SAS 3. Working with Your Data

1. Methods for Getting Your Data into SAS

You create and redefine variables with assignment statements using this basic form:

variable = expression;

Here are examples of these basic types of assignment statements: Type of

expression	Assignment statement
Numeric (character) constant	Qwerty = 10; (Qwerty = 'ten';)
a variable	Qwerty = OldVar;
addition	Qwerty = OldVar + 10;
subtraction	Qwerty = OldVar - 10;
multiplication	Qwerty = OldVar * 10;
division	Qwerty = OldVar / 10;
exponentiation	Qwerty = OldVar ** 10;

Example The following raw data are from a survey of home gardeners. Gardeners were asked to estimate the number of pounds they harvested for four crops: tomatoes, zucchini, peas, and grapes.

Gregor 10 2 40 0
Molly 15 5 10 1000
Luther 50 10 15 50
Susan 20 0 . 20

The following program inputs and then modifies the data.

```
DATA homegarden;
INPUT Name $ 1-7 Tomato Zucchini Peas Grapes;
Zone = 14;
Type = 'home';
Zucchini = Zucchini * 10;
Total = Tomato + Zucchini + Peas + Grapes;
PerTom = (Tomato / Total) * 100;
DATALINES;
Gregor 10 2 40 0
Molly  15 5 10 1000
Luther 50 10 15 50
Susan  20 0 . 20
;
PROC PRINT data=homegarden;
TITLE 'Home Gardening Survey';
RUN;
```

Notice that the variable Zucchini appears only once because the new value replaced the old value. The other four assignment statements each created a new variable. The variable Peas had a missing value for the last observation. Because of this, the variables Total and PerTom, which are calculated from Peas, were also set to missing and a message appeared in the log.

Exercise: Save the data as ‘Garden.txt’ in your local directory, and use the following code to implement the same task as the above.

```
DATA homegarden;
INFILE ' D:\Users\mshu\Desktop\Garden.txt';
INPUT Name $ 1-7 Tomato Zucchini Peas Grapes;
Zone = 14;
Type = 'home';
Zucchini = Zucchini * 10;
Total = Tomato + Zucchini + Peas + Grapes;
PerTom = (Tomato / Total) * 100;
PROC PRINT DATA = homegarden;
TITLE 'Home Gardening Survey';
RUN;
```

2. Using SAS functions

Sometimes a simple expression, using only arithmetic operators, does not give you the new value you are looking for. This is where functions are handy, simplifying your task because SAS has already done the programming for you. SAS has over 400 functions in the following general areas:

Character	Probability	Date and Time	Random Number
Financial	Sample Statistics	Macro	State and ZIP Code
Mathematical			

SAS functions have the following general form:

function-name(argument, argument, ...)

All functions must have parentheses even if they don't require any arguments.

Example Data from a pumpkin carving contest illustrate the use of several functions. The contestants' names are followed by their age, type of pumpkin (carved or decorated), date of entry, and the scores from five judges:

```
Alicia Grossman  13 c 10-28-2003 7.8 6.5 7.2 8.0 7.9
Matthew Lee       9 D 10-30-2003 6.5 5.9 6.8 6.0 8.1
Elizabeth Garcia 10 C 10-29-2003 8.9 7.9 8.5 9.0 8.8
Lori Newcombe     6 D 10-30-2003 6.7 5.6 4.9 5.2 6.1
Jose Martinez     7 d 10-31-2003 8.9 9.510.0 9.7 9.0
Brian Williams   11 C 10-29-2003 7.8 8.4 8.5 7.9 8.0
```

The following program reads the data, creates two new variables (AvgScore and DayEntered) and transforms another (Type):

```
DATA contest;
INPUT Name $16. Age 3. +1 Type $1. +1 Date MMDDYY10.
(Scr1 Scr2 Scr3 Scr4 Scr5) (4.1);
AvgScore = MEAN(Scr1, Scr2, Scr3, Scr4, Scr5);
DayEntered = DAY(Date);
Type = UPCASE(Type);
DATALINES;
Alicia Grossman  13 c 10-28-2003 7.8 6.5 7.2 8.0 7.9
Matthew Lee       9 D 10-30-2003 6.5 5.9 6.8 6.0 8.1
Elizabeth Garcia 10 C 10-29-2003 8.9 7.9 8.5 9.0 8.8
Lori Newcombe     6 D 10-30-2003 6.7 5.6 4.9 5.2 6.1
Jose Martinez     7 d 10-31-2003 8.9 9.510.0 9.7 9.0
```

```
Brian Williams   11 C 10-29-2003 7.8 8.4 8.5 7.9 8.0
;
PROC PRINT DATA = contest;
TITLE 'Pumpkin Carving Contest';
RUN;
```

