SAS Data Management

Math 3210
Dr. Zeng
Department of Mathematics
California State University, Bakersfield

Outline

- Creating and Redefining Variables
- Using IF-THEN/ELSE Statement
  -Case I: The Basic Form
  -Case II: More than One Actions
  -Case III: Grouping Observations
  -Case IV: Subsetting Data
- Loops in SAS
  -The DO Statement
  -The ARRAY Statement

Overview of the DATA Step

Figure 2.2 From Raw Data to a SAS Data Set

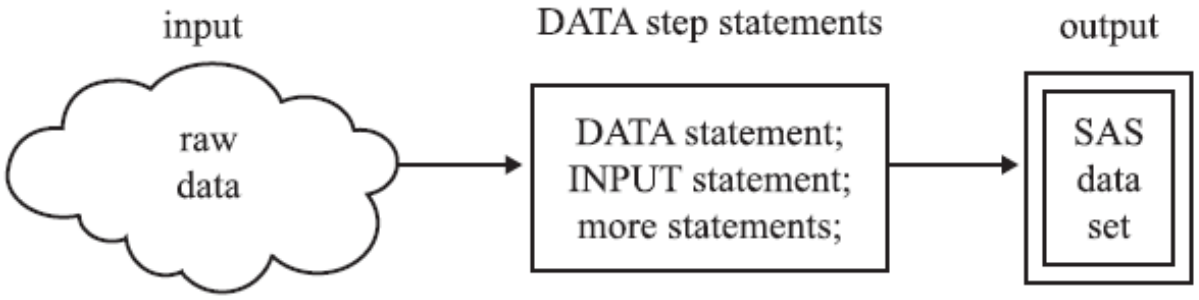
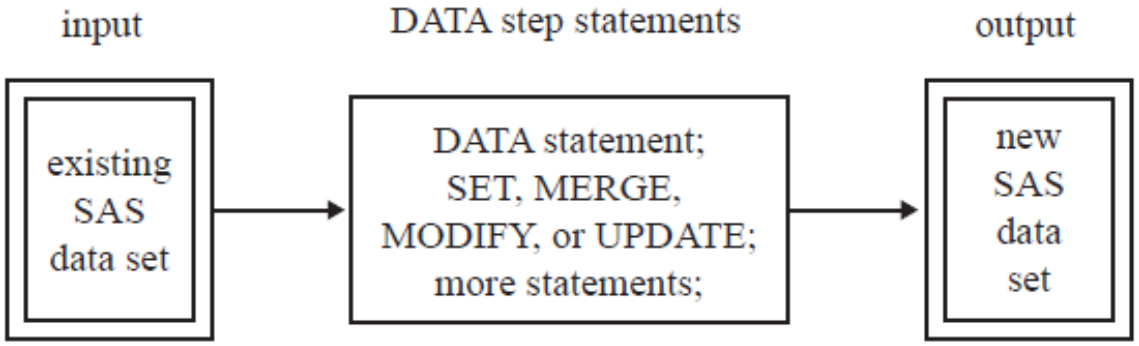


Figure 2.3 Using One SAS Data Set to Create Another



Creating and Redefining Variables

- One of the typical practices in the DATA step is to produce new information from the original information or to change the information read by the INPUT or SET/MMERGE/MODIFY/UPDATE statement.
- The basic method of adding information to a SAS data set is to create a new variable in a DATA step with an assignment statement.
- The basic form of an assignment statement is:

```
variable=expression;
```

Basic Types of Assignment Statements

Type of Expression	Assignment Statement
numeric constant	new=10;
character constant	new='ten';
a variable	new=new_variable;
addition	new=old+10;
subtraction	new=old-10;
multiplication	new=old*10;
division	new=old/10;
exponentiation	new=old**10;

Here is a complete list of SAS functions by category:
http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000245860.htm

Selected SAS Character Functions

Function name	Syntax[1]	Definition
Character		
CAT	CAT(*arg-1,arg-2,...arg-n*)	Concatenates two or more character strings together leaving leading and trailing blanks
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)
PROPCASE	PROPCASE(*arg*)	Converts first character in word to uppercase and remaining characters to lowercase
SUBSTR	SUBSTR(*arg,position,n*)	Extracts a substring from an argument starting at *position* for *n* characters or until end if no *n*[2]
TRIM	TRIM(*arg*)	Removes trailing blanks from character expression
UPCASE	UPCASE(*arg*)	Converts all letters in argument to uppercase

[1] *arg* is short for argument, which means a literal value, variable name, or expression.

