Chapter 3: The DATA Step – Part I

- Creating new SAS data sets with the SET statement – how the DATA step works
- Creating and transforming variables – assignment statements, numeric and character operators and functions
- Subsetting observations – IF and WHERE
- Subsetting variables – KEEP and DROP
- Variable lengths
- SAS dates
- Important messages in the SAS log

Creating New SAS Data Sets

It will often be desirable to modify an existing SAS data set in some way, for example, selecting only a subset of the original observations, transforming variables, creating new variables, etc. These kinds of modifications are accomplished within a **DATA step.**

- A DATA step:
    - Reads one or more input files (SAS data sets and/or non-SAS files)
    - Performs processing (transformations, selections, etc.), if specified
    - Creates one or more output files (SAS data sets or non-SAS files)
- In this Chapter, we will only discuss reading a single input SAS data set and creating a single output SAS data set.
- All of the modification statements we will discuss can be used with any combination of input and output sources.

Structure of a DATA Step

A DATA step that creates a single output SAS data set by modifying a single input SAS data set has a five-part structure:

1. A DATA statement to start the step and name the output data set.

2. A SET statement to read an observation from the input data set (also tells SAS which input data set to use).

3. Programming statements to perform the processing required for the input observation.

4. An OUTPUT statement to write the observation to the output data set.

5. A RETURN statement to end processing of this observation and return to the top of the step (to process the next observation).

Figure 1: A Basic DATA Step

```
data work.myclass;
 set sashelp.class;

   /*** programming statements ***/

   output;
   return;
run;
```

Processing of a DATA Step

The processing of every DATA step involves two distinct phases.

1. **Compilation Phase**: SAS compiles the statements within the step, creating a program to perform the requested processing.
2. **Execution Phase**: The program created is executed, processing the data and creating the new data set.

The Compilation Phase

During the compilation phase, the DATA step compiler:

- Reads the descriptor portion of the existing SAS data set.
- Creates the descriptor part of the output data set.
- Creates the program data vector (PDV) which will contain all of the variables found in the existing SAS data set plus any new variables created with assignment statements.
- Creates a machine language program to perform the processing.
- Detects syntax errors.

The Execution Phase (DATA step as a loop)

During the execution phase:

- The SET statement is executed once for each observation in the input SAS data set.
- Each time the SET statement is executed, an observation is read from the input data set.
- Any program statements in the DATA step are executed once for each observation in the input data set.
- The values in the PDV are written to the new SAS data set after the last executable statement in the DATA step or any time an OUTPUT statement is executed.

Steps of Execution

1. Initialize PDV to missing - DATA work.myclass;

2. End of input?

    a. If yes- then exit

    b. If no- then continue to step 3

3. Read next observation into the PDV - SET sashelp.class;

4. Modify data values in the PDV

5. Write values from the PDV to the output data set - OUTPUT;

6. Return to the top of the DATA step in step 1 - RETURN;

To learn more:
(1) Do e-learning courses
(2) Google: “How the DATA Step Works: A Basic Introduction” and read for homework.

The DATA Statement

The DATA statement has two functions:

- It defines the start of a DATA step.
- It names the SAS data sets to be created.

Syntax: `DATA LIBREF.FILENAME;`

Where:

FILENAME	is the name of the SAS data set to be created
LIBREF	is the libref for a SAS data library in which the data set will be stored

The SET Statement

- The SET statement tells SAS the input SAS data set you want to use and reads an observation from this input SAS data set each time it is executed.
- All variables in the input SAS data set are automatically passed to the new SAS data set (unless otherwise directed with programming statements).
- All observations in the input SAS data set are automatically passed to the new SAS data set (unless otherwise directed with programming statements).
- New variables may be added with assignment statements.
- Note that reading a data set does not modify it in any way.

Syntax: `SET LIBREF.FILENAME;`

Where:

FILENAME	is the name of an existing SAS data set to be read
LIBREF	is a libref for the SAS data library where the data set is located

- The SET statement supports many useful options and we will discuss some in later chapters.

The OUTPUT Statement

- The OUTPUT statement controls when the values in the program data vector (PDV) are written to the output SAS data set.

- When an OUTPUT statement is executed, SAS **immediately** outputs the current PDV values to a SAS data set.

- Implicit OUTPUT – When an OUTPUT statement does not appear in the DATA step, SAS outputs the values of the PDV at the end of the DATA step.

- Explicit OUTPUT – When an OUTPUT statement does appear in the DATA step, there is no implicit output at the end of the step.

- **The OUTPUT statement is <u>optional</u> and is omitted by convention unless explicit output is required for the program.**

- Execution of the OUTPUT statement does not return control to the beginning of the DATA step.

Syntax: `OUTPUT;` or `OUTPUT LIBREF.FILENAME;`

The RETURN Statement

- When a RETURN statement is encountered during processing of an observation, control returns to the SET statement where the next observation is read (or the DATA step terminates if no more observations need to be read).

- When a RETURN statement does not appear in a DATA step, SAS assumes an implicit RETURN directly after the last programming statement in the DATA step.

- **The RETURN statement is <u>optional</u> and is omitted by convention unless explicit user-controlled return is required for the program. It almost all applications (in this course), the RETURN statement can be omitted.**

Syntax: `RETURN;`

Summary of Creating New SAS Data Sets

- The four statements just described (DATA, SET, OUTPUT, RETURN) are used whenever we want to create a new SAS data set from an existing one.

- Other statements are added to the step in order to make the output data set a modified version of the input data set, rather than an exact copy.

- In this chapter, we only discuss creating SAS data sets from other, already existing SAS data sets.

- Creating a SAS data set from a non-SAS data set (e.g., an ASCII or Excel file) is a more complex task, which will be covered in detail later in the course.

- Creating a new data set does not delete or modify the input data set; it is still available for use in subsequent steps.

Figure 2: Equivalent Data Steps

```
data work.class;
 set sashelp.class;

 output;
 return;
run;
```

```
data work.class;
 set sashelp.class;

run;
```

```
data work.class;
 set sashelp.class;

 output work.class;
run;
```

```
data work.class;
 set sashelp.class;

 return;
run;
```

Creating and Transforming Variables with Programming Statements

- In many cases, the reason for creating a new SAS data set will be to create new variables that are some combination of existing variables, or to transform an existing variable.
- For example, we might want to add a new variable to the CLASS data set called RELWT (for relative weight) whose value for each observation is defined by the algebraic formula:

  relwt = wt/ht;

  That is, RELWT will be the person's weight divided by their height.
- An example of transforming an existing variable would be recoding the values of height from English units (inches) to metric units (centimeters). The formula in this case is:

  ht = 2.54*ht ;

  That is, take each person's current value of height, multiply it by 2.54, and use that result to replace the original value.
- These kinds of operations are performed in a DATA step using <u>assignment statements</u>.

