qualified path in your operating environment.

Using PROC IMPORT to Create a SAS Data Set from an Excel Worksheet

```
proc import out=work.subset2a;
            datafile='my-file-path\sales.xls';
   range="Australia$"n;
   getnames=yes;
   mixed=no;
   scantext=yes;
   usedate=yes;
   scantime=yes;
run;
```

Using the DATA Step to Create an Excel Workbook from SAS Data Sets

```
libname orionxls 'my-file-path\qtr20007a.xls';

data orionxls.qtr1_2007;
   set orion.qtr2_2007;
run;

data orionxls.qwtr2_2007;
   set orion.qtr2_2007;
run;
```

Using PROC COPY to Create an Excel Workbook from SAS Data Sets

```
libname orionxls 'my-file-path\qtr20007b.xls';

proc copy in=orion out=orionxls;
   select qtr1_2007 qtr2_2007;
run;
```

Lesson 7: Creating SAS Data Sets from Delimited Raw Data Files

Summary

Main Points

Creating a SAS Data Set from a Raw Data File

```
DATA output-SAS-data-set-name;
     INFILE 'raw-data-file-name' <DLM='delimiter'>;
     INPUT specifications;
RUN;
```

- A raw data file is an external file whose records contain data values that are organized in fields. Raw data files are not software-specific. A delimited raw data file is a file in which the data values are separated by spaces or other special characters.
- You can use a DATA step to read input from a raw data file and create a SAS data set from it. Instead of a SET statement, you use an INFILE statement and an INPUT statement. The INFILE statement specifies the name and location of the raw data file, and the INPUT statement provides specifications for SAS to read the data values as variables.
- You use the DLM= option in the INFILE statement to specify a delimiter if the raw data file uses a character other than a blank space to delimit data values.
- Delimited data is also known as list input. Other types of raw data files might contain column input or formatted input. Raw data files can contain either standard or nonstandard data values.
- A raw data file does not contain names for the data fields. You specify variable names for the values in the INPUT statement, and the names that you use must follow the rules for SAS names.

Understanding DATA Step Processing

- Remember, SAS processes the DATA step in two phases: the compilation phase and the execution phase.
- During the compilation phase, SAS scans each statement for syntax errors. Then, SAS creates the PDV and the descriptor portion of the output data set.
- During the execution phase, SAS reads and processes the input data and fills in the data portion of the output data set.

Specifying Lengths of Variables

LENGTH *variable(s)* **$** *length*;

- By default, SAS creates each variable with a length of 8 bytes. So, SAS truncates character values that are longer than 8 characters. You can use the LENGTH statement in the DATA step to specify a variable length that is longer or shorter than 8 bytes.
- Character variables can have a length of 1 to 32,767 characters. In this lesson, you don't alter the lengths of any numeric variables because you can stored a numeric value in a length of 8 bytes no matter how many digits it contains.
- You can specify multiple variables in one LENGTH statement. You can either specify different lengths for each variable that you list in the LENGTH statement or you can specify one length for all of the variables listed.
- The LENGTH statement must precede the INPUT statement in a DATA step, because SAS determines variable attributes such as length the first time it encounters a variable. You should also list the variable names in the LENGTH statement exactly as you want SAS to store them in the data set.

Working with Nonstandard Data

INPUT *variable* <**$**> *variable* <**:***informat*>**;**

- Nonstandard data is data that SAS cannot read without extra instructions. To read nonstandard data, you use the colon format modifier and an informat in the INPUT statement.
- Informats are similar to formats. An informat is the instruction that SAS uses to read data values into a variable. An informat also provides a length for a character variable.

Further DATA Step Processing

- You can use a DROP statement or a KEEP statement to subset variables in a DATA step that reads raw data.
- You can use FORMAT and LABEL statements to add permanent attributes to your output data set in a DATA step that reads raw data.
- You **cannot** use a WHERE statement in a DATA step that reads raw data. You can use a subsetting IF statement instead, which you learn about in a later lesson.

Reading Instream Data

```
DATA output-SAS-data-set;
      INPUT specifications;
      DATALINES;
instream data
;
RUN;
```

- You use a DATALINES statement to read instream data. Instream data is data that you specify within your program. You use a NULL statement to indicate the end of the input data.
- You use the INPUT statement for list input to specify that the instream data is delimited with blanks. You could also use a LENGTH statement before a DATALINES statement to specify lengths for variables within the instream data that are shorter or longer than 8 bytes.