Function name	Example	Result	Example	Result
Character				
CAT	`a=' cat';b='dog ';` `x=CAT(a,b);`	`x=' catdog '`	`a='cat ';b=' dog';` `y=CAT(a,b);`	`y='cat  dog'`
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`
PROPCASE	`a='MyCat';` `x=PROPCASE(a);`	`x='Mycat'`	`a='TIGER';` `y=PROPCASE(a);`	`y='Tiger'`
SUBSTR	`a='(916)734-6281';` `x=SUBSTR(a,2,3);`	`x='916'`	`y=SUBSTR('1cat',2);`	`y='cat'`
TRIM	`a='my  '; b='cat';` `x=TRIM(a)||b;`[3]	`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'`

Selected SAS Numeric Functions

Function name	Syntax[1]	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-1,arg-2,...arg-n*)	Largest non-missing value
MEAN	MEAN(*arg-1,arg-2,...arg-n*)	Arithmetic mean of non-missing values
MIN	MIN(*arg-1,arg-2,...arg-n*)	Smallest non-missing value
N	N(*arg-1,arg-2,...arg-n*)	Number of non-missing values
NMISS	NMISS(*arg-1,arg-2,...arg-n*)	Number of missing values
ROUND	ROUND(*arg, round-off-unit*)	Rounds to nearest round-off unit
SUM	SUM(*arg-1,arg-2,...arg-n*)	Sum of non-missing values
Date		
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
WEEKDAY	WEEKDAY(date)	Returns day of week (1=Sunday) from SAS date value
YEAR	YEAR(date)	Returns year from a SAS date value

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`
N	`x=N(1,.,7,2);`	`x=3`	`y=N(.,4,.,.);`	`y=1`
NMISS	`x=NMISS(1,.,7,2);`	`x=1`	`y=NMISS(.,4,.,.);`	`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`
Date				
DAY	`a=MDY(4,18,2008);` `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(4,18,2008);` `x=MONTH(a);`	`x=4`	`a=MDY(9,3,60);` `y=MONTH(a);`	`y=9`
QTR	`a=MDY(4,18,2008);` `x=QTR(a);`	`x=2`	`a=MDY(9,3,60);` `y=QTR(a);`	`y=3`
TODAY	`x=TODAY();`	`x=today's date`	`y=TODAY()-1;`	`y=yesterday's date`
WEEKDAY	`a=MDY(4,13,2008);` `x=WEEKDAY(a);`	`x=1`	`a=MDY(4,18,2008);` `y=WEEKDAY(a);`	`y=6`
YEAR	`a=MDY(4,13,2008);` `x=YEAR(a);`	`x=2008`	`a=MDY(1,1,1960);` `v=YEAR(a);`	`y=1960`

[1] *arg* is short for argument, which means a literal value, variable name, or expression.
[2] A SAS date value is the number of days since January 1, 1960.

Example 1

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

```
DATA homegarden;
infile '/home/bzeng/my_content/Garden.dat';
INPUT Name $ 1-7 Tomato Zucchini Peas Grapes;
Zone = 14;
Type = 'home';
Zucchini = Zucchini * 10;
Total = Tomato + Zucchini + Peas + Grapes;
All=sum(Tomato, Zucchini, Peas, Grapes); /* non-missing values only*/
Average=mean(Tomato, Zucchini, Peas, Grapes); /*non-missing*/
PerTom = (Tomato / Total) * 100;
PerTomato=(Tomato / All) * 100;
RUN;
```

Home Gardening Survey

Obs	Name	Tomato	Zucchini	Peas	Grapes	Zone	Type	Total	All	Average	PerTom	PerTomato
1	Gregor	10	20	40	0	14	home	70	70	17.500	14.2857	14.2857
2	Molly	15	50	10	1000	14	home	1075	1075	268.750	1.3953	1.3953
3	Luther	50	100	15	50	14	home	215	215	53.750	23.2558	23.2558
4	Susan	20	0	.	20	14	home	.	40	13.333	.	50.0000

Example 2

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 10-28-2012 7.8 6.5 7.2 8.0 7.9
Matthew Lee       9 D 10-30-2012 6.5 5.9 6.8 6.0 8.1
Elizabeth Garcia 10 C 10-29-2012 8.9 7.9 8.5 9.0 8.8
Lori Newcombe     6 D 10-30-2012 6.7 5.6 4.9 5.2 6.1
Jose Martinez     7 d 10-31-2012 8.9 9.510.0 9.7 9.0
Brian Williams   11 C 10-29-2012 7.8 8.4 8.5 7.9 8.0
```