Assignment Statements

An assignment statement is used to create new variables or to transform existing variables.

<u>Syntax:</u> *VARIABLE* = *EXPRESSION*;

Where:

VARIABLE	is the name of a variable in (or to be added to) the data set
EXPRESSION	is an arithmetic expression, as defined below

<u>Notes</u>

- The assignment is one (of two) exceptions to the rule that every SAS statement begins with a keyword.
- If VARIABLE is the name of an already existing variable, the value of "expression" replaces the previous value.
- Otherwise, the assignment statement creates a new variable, which is added to the output data set.

Expressions

- An expression consists of one or more constants, variables, and functions, combined by operators.
- A constant is a number (e.g., 1, -23.6) or a character string (e.g., 'JOHN', 'MALE', 'X#!').
    - Character constants must be enclosed in single or double quotes.
    - SAS also allows other specialized types of constants; we will discuss some of them later in the course.
- A function is a program "built in" to SAS that performs some computation on character or numeric values in a SAS variable.
    - Users can write their own function using PROC FCMP (beyond the scope of this course)
- An operator is a mathematical, logical, or character operation or manipulation that combines, compares, or transforms numeric or character values.

Arithmetic Operators

Symbol	Action
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Prefix Operators

+	Makes a number positive, for example +5
-	Makes a number negative, for example -9999

Comparison operators

Symbol	Mnemonic Equivalent	Action
=	EQ	Equal to
^- or ~=	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to
<=	LE	Less than or equal to
	IN	Equals to one in a list

Logical or Boolean operators

Symbol	Mnemonic Equivalent
&	AND
|	OR
^ or ~	NOT

Other operators

Symbol	Description
><	Minimum
<>	Maximum
||	Concatenation

Note: Modern alternatives to || include the CAT, CATS, and CATX functions described later in the chapter.

Order of Operator Execution

For homework, google "SAS Operators in Expressions 9.4" and read the documentation.

In order of first executed to last executed:

1. Parentheses
2. Exponentiation, Prefix operators, Not, Minimum, Maximum
3. Multiplication, Division
4. Addition, Subtraction
5. Concatenation
6. Comparison Operators
7. AND
8. OR

Examples

The compound expression x ** 2 - 7 / 2 ** 3 * 9 + 4 evaluates as (((x**2) - ((7 / (2**3)) * 9)) + 4)

The compound expression (trt=1) OR (sex="F") AND (age>70) evaluates as ((trt=1) OR ((sex="F") AND (age>70)))

NOTE: Use parentheses so that you control the order of operator execution to avoid unintended behavior.

Expressions Examples

Assigning constants

```
z = 4;                          numeric constant
sex = 'Female';                 character constant
```

Basic arithmetic operators

```
x2 = x;                         copy the value
sum = x + y;                    addition
diff = x - y;                   subtraction
twice = x * 2;                  multiplication
half = x / 2;                   division
cubic = x**3;                   exponentiation
y = -x;                         change sign
```

Comparison operators

```
IF x < y THEN c = 5 ;
IF name = 'PAT' ;
IF 5 <= age <= 20 ;
IF age IN (10,20,30) THEN x = 5 ;
IF sex IN ('M','F') THEN s =1 ;
```

Logical operators

```
IF a < b AND c > 0 ;
IF x = 2 OR x = 4 ;
IF ^(x = 2) ;
IF NOT(name = 'SMITH');
```

Other operators:

```
x = a >< b ;
name = first || last ;
```

Figure 3: Using Assignment Statements

```
data work.newclass;
    set bios511.class;
    age = age * 12;
    bmi = ((wt/2.2)/((ht*2.54)**2)) * 10000;
run;

proc print data=work.newclass;
  title1 'Creating new variables with assignment statements';
run;
```

Creating new variables with assignment statements

Obs	NAME	SEX	AGE	HT	WT	bmi
1	CHRISTIANSEN	M	444	71	195	27.2538
2	HOSKING J	M	372	70	160	23.0056
3	HELMS R	M	492	74	195	25.0889
4	PIGGY M	F	.	48	.	.
5	FROG K	M	36	12	1	4.8927
6	GONZO		168	25	45	50.7274

Figure 4: Using Assignment Statements and Explicit OUTPUT Statements

```
data work.newclass;
    set bios511.class;

    unit = 'inches';
    output;

    unit = 'feet';
    ht = ht/12;
    output;

    ht = .; ** for illustration only;
run;

proc print data=work.newclass label;
  title1 'Using OUTPUT to Construct Multiple Observations';
  label ht = 'Height' unit = 'Unit';
  var Name Sex ht unit;
run;
```

Using OUTPUT to Construct Multiple Observations

Obs	NAME	SEX	Height	Unit
1	CHRISTIANSEN	M	71.0000	inches
2	CHRISTIANSEN	M	5.9167	feet
3	HOSKING J	M	70.0000	inches
4	HOSKING J	M	5.8333	feet
5	HELMS R	M	74.0000	inches
6	HELMS R	M	6.1667	feet
7	PIGGY M	F	48.0000	inches
8	PIGGY M	F	4.0000	feet
9	FROG K	M	12.0000	inches
10	FROG K	M	1.0000	feet
11	GONZO		25.0000	inches
12	GONZO		2.0833	feet

Functions

- General form of a SAS function:

  VARIABLE = *FUNCTION*(*ARGUMENT1, ARGUMENT2*, *ETC.*);

- Each function has its own particular set of arguments.
- Each argument is separated from the others by a comma.
- Most functions accept arguments that are constants, variables, expressions, or functions.
- Typically, you will need to look up the arguments required by a function before attempting to use the function in a program.
- Examples:

s = SQRT(x);
a = ABS(x);
b = MAX(2,7);
c = SUBSTR('INSIDE',3,4);
d = MIN(x,7,a+b);
e = SUM(OF x1-x5);

- Types of functions:

Arithmetic (absolute value, square root, mean, variance…..)
Other mathematical and statistical (natural logarithm, exponential….)
Character string functions
Pseudo-random number generators

- Selected functions that compute simple statistics:

Name	**Purpose**
SUM	sum of non-missing arguments
MEAN	mean of non-missing arguments
VAR	variance of non-missing arguments
MIN	minimum of non-missing arguments
MAX	maximum of non-missing arguments
STD	standard deviation of non-missing arguments
N	number non-missing
NMISS	number missing
RANGE	difference between minimum and maximum

- DATA step statistics functions compute statistics for each observation or row in the SAS data set.

- Procedures produce statistics for each variable or column in the SAS data set.

