

Introduction to Data Management and Programming in SAS

Harvard School of Public Health

Text book

The Little SAS Book (Second Edition), Delwiche LD and Slaughter SJ, SAS Institute, Inc., 1998. Available at the Harvard Coop. Identified as LSB.

Reference

SAS User's Guide: Basics and *SAS User's Guide: Statistics*.
SAS System Help (online)

General Information (LSB 1.1-1.10, 3.11, 8.1-8.2)

What is as SAS data set	1
What is as SAS program.....	1
SAS names	3
SAS variables	3
Lists of variables	3
SAS statements	3
Coding style	4
SAS Windowing Environment	4

Creating SAS data sets (LSB 2.1-2.7, 2.12-2.16, 8.3-8.6, 8.13-8.14)

Begin the data step: DATA.....	10
Where's the data: DATALINES, INFILE	10
Describe the 'raw' data to SAS: INPUT	
List, column, formatted and mixed input statements	11
INFORMAT (special formats)	11
Delimited files	12
Pointer commands: @, +n.....	15
Conditional input	18
Multiple records/observation, multiple observations/record.....	20
Options on INFILE: MISOVER, TRUNCOVER, LRECL.....	21

Accessing external raw data sets and SAS data sets (LSB 2.8-2.10)

External raw data sets: FILENAME and <i>fileref</i>	26
SAS libraries	27
Temporary vs. permanent SAS data sets	28
Create a temporary SAS data set	28
Create a permanent SAS data set: LIBNAME and <i>libref</i>	30
Using a permanently stored SAS data set	31
Another way to create and later use a permanent SAS data set	32
See what's in the SAS data set: PROC CONTENTS, Explorer/Libraries	34
Input from other programs: Stat/Transfer™	36

Create new variables and change existing variables (LSB 3.1-3.3, 3.7-3.8, 8.7-8.9)

Assignment statements	37
Create new variables	37
Change existing variables	38
SAS functions	39
Date functions	40
Year 2K	41
Date difficulties	42
Useful SAS functions	43

Conditional processing (LSB 3.4-3.6)

Conditional assignment: IF-THEN, IF-THEN-DO-END	44
Multiple choice processing: IF-THEN-ELSE	46
Subsetting IF	47

Arrays (LSB 3.10)

Define arrays: ARRAY	48
Process arrays: DO...END	48
Create observations	53

Sorting and printing data with PROCs (LSB 4.1-4.7)

Basics: PROC PRINT, PROC SORT	55
Dress up the output: LABEL, TITLE, FOOTNOTE, FORMAT	57
Create you own permanent format library	59

Simple data summarization (LSB 4.9-4.11, 4.14, 7.2-7.3)

Summarize categorical data: PROC FREQ	61
Summarize continuous data: PROC MEANS.....	64
Display relationships between variables: PROC PLOT, PROC GPLOT	68

Data management I - SET (LSB 5.1-5.3)

Basics	71
Concatenate 2 SAS data sets	72
Interleave 2 SAS data sets: BY	72
Create a subset: IF	73

Data management II - MERGE (LSB 5.4-5.7, 5.9-5.10, 5.14)

Basics	74
One-to-one join	75
One-to-many join	76
Join with IN= option.....	77
"Many-to-one" join (doesn't use MERGE)	79

Data management III - OUTPUT (LSB 5.11)80**Data management IV - Change unit of observation (LSB 3.9-3.10, 5.12-5.14)**

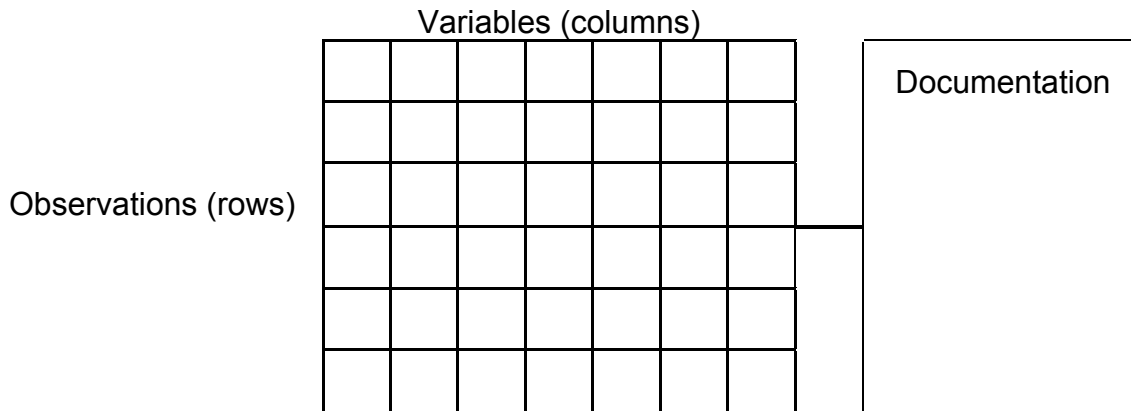
Make several observations from one	81
Make one observation from several	82

Data analysis - PROC step (LSB 7.1-7.8, SAS User's Guide: Statistics)

PROC UNIVARIATE	83
PROC TTEST	85
PROC NPAR1WAY	86
PROC REG.....	87
PROC ANOVA.....	89
PROC GLM.....	91
PROC LOGIST	92
PROC LIFETEST	94

GENERAL INFORMATION (LSB 1.1-1.10, 3.11, 8.1-8.2)

What is a SAS data set? a rectangular array of observations and variables with documentation.



What is a SAS program? a series of SAS statements, each terminated by a semicolon (;), executed *in order* on *each observation* (an *implied* loop with *implicit* output).