```
DATA contest;
  infile '/home/bzeng/my_content/Pumpkin.dat';
  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);
RUN;
PROC PRINT DATA = contest;
  TITLE 'Pumpkin Carving Contest';
RUN;
```

Pumpkin Carving Contest

Obs	Name	Age	Type	Date	Scr1	Scr2	Scr3	Scr4	Scr5	AvgScore	DayEntered
1	Alicia Grossman	13	C	19294	7.8	6.5	7.2	8.0	7.9	7.48	28
2	Matthew Lee	9	D	19296	6.5	5.9	6.8	6.0	8.1	6.66	30
3	Elizabeth Garcia	10	C	19295	8.9	7.9	8.5	9.0	8.8	8.62	29
4	Lori Newcombe	6	D	19296	6.7	5.6	4.9	5.2	6.1	5.70	30
5	Jose Martinez	7	D	19297	8.9	9.5	10.0	9.7	9.0	9.42	31
6	Brian Williams	11	C	19295	7.8	8.4	8.5	7.9	8.0	8.12	29

Conditional Execution

- Often you need to add information to some observations but not to others.
- IF-THEN/ELSE statements can cause assignment statements to be carried out only when condition(s) is/are met.
- This section will introduce a few types of commonly seen conditional execution.

Case I: The Basic Form

- The basic form is

IF condition THEN action;

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

IF condition AND condition THEN action;

- The *condition* is one or more comparisons, for example,
  - City = 'Rome'
  - NumberOfEvents > Nights
  - TourGuide = 'Lucas' and Nights > 7
  - Model in ('Model T', 'Model A')
- The most common executable statements are assignment statements, such as
  - LandCost = LandCost + 30;
  - Calendar = 'Check schedule';
  - TourGuide = 'Torres';

SAS Comparison Operators

Symbol	Mnemonic Equivalent	Definition	Example
=	EQ	equal to	a=3
^=	NE	not equal to	a ne 3
¬=	NE	not equal to	
~=	NE	not equal to	
>	GT	greater than	num>5
<	LT	less than	num<8
>=	GE	greater than or equal to	sales>=300
<=	LE	less than or equal to	sales<=100
	IN	equal to one of a list	num in (3, 4, 5)

Using Complex Comparisons That Require AND and OR

A condition can contain both AND operators and OR operators. When it does, SAS evaluates the AND operators before the OR operators. SAS evaluates the comparisons within parentheses first and uses the results as the terms of the larger comparison. For example,

If (season='summer' or season='fall') and (grape > 2000 or apple > 4000) then Level='High';

Case II: More than One 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;
```

SAS Logical Operators

The logical Operators AND and OR may be symbolic or mnemonic:

SAS Logical (Boolean) Operators		
Symbol	**Mnemonic**	**Definition**
&	AND	Logical AND
|	OR	Logical OR
^	NOT	Logical NOT

For example,

IF Season='Spring' and Temperature <70 THEN Type='A';
IF Season='Summer' & Temperature GT 110 THEN Type='B';

Case II: Example

The following data show information about rare antique cars sold at auction. The data values are the make, model, the year the car was made, the number of seats, and the selling price in millions of dollars.

DeDion	LaMarquise	1884	4	4.6
Rolls-Royce	Silver Ghost	1912	4	1.7
Mercedes-Benz	SSK	1929	2	7.4
	F-88	1954	.	3.2
Ferrari	250 Testa Rossa	1957	2	16.3

Case II: Example

```
DATA oldcars;
  INFILE '/home/bzeng/my_content/Auction.dat';
  INPUT Make $ 1-13 Model $ 15-29 YearMade Seats MillionsPaid;
  IF YearMade < 1890 THEN Veteran = 'Yes';
  IF Model = 'F-88' THEN DO;
     Make = 'Oldsmobile';
     Seats = 2;
  END;
RUN;
PROC PRINT DATA = oldcars;
  TITLE 'Cars Sold at Auction';
RUN;
```

Cars Sold at Auction

Obs	Make	Model	YearMade	Seats	MillionsPaid	Veteran
1	DeDion	LaMarquise	1884	4	4.6	Yes
2	Rolls-Royce	Silver Ghost	1912	4	1.7	
3	Mercedes-Benz	SSK	1929	2	7.4	
4	Oldsmobile	F-88	1954	2	3.2	
5	Ferrari	250 Testa Rossa	1957	2	16.3	