- Complete list of functions available in SAS 9.4: Google “SAS Functions 9.4”

Selected SAS Functions

Function	Syntax	Definition	Example	Result
Numeric				
INT	INT(argument)	Returns the integer portion of argument	x = INT(2.17); y = INT(3.645);	x = 2 y = 3
LOG	LOG(positive argument)	Natural logarithm	x = LOG(1); y = LOG(10);	x = 0.0 y = 2.3026
LOG10	LOG10(positive argument)	Logarithm to the base 10	x = LOG10(1); y = LOG10(10);	x = 0.0 y = 1.0
MAX	MAX(arg,arg,…)	Largest non-missing value	x = MAX(9.4,6,7.2); y = MAX(-2,.,6);	x = 9.4 y = 6
MIN	MIN(arg,arg,…)	Smallest non-missing value	x = MIN(9.4,6,7.2); y = MIN(-2,.,6);	x = 6 y = -2
MEAN	MEAN(arg,arg,…)	Arithmetic mean of non-missing values	x = MEAN(1,3,4,8); y = MEAN(.,2,3);	x = 4 y = 2.5
N	N(arg,arg,…)	Number of non-missing values	x = N(1,3,4,8); y = N(.,2,3);	x = 4 y = 2
NMISS	NMISS(arg,arg,…)	Number of missing values	x = NMISS(1,2,4,8); y = NMISS(.,2,3);	x = 0 y = 1
ROUND	ROUND(arg,round-off-unit)	Rounds to nearest round-off unit	x = ROUND(3.645); y = ROUND(3.645,.1);	x = 4 y = 3.6
SQRT	SQRT(arg)	Square root	x = SQRT(25);	x = 5
SUM	SUM(arg,arg,…)	Sum of non-missing values	x = SUM(1,3,4,8); y = SUM(.,2,3);	x = 16 y = 5

Function	Syntax	Definition	Example	Result
Character				
COMPRESS	COMPRESS(source, <chars to remove>)	Removes specific characters from a string; by default, blanks are removed	x = COMPRESS('X Y'); y = COMPRESS ('a.b= c','. ');	x = 'XY' y = 'ab=c'
COUNT	COUNT(source,string to search for,<'I'>)	Counts the number of times a string appears within a longer string; optionally ignore case	x = COUNT('Applesauce','a'); y = COUNT('Applesauce','a','I');	x = 1 y = 2
INDEX (see also INDEXC, INDEXW, FIND, FINDC)	INDEX(source,string to search for)	Locates a string within a longer string and returns the position of the first occurrence	x = INDEX('ABCDEFG', 'DEF'); y = INDEX('ABCDEFG', 'X');	x = 4 y = 0
LEFT (RIGHT)	LEFT(arg)	Left aligns a SAS character expression	x = LEFT(' dog'); y = LEFT(' a dog');	x = 'dog ' y = 'a dog '
LENGTH	LENGTH(arg)	Returns the length of an argument not counting trailing blanks (missing values have a length of 1)	x = LENGTH('a dog'); y = LENGTH(' a dog ');	x = 5 y = 6
LOWCASE (UPCASE)	LOWCASE(arg)	Converts all letters in argument to lowercase	x = LOWCASE('A DOG'); y = LOWCASE('Noah');	x = 'a dog' y = 'noah'
PROPCASE	PROPCASE(arg)	Capitalizes the first letter of each word in a string	x = PROPCASE('hello world');	x = 'Hello World'
REVERSE	REVERSE(arg)	Reverses string's characters	x = REVERSE('4321');	x = '1234'

SCAN	SCAN(arg,n, <delimiters>)	Returns the nth 'word' from a string, where a 'word' is defined as anything between two delimiters; default group of delimiters is blank . < (+ & ! $ *) ; ^ - / , % |	x = SCAN('A.BC(X=Y)',3);
			y = SCAN('A.BC(X=Y)',-3);
			z = SCAN('A.BC(X=Y)',2,'(');
				x = 'X=Y'
				y = 'A'
				z = 'X=Y)'

STRIP	STRIP(arg)	Strips leading and trailing blanks from a string	x = STRIP(' abc ');	x = 'abc'

SUBSTR	SUBSTR(arg,position,<n>)	Extracts a substring from an argument starting at 'position' for 'n' characters or until end if no 'n'	p = '(919) 677-8000';
			x = SUBSTR(p,2,3);
			y = SUBSTR('adog',2);
				x = '919'
				y = 'dog'

TRANSLATE	TRANSLATE(source,
	to-1,from-1,…,
	to-n,from-n)
		Replaces 'from' characters in 'source' with 'to' characters (one-to-one replacement only)	d = '9/29/62';
			x = TRANSLATE(d,'-','/');
			y = TRANSLATE('furry','z','r');
				x = '9-29-62'
				y = 'fuzzy'

VERIFY	VERIFY(source,verify string)	Checks if source contains any unwanted values and returns first location	x = VERIFY('AXBC', 'AB'); y = VERIFY('ABB', 'AB');	x = 2 y = 0
Date				
DATEPART	DATEPART(datetime)	Returns date part of datetime value	dt = DATEPART('17JUL1997:00:00:00'dt);	dt = 13712
DAY	DAY(date)	Returns day of the month from a SAS date value	d = MDY(9,29,62); x = DAY(d);	x = 29
HOUR	HOUR(<time | datetime>)	Returns hour (0-23) from a SAS time or datetime value	x = HOUR('1:30't);	x = 1
MDY	MDY(month,day,year)	Returns a SAS date value from month, day, and year numeric values	x = MDY(1,1,1960); y = MDY(2,5,1960);	x = 0 y = 35
MONTH	MONTH(date)	Returns the month (1-12) from a SAS date value	d = MDY(9,29,1962); x = MONTH(d);	x = 9
QTR	QTR(date)	Returns the quarter (1-4) from a SAS date value	d = MDY(9,29,1962); x = QTR(d);	x = 3
TODAY	TODAY()	Returns the current date as a SAS date value	x = TODAY(); y = TODAY() – 1;	x = *today's date* y=*yesterday*
WEEKDAY	WEEKDAY(date)	Returns day of the week from a SAS date value	d = MDY(9,29,1962); x = WEEKDAY(d);	x = 7
YEAR	YEAR(date)	Returns the year from a SAS date value	d = MDY(9,29,1962); x = YEAR(d);	x = 1962

Other				
LAG	LAG(variable)	Value of the argument in the previous observation		
RANUNI	RANUNI(integer seed)	Returns a random variate from a uniform distribution; typically used to generate random numbers		
RAND	**RAND('Distribution', parm1,…,parmN)**	**Used to generate random numbers from many distributions (e.g. beta, normal, geometric, etc).**	**X = rand('normal',0,1);** **Y = rand('beta',3,7);** **Z = rand('weibull',2,5);**	
MISSING	MISSING(variable)	Returns 1 if variable value is missing, 0 otherwise		

The Concatenation Operator

- The concatenation operator || concatenates character values.
- The results of a concatenation are usually stored in a variable using an assignment statement.
- The length of the resulting variable is the sum of the lengths of each variable or constant in the concatenation operation.
- The concatenation operator does not trim trailing or leading blanks.
- The STRIP function can be used to trim leading and trailing blanks from values before concatenating them.