Components:

- OPTIONS statements
- Data set definition (LIBNAME and FILENAME) statements
- DATA step(s)
- PROC step(s)
- Comments
- Run statements

OPTIONS:

- begins with keyword OPTIONS
- governs the appearance of output

Data set definition:

- begins with the keyword LIBNAME or FILENAME
- tells SAS where to find the data and where to write data

DATA step:

- begins with the keyword DATA
- accesses 'raw' data or existing SAS data set(s)
- modifies accessed data
- creates a new SAS data set(s)

PROC step:

- begins with the keyword PROC
- takes an action or performs an analysis
- frequently generates a report

Comments:

- annotate and document SAS programs

Run:

- execute preceding lines in DATA or PROC step

Example of a SAS program:

```
* example of a SAS program;
options nocenter;
data sasclass;
input name $ height;
datalines;
roland 180
rhonda 172
;
run;

proc print;
run;
```

SAS LOG

```
1  * example of a SAS program;
2  options nocenter;
3  data sasclass;
4  input name $ height;
5  datalines;

NOTE: The data set WORK.SASCLASS has 2 observations and 2 variables.
NOTE: DATA statement used:
      real time           0.35 seconds
      cpu time            0.13 seconds

8  ;
9  run;
10
11 proc print;
12 run;

NOTE: There were 2 observations read from the dataset WORK.SASCLASS.
NOTE: PROCEDURE PRINT used:
      real time           0.19 seconds
      cpu time            0.07 seconds
```

SAS OUTPUT

```
The SAS System

Obs      name      height
1        roland    180
2        rhonda    172
```

SAS names: You, the SAS programmer, get to name variables, arrays and data sets but there are a few rules.

Names consist of 32 or fewer letters, numbers or underscores (_) EXCEPT dataset (member) names are ≤ 28 characters and librefs, filerefs, and formats are ≤ 8 characters

Names must begin with a letter or an underscore

Names may be lower or upper case (SAS doesn't recognize that upper and lower case are different)

Examples:

SAS variables: variables are either character or numeric.

TYPE	WHAT	LENGTH	MISSING VALUE
character	everything	1-32767	blank
numeric	0, 1, 2...9 -, +, decimal point scientific notation (E)	1-32	. (dot)

Lists of variables: (LSB 3.11)

regular list: name age sex wt bp1 bp2 bp3 bp4 bp5
'type' list: _numeric_, _character_
single dash list: bp1-bp5
double dash list: name--bp5

SAS statements: the building blocks of the SAS program. A few rules here, too.

Statements (except assignment statements) begin with a SAS keyword

Every statement ends with a semicolon (;)

Words in statements are separated by one or more spaces or by operators such as +, -, *, /, <, <=, >, >=, =, ^=, &, |, !, le, ge, eq, ne, and, or

Statements may be lower or upper case

Statements may continue over multiple lines

Statements may start in any column

Most statements are unique to the DATA step (SET, MERGE, IF) or to the PROC step (VAR, TABLES, MODEL). A few appear in both (BY, LABEL, FORMAT, TITLE, RUN, OPTIONS)

Coding style: Develop a style that works for you. The idea is have a program that you and people you work with can read and debug easily. Some suggestions...

Begin each SAS statement on a new line (but, I don't always...)

When the statement extends beyond a single line, indent succeeding lines

Separate tasks in the program with blank lines

Use comments: `* comment;` or `/* comment */`

Complete each DATA and PROC step with a RUN statement

SAS Windowing Environment: Consists of five windows.

PROGRAM EDITOR: enter and modify SAS programs and data
execute programs

LOG: SAS's annotation of programs after execution (black)
has NOTES about data set names, number of observations and
variables, execution time (blue)
displays WARNINGS (green)
displays ERRORS (red)