Case III: Grouping Observations

- A series of IF-THEN/ELSE statements can be used for grouping observations.
- The ELSE keyword tells SAS that these IF statements are all related.
- The ELSE logic ensures that your groups are ***mutually exclusive*** so you don't accidently have an observation fitting into more than one group.
- The basic form is

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

- Alternatively, you can skip the condition part only in the last ELSE statement. An ELSE of this kind becomes a default which is automatically executed for all observations failing to satisfy any of the previous IF statements. For example,

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

Case III: Example

Here are data from ma 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
```

Case III: Example

```
* Group observations by cost;
DATA homeimprovements;
  INFILE '/home/bzeng/my_content/Home.dat';
  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';
RUN;
```

Home Improvement Cost Groups

Obs	Owner	Description	Cost	CostGroup
1	Bob	kitchen cabinet face-lift	1253.00	low
2	Shirley	bathroom addition	11350.70	high
3	Silvia	paint exterior	.	missing
4	Al	backyard gazebo	3098.63	medium
5	Norm	paint interior	647.77	low
6	Kathy	second floor addition	75362.93	high

Case IV: Subsetting Data

- Sometimes, you just need a subset of the observations instead of the entire dataset.
- The IF statement can be used to subset your data.
- The basic form is:

```
IF expression;
```

- For example,

```
IF gender='female';
```

This means to only include the female data.

- Alternatively, you can use

```
IF expression THEN DELETE;
```

In this example, it is equivalent to

```
IF gender='male' THEN DELETE;
```

Case IV: 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
```

Case IV: Example

```
* Choose only comedies;
DATA comedy;
   INFILE '/home/bzeng/my_content/Shakespeare.dat';
   INPUT Title $ 1-26 Year Type $;
   IF Type = 'comedy';
RUN;
PROC PRINT DATA = comedy;
   TITLE 'Shakespearean Comedies';
RUN;
```

Shakespearean Comedies

Obs	Title	Year	Type
1	A Midsummer Night's Dream	1595	comedy
2	Comedy of Errors	1590	comedy
3	Taming of the Shrew	1593	comedy

Note

Since there are four types of play (tragedy, romance, history and comedy). Equivalently, you can also replace

IF Type = 'comedy';

by the following statement

IF Type=‘tragedy’ OR Type=‘Romance’ OR Type =‘history’ THEN DELETE;

Loops in SAS

The Basic DO statement:

To tell SAS to repeat the same action several times, use an iterative DO statement in the DATA step. For example,

```
data A;
do i = 1 to 5;
  y = i**2; /* values are 1, 4, 9, 16, 25 */
  output;
end;
run;
```

Obs	i	y
1	1	1
2	2	4
3	3	9
4	4	16
5	5	25

```
data B;
do i = 1 to 5 by 0.5;
  y = i**2; /* values are 1, 2.25, 4, ..., 16, 20.25, 25 */
  output;
end;
run;
```

Obs	i	y
1	1.0	1.00
2	1.5	2.25
3	2.0	4.00
4	2.5	6.25
5	3.0	9.00
6	3.5	12.25
7	4.0	16.00
8	4.5	20.25
9	5.0	25.00

Performing the Same Action for a Series of Variables

Sometimes, you need to perform the same action for a series of variables. For example, you might want to change every occurrence of zero to a missing value or making mathematical transformation to numerical variables. You have the following two methods:

- Use multiple IF statements in the DATA step
- Use ARRAY statement in the DATA step

Simplifying Programs with ARRAYS

The ARRAY statement has the following general form:

ARRAY name (n) $ variable-list;

- name is a name you give to the array
- n is the number of variables in the array
- $ is needed if the variables are character
- The rules for naming arrays are the same as those for naming variables
- For example,

ARRAY seasons (4) $ Spring Summer Fall Winter;

- Provide the array name and the subscript for that variable to reference a variable using the array name. For example, seasons(1) is Spring and seasons(4) is Winter.

Example

The radio station KBRK is conducting a survey asking people to rate five different songs. Songs are rated on a scale of 1 to 5, where 1 equals change the station when it comes on, and 5 equals turn up the volume when it comes on. If listeners had not heard the song or didn't care to comment on it, a 9 was entered for that song. The following are the data collected:

Albany	54 3 9 4 4 9
Richmond	33 2 9 3 3 3
Oakland	27 3 9 4 2 3
Richmond	41 3 5 4 5 5
Berkeley	18 4 4 9 3 2