<u>Alternatives to || :</u>

Function	Syntax	Definition	Example	Result
CAT	CAT(string-1, string-2 <, string-n>)	Concatenates two or more strings, leaving leading and trailing blanks (same as ||)	a='UNC '; b=' CH'; y=CAT(a,b);	y='UNC CH'
CATS	CATS(string-1, string-2 <, string-n>)	Concatenates two or more strings, stripping leading and trailing blanks	a='UNC '; b=' CH'; y=CATS(a,b);	y= 'UNCCH'

CATX	CATX(separator, string-1, string-2 <, string-n>)	Concatenates two or more strings, stripping leading and trailing blanks and inserting a separator between the strings	a='UNC ';
			b=' CH';
			y=CATX('-',a,b);
			
			y=CATX(' ',a,b);
				y='UNC-CH'
				
				y='UNC CH'

The length of any variable constructed with CAT, CATS, or CATX is 200 by default.

Figure 5: A Concatenation Example

```
data one;
  c1 = 'dept';
  c2 = 'bios';
  c3 = c1 || c2;

  length c4 $ 8;

  c4 = 'dept';

  c5       = c4 || c2;
  c5_strip = strip(c4) || c2;
  cat5     = cat(c4,c2);
  cats5    = cats(c4,c2);

  c6    = c1 || ' of ' || c2;
  catx6 = catx(' of ',c4,c2);

  keep c1-c6 c5_strip cat:;
run;

title 'concatenation example';
proc print data=one noobs;
    var c3-c5 c5_strip c6 cat:;
run;

proc contents data=one; run;
```

---- PDF Destination ----

concatenation example

c3	c4	c5	c5_strip	c6	cat5	cats5	catx6
deptbios	dept	dept bios	deptbios	dept of bios	dept bios	deptbios	dept of bios

---- HTML Destination ----

concatenation example

c3	c4	c5	c5_strip	c6	cat5	cats5	catx6
deptbios	dept	dept bios	deptbios	dept of bios	dept bios	deptbios	dept of bios

Note: The HTML output format displays multiple spaces as one space, so the output doesn't clearly show that extra spaces are inserted in the middle of variable c5.

(Select PROC Contents Output)

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
1	c1	Char	4
2	c2	Char	4
3	c3	Char	8
4	c4	Char	8
5	c5	Char	12
9	c6	Char	12
6	c5_strip	Char	12
7	cat5	Char	200
8	cats5	Char	200
10	catx6	Char	200

Subsetting Observations with a Subsetting IF Statement

- A common type of transformation is subsetting observations, creating a new SAS data set with the same variables as the input data set, but with only those observations that satisfy some selection criteria.
- The subsetting IF statement can be used to accomplish this.
- Syntax: `IF LOGICAL-EXPRESSION;`

  Where *LOGICAL-EXPRESSION* is given by one of the following:

expression	GT LT GE LE EQ NE IN	expression

logical expression1	OR AND	logical expression2

- Each "expression" can be any of the forms discussed for assignment statements.
- If the expression is true, execution of the step continues for this observation.
- If the expression is false:
  - SAS stops executing statements for this observation immediately.
  - SAS returns to the top of the DATA step and begins processing the next observation.
- Complex logical expressions can be constructed by combining simple logical expressions with the operators AND and/or OR.

- <u>Examples of subsetting IF Statement:</u>

```
IF age GT 35;
```

```
IF dept EQ 'FURS';
```

```
IF (ht GT 70) AND (wt GT 180);
```

```
IF (dept EQ 'FURS') OR (clerk EQ 'AGILE');
```

Figure 6: Using a subsetting IF statement

Program

```
data work.males;
  set bios511.class;

    if sex eq 'M';

    output;
    return;
run;
```

Log

Log - (Untitled)

```
35    data work.males;
36      set bios511.class;
37        if sex eq 'M';
38        output;
39        return;
40    run;

NOTE: There were 6 observations read from the data set
      BIOS511.CLASS.
NOTE: The data set WORK.MALES has 4 observations and 5
      variables.
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
      cpu time            0.01 seconds
```

Bios511.class data set

	NAME	SEX	AGE	HT	WT
1	CHRISTIANSEN	M	37	71	195
2	HOSKING J	M	31	70	160
3	HELMS R	M	41	74	195
4	PIGGY M	F	.	48	.
5	FROG K	M	3	12	1
6	GONZO		14	25	45

Work.males data set

	NAME	SEX	AGE	HT	WT
1	CHRISTIANSEN	M	37	71	195
2	HOSKING J	M	31	70	160
3	HELMS R	M	41	74	195
4	FROG K	M	3	12	1

- All observations are read into the PDV via the SET statement.
- Only observations where sex="M" are processed beyond the subsetting IF statement and therefore only those observations are written to work.males data set.

Comparison Operators

- The left and right columns are equivalent syntax for ranges:

IF age > 35 ;	IF age GT 35 ;
IF age < 35 ;	IF age LT 35 ;
IF age >= 35 ;	IF age GE 35 ;
IF age <= 35 ;	IF age LE 35 ;
IF age = 35 ;	IF age EQ 35 ;
IF age ^= 35 ;	IF age NE 35 ;

- Text comparisons are case-sensitive, so these are not equivalent:

IF sex = 'female' ;	IF sex = 'FEMALE' ;

- SAS treats a string of any number of blanks as equal to a missing character value, so these yield equivalent results:

IF sex = ' ' ;	IF sex = ' ' ;

- One can compare to character variables/expressions with inequality operators though it is not commonly necessary.

```
IF value1 > value2 ;
```

- The above expression essentially asks the question “Does the value of value1 sort AFTER the value of value2?”
- Character strings of numbers sort BEFORE uppercase letters which sort BEFORE lowercase letters.

```
blank ! “ # $ % & ‘ ( ) * + , - / 0 1 2 3 4 5 6 7 8 9
: ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T
U V W X Y Z [ \ ] ^ _ ` a b c d e f g h I g h I j k l
m n o p q r s t u v w x y z { | } ~
```

- Comparisons can be done on numeric variables with missing values.

IF ht <= .z ;	IF MISSING(ht);

- Missing values always sort as less than other numeric values.

._ . .A .B to .Z numeric values

- Using the UPCASE function for character comparison ensures the comparison will catch data values with different cases.

IF UPCASE(sex) IN ('MALE','FEMALE');

- The IN operator provides a short-hand for a list of values; it does NOT specify a range. Hence, the below are not equivalent.

IF age IN (30,34) ;	*Equivalent to IF age=30 or age=34;
IF 30 <= age <= 34 ;	*Equivalent to IF age takes any real number between and including 30 and 34.