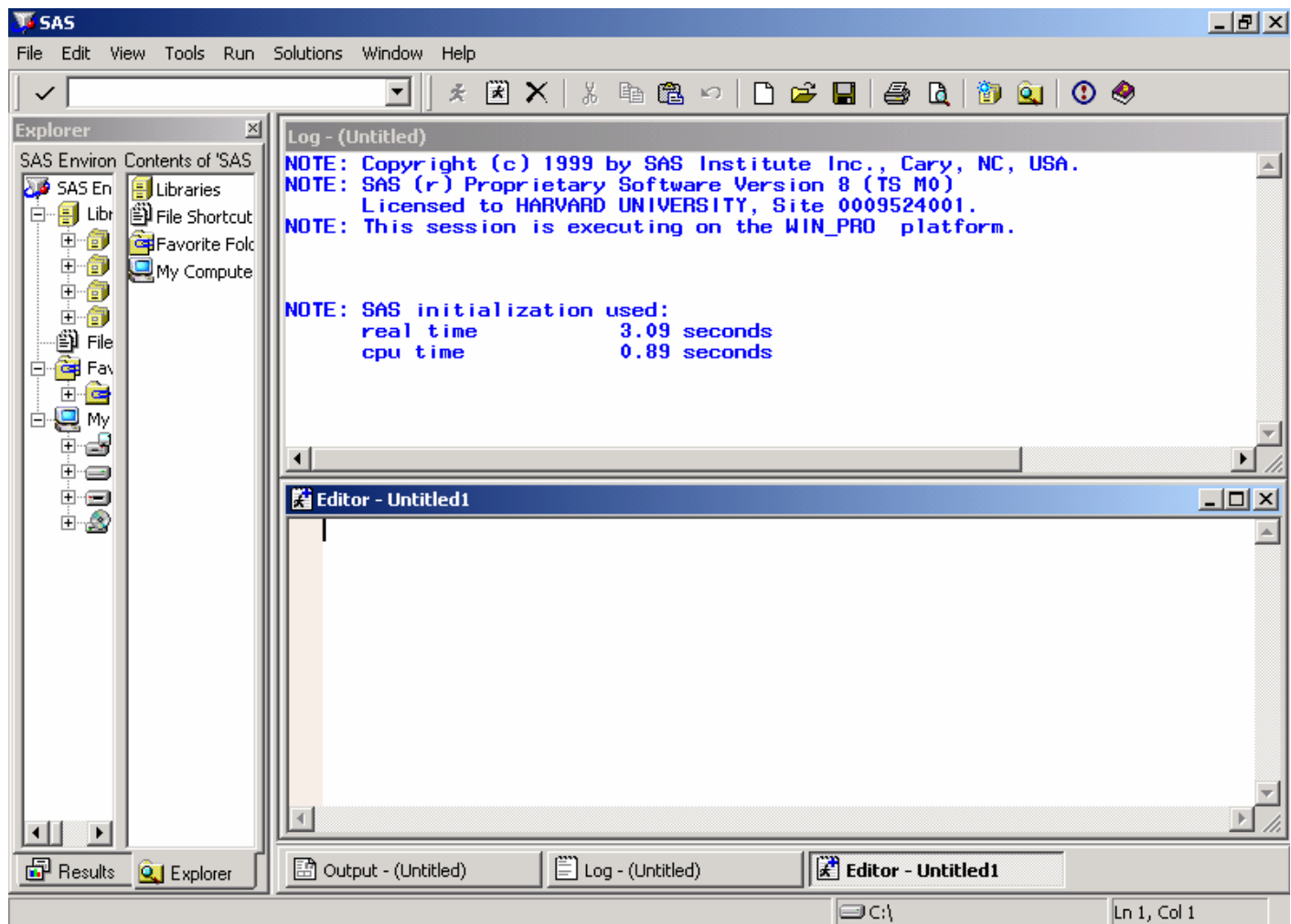
OUTPUT: Results of PROCedures

RESULTS: Table of contents/index of your output

EXPLORER: Access to data libraries and files

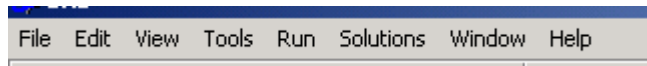
1. Start SAS

Click Start, Statistics and SAS8 to open the SAS Windowing Environment. Three windows (Editor, Log, Explorer) are visible when you start. Output and Results may be accessed by clicking the appropriate tab at the bottom of the screen.



The title bar of the active window is blue while the others are gray. Click anywhere in a window to make it the active window.

2. The SAS menu bar



<u>F</u> ile	open, save and print files and exit SAS
<u>E</u> dit	editing commands such as cut, copy and paste, and search/replace
<u>V</u> iew	select the active window
<u>T</u> ools	various editors and SAS options/preferences and options/'keys' settings
<u>R</u> un	submit and recall SAS programs
<u>S</u> olutions	some point-and-click programs (may not be available)
<u>W</u> indow	select tiled or cascaded windows, select active window
<u>H</u> elp	access SAS on-line help

Notice the Run item on the menu bar disappears when the editor window is NOT the active window.

3. The SAS tool bar (command bar and various buttons)



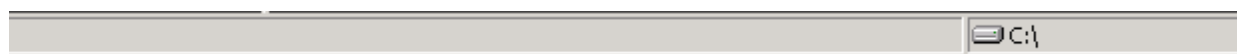
<u>C</u> ommand	submit short commands to SAS
<u>b</u> uttons	shortcuts to actions you may also access from the menu bar. Move the cursor over a button and a short description appears. The button at the right end, for example, accesses SAS help.

4. The SAS window bar




Click on a tab to see a hidden window and make it active.

5. The SAS status bar



The status bar is at the bottom of the screen. Error messages will appear in the area on the left. The area on the right displays the current working folder (directory). You may change the working folder by double-clicking this area and navigating through folders in the usual Windows way (we will demonstrate this in class).

6. Enter and save a SAS program

Make the p:\bio113 your current working folder and make the editor window active (click anywhere in the window). Click on the  in the upper right corner of the window to make it fill the screen. Type in the following program. The editor is a primitive word processor—

```
* This is my first SAS program;
*   Input: data in the program;
*   Output: listing of the data;
*   Author: Sadie Sas;

options nocenter;


data one;
input name $ height;
datalines;
roland 180
ronda 172
;
run;

proc print;
title1 'Sadie Sas';
title2 'My first SAS program';
run;
```

arrow keys move you around, the End key moves you to the end of a line, Home moves you to the beginning of a line, PageUp and PageDown move you up or down one page, Enter moves you to the beginning of a new line. You can block areas of the program and cut and paste in the usual way. Pay close attention to spelling and punctuation as you type the program. If you misspell a SAS key word, it will be colored red until you correct it. Notice that comments are green, key words are blue, data in the program are highlighted in yellow, title text is purple, and everything else is black. In addition, rules appear to separate parts of the program (data and proc steps). It's a very good idea to save your program before you submit. It can be recalled from memory during this SAS session but, should you exit SAS before saving it, it will be lost forever. Click File and Save and give the file the name *first.sas*. Because your working folder is p:\bio113, that is where the file will be saved.

By the way, you could click on the save button  and fill in the dialogue to save the file.

6. Submit the SAS program for processing

Click Window and Tile (vertical or horizontal) so that the editor, log and output windows are visible and then on the little man  in the tool bar. Text will whiz by in the LOG window and, if the program runs successfully, output will appear in the OUTPUT window.

7. Examine the LOG window first

Make the LOG window active and click ☐ to make the window large.

```
23  * This is my first SAS program;
24  *   Input: data in the program;
25  *   Output: listing of the data;
26  *   Author: Sadie Sas;
27
28  options nocenter;
29
30  data one;
31  input name $ height;
32  datalines;

NOTE: The data set WORK.ONE has 2 observations and 2 variables.
NOTE: DATA statement used:
      real time           0.02 seconds
      cpu time            0.02 seconds

35  ;
36  run;
37
38  proc print;
39  title1 'Sadie Sas';
40  title2 'My first SAS program';
41  run;

NOTE: There were 2 observations read from the dataset WORK.ONE.
NOTE: PROCEDURE PRINT used:
      real time           0.02 seconds
      cpu time            0.01 seconds
```

Your SAS program and messages from SAS are displayed in different colors (you'll see the colors in class).

Black	the program you submitted
Blue	SAS notes--number of observations and processing time
Green	warnings
Red	errors

Green and red messages suggest that you should examine your program carefully for *program* and/or *logical* errors. We'll review recalling, correcting and resubmitting problem programs in class.

