Data Step Information - 2

New Variable Assignment or Data Transformations
Data tables may be created using an INPUT statement in your program. Often times, variables of interest are functions of the variables that you have measured and recorded in your data table.

Example Suppose you have recorded the lengths in millimeters of the items you are studying along with other characteristics. Such as,

```
DATA  one;
INPUT   id   location $   length   weight   age   gender $   ;
DATALINES;
23   payne   75    14   2.5   f
41   payne   68    16   2     m
17   payne   57    12   1.5   f
```

The data table named 'one' will contain the above six variables: id location length weight age gender. Suppose that you wish to convert the length in millimeters to length in inches. You could create a new variable in data table 'one' that is a measure in inches by doing the following.

```
DATA one;
INPUT  id   location $  length   weight   age   gender $ ;
L_IN = length / 25.4;
DATALINES;
23   payne   75    14   2.5   f
41   payne   68    16   2     m
17   payne   57    12   1.5   f
```

In the above SAS code, a new variable called L_IN is created. The values of this new variable are obtained by dividing each of the values read in for length. The values you input after the DATALINES (or CARDS) statement is the same as before. When the DATA step executes, there will be seven variables in it: six of them you input (id location length weight age gender) and one (L_IN) that is a function of those values input.

Variable assignment or data transformation statements must be placed ***after*** the INPUT statement and ***before*** the DATALINES (or CARDS) statement. You can use any number of assignments in one DATA step execution. The form of the variable assignment is as follows:
newvariablename = function of variables input or created in prior transformations;

In these statements you can use the following symbols for arithmetic operations:

Symbol	**Operation**
+	addition
-	subtraction
*	multiplication

/	division
**	exponentiation
( )	parenthesis as needed

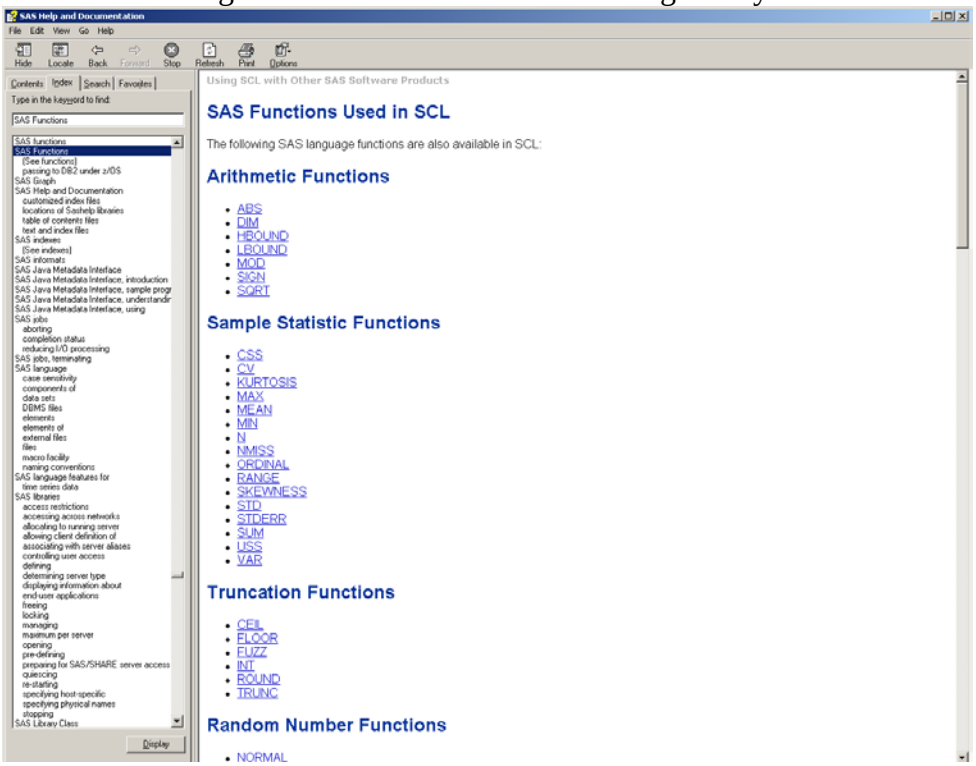
There are also many, many algebraic functions available. Some of the more commonly used functions are given in the table below. The word *argument* in the syntax refers to a variable name in the data table or a function of variables.