- Since .Z is the highest missing values in the numeric sorting sequence, the following provide equivalent syntax:

IF .z < age <= 50 ;	IF NOT(MISSING(age)) AND age <= 50;

Logical Boolean Operators and Expressions

- The following table includes Boolean operators and their mnemonics.

&	AND
|	OR
^	NOT

- Each pair shows statements with equivalent syntax.

```
IF age = 35 AND ht = 40 ;
IF (age = 35) & (ht = 40) ;

IF age >= 16 AND age <= 65 ;
IF 16 <= age <= 65 ;

IF ht > wt OR age = 40 ;
IF (ht > wt) | (age = 40) ;

IF age = 20 OR age = 30 OR age = 40 ;
IF age IN(20,30,40) ;

IF (age = 20 OR age = 30) AND sex='M';
IF age IN (20,30) AND sex='M';

IF NOT(sex = 'Male') ;
IF UPCASE(sex) NE 'MALE' ;
```

Boolean Numeric Expressions

- Boolean expressions are evaluated as either TRUE or FALSE.
  - A numeric value other than 0 or missing is evaluated as TRUE.
  - A value of 0 or missing is evaluated as FALSE.
- A numeric variable or expression can stand alone as a Boolean expression.

SAS Code	Interpretation
IF age ;	If the "age" variable is not missing and also greater than zero, continue processing the current observation. Otherwise, stop processing the observation.
IF (ht > 70) ;	If the "ht" variable has a value that is larger than 70, continue processing the current observation. Otherwise, stop processing the observation.
IF (age) & (ht > 70) ;	If the "age" variable is not missing and also greater than zero AND the "ht" variable is greater than 70, continue processing the current observation. Otherwise, stop processing the observation.

- In assignment statements:
  - A TRUE Boolean expression is equivalent to a numeric value of 1.
  - A FALSE Boolean expression is equivalent to a numeric value of 0.

SAS Code	Interpretation
newVar = (ht > 70) ;	If the value of the "ht" variable is greater than 70, then set the value of the "newVar" variable to 1. Otherwise, set the value of the "newVar" variable to 0.
newVar = (age=40);	If the value of the "age" variable is equal than 40, then set the value of the "newVar" variable to 1. Otherwise, set the value of the "newVar" variable to 0.

Figure 7: Examples with Boolean Expressions

```
data flags;
      set bios511.class;
      heightIndicator = (ht>70);
      weightIndicator = (wt>100);
run;

proc print data=flags;run;
```

Obs	NAME	SEX	AGE	HT	WT	heightIndicator	weightIndicator
1	CHRISTIANSEN	M	37	71	195	1	1
2	HOSKING J	M	31	70	160	0	1
3	HELMS R	M	41	74	195	1	1
4	PIGGY M	F	.	48	.	0	0
5	FROG K	M	3	12	1	0	0
6	GONZO		14	25	45	0	0

```
data class2;
    set bios511.class;
    if age;
run;

proc print data=class2; run;
```

Obs	NAME	SEX	AGE	HT	WT
1	CHRISTIANSEN	M	37	71	195
2	HOSKING J	M	31	70	160
3	HELMS R	M	41	74	195
4	FROG K	M	3	12	1
5	GONZO		14	25	45

Subsetting Observations with a WHERE Statement

- The WHERE statement allows you to select observations from an existing SAS data set that meet a particular condition BEFORE SAS puts the observation into the PDV.
- WHERE selection is the first operation the SAS System performs in each execution of a SET, MERGE, or UPDATE operation.
- The WHERE statement is not executable; that is, it can't be used as part of an IF/THEN statement.
- A WHERE expression is a sequence of operands and operators. You cannot use variables created within the DATA step or variables created in assignment statements.
- The WHERE statement is not a replacement for the IF statement. The two are similar in effect, but they work differently and can produce different output data sets. A DATA step can use either statement, both, or neither.
- Syntax: `WHERE EXPRESSION ;`

  In which EXPRESSION is an arithmetic or logical expression
- Examples of observation selection with WHERE statements:

WHERE age > 50 ;
WHERE UPCASE(sex) = 'FEMALE' AND ht = . ;

WHERE vs. IF

- Statement timing:
    - The WHERE statement filters observations before they are brought into the DATA step and program data vector.
    - The IF statement works on observations that are already in the DATA step and program vector.
- Since a WHERE statement works before the PDV is created and an IF statement works after the PDV is created:
    - An IF statement can use variables created in the DATA step, but WHERE cannot.
    - The location of a WHERE statement in the DATA step makes no difference, but the location of an IF statement in the DATA step can make a difference.
    - The IF statement can only use variables that are created in statements prior to the IF statement, not in statements afterwards.
- The IF statement is executable, whereas the WHERE statement is not.
- The WHERE statement can be used in SAS procedure steps and as a data set option (which we will cover soon). The IF statement cannot be used in either of these cases.
- You can stack multiple IF statements as a logical AND. That is, you can use together:

IF age > 50; IF UPCASE(sex) = ‘F’;	⇔	IF age > 50 and UPCASE(sex)=’F’;

- You cannot directly stack WHERE statements in the same way:

WHERE age > 50; WHERE UPCASE(sex) = ‘F’;	⇔	WHERE UPCASE(sex) = ‘F’;

In this case, SAS will replace the first WHERE statement with the second one, so that ultimately only the second WHERE statement is applied.

- In order to stack WHERE statements, you must use WHERE ALSO in statements after the first or simply connect the conditions with AND.

WHERE age > 50; WHERE ALSO UPCASE(sex) = ‘F’;	⇔	WHERE age > 50 and UPCASE(sex) = ‘F’;

- The WHERE statement, but not the IF statement, can be used in SAS procedure steps and as a “data set option” (which we will cover soon).

Figure 8: Comparing Subsetting IF and WHERE Statements

```
data one;
    set bios511.class;
    where age>35;
run;
```

```
data two;
    set bios511.class;
    if age>35;
run;
```

Log - (Untitled)

```
399  data one;
400      set bios511.class;
401      where age>35;
402  run;

NOTE: There were 2 observations read from the data set
      BIOS511.CLASS.
      WHERE age>35;
NOTE: The data set WORK.ONE has 2 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.00 seconds


403
404  data two;
405      set bios511.class;
406      if age>35;
407  run;

NOTE: There were 6 observations read from the data set
      BIOS511.CLASS.
NOTE: The data set WORK.TWO has 2 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Figure 9: Comparing Subsetting IF and WHERE Statements

Functions / Correct	Does Not Function

`proc means data=bios511.class;`
`  where 25 < age <= 35 and sex = 'M' ;`
`run;`
	`proc means data=bios511.class;`
	`  if 25 < age <= 35 and sex = 'M' ;`
	`run;`

Log - (Untitled)

```
413  proc means data=bios511.class;
414      where 25 < age <= 35 and sex = 'M' ;
415  run;