8. Examine the OUTPUT window


Make the output window active (click Output in the window bar) and click ☐ to make it large.




Sadie Sas			16:00 Thursday, C		
My first SAS program					
Obs	name	height			
1	roland	180			
2	ronda	172			

Scroll through the output with PageUp and PageDown or by moving the bar on the right side of the window.

9. Print the contents of any window

Make the window you want to print active. Click File, Print and OK or click  (the printer button) on the tool bar.

10. Save the contents of the LOG and OUTPUT windows

Make the window you want to save active, click File and Save as or click on  (the save button) and fill in the dialogue. Name the LOG file *first.log* and the OUTPUT *first.lst*. Files saved in this way may be imported into your favorite word processing program for formatting and editing (notice that you cannot edit in the LOG or OUTPUT windows). You may also use the usual copy and paste buttons or key strokes to copy LOG or OUTPUT material to a word processing document.

CREATING SAS DATA SETS (LSB 2.1-2.7, 2.12-2.16, 8.3-8.6, 8.13-8.14)

Begin the DATA step: DATA

Start with the keyword, DATA

Follow with your chosen data set name (if you don't, SAS will assign the name *data1* to the first data set, *data2* to the second, etc.)

End the statement with ;

Example: data;
 data whatever;

Where's the data? DATALINES and INFILE statements tell SAS where to find 'raw' data

DATALINES: The data are in the program

INFILE: The data are in an *external* file

Example:

<pre>* datalines example; data one; input a b; datalines; 11 35 95 7 ; run; proc print; run;</pre>	<pre>* infile example; data one; infile 'ex01.dat'; *infile 'p:\bio113\ex01.dat'; *infile 'g:\shared\bio113\ex1.dat'; input a b; run; proc print; run;</pre>	<p>external file (ex01.dat)</p> <pre>11 35 95 7</pre>
--	--	--

Result (both programs produce this output):

The SAS System	15:15 Sunday, August 15, 2004	1									
<table border="0" style="width: 100%;"> <tr> <td style="text-align: left;">Obs</td> <td style="text-align: center;">a</td> <td style="text-align: center;">b</td> </tr> <tr> <td style="text-align: left;">1</td> <td style="text-align: center;">11</td> <td style="text-align: center;">35</td> </tr> <tr> <td style="text-align: left;">2</td> <td style="text-align: center;">95</td> <td style="text-align: center;">7</td> </tr> </table>			Obs	a	b	1	11	35	2	95	7
Obs	a	b									
1	11	35									
2	95	7									

Describe the 'raw' data to SAS: The input statement begins with the SAS keyword **INPUT** and ends with a semicolon (;).

LIST input may be used if:

1. Data values are separated by one or more spaces or by a delimiter such as a tab or comma
2. Character data contain no imbedded spaces and are 8 or fewer characters
3. Missing data values are indicated by a dot (.)
4. There's nothing funny like a date (if so, use an **INFORMAT** statement, too)
5. You plan to read **every** variable on the data line

How? List the names you have chosen for the variables
 Indicate character variables by following their names with \$

Example program:
 (♦ in data lines indicate blanks)

```
data one;
infile 'p:\bio113\ex02.dat';
input name $ age;
run;
proc print;
run;
```

Data:

sam♦42

Result:

Obs	name	age
1	sam	42

```
data one;
infile 'p:\bio113\ex03.dat';
input id a b c;
run;
proc print;
run;
```

Data:

21♦1♦123♦4456

Result:

Obs	id	a	b	c
1	21	1	123	4456

```
data one;
infile 'p:\bio113\ex04.dat';
input mrn bp1-bp5;
run;
proc print;
run;
```

Data:

90043♦58♦67♦98♦72♦71

Result:

Obs	mrn	bp1	bp2	bp3	bp4	bp5
1	90043	58	67	98	72	71

```
data one;
infile 'p:\bio113\ex05.dat';
informat dob mmddyy10.;
input name $ dob;
run;
proc print;
run;
```

Data:

sam♦01/16/1960
william♦01/18/1960

Result:

Obs	dob	name
1	15	sam
2	17	william

```
data one;
infile 'p:\bio113\ex05.dat';
informat name $8.
      dob mmddyy10.;
input name dob;
run;
proc print;
run;
```

Result:

Obs	name	dob
1	sam	15
2	william	17

```
data one;
infile 'p:\bio113\ex05.dat';
input name $ dob : mmddyy10.;
run;
proc print;
format dob date9.;
run;
```

Result:

Obs	name	dob
1	sam	16JAN1960
2	william	18JAN1960

```
data one;
infile 'p:\bio113\ex06.dat' dlm=',';
input x y z;
run;
proc print;
run;
```

Data:

2,5,41
11,2,54

Result:

Obs	x	y	z
1	2	5	41
2	11	2	54

COLUMN input may be used if:

1. Data values for each variable are always in the **same columns** in all the data lines in the raw data set
2. There's nothing funny like a date (INFORMAT **may not** be used with column input)

How? List the names you have chosen for the variables
 Follow each name with the column location of the variable
 Indicate character variables with \$ before the column location

Example program:

(♦ in data lines indicate blanks)

```
data one;
infile 'p:\bio113\ex07.dat';
input name $1-10 age 11-12;
run;
proc print;
run;
```

Data:

```
sam♦♦♦♦♦♦♦♦42
alexander♦12
```

Result:

Obs	name	age
1	sam	42
2	alexander	12

```
data one;
infile 'p:\bio113\ex08.dat';
input id 1-2 a 3 b 4-6
      c 8-11;
run;
proc print;
run;
```

Data:

```
311123♦4456
```

Result:

Obs	id	a	b	c
1	31	1	123	4456


```
data one;
infile 'p:\bio113\ex09.dat';
input mrn 1-5 bp1 9-10
      bp2 11-12 bp3 13-14
      bp4 15-16 bp5 17-18;
run;
proc print;
run;
```

