Print

# Summary of Lesson 3: Exploring and Validating Data

## Exploring Data

- PROC PRINT lists all columns and rows in the input table by default. The OBS= data set option limits the number of rows listed. The VAR statement limits and orders the columns listed.

```
PROC PRINT DATA=input-table(OBS=n);
    VAR col-name(s);
RUN;
```

- PROC MEANS generates simple summary statistics for each numeric column in the input data by default. The VAR statement limits the variables to analyze.

```
PROC MEANS DATA=input-table;
    VAR col-name(s);
RUN;
```

- PROC UNIVARIATE also generates summary statistics for each numeric column in the data by default, but includes more detailed statistics related to distribution and extreme values. The VAR statement limits the variables to analyze.

```
PROC UNIVARIATE DATA=input-table;
    VAR col-name(s);
RUN;
```

- PROC FREQ creates a frequency table for each variable in the input table by default. You can limit the variables analyzed by using the TABLES statement.

```
PROC FREQ DATA=input-table;
    TABLES col-name(s) < / options>;
RUN;
```

## Filtering Rows

- The WHERE statement is used to filter rows. If the expression is true, rows are read. If the expression is false, they are not.
- Character values are case sensitive and must be in quotation marks.
- Numeric values are not in quotation marks and must only include digits, decimal points, and negative signs.
- Compound conditions can be created with AND or OR.

- The logic of an operator can be reversed with the NOT keyword.
- When an expression includes a fixed date value, use the SAS date constant syntax: “ddmmmyyyy”d, where *dd* represents a 1- or 2-digit day, *mmm* represents a 3-letter month in any case, and *yyyy* represents a 2- or 4-digit year.

```
PROC procedure-name ... ;
     WHERE expression;
RUN;
```

### WHERE Operators

= or EQ
^= or ~= or NE
> or GT
< or LT
>= or GE
<= or LE

### SAS Date Constant

"ddMONyyyy"d

### IN Operator

**WHERE** *col-name* **IN(***value-1<...,value-n>***);**
**WHERE** *col-name* **NOT IN (***value-1<…,value-n>***);**

### Special WHERE Operators

**WHERE** *col-name* **IS MISSING;**
**WHERE** *col-name* **IS NOT MISSING;**
**WHERE** *col-name* **IS NULL;**
**WHERE** *col-name* **BETWEEN** *value-1* **AND** *value-2***;**
**WHERE** *col-name* **LIKE "***value***";**
**WHERE** *col-name* **=* "***value***";**

### Filtering Rows with Macro Variables

**%LET** *macro-variable=value***;**

### Example WHERE Statements with Macro Variables:

**WHERE** *numvar=&macrovar***;**
**WHERE** *charvar="&macrovar"***;**
**WHERE** *datevar="&macrovar"*d

- A macro variable stores a text string that can be substituted into a SAS program.
- The %LET statement defines the macro variable name and assigns a value.

- Macro variable names must follow SAS naming rules.
- Macro variables can be referenced in a program by preceding the macro variable name with an &.
- If a macro variable reference is used inside quotation marks, double quotation marks must be used.

## Formatting Columns

- Formats are used to change the way values are displayed in data and reports.
- Formats do not change the underlying data values.
- Formats can be applied in a procedure using the FORMAT statement.
- Visit SAS Language Elements documentation to access a list of available SAS formats.

```
PROC PRINT DATA=input-table;
     FORMAT col-name(s) format;
RUN;
```

```
<$>format-name<w>.<d>
```

## Sorting Data and Removing Duplicates

- PROC SORT sorts the rows in a table on one or more character or numeric columns.
- The OUT= option specifies an output table. Without this option, PROC SORT changes the order of rows in the input table.
- The BY statement specifies one or more columns in the input table whose values are used to sort the rows. By default, SAS sorts in ascending order.

```
PROC SORT DATA=input-table <OUT=output-table>;
     BY <DESCENDING> col-name(s);
RUN;
```

- The NODUPKEY option keeps only the first row for each unique value of the column(s) listed in the BY statement.
- The NODUPKEY option together with the BY _ALL_ statement removes adjacent rows that are entirely duplicated.
- The DUPOUT= option creates an output table containing duplicates removed.

```
PROC SORT DATA=input-table <OUT=output-table>
                NODUPKEY <DUPOUT=output-table>;
     BY _ALL_;
RUN;
```

```
PROC SORT DATA=input-table <OUT=output-table>
```

```
            NODUPKEY <DUPOUT=output-table>;
    BY col-name(s);
RUN;
```