NOTE: Writing HTML Body file: sashtml11.htm
NOTE: There were 1 observations read from the data set
      BIOS511.CLASS.
      WHERE (age>25 and age<=35) and (sex='M');
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.08 seconds
      cpu time            0.03 seconds


416
417  proc means data=bios511.class;
418      if 25 < age <= 35 and sex = 'M' ;
         --
         180
ERROR 180-322: Statement is not valid or it is used out of
               proper order.
419  run;

NOTE: The SAS System stopped processing this step because of
      errors.
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.07 seconds
      cpu time            0.01 seconds
```

Figure 10: Comparing Subsetting IF and WHERE Statements

Functions / Correct	Functions / Incorrect	Does Not Function

`data ratio;`
`  set bios511.class;`
`  ratio = wt/ht;`
`  if ratio>.;`
`run;`
	`data ratio;`
	`  set bios511.class;`
	`  if ratio>.;`
	`  ratio = wt/ht;`
	`run;`
		`data ratio;`
		`  set bios511.class;`
		`  ratio = wt/ht;`
		`  where ratio>.;`
		`run;`

Log - (Untitled)

```
438  data ratio;
439      set bios511.class;
440      ratio = wt/ht;
441      if ratio>.;
442  run;

NOTE: Missing values were generated as a result of performing
      an operation on missing values.
      Each place is given by:
      (Number of times) at (Line):(Column).
      1 at 440:15
NOTE: There were 6 observations read from the data set
      BIOS511.CLASS.
NOTE: The data set WORK.RATIO has 5 observations and 6
      variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds


443
444  data ratio;
445      set bios511.class;
446      if ratio>.;
447      ratio = wt/ht;
448  run;

NOTE: There were 6 observations read from the data set
      BIOS511.CLASS.
NOTE: The data set WORK.RATIO has 0 observations and 6
      variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


449
450  data ratio;
451      set bios511.class;
452      ratio = wt/ht;
453      where ratio>.;
ERROR: Variable ratio is not on file BIOS511.CLASS.
454  run;

NOTE: The SAS System stopped processing this step because of
      errors.
WARNING: The data set WORK.RATIO may be incomplete.  When this
         step was stopped there were 0 observations and 6
         variables.
WARNING: Data set WORK.RATIO was not replaced because this step
         was stopped.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

Subsetting Variables with DROP and KEEP Statements

- Another type of transformation that can be performed with a DATA step is to create a data set containing a subset of the variables from the input data set.
- The DROP or KEEP statements can be used to accomplish this.
- Syntax for DROP statement:

  `DROP VARIABLE LIST;` or `KEEP VARIABLE LIST;`

  where VARIABLE LIST is a collection of variable names separated by spaces.
- Never use both KEEP and DROP statements in a single DATA step.
- If the DROP statement is used, the variables listed are not included in the output data set.
- If the KEEP statement is used, the variables listed are the only ones included in the output data set.
- The KEEP or DROP statement defines only which variables are written from the program data vector to the output data set. All variables are available during the execution of the DATA step.

Figure 11: Use of DROP/KEEP Statements

```
data noname;
    set bios511.class;
    drop name;
run;
proc print data = noname; run;
```

SEX	AGE	HT	WT
M	37	71	195
M	31	70	160
M	41	74	195
F	.	48	.
M	3	12	1
	14	25	45

```
data nameonly;
    set bios511.class;
    keep name;
run;
proc print data =nameonly; run;
```

NAME
CHRISTIANSEN
HOSKING J
HELMS R
PIGGY M
FROG K
GONZO

The LENGTH Statement

- The LENGTH statement is used to control the number of bytes allocated for a variable.
- It may also be used to establish whether the variable is character or numeric.
- When used as described below, the LENGTH statement can be safely used with numeric variables which have only integer values, thus decreasing the disk space required to store SAS data sets.
- Syntax: `LENGTH VARIABLE-LIST ($) N ...;`
  where

• VARIABLE-LIST is a collection of variables that have the same type and length
• $ denotes that the variables in the preceding VARIABLE-LIST are character, not numeric
• N is the number of bytes (length) to be assigned to the variables in the preceding VARIABLE-LIST ○ For character variables, 1 <= N <= 32,767 ○ For numeric variables on a PC or UNIX, 3 <= N <= 8
• "…" indicates that the pattern may be repeated to define multiple sets of VARIABLE-LISTs using the same convention.

LENGTH Statement Usage Notes

- For character variables, the placement of the LENGTH statement in the DATA step determines its effectiveness.
  - If it is placed before the first reference to a character variable, it will store the variable's values in the indicated number of bytes.
  - If it is placed after the step's first reference to a character variable, it will have no effect (and you will see a warning in the log).
- It is usually a good idea to specify lengths for all calculated or assigned character variables. If you do not, then the length is determined by the first occurrence of the new variable name. This could result in a shorter variable than you intend, and thus truncated values.

Notes for character variables

- The length of a character variable is determined by the first statement in which the compiler sees the variable.
- When used, the LENGTH statement should precede any assignment or SET statement involving the variable in question.
- When character variables of different lengths are compared, the shorter value is padded with blanks on the right to match the length of the longer variable (in memory only).

Notes for numeric variables

- The valid length of a numeric variable is 3-8 bytes on a PC or UNIX.
- The default length for numeric variables is 8 bytes.
    - Do not consider specifying a shorter length unless a variable contains integers only, being sure to take into account the maximum integer that can be safely stored in a given number of bytes as specified in the length tables on the bottom of this page.
    - Non-integers stored in less than 8 bytes could lose precision, so you should NEVER specify a length shorter than 8 for a numeric variable that might contain a non-integer.
- **The only time you should consider using a numeric variable length less than 8 is when your dataset is large (disk space)**
- In the PDV, all numbers are stored in 8 bytes.