Data:

90043♦♦♦5867987271

Result:

Obs	mrn	bp1	bp2	bp3	bp4	bp5
1	90043	58	67	98	72	71

```
data one;
infile 'p:\bio113\ex10.dat';
input name $1-3 bm 4-5
      bd 8-9 by 12-15;
dob=mdy(bm,bd,by);
run;
proc print;
run;
```

Data:

pam01♦♦19♦♦1960

Result:

Obs	name	bm	bd	by	dob
1	pam	1	19	1960	18

FORMATTED input may be used if:

1. Data values for each variable are always in the same columns in all the data lines in the raw data set
2. There *are* funny variables like dates
3. You want to reuse formatting instructions

How? List the names you have chosen for the variables
 Follow each name with the format instruction for the variable (note the
 '.' at the end of the format instruction)
 Indicate character variables with \$ before the format instruction
 Use pointer instructions (@n and +n) to move to correct columns

Example program:
 (♦ in data lines indicate blanks)

```
data one;
infile 'p:\bio113\ex11.dat';
input name $10. age 2.
      dob mmddyy8.;
run;
proc print;
run;
```

Data:

sam♦♦♦♦♦♦♦♦4201161960

Result:

Obs	name	age	dob
1	sam	42	15

```
data one;
infile 'p:\bio113\ex12.dat';
input id 2. a 1. b 3. @8 c 4.;
run;
proc print;
run;
```

Data:

311123♦4456

Result:

Obs	id	a	b	c
1	31	1	123	4456

```
data one;
infile 'p:\bio113\ex13.dat';
input mrn 5. +3 (bp1-bp5) (2.);
run;
proc print;
run;
```

Data:

90043♦♦♦5867987271

Result:

Obs	mrn	bp1	bp2	bp3	bp4	bp5
1	90043	58	67	98	72	71

```
data one;
infile 'p:\bio113\ex14.dat';
input @1 (x1-x3) ($1. +3)
      @1 (y1-y3) (+1 1. +2)
      @1 (z1-z3) (+2 2.);
run;
proc print;
run;
```

Data:

a342d165f209
m111x893p585

Result:

Obs	x1	x2	x3	y1	y2	y3	z1	z2	z3
1	a	d	f	3	1	2	42	65	9
2	m	x	p	1	8	5	11	93	85

Compare the above to this:

```
data one;
infile 'p:\bio113\ex14.dat';
input x1 $1. y1 1. z1 2. x2 $1. y2 1. z2 2. x3 $1. y3 1. z3 2.;
run;
proc print;
run;
```

Result:

Obs	x1	y1	z1	x2	y2	z2	x3	y3	z3
1	a	3	42	d	1	65	f	2	9
2	m	1	11	x	8	93	p	5	85

MIXED input:

In most situations, you'll mix input styles, using what's convenient and requires the least typing. Typically, you use list style or you use a mix of column and formatted style. This example is a bit contrived to show that you might use all three styles.

Example program:

(♦ in data lines indicate blanks)

```
data one;
infile 'p:\bio113\ex15.dat';
input name $ age mrn 14-18 dob mmddyy8.
      +1 sex $1. @33 (bp1-bp5) (2.);

run;
proc print;
run;
```

Data:

```
sam♦♦♦♦43♦♦♦9004301161960♦m♦♦♦♦5867987271
pam♦♦♦♦43♦♦♦9011301191960♦f♦♦♦♦6165807870
```

Result:

Obs	name	age	mrn	dob	sex	bp1	bp2	bp3	bp4	bp5
1	sam	43	90043	15	m	58	67	98	72	71
2	pam	43	90113	18	f	61	65	80	78	70

Conditional input may be used if:

1. You want to read some, but not all, data lines.
2. You want to read different data lines with different input statements.

How? Input the conditional variable and end the statement with @ (trailing at)
 Test the value of the conditional variable
 Execute additional input statements or delete the observation based on the
 value found

Example program: Read some, but not all, data lines.

```
data one;
infile 'p:\bio113\ex16.dat';
input grade @;

if grade < 8
    then input name $ sex $ age;
else delete;

run;

proc print;
run;
```

Data:

```
6 bob m 11
8 bill m 13
5 jane f 10
```

Result:

Obs	grade	name	sex	age
1	6	bob	m	11
2	5	jane	f	10

```
* read parent data;
data one;
infile 'p:\bio113\exfam.dat';
input rec $ @;
if rec='parent'
    then input sex age employed;
else delete;

run;

proc print;
run;
```

Data:

```
house 8 5 3 2
parent 1 33 1
parent 0 33 0
child 1 10
child 0 9
child 1 6
pet 0 1
pet 0 2
```

Result:

Obs	rec	sex	age	employed
1	parent	1	33	1
2	parent	0	33	0

Example program: Read different data lines with different input statements.

```
data one;
infile 'p:\bio113\ex17.dat';
input sex $6 @;
```

```
if sex='f'
  then input name $1-4 @7 (softball fldhcky)(1.) age 9-10;
else if sex='m'
  then input name $1-4 @7 (baseball soccer)(1.) age 10-11;
else delete;
```

```
run;
proc print;
run;
```

Data:

123456789012

```
bob    m11 10
jane   f01 9
jill   f1012
bill   m01 12
lee     11
```

Result:

Obs	sex	name	softball	fldhcky	age	baseball	soccer
1	m	bob	.	.	10	1	1
2	f	jane	0	1	9	.	.
3	f	jill	1	0	12	.	.
4	m	bill	.	.	12	0	1

```
* read all data;
data one;
infile 'p:\bio113\ex16fam.dat';
input rec $ @;
if rec='house' then input rooms num kids pets;
else if rec='parent' then input sex age employed;
else if rec='child' then input sex age;
else if rec='pet' then input sex kind;
else delete;
run;

proc print;
run;
```

Data:

```
house  8 5 3 2
parent 1 33 1
parent 0 33 0
child  1 10
child  0 9
child  1 6
pet    0 1
pet    0 2
```

Result:

Obs	rec	rooms	num	kids	pets	sex	age	employed	kind
1	house	8	5	3	2
2	parent	1	33	1	.
3	parent	0	33	0	.
4	child	1	10	.	.
5	child	0	9	.	.
6	child	1	6	.	.
7	pet	0	.	.	1
8	pet	0	.	.	2

Read multiple observations on a single record -- double trailing at (@@):

```
data one;
infile 'p:\bio113\ex18.dat';
input id hr @@;
run;
proc print;
run;
```

Data:

1001	121	1005	93	1011
107	1034	82		

From SAS LOG:

NOTE: 2 records were read from the infile 'p:\bio113\ex18.dat'.
 The minimum record length was 11.
 The maximum record length was 21.
 NOTE: SAS went to a new line when INPUT statement reached past
 the end of a line.