Exercise: Reconstruct the table above by adding a new variable TotalScore=SUM(Scr1, Scr2, Scr3, Scr4, Scr5);

3. Selected SAS functions

Function name	Syntax[2]	Definition
Numeric		
INT	INT(*arg*)	Returns the integer portion of argument
LOG	LOG(*arg*)	Natural logarithm
LOG10	LOG10(*arg*)	Logarithm to the base 10
MAX	MAX(*arg,arg,...*)	Largest non-missing value
MEAN	MEAN(*arg,arg,...*)	Arithmetic mean of non-missing values
MIN	MIN(*arg,arg,...*)	Smallest non-missing value
ROUND	ROUND(*arg, round-off-unit*)	Rounds to nearest round-off unit
SUM	SUM(*arg,arg,...*)	Sum of non-missing values
Character		
LEFT	LEFT(*arg*)	Left aligns a SAS character expression
LENGTH	LENGTH(*arg*)	Returns the length of an argument not counting trailing blanks (missing values have a length of 1)
SUBSTR	SUBSTR(*arg,position,n*)	Extracts a substring from an argument starting at '*position*' for '*n*' characters or until end if no '*n*'[3]
TRANSLATE	TRANSLATE(*source,to-1, from-1,...to-n,from-n*)	Replaces '*from*' characters in '*source*' with '*to*' characters (one to one replacement only—you can't replace one character with two, for example)
TRIM	TRIM(*arg*)	Removes trailing blanks from character expression
UPCASE	UPCASE(*arg*)	Converts all letters in argument to uppercase
Date		
DATEJUL	DATEJUL(*julian-date*)	Converts a Julian date to a SAS date value[4]
DAY	DAY(*date*)	Returns the day of the month from a SAS date value
MDY	MDY(*month,day,year*)	Returns a SAS date value from month, day, and year values
MONTH	MONTH(*date*)	Returns the month (1-12) from a SAS date value
QTR	QTR(*date*)	Returns the yearly quarter (1-4) from a SAS date value
TODAY	TODAY()	Returns the current date as a SAS date value

Here are examples using the selected functions.

Function name	Example	Result	Example	Result
Numeric				
INT	`x=INT(4.32);`	`x=4`	`y=INT(5.789);`	`y=5`
LOG	`x=LOG(1);`	`x=0.0`	`y=LOG(10);`	`y=2.30259`
LOG10	`x=LOG10(1);`	`x=0.0`	`y=LOG10(10);`	`y=1.0`
MAX	`x=MAX(9.3,8,7.5);`	`x=9.3`	`y=MAX(-3,.,5);`	`y=5`
MEAN	`x=MEAN(1,4,7,2);`	`x=3.5`	`y=MEAN(2,.,3);`	`y=2.5`
MIN	`x=MIN(9.3,8,7.5);`	`x=7.5`	`y=MIN(-3,.,5);`	`y=-3`
ROUND	`x=ROUND(12.65);`	`x=13`	`y=ROUND(12.65,.1);`	`y=12.7`
SUM	`x=SUM(3,5,1);`	`x=9.0`	`y=SUM(4,7,.);`	`y=11`
Character				
LEFT	`a='  cat';` `x=LEFT(a);`	`x='cat  '`	`a='  my cat';` `y=LEFT(a);`	`y='my cat  '`
LENGTH	`a='my cat';` `x=LENGTH(a);`	`x=6`	`a=' my cat ';` `y=LENGTH(a);`	`y=7`
SUBSTR	`a='(916)734-6281';` `x=SUBSTR(a,2,3);`	`x='916'`	`y=SUBSTR('1cat',2);`	`y='cat'`

TRANSLATE	`a='6/16/99';`
	`x=TRANSLATE`
	`(a,'-','/');`
		`x='6-16-99'`	`a='my cat can';`
			`y=TRANSLATE`
			`(a, 'r','c');`
				`y='my rat ran'`

TRIM	`a='my  '; b='cat';` `x=TRIM(a)||b;`[5]	`x='mycat  '`	`a='my cat '; b='s';` `y=TRIM(a)||b;`	`y='my cats '`
UPCASE	`a='MyCat';` `x=UPCASE(a);`	`x='MYCAT'`	`y=UPCASE('Tiger');`	`y='TIGER'`
Date				
DATEJUL	`a=60001;` `x=DATEJUL(a);`	`x=0`	`a=60365;` `y=DATEJUL(a);`	`y=364`
DAY	`a=MDY(4,18,1999);` `x=DAY(a);`	`x=18`	`a=MDY(9,3,60);` `y=DAY(a);`	`y=3`
MDY	`x=MDY(1,1,1960);`	`x=0`	`m=2;  d=1;  y=60;` `Date=MDY(m,d,y);`	`Date=31`
MONTH	`a=MDY(4,18,1999);` `x=MONTH(a);`	`x=4`	`a=MDY(9,3,60);` `y=MONTH(a);`	`y=9`
QTR	`a=MDY(4,18,1999);` `x=QTR(a);`	`x=2`	`a=MDY(9,3,60);` `y=QTR(a);`	`y=3`
TODAY	`x=TODAY();`	*x=today's date*	`x=TODAY()-1;`	*x=yesterday's date*