Length table for PC and UNIX environments

Length in Bytes	Significant Digits Retained	Largest Integer Represented Exactly
3	3	8,192
4	6	2,097,152
5	8	536,870,912
6	11	137,438,953,472
7	13	35,184,372,088,832
8	15	9,007,199,254,740,992

Figure 12: An Example with Length Statements

SAS Code

```
data newName;
    set bios511.class;

    length last first $30 htft 8;

    label last  = 'Last Name'
          first = 'First Initial'
          htft  = 'Height (ft.)'
          sex   = 'Gender';

    format htft 6.1;

    last  = scan(propcase(Name),1,' ');
    first = scan(propcase(Name),2,' ');
    htft  = ht/12;

    keep last first sex htft ;
run;

proc print data = newName noobs label;
 var last first sex htft;
run;
```

PROC PRINT Output

Last Name	First Initial	Gender	Height (ft.)
Christiansen		M	5.9
Hosking	J	M	5.8
Helms	R	M	6.2
Piggy	M	F	4.0
Frog	K	M	1.0
Gonzo			2.1

Viewing the "newName" data set

VIEWTABLE: _last_

	SEX	last	first	htft
1	M	Christiansen		5.9
2	M	Hosking	J	5.8
3	M	Helms	R	6.2
4	F	Piggy	M	4.0
5	M	Frog	K	1.0
6		Gonzo		2.1

Figure 13: An Example with Length Statements

Functions / Correct

```
data one ;
   set bios511.class;
   length size $ 6 ;
   if ht < 15 then size = 'small';
   if ht >= 15 then size = 'medium';
run;
proc print data=one; run;
```

Log - (Untitled)

```
726  data one ;
727     set bios511.class;
728     length size $ 6 ;
729     if ht < 15 then size = 'small';
730     if ht >= 15 then size = 'medium';
731  run;

NOTE: There were 6 observations read from the data set
      BIOS511.CLASS.
NOTE: The data set WORK.ONE has 6 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time           0.10 seconds
      cpu time            0.01 seconds
```

Obs	NAME	SEX	AGE	HT	WT	size
1	CHRISTIANSEN	M	37	71	195	medium
2	HOSKING J	M	31	70	160	medium
3	HELMS R	M	41	74	195	medium
4	PIGGY M	F	.	48	.	medium
5	FROG K	M	3	12	1	small
6	GONZO		14	25	45	medium

Functions / Incorrect

```
data two ;
   set bios511.class;
   if ht < 15 then size = 'small';
   if ht >= 15 then size = 'medium';
   length size $ 6 ;
run;
proc print data=two; run;
```

Log - (Untitled)

```
742  data two ;
743     set bios511.class;
744     if ht < 15 then size = 'small' ;
745     if ht >= 15 then size = 'medium' ;
746     length size $ 6 ;        /* has no effect! */
WARNING: Length of character variable size has already been
         set.
         Use the LENGTH statement as the very first statement
         in the DATA STEP to declare the length of a character
         variable.
747  run;

NOTE: There were 6 observations read from the data set
      BIOS511.CLASS.
NOTE: The data set WORK.TWO has 6 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.04 seconds
```

Obs	NAME	SEX	AGE	HT	WT	size
1	CHRISTIANSEN	M	37	71	195	mediu
2	HOSKING J	M	31	70	160	mediu
3	HELMS R	M	41	74	195	mediu
4	PIGGY M	F	.	48	.	mediu
5	FROG K	M	3	12	1	small
6	GONZO		14	25	45	mediu

Figure 14: An Example with Length Statements / Assigning Constants

```
data one;
    set bios511.class;

    ** create c1 as a character constant **;
    c1 = 'bios';

    ** create c2 as a character constant **;
    c2 = 'csc';

    ** what will be the value of c3 after these two statements? **;
    c3 = c2;
    c3 = 'pqrv';

    ** create some numeric constants **;
    n1 = 100;
    n2 = 100.00;
    n3 = 1e2;  /* scientific notation */
run;
```

Log - (Untitled)

```
788  data one;
789      set bios511.class;
790
791      ** create c1 as a character constant **;
792      c1 = 'bios';
793
794      ** create c2 as a character constant **;
795      c2 = 'csc';
796
797      ** what will be the value of c3 after these two
797! statements? **;
798      c3 = c2;
799      c3 = 'pqrv';
800
801      ** create some numeric constants **;
802      n1 = 100;
803      n2 = 100.00;
804      n3 = 1e2;  /* scientific notation */
805  run;

NOTE: There were 6 observations read from the data set
      BIOS511.CLASS.
NOTE: The data set WORK.ONE has 6 observations and 11 variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.06 seconds
```

Obs	NAME	SEX	AGE	HT	WT	c1	c2	c3	n1	n2	n3
1	CHRISTIANSEN	M	37	71	195	bios	csc	pqr	100	100	100
2	HOSKING J	M	31	70	160	bios	csc	pqr	100	100	100
3	HELMS R	M	41	74	195	bios	csc	pqr	100	100	100
4	PIGGY M	F	.	48	.	bios	csc	pqr	100	100	100
5	FROG K	M	3	12	1	bios	csc	pqr	100	100	100
6	GONZO		14	25	45	bios	csc	pqr	100	100	100

SAS Date, Time, and Date-Time Values

- Although dates and times are typically written in a numeric form (12/25/83, 12:15), when written this way they are not truly numeric in that we cannot directly perform arithmetic operations on them.
- SAS allows date and time data to be stored as numeric variables by counting the number of days or seconds from a reference point.

Date	Number of days since January 1, 1960
Time	Number of seconds and hundredths of seconds since midnight
Date-time	Number of seconds since midnight, January 1, 1960

- Examples of Date Representations:

Jul 4, 1776	Stored as -67019
May 8, 1945	Stored as -5351
Jan 1, 1960	Stored as 0
Nov 9, 1989	Stored as 10905
Apr 6, 2009	Stored as 18207

- Using the baseline of January 1, 1960 is arbitrary.
- Any dates from 1582 to 20,000 AD are valid.
- SAS accounts for leap years, century adjustments, etc.
- Although date and date-time values have implied baseline times, differences in these values are directly interpretable. For example, the number of days from January 1, 1953 to November 24, 1983 is:

  8728 – (-2556) = 11284 days

Date Constants

- You can use SAS **date constants** to generate a SAS date from a specific date entered in your SAS code.
- Date constants are special constants that SAS converts into SAS date values.
- The syntax of a date constant is: *'ddmmmyyyy'd*
  where:

dd	is a one or two digit value for day
mmm	is a three letter abbreviation for month (JAN,FEB…)
yyyy	is a two or four digit value for year (four recommended)

The *d* at the end of the constant ensures that SAS does not confuse the string with a character constant.

Figure 15: Example Using Date Constants

SAS Code

```
proc print data = bios511.UFO noobs;
run;


data oldUFO;
 set bios511.UFO;
 where sightDate <= '16JUL2004'd;
run;

proc print data = oldUFO noobs; run;
```

PROC PRINT for "oldUFO" data set

SightDate	HowMany
05JUL2004	1
06JUL2004	1
07JUL2004	2
11JUL2004	3
16JUL2004	1

PROC PRINT for "BIOS511.UFO" Data Set

SightDate	HowMany
05JUL2004	1
06JUL2004	1
07JUL2004	2
11JUL2004	3
16JUL2004	1
19JUL2004	5
23JUL2004	1
24JUL2004	1
29JUL2004	1

Date Functions

- There are <u>many</u> useful functions available for handling SAS dates. These are just a sample, and see a larger sample in the tables earlier in the chapter.