OUTPUT:

Obs	id	hr
1	1001	121
2	1005	93
3	1011	107
4	1034	82

Read a single observation from multiple records – slash (/) or pound sign (#) pointer instruction :

```
data one;
infile 'p:\bio113\ex19.dat';
input id 4. hr1 2. / hr2 2-3 #3 hr3;
run;
proc print;
run;
```

Data:

1021	95		
f88			
	101		

From SAS LOG:

NOTE: 3 records were read from the infile 'p:\bio113\ex19.dat'.
 The minimum record length was 3.
 The maximum record length was 6.
 NOTE: The data set WORK.ONE has 1 observations and 4 variables.

OUTPUT:

Obs	id	hr1	hr2	hr3
1	1021	95	88	101

Options on the INFILE statement:

With no options on the infile statement, we get into trouble trying to read the following data with a list input statement. Notice, there are only 3 data values on the first data record.

Example program:

```
* infile with no options;
data one;
infile 'p:\bio113\ex20.dat';
input bp1-bp4;
run;
proc print;
run;
```

Data:

```
121 83 100
84 92 120 95
```

Result:

From SAS LOG:

```
NOTE: 2 records were read from the infile 'p:\bio113\ex20.dat'.
      The minimum record length was 10.
      The maximum record length was 12.
NOTE: SAS went to a new line when INPUT statement reached past
      the end of a line.
NOTE: The data set WORK.ONE has 1 observations and 4 variables.
```

OUTPUT:

Obs	bp1	bp2	bp3	bp4
1	121	83	100	84

MISSOVER If the list input statement describes variables beyond the end of the data record, as above, SAS will go to the *next line* to complete the input statement. MISSOVER tells SAS to stop reading variables at the end of the line; any variables in the input statement not read before SAS finds the end of the line are assigned missing (.). (Note: TRUNCOVER, page 23, will achieve the same effect but the tradition is to use missover with list input and truncover with column and formatted input)

Example program:

```
* infile with missover - data from ex20;
data one;
infile 'p:\bio113\ex20.dat' missover;
input bp1-bp4;
run;
proc print;
run;
```

Data:

```
121 83 100
84 92 120 95
```

Result:

From SAS LOG:

```
NOTE: 2 records were read from the infile 'p:\bio113\ex20.dat'.
      The minimum record length was 10.
      The maximum record length was 12.
NOTE: The data set WORK.ONE has 2 observations and 4 variables.
NOTE: DATA statement used:
```

OUTPUT:

Obs	bp1	bp2	bp3	bp4
1	121	83	100	.
2	84	92	120	95

With no options on the infile statement, we get into trouble trying to read the following data with a column or formatted input statement. You cannot ordinarily see that a record is "short", but here I've marked the end of the data record with ®.

Example program:

```
* infile with no options;
data one;
infile 'p:\bio113\ex21.dat';
input x 1-3 y 4-6;
run;
proc print;
run;
```

Data

```
45385®
763943®
```

Result:**From SAS LOG:**

```
NOTE: 2 records were read from the infile 'p:\bio113\ex21.dat'.
      The minimum record length was 5.
      The maximum record length was 6.
NOTE: SAS went to a new line when INPUT statement reached past
      the end of a line.
NOTE: The data set WORK.ONE has 1 observations and 2 variables.
```

OUTPUT:

Obs	x	y
1	453	763

TRUNCOVER If the column or formatted input statement describes variables beyond the end of the data record, as above, SAS will go to the *next line* to complete the input statement. TRUNCOVER tells SAS to read to the last column specified in the input statement or to the end of the line, whichever happens first. By the way, MISCOVER will not work here.

Example program:

```
* infile with truncover;
data one;
infile 'p:\bio113\ex21.dat' truncover;
input x 1-3 y 4-6;
run;
proc print;
run;
```

Data

```
45385®
763943®
```

From SAS LOG:

```
NOTE: 2 records were read from the infile 'p:\bio113\ex21.dat'.
      The minimum record length was 5.
      The maximum record length was 6.
NOTE: The data set WORK.ONE has 2 observations and 2 variables.
```

OUTPUT:

Obs	x	y
1	453	85
2	763	943

LRECL Short for **Logical RECORD Length**. Somewhere deep inside SAS it is written that data lines (records) are of maximum length 256 unless otherwise specified. If the data lines are longer than 256 characters, then you must describe the longer record length with LRECL in order to read the data beyond column 256.

Example: `infile 'loooong.data' lrecl=2001;`

PAD PAD, used with column or formatted input, will 'pad' the data line with spaces out to the LRECL. This is another way to even out the 'ragged' right side of a data set (you would not need to use trunccover).

NOTE: You may end up using PAD on your infile statement but it is NOT a good idea to use it unless it's needed. Ragged data lines in formatted files often indicate some problem in the creation or copying of the data set. The 'SAS went to a new line' message will warn you to look at the data file more carefully.