Method 1: Use Multiple IF Statements

```
DATA songs;
  INFILE '/home/bzeng/my_content/KBRK.dat';
  INPUT City $ 1-15 Age wj kt tr filp ttr;
 if wj=9 then wj=.;
 if kt=9 then kt=.;
 if tr=9 then tr=.;
 if filp=9 then filp=.;
 if ttr=9 then ttr=.;
RUN;
PROC PRINT DATA = songs;
  TITLE 'KBRK Song Survey';
RUN;
```

KBRK Song Survey

Obs	City	Age	wj	kt	tr	filp	ttr	i
1	Albany	54	3	.	4	4	.	6
2	Richmond	33	2	.	3	3	3	6
3	Oakland	27	3	.	4	2	3	6
4	Richmond	41	3	5	4	5	5	6
5	Berkeley	18	4	4	.	3	2	6

Method 2: Use ARRAY Statement

- Change all 9s to missing values;

```
DATA songs;
  INFILE '/home/bzeng/my_content/KBRK.dat';
  INPUT City $ 1-15 Age wj kt tr filp ttr;
  ARRAY song (5) wj kt tr filp ttr;
  DO i = 1 TO 5;
    IF song(i) = 9 THEN song(i) = .;
  END;
RUN;
PROC PRINT DATA = songs;
  TITLE 'KBRK Song Survey';
RUN;
```

KBRK Song Survey

Obs	City	Age	wj	kt	tr	filp	ttr	i
1	Albany	54	3	.	4	4	.	6
2	Richmond	33	2	.	3	3	3	6
3	Oakland	27	3	.	4	2	3	6
4	Richmond	41	3	5	4	5	5	6
5	Berkeley	18	4	4	.	3	2	6

Using Shortcuts for Lists of Variable Names

- When you need to write a very long list of variable names, sometimes it is easier to use a shortcut.
- You can use an abbreviated list of variable names almost anywhere you can use a regular variable list.
- In functions, abbreviated lists must be preceded by the keyword OF (for example, SUM(OF Cat1-Cat12)). Otherwise, you simply replace the regular list with the abbreviated one.

Numbered range lists

Variables which start with the same characters and end with consecutive numbers can be part of a numbered range list. The numbers can start and end anywhere as long as the number sequence between is complete. For example, the following INPUT statement shows a variable and its abbreviated form:

Variable list	**Abbreviated list**
INPUT Song1 Song2 Song3 Song4 Song5;	INPUT Song1-Song5;
INPUT Cat8 Cat9 Cat10 Cat11 Cat12;	INPUT Cat8-Cat12;
Total= sum(Song1, Song2, Song3);	Total=sum(OF Song1-Song3);

Name Prefix lists

Variables which start with the same characters can be part of a name prefix list, and can be used in some SAS statements and functions. For example:

Variable list	**Abbreviated list**
DogBills=sum(DogVet, DogFood, Dog_Care);	DogBills=sum(OF Dog:);

Name Range lists

Name range lists depend on the internal order, or position, of the variables in the SAS data set. This is determined by the order of appearance of the variables in the DATA step. For example, given the following DATA step, the internal variable order would be Y A C H R B:

```
DATA example;
INPUT y a c h r;
Run;
```

To specify a name range list, put the first variable, then two hyphens, then the last variable. For example,

Variable list	**Abbreviated list**
sum(y,a,c,h,r);	sum (OF y--r)

Example

The radio station KBRK wants to modify the program from the previous section, which changes all 9s to missing values. Now, instead of changing the original variable, they create new variables (Song1 through Song5) which will have the new missing values. We also need to computes the average score using the MEAN function.

```
DATA songs;
  INFILE '/home/bzeng/my_content/KBRK.dat';
  INPUT City $ 1-15 Age wj kt  tr filp ttr;
  ARRAY new (5) Song1 - Song5;
  ARRAY old (5) wj -- ttr;
  DO i = 1 TO 5;
    IF old(i) = 9 THEN new(i) = .;
      ELSE new(i) = old(i);
  END;
  AvgScore = MEAN(OF Song1 - Song5);
PROC PRINT DATA = songs;
  TITLE 'KBRK Song Survey';
RUN;
```

KBRK Song Survey

Obs	City	Age	wj	kt	tr	filp	ttr	Song1	Song2	Song3	Song4	Song5	i	AvgScore
1	Albany	54	3	9	4	4	9	3	.	4	4	.	6	3.66667
2	Richmond	33	2	9	3	3	3	2	.	3	3	3	6	2.75000
3	Oakland	27	3	9	4	2	3	3	.	4	2	3	6	3.00000
4	Richmond	41	3	5	4	5	5	3	5	4	5	5	6	4.40000
5	Berkeley	18	4	4	9	3	2	4	4	.	3	2	6	3.25000