Sample Code

Windows: Replace *my-file-path* with the location where you stored the practice files.
UNIX and **z/OS**: Specify the fully qualified path in your operating environment.

Creating a SAS Data Set from a Raw Data File

```
data work.subset3;
   infile 'my-file-path\sales.csv' dlm=',';
   length First_Name $ 12 Last_Name $ 18 Gender $ 1
          Job_Title $ 25 Country $ 2;
   input Employee_ID First_Name $ Last_Name $ Gender $
         Salary Job_Title $ Country $ Birth_Date :date9.
         Hire_Date :mmddyy10.;
   keep First_Name Last_Name Salary Job_Title Hire_Date;
   label Job_Title='Sales Title' Hire_Date='Date Hired';
   format Salary dollar12. Hire_Date monyy7.;
run;
```

Reading Instream Data

```
data work.subset4;
   input Employee_ID First_Name $ Last_Name $;
   datalines;
120102 Tom Zhou
120103 Wilson Dawes
120121 Irenie Elvish
;
run;
```

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;
```

Lesson 9: Manipulating Data

Summary

Main Points

Creating Variables

variable=expression;

SUM(*argument1,argument2,…***)**

MONTH(*SAS-date***)**

- You can use an assignment statement to create a new variable in a DATA step.
- The SUM function returns the sum of the specified arguments. If an argument contains a missing value, the SUM function ignores the missing value and returns the sum of the non-missing values.
- If the expression in an assignment statement contains a missing value, the result of the expression is also a missing value.
- You can use SAS date functions in assignment statements to specify SAS date values as part of your expressions or to create SAS date values.
- The MONTH function extracts a month from a SAS date value.

Subsetting Variables

- Remember that you can use a DROP statement or KEEP statement in a DATA step to subset variables in the output data set.
- When SAS processes a DROP statement, the variables listed in the DROP statement are included in the PDV. SAS reads values for those variables into the PDV for each observation, but does not write those variables to the output data set.

Creating Variables Conditionally

```
IF expression THEN
    DO;
      executable statements
    END;
ELSE IF expression THEN
    DO;
      executable statements
    END;
```

- You can use a DO statement within an IF-THEN/ELSE statement to execute multiple statements, which enables you to create variables conditionally. The DO statement, together with the other SAS statements it contains, is known as a DO group. Each DO group must end with an END statement.
- SAS executes the statements withing the first DO group only if the expression in the IF-THEN statement is true. SAS executes the statements within the second DO group only if the expression in the first IF expression is false and the expression in the ELSE IF expression is true.
- Remember that SAS creates the attributes for a variable, including length, based on the first time it encounters that variable. You use a LENGTH statement to specify the length for a variable explicitly.

Subsetting Observations

```
IF expression;
```

```
IF expression THEN DELETE;
```

- The WHERE statement selects observations when they are read from the input data set, so you can't use the WHERE statement to subset observations based on a variable that you create in the same DATA step.
- You can use the subsetting IF statement to subset observations based on the value of a variable you create. You can specify multiple expressions in a subsetting IF statement.
- You cannot use special WHERE operators in IF expressions.

- You must use the WHERE statement rather than a subsetting IF statement in a PROC step You can always use a subsetting IF statement in a DATA step. You can use a WHERE statement in a DATA step if the expression in the WHERE statement references only variables from the input data set, or from all input data sets if the step uses multiple input data sets.
- You can use a DELETE statement within an IF-THEN statement as an alternative to the subsetting IF statement. The DELETE statement is especially useful if it is easier to specify a condition for excluding observations than for selecting observations.

Sample Code

Creating Variables

```
data work.comp;
   set orion.sales;
   Bonus=500;
   Compensation=sum(Salary,Bonus);
   BonusMonth=month(Hire_Date);
run;
```

Subsetting Variables

```
data work.comp;
   set orion.sales;
   drop Gender Salary Job_Title Country
        Birth_Date Hire_Date;
   Bonus=500;
   Compensation=sum(Salary,Bonus);
   BonusMonth=month(Hire_Date);
run;
```

Creating Variables Conditionally

```
data work.bonus;
   set orion.sales;
   length Freq $ 12;
   if Country='US' then
      do;
         Bonus=500;
         Freq='Once a Year';
      end;
   else if Country='AU' then
      do;
         Bonus=500;
         Freq='Twice a Year';
      end;
run;
```

Subsetting Observations

```
data work.december;
   set orion.sales;
   BonusMonth=month(Hire_Date);
   if Country='AU' and BonusMonth=12;
   Bonus=500;
   Compensation=sum(Salary,Bonus);
run;
```