Function	Syntax	Description
MAX	y = MAX(argument1, . . . , argumentn)	returns the largest value of those arguments specified in parenthesis
MIN	y = MIN(argument1, . . . , argumentn)	returns the smallest value of those arguments specified in parenthesis
MOD	y = MOD(argument1, argument2)	returns the remainder
SIGN	y = SIGN(argument)	returns the sign of the value or 0
SQRT	y = SQRT(argument)	computes a square root
ABS	y = ABS(argument)	computes the absolute value
INT	y = INT(argument)	computes the greatest integer of the value
ROUND	y = ROUND(argument)	rounds the value to the nearest unit
EXP	y = EXP(argument)	computes e^x where x is specified
LOG	y = LOG(argument)	computes the natural log or log base e
LOG2	y = LOG2(argument)	computes the log base 2
LOG10	y = LOG10(argument)	computes the log base 10 or common log
LAGn	y = LAGn(argument)	returns the value occurring n observations before this one. n must be a whole number between 1 and 100
MEAN	y = MEAN(argument1, . . . , argumentn)	computes the arithmetic mean of the arguments listed
SIN	y = SIN (argument)	computes the sine of an angle
STD	y = STD(argument1, . . . , argumentn)	returns the standard deviation of the arguments listed
COS	y = COS(argument)	computes the cosine of an angle
TAN	y = TAN(argument)	computes the tangent of an angle
ARCOS	y = ARCOS(argument)	computes the inverse cosine
ARSIN	y = ARSIN(argument)	computes the inverse sine
ATAN	y = ATAN(argument)	computes the inverse tangent
UPCASE	y = UPCASE(argument)	converts the value to upper case
_n_	y = _n_	sets y equal to the value of the observation number

If you wish to see more SAS functions, you can obtain a list from SAS Help. From the pull-down menu select **Help – SAS Help and Documentation** (version 9). After SAS Help opens, select the tab labeled **Index** and type SAS Functions in the blank. Click on SAS Functions in the list

below the blank to display more information about the available functions. You should be able to view the following screen image version 9 of SAS you. Note the many categories of functions that SAS has available.

version 9 – the right side of this screen scrolls through many SAS functions



***Objective 11*:** Create a data table containing the physical measurements of fish caught in Payne County. Examine the output from the suggested functions in the program below. Experiment with other data functions.

```
DATA one;
INPUT  id   location $  length   weight   age   gender $ ;
L_IN = length / 25.4;
a = log(length);
b = exp(age);
c = lag1(age);
d = lag2(age);
g = upcase(gender);
DATALINES;
23   payne   75    14   2.5   f
41   payne   68    16   2     m
17   payne   57    12   1.5   f
PROC PRINT DATA = one;
RUN;
QUIT;
```

The data table ONE will now have 12 variables in it. Six variables are all that are included on the INPUT statement. The UPCASE function is a quick way to correct uppercase/lowercase error problems in the data. Remember data are case sensitive.

IF - THEN Statements

Variables can also be created or deleted by using an IF - THEN statement. An IF - THEN statement must also appear after the INPUT line and before the DATALINES (or CARDS) statement.

The form of the IF-THEN statement is :

```
IF  condition specified  THEN  assignment1;
```

If the condition specified is true, then assignment1 will be performed. If the condition specified is not true, then the transformation is not done. This may create a missing value in the data set.

Another form of the IF – THEN statement includes an ELSE statement.

```
If condition specified THEN assignment1; ELSE  assignment2;
```

The ELSE statement is optional. If the condition specified is true, then assignment1 will be performed. If it is not true, then assignment2 will be performed.

The condition specified can use any of the following symbols:

Condition	Symbol	Or Text	SAS Example
equal	=	eq	x = 5
not equal	^=	ne	color ne 'Red'
less than	<	lt	6.7 < x < 8.2
less than or equal to	<=	le	y le 15.8
greater than	>	gt	t > 10.2
greater than or equal to	>=	ge	r ge 14.3
or		or	x < 3 or x ge 7
and		and	y > 12 and t ne 13

WHERE statement

The WHERE statement is not a type of variable assignment but is a convenient tool for creating a subgroup on which a procedure is to operate. The WHERE statement also requires that a condition be specified. The condition follows the same syntax as the IF-THEN statement conditions above. Examine the following example for illustrations of IF-THEN, WHERE statement and other transformations and variable assignments.

Example The text in italics to the right describes what the transformation is doing. It is NOT a part of the SAS code. Notice that the values of character variables appear in quotes when they are specified in the SAS code.

```
DATA one;
INPUT  id   location $  length   weight   age   gender $ ;
IF gender = 'f' THEN DELETE;                    only males will be kept in the data table

IF length < 60 THEN size = 'small';          all items smaller than 60 are small
ELSE size = 'large';                      all items greater than or equal to 60 are large
                        Notice that size is a string variable and the value is enclosed in quotes.

state = 'OK';                                  variable state has value OK for all items

IF age LE 2.0 then group = 1;          Three groups are created.  Group 1 is youngest.
IF 2.0 < age < 3.0 then group = 2;                          All items with ages between
                                                                    2 and 3 in group 2.
IF age GE 3.0 THEN group = 3;                  The oldest specimens are in group 3.

IF location="Payne" THEN length2 = length/25.4;    This transformation is never done
                                    since payne is spelled in the data set in all lowercase letters.
DATALINES;
23   payne   75    14   2.5   f
41   payne   68    16   2     m
17   payne   57    12   1.5   f
;
PROC PRINT DATA=one; WHERE weight le 15;   Only observations with weights smaller than
                                        15 are printed.  All variables are printed since
                                                        no VAR statement is included.
                                In this program, no data prints since only males are in the data set,
                                                and the male remaining the data set weighs more than 15.

PROC MEANS DATA=one; WHERE gender='m';                  Summary statistics for the
VAR weight length;                                                  males are requested.
run;
quit;
```

SET statement

The SET statement can be used within a DATA step operation. The set command can be used in two ways.