Function	Description
YEAR(SAS-date)	extracts the year from a SAS date
MONTH(SAS-date)	extracts the month from a SAS date and returns a number between 1 and 12
DAY(SAS-date)	extracts the day from a SAS date and returns a number between 1 and 31
TODAY()	extracts the date from the computer system's clock and stores the value as a SAS date. This function does not require any arguments
MDY(month,day,year)	creates a SAS date from separate month, day, and year variables. Arguments can be SAS numeric variables or constants. A missing or out of range value creates a missing value

Simple Calculations Using Date Variables

- Using SAS date variables, you can find the time elapsed between two dates.
- Simply subtract the dates to find the number of elapsed days, then, if necessary, divide the number to scale it to months, years, weeks, or any other unit of interest.

```
Days =  date2 - date1 ;
Weeks = (date2 - date1) / 7 ;
Months =  (date2 - date1) / 30.4 ;
Years =  (date2 - date1) / 365.25 ;
```

Figure 16: Working with Dates in SAS

```
data recentUFO;
 set bios511.UFO;

 targetDate = '16JUL2004'd;
 if sightDate >= targetDate;

 sightYear   = year(sightDate);
 sightMonth  = month(sightDate);

 elapsedDays = (sightDate-targetDate)+1;

run;

proc print data = recentUFO noobs; run;
```

SightDate	HowMany	targetDate	sightYear	sightMonth	elapsedDays
16JUL2004	1	16268	2004	7	1
19JUL2004	5	16268	2004	7	4
23JUL2004	1	16268	2004	7	8
24JUL2004	1	16268	2004	7	9
29JUL2004	1	16268	2004	7	14

How does SAS interpret two-digit years?

What if SAS is provided with a date value of 10/27/08? In converting that to a SAS date value, how does SAS know whether the 08 means 1908 (it's a very old person) or 2008 (it's a very young person)?

The YEARCUTOFF= system option controls how SAS interprets two-digit years. The YEARCUTOFF value specifies the first year of a hundred-year span for SAS to use. By default in 9.4, the YEARCUTOFF value is 1926, so two-digit years are interpreted as occurring between 1926 and 2025.

A good lesson: When you are in control, always supply dates using four-digit years.

Miscellaneous Examples:

Figure 17: Using Incorrect Arguments Data Types with Functions

```
data one;
    set bios511.class;

    n1 = min(ht,wt);
    n2 = min(age,35);
    n3 = min(ht,sex);
    n4 = mean(sex,name);

run;
```

```
Log - (Untitled)
927
928  data one;
929      set bios511.class;
930
931      n1 = min(ht,wt);
932      n2 = min(age,35);
933      n3 = min(ht,sex);
934      n4 = mean(sex,name);
935
936  run;

NOTE: Character values have been converted to numeric
      values at the places given by: (Line):(Column).
      933:17   934:15   934:19
NOTE: Invalid numeric data, SEX='M' , at line 933 column 17.
NOTE: Invalid numeric data, SEX='M' , at line 934 column 15.
NOTE: Invalid numeric data, NAME='CHRISTIANSEN' , at line 934
      column 19.
NAME=CHRISTIANSEN SEX=M AGE=37 HT=71 WT=195 n1=71 n2=35 n3=71
n4=. _ERROR_=1 _N_=1
NOTE: Invalid numeric data, SEX='M' , at line 933 column 17.
NOTE: Invalid numeric data, SEX='M' , at line 934 column 15.
NOTE: Invalid numeric data, NAME='HOSKING J' , at line 934
      column 19
```

Obs	NAME	SEX	AGE	HT	WT	n1	n2	n3	n4
1	CHRISTIANSEN	M	37	71	195	71	35	71	.
2	HOSKING J	M	31	70	160	70	31	70	.
3	HELMS R	M	41	74	195	74	35	74	.
4	PIGGY M	F	.	48	.	48	35	48	.
5	FROG K	M	3	12	1	1	3	12	.
6	GONZO		14	25	45	25	14	25	.

Note: "Invalid numeric data" is a term one should ALWAYS check for in SAS logs.

Figure 18: Using Ill-formed Assignment Statements

```
data one;
    set bios511.class;
    ** use a character variable with a numeric operand **;
    c1 = sex + 2;
    ** numeric variable on left, character variable on right **;
    length n1 8;
    n1 = sex;
    ** character variable on left, numeric variable on right **;
    length c2 $ 3;
    c2 = age;
run;
proc print data = one ; run;
proc contents data = one ; run;
```

Log - (Untitled)

```
973  data one;
974      set bios511.class;
975      ** use a character variable with a numeric operand **;
976      c1 = sex + 2;
977      ** numeric variable on left, character variable on
977! right **;
978      length n1 8;
979      n1 = sex;
980      ** character variable on left, numeric variable on
980! right **;
981      length c2 $ 3;
982      c2 = age;
983  run;

NOTE: Character values have been converted to numeric
      values at the places given by: (Line):(Column).
      976:10   979:10
NOTE: Numeric values have been converted to character
      values at the places given by: (Line):(Column).
      982:10
NOTE: Invalid numeric data, SEX='M' , at line 976 column 10.
NOTE: Invalid numeric data, SEX='M' , at line 979 column 10.
NAME=CHRISTIANSEN SEX=M AGE=37 HT=71 WT=195 c1=. n1=. c2=37
 ERROR =1  _N =1
```

Obs	NAME	SEX	AGE	HT	WT	c1	n1	c2
1	CHRISTIANSEN	M	37	71	195	.	.	37
2	HOSKING J	M	31	70	160	.	.	31
3	HELMS R	M	41	74	195	.	.	41
4	PIGGY M	F	.	48	.	.	.	.
5	FROG K	M	3	12	1	.	.	3
6	GONZO		14	25	45	.	.	14

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
3	AGE	Num	8
4	HT	Num	8
1	NAME	Char	12
2	SEX	Char	1
5	WT	Num	8
6	c1	Num	8
8	c2	Char	3
7	n1	Num	8

Note: "values have been converted to" is a term one should ALWAYS check for in SAS logs.

Figure 19: Misspelling a Variable Name

```
data one;
    set bios511.class;

      female=(sexc="F");
run;
proc print data = one; run;
```

Log - (Untitled)

```
1016  data one;
1017      set bios511.class;
1018
1019      female=(sexc="F");
1020  run;

NOTE: Variable sexc is uninitialized.
NOTE: There were 6 observations read from the data set
      BIOS511.CLASS.
NOTE: The data set WORK.ONE has 6 observations and 7 variables.
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
      cpu time            0.01 seconds
```

Obs	NAME	SEX	AGE	HT	WT	female	sexc
1	CHRISTIANSEN	M	37	71	195	0	
2	HOSKING J	M	31	70	160	0	
3	HELMS R	M	41	74	195	0	
4	PIGGY M	F	.	48	.	0	
5	FROG K	M	3	12	1	0	
6	GONZO		14	25	45	0	

Note: "uninitialized" is a term one should ALWAYS check for in SAS logs.