Example: `infile 'ex21.dat' pad;`
 `infile 'loooong.data' lrecl=2001 pad;`

Example program:

Data

```
* infile with pad;
data one;
infile 'p:\bio113\ex21.dat' pad;
input x 1-3 y 4-6;
run;
proc print;
run;
```

45385®

From SAS LOG:

NOTE: 2 records were read from the infile 'p:\bio113\ex21.dat'.
 The minimum record length was 5.
 The maximum record length was 6.
 NOTE: The data set WORK.ONE has 2 observations and 2 variables

OUTPUT:

Obs	x	y
1	453	85
2	763	943

ACCESSING EXTERNAL RAW DATA SETS AND SAS DATA SETS (LSB 2.8-2.10)

External raw data sets: FILENAME and *fileref*

We've seen:

Data:

```
* what we've seen;
data one;
infile 'p:\bio113\ex01.dat'; * note: name of external raw data set in single quotes *;
input a b;
run;
```

```
11 35
95 7
```

You may also describe the external 'raw' data set with a FILENAME statement. A link is created between the file named in the FILENAME statement and the INFILE statement with the *fileref*. You choose the *fileref* but it can only be **8 characters** long. The FILENAME statement must precede the INFILE statement. Options (MISSOVER, TRUNCOVER, LRECL, PAD) appear on the INFILE statement as before.

How? FILENAME *fileref* 'raw-data set-name';

Example:

```
* filename with fileref (fluffy is the fileref);
filename fluffy 'p:\bio113\ex01.dat';
data one;
infile fluffy;
input a b;
run;
```

SAS LOG:

```
267 * filename with fileref (fluffy is the fileref);
268 filename fluffy 'p:\bio113\ex01.dat';
269 data one;
270 infile fluffy;
271 input a b;
272 run;
```

```
NOTE: The infile FLUFFY is:
      File Name=p:\bio113\ex01.dat,
      RECFM=V,LRECL=256
```

```
NOTE: 2 records were read from the infile FLUFFY.
      The minimum record length was 4.
      The maximum record length was 5.
```

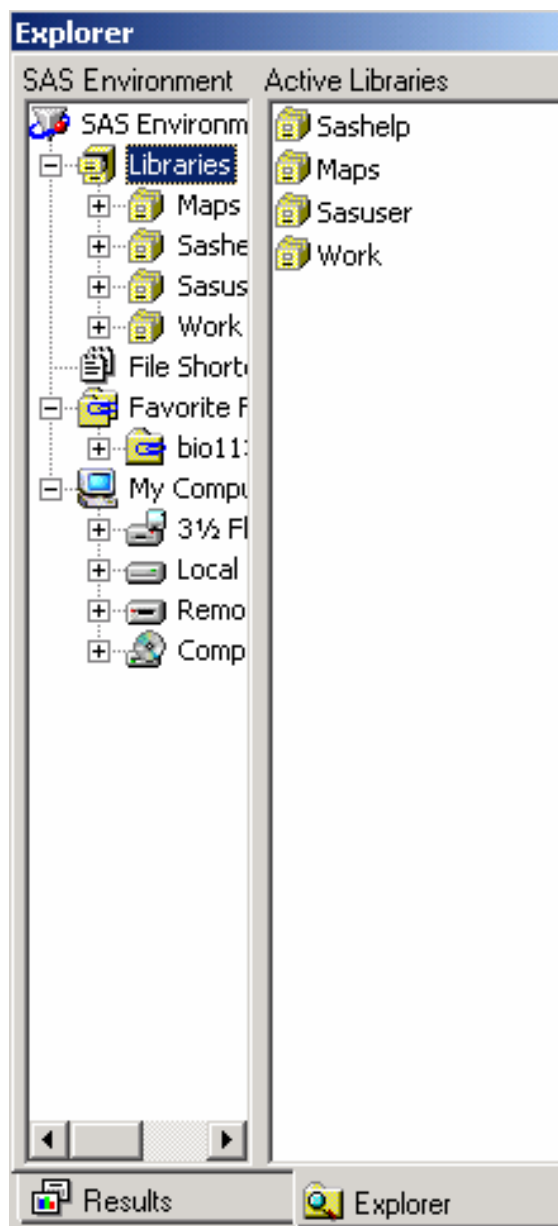
```
NOTE: The data set WORK.ONE has 2 observations and 2 variables.
```

```
NOTE: DATA statement used:
      real time          0.06 seconds
      cpu time           0.01 seconds
```

Test: Write FILENAME and INFILE statements to read the 'raw' data set, ivh.dat, from p:\bio113.

Answer:

SAS libraries: click on the Explorer tab and then on Libraries



What is a library? A library is a folder that contains special SAS files including SAS data sets. Three libraries (at least), exist when you start SAS. Other libraries may be added by you when you create permanent SAS data sets.

Default SAS Libraries:

SASHELP contains SAS help files

SASUSER contains device drivers, fonts and the user profiles

WORK will contain **temporary** SAS data sets

Here we also see a library called MAPS.

Temporary vs. permanent SAS data sets:

Temporary SAS file:	Exists only for the duration of the SAS session Is stored in the library, WORK (click on WORK to see what's been stored there) You gave it a 'single-level' name (though SAS gives it a 'two-level' name)
----------------------------	---

Permanent SAS file:

- Exists after you exit a SAS session (we'll make one shortly!)
- Is stored in your own library
- You gave it a 'two-level' name

Create a *temporary* SAS data set: (we've been doing it)

Example:

```
* creating a temporary SAS dataset;
filename pups 'p:\bio113\ex22.dat';
data dogs;
infile pups;
input name $ age;
run;
```

Data

```
spot 2
dot 4
```

```
LOG: 15 * creating a temporary SAS dataset;
16 filename pups 'p:\biol113\ex22.dat';
17 data dogs;
18 infile pups;
19 input name $ age;
20 run;
```

NOTE: The infile PUPS is:
File Name=P:\bio113\ex22.dat,
RECFM=V,LRECL=256

NOTE: 2 records were read from the infile PUPS.
The minimum record length was 5.
The maximum record length was 6.