1. **Modify an existing data table**

   One existing data table can be called into a DATA step to be modified. This new data table can be saved by a new name, or can overwrite the old data table. This is illustrated below.

`DATA newname;`
`SET oldname;`
. . . data step operations . . .
	Create a new data table
	Call in an existing table
	Modify the existing table with data step operations. When this DATA step is completed, two DATA sets: OLDNAME and NEWNAME will exist.
	
	This is useful if one wishes to modify variables in a data table, and keep the original data table separate from the modified one.
	
	Example
	
	`DATA new;` *Creates a new data table*
	`SET original;` *Gets the original data table*
	`x = log(y);`
	`IF gender = "F" THEN DELETE;`

`DATA oldname;`
`SET oldname;`
. . . data step operations . . .
	Call in an existing data table.
	Modify the existing table with DATA step operations. When this DATA step is completed, only one DATA set, OLDNAME, will exist. It will contain all of the new information specified in the DATA step operations and will replace the previous version of the DATA table OLDNAME.

Objective 7: Add the additional variable 'species' to data table one. Call this new data table 'two.' Set the value of species equal to "perch" for all observations. Print the id, location and species.

2. **Concatenating and Merging Data tables**

   More than one data table can be a part of a SAS program. Sometimes it is necessary to combine the information in two or more sets. This can be done in the DATA step in either of the following ways.

   If two or more data tables are to be concatenated (combined) the SET command can do this. The syntax of this command is:

DATA newtable;
SET *datatable1 ... datatablen;*

When using the SET command, the data tables must have the same variable names in each of the data tables. The SET command vertically concatenates (or stacks) the data. The number of observations in a data table is increased when concatenating two or more data tables.

Objective 12: Form one data set from two family income data sets recorded for Payne and Noble Counties.

```
DATA   payne;
INPUT  family   income  children  home $ ;
county = 'payne';
DATALINES;
14  36000   3   rent
15  47000   2   own
DATA  noble;
INPUT  family  income  children  home $;
county = 'noble';
DATALINES;
23  24000   2   own
45  27000   1   own
DATA combine;
SET  payne  noble;
PROC PRINT DATA=COMBINE;
RUN;
QUIT;
```

Execute the above program and note how the data in the set COMBINE appear.

Two data tables PAYNE and NOBLE are read. Each of the data tables has five variables. In the set COMBINE all of the PAYNE information and all of the NOBLE information are combined into one set. In this particular example, all items in the PAYNE set were assigned the value 'payne' to a variable county and all items in the NOBLE set were assigned the value 'noble' to a variable county. With all of these items in the same program, SAS procedures may be used to operate on any of the three data tables.

Modifications to Objective 12:

1. Delete COUNTY='noble' from the program code and run. The result is that the second data set has fewer variables than the first. SAS can still concatenate them.

2. With the deletion made in the first modification, change the statement
SET payne noble;

to

SET noble payne;

Notice that the SET command "stacks" the data tables in the order you specify in the SET statement. This is true whether you are concatenating two data sets or several.

Merge statement

Data tables that have one or more common variables can be merged. The MERGE command increases the number of variables or columns in the data table. **Each of the data tables must first be sorted by the variable(s) that they have in common.** If the data tables do not have at least one common variable, they cannot be merged.

```
PROC SORT DATA=a; BY variablelist;
PROC SORT DATA=b; BY variablelist;
DATA new;
MERGE a b ; BY variablelist;
```

The above SAS code works for two or more data tables. Each data table must contain the same common information by which the tables are to be merged. Each data table must be sorted by that common variable list prior to merging.

Objective 13: The Personnel and Accounting departments each have recorded sick days for the first quarter of the year. Put these into one data set so that each month has associated with it two responses: one for personnel and one for accounting.

```
DATA  one;
INPUT  month  $  acctg;
DATALINES;
MAR   7
JAN    23
FEB   20
PROC SORT DATA=one;   BY month;
DATA  two;
INPUT month $  prsnl ;
DATALINES;
JAN   19
FEB   14
MAR  9
PROC SORT  DATA=two;  BY  month;
DATA  new;
MERGE  one  two ;
BY  month;
total  = acctg  + prsnl;
PROC PRINT DATA=new;
TITLE  'Number of Sick Days for Departments -  First Quarter ';
RUN;
QUIT;
```

Note: When the data are sorted by month, the months are arranged alphabetically. How could you arrange the months in chronological order?

Objective 14: Modify the above demonstration program to arrange the months in chronological order. (Option 1: Add a variable to each data table called monthn where monthn=1 for January, monthn = 2 for February, etc. Option 2: For the final data set, write IF – THEN statements that create the monthn variable.)