4. Using if-then statement

The IF-THEN statement has the form

```
IF condition THEN action;
```

A single IF-THEN statement can only have one action. If you add the keywords DO and END, then you can execute more than one action. For example

```
IF condition THEN DO;
      action;
      action;
END;
```

You can also specify multiple conditions with the keywords AND and OR:

```
IF condition AND condition THEN action;
```

Example The following data about used cars contain values for model, year, make, number of seats, and color:

```
Corvette 1955 . 2 black
XJ6 1995 Jaguar 2 teal
Mustang 1966 Ford 4 red
Miata 2002 . . silver
CRX 2001 Honda 2 black
Camaro 2000 . 4 red
```

This program reads the data and uses a series of IF-THEN statements to fill in missing data, and creates a new variable, Status:

```
DATA sportscars;
INPUT Model $ Year Make $ Seats Color $;
IF Year < 1975 THEN Status = 'classic';
IF Model = 'Corvette' OR Model = 'Camaro' THEN Make = 'Chevy';
IF Model = 'Miata' THEN DO;
Make = 'Mazda';
Seats = 2;
END;
DATALINES;
Corvette 1955 . 2 black
XJ6 1995 Jaguar 2 teal
Mustang 1966 Ford 4 red
Miata 2002 . . silver
CRX 2001 Honda 2 black
Camaro 2000 . 4 red
;
PROC PRINT DATA = sportscars;
TITLE “Eddy’s Excellent Emporium of Used Sports Cars”;
RUN;
```

Exercise: Save the data as ‘Cars.txt’ or ‘Cars.dat’ in your local directory, and use the following code to implement the same task as the above.

```
DATA sportscars;
INFILE ' (path) '; INPUT Model $ Year Make $
Seats Color $;
IF Year < 1975 THEN Status = 'classic';
IF Model = 'Corvette' OR Model = 'Camaro' THEN Make = 'Chevy';
IF Model = 'Miata' THEN DO;
Make = 'Mazda';
Seats = 2;
END;
PROC PRINT DATA = sportscars;
TITLE “Eddy’s Excellent Emporium of Used Sports Cars”;
RUN;
```

5. Grouping observations with IF-THEN/ELSE statements

One of the most common uses of IF-THEN statements is for grouping observations. IF-THEN/ELSE logic takes this basic form:

```
IF condition THEN action;
ELSE IF condition THEN action;
ELSE IF condition THEN action;
ELSE action;
```

Example Here are data from a survey of home improvements. Each record contains three data values: owner's name, description of the work done, and cost of the improvements in dollars:

```
Bob     kitchen cabinet face-lift 1253.00
Shirley bathroom addition         11350.70
Silvia  paint exterior            .
Al      backyard gazebo           3098.63
Norm    paint interior            647.77
Kathy   second floor addition     75362.93
```

This program reads the raw data from a file called Home.dat and then assigns a grouping variable called CostGroup. This variable has a value of high, medium, low, or missing, depending on the value of Cost:

```
DATA homeimprovements;
INPUT Owner $ 1-7 Description $ 9-33 Cost;
IF Cost = . THEN CostGroup = 'missing';
ELSE IF Cost < 2000 THEN CostGroup = 'low';
ELSE IF Cost < 10000 THEN CostGroup = 'medium';
ELSE CostGroup = 'high';
DATALINES;
Bob     kitchen cabinet face-lift 1253.00
Shirley bathroom addition         11350.70
Silvia  paint exterior            .
Al      backyard gazebo           3098.63
Norm    paint interior            647.77
Kathy   second floor addition     75362.93
;
PROC PRINT DATA = homeimprovements;
TITLE 'Home Improvement Cost Groups';
RUN;
```

Exercise: Reconstruct the above table by changing the threshold 2000 and 10000 to 3000 and 9000, respectively.

6. Subseting your data

Often programmers find that they want to use some of the observations in a data set and exclude the rest. The most common way to do this is with a subsetting IF statement in a DATA step.1 The basic form of a subsetting IF is

```
IF expression THEN DELETE;
```

Example The members of a local amateur playhouse want to choose a Shakespearean comedy for this spring's play. You volunteer to compile a list of titles using an online encyclopedia. For each play your data file contains title, approximate year of first performance, and type of play:

```
A Midsummer Night's Dream 1595 comedy
Comedy of Errors          1590 comedy
Hamlet                     1600 tragedy
Macbeth                    1606 tragedy
Richard III                1594 history
Romeo and Juliet          1596 tragedy
Taming of the Shrew       1593 comedy
Tempest                   1611 romance
```

This program reads the data from a raw data file called Shakespeare.dat and then uses a subsetting IF statement to select only comedies:

```
* Choose only comedies;
DATA comedy;
INPUT Title $ 1-26 Year Type $;
IF Type = 'comedy';
DATALINES;
A Midsummer Night's Dream 1595 comedy
Comedy of Errors          1590 comedy
Hamlet                     1600 tragedy
Macbeth                    1606 tragedy
Richard III                1594 history
Romeo and Juliet          1596 tragedy
Taming of the Shrew       1593 comedy
Tempest                   1611 romance
;
PROC PRINT DATA = comedy;
TITLE 'Shakespearean Comedies';
RUN;
```

Exercise: Substituting for the IF statement by

```
(1) IF Type = 'tragedy' OR Type = 'romance' OR Type = 'history' THEN DELETE;
(2) IF Type = 'tragedy' OR Type = 'romance' THEN DELETE;
(3) IF Type = 'tragedy';
```

7. Working with SAS dates

Dates can be tricky to work with. Some months have 30 days, some 31, some 28, and don't forget leap year. SAS dates simplify all this. A SAS date is a numeric value equal to the number of days since January 1, 1960.1 The table below lists three dates and their values as SAS dates:

Date	**SAS date value**
January 1, 1959	-365
January 1, 1960	0
January 1, 1961	366

SAS has special tools for working with dates: informats for reading dates, functions for manipulating dates, and formats for printing dates.

Informats To read variables that are dates, you use formatted style input. The INPUT statement below tells SAS to read a variable named BirthDate using the MMDDYY10. informat:

```
INPUT BirthDate MMDDYY10.;
```

SAS has a variety of date informats for reading dates in many different forms. All of these informats convert your data to a number equal to the number of days since January 1, 1960.

Setting the default century When SAS sees a date with a two-digit year like 07/04/76, SAS has to decide in which century the year belongs. Is the year 1976, 2076, or perhaps 1776? The system option YEARCUTOFF= specifies the first year of a hundred-year span for SAS to use. The default value for this option is 1920, but you can change this value with the OPTIONS statement. To avoid problems, you may want to specify the YEARCUTOFF= option whenever you have data containing two-digit years. This statement tells SAS to interpret two-digit dates as occurring between 1950 and 2049:

```
OPTIONS YEARCUTOFF = 1950;
```

Dates in SAS expressions Once a variable has been read with a SAS date informat, it can be used in arithmetic expressions like other numeric variables. For example, if a library book is due in three weeks, you could find the due date by adding 21 days to the date it was checked out:

```
DateDue = DateCheck + 21;
```

You can use a date as a constant in a SAS expression by adding quotation marks and a letter D. The assignment statement below creates a variable named EarthDay05, which is equal to the SAS date value for April 22, 2005:

```
EarthDay05 = '22APR2005'D;
```

Example A local library has a data file containing details about library cards. Each record contains three data values—the card holder's name, birthdate, and the date that card was issued:

```
A. Jones     1jan60     9-15-03
M. Rincon    05OCT1949  02-29-2000
Z. Grandage  18mar1988  10-10-2002
K. Kaminaka  29may2001  01-24-2003
```

The program below reads the raw data, and then computes the variable ExpireDate (for expiration date) by adding three years to the variable IssueDate. The variable ExpireQuarter (the quarter the card expires) is computed using the **QTR** function and the variable ExpireDate. Then an IF statement uses a date constant to identify cards issued after January 1, 2003:

```
DATA librarycards;
INPUT Name $11. +1 BirthDate DATE9. +1 IssueDate MMDDYY10.;
ExpireDate = IssueDate + (365.25 * 3);
ExpireQuarter = QTR(ExpireDate);
IF IssueDate > '01JAN2003'D THEN NewCard = 'yes';
DATALINES;
```

```
A. Jones     1jan60     9-15-03
M. Rincon    05OCT1949 02-29-2000
Z. Grandage  18mar1988 10-10-2002
K. Kaminaka  29may2001 01-24-2003
PROC PRINT DATA = librarycards;
FORMAT IssueDate MMDDYY8. ExpireDate WEEKDATE17.;
TITLE 'SAS Dates without and with Formats';
RUN;
```