NOTE: The data set WORK.DOGS has 2 observations and 2 variables.

```
NOTE: DATA statement used:
      real time          0.09 seconds
      cpu time           0.01 seconds
```

Result: Examine the SAS log. SAS created DOGS with 2 observations and 2 variables. While you gave the data set the single-level name, DOGS, SAS knows it by a **two-level name**, WORK.DOGS. The first level of the name WORK) is the *libref* and it points to the library (*physically*, a subfolder in c:\SASTemp_Txxxx; xxxx is a number generated for this session) where **temporary** SAS data files are stored. The second level is DOGS, the name you gave the data set. Click Libraries in Explorer and the library WORK to verify that DOGS is there.

Test: Write a program to create a **temporary** data set with the single-level name, INSECTS, and with variables, BUG and LEGS. Then see if you can find it in a SAS library. Which library should you look in?

Here are the data lines in ex23.dat:

```
spider 8  
cricket 6
```

Answer:

Create a permanent SAS data set:

How? Tell SAS the name of the library (folder) where you want to store the SAS data set.

LIBNAME *libref* 'library-name';

Example:

```
* creating a permanent SAS data set (libref is muffy);
libname muffy 'p:\bio113';
data muffy.dogs;
infile 'p:\bio113\ex22.dat';
input name $ age;
run;
```

LOG:

```
17 * creating a permanent SAS dataset (libref is muffy);
18 libname muffy 'p:\bio113';
NOTE: Libref MUFFY was successfully assigned as follows:
      Engine:          V8
      Physical Name: p:\bio113
19 data muffy.dogs;
20 infile 'p:\bio113\ex22.dat';
21 input name $ age;
22 run;
```

NOTE: The infile 'p:\bio113\ex22.dat' is:
 File Name=P:\bio113\ex22.dat,
 RECFM=V, LRECL=256

NOTE: 2 records were read from the infile 'p:\bio113\ex22.dat'.
 The minimum record length was 5.
 The maximum record length was 6.

NOTE: The data set MUFFY.DOGS has 2 observations and 2 variables.

NOTE: DATA statement used:
 real time 0.05 seconds
 cpu time 0.02 seconds

Result: Examine the SAS Log. SAS created MUFFY.DOGS with 2 observations and 2 variables. The first level of the name (muffy) is the *libref* I chose and it points to a library that's *physically* a folder (p:\bio113) I created *before* I ran the program. This permanent SAS data set will be stored in p:\bio113. Click Libraries and under Active Libraries, muffy, to verify; notice that SAS calls the file DOGS (Not muffy.dogs). Just to confuse things, the file will have the name dogs.sas7bdat (sas7bdat is the special suffix reserved for version 7 and 8 SAS data sets) when you look for it with Windows "My Computer" or "Search".

Test: Write a program to read the insect data set (ex23.dat), as before, but, this time, save it as a permanent SAS data set, insects.sas7bdat, on your P drive.

Answer:

Using a *permanently* stored SAS data set:

Example: Come back later to read the ***permanent*** data set, dogs.sas7bdat, into a ***temporary*** data set (foo) and print the data. Note, this time the *libref* is alvin but it points to the same physical location, p:\bio113.

Program: * read from an existing SAS data set;
 libname alvin 'p:\bio113';
 data foo;
 set alvin.dogs; * set is the SAS command that reads *existing* SAS data sets *;
 run;
 proc print; run;

LOG: 23 * read from an existing SAS data set;
 24 libname alvin 'p:\bio113';
 NOTE: Libname ALVIN refers to the same physical library as MUFFY.
 NOTE: Libref ALVIN was successfully assigned as follows:
 Engine: V8
 Physical Name: p:\bio113
 25 data foo;
 26 set alvin.dogs; * set is a SAS command that reads
 existing SAS data sets *;
 27 run;

 NOTE: There were 2 observations read from the dataset ALVIN.DOGS.
 NOTE: The data set WORK.FOO has 2 observations and 2 variables.
 NOTE: DATA statement used:
 real time 0.08 seconds
 cpu time 0.00 seconds

Test: Write SAS statements to read the insect data set (ex23.dat) and save insects.sas7bdat permanently in your own folder, p:\bio113. Write statements that read insects.sas7bdat from your folder into a temporary data set (call it ONE) and then print the data.

Answer:

Another way to create and later use a permanent SAS data set:

Starting with SAS version 7, you can directly reference SAS data sets in programs. The syntax is similar to that of the INFILE statement where the name of a raw data set is included in quotation marks. On the DATA (or SET, MERGE, etc) statement, include the path and member name of the SAS data set in quotes. Notice that SAS, in fact, creates a libref for you (Wc000001, 2, etc) and lists it under Active Libraries. In this program the SAS file created and later read is dogs.sas7bdat and it can be found in the folder, p:\bio113.

Example: * create a permanent SAS data set without a libref;
data 'p:\bio113\dogs';
infile 'p:\bio113\ex22.dat';
input name \$ age;
run;

LOG: 15 data 'p:\bio113\dogs';
 16 infile 'p:\bio113\ex22.dat';
 17 input name \$ age;
 18 run;

NOTE: The infile 'p:\bio113\ex22.dat' is:
 File Name=P:\bio113\ex22.dat,
 RECFM=V,LRECL=256

NOTE: 2 records were read from the infile 'p:\bio113\ex22.dat'.
 The minimum record length was 5.
 The maximum record length was 6.

NOTE: The data set p:\bio113\dogs has 2 observations and 2 variables.

NOTE: DATA statement used:
 real time 0.51 seconds
 cpu time 0.03 seconds

Test: Write SAS statements to read the insect data set (ex23.dat) and save insects.sas7bdat permanently in your own folder, p:\bio113 **without** using a libref statement.

Answer:

Read a permanent SAS data set without using a libref statement:

Example: * read a permanent SAS data set without a libref;
 data foo;
 set 'p:\bio113\dogs';
 run;

LOG: 21 data foo;
 22 set 'p:\bio113\dogs';
 23 run;

NOTE: There were 2 observations read from the dataset
p:\bio113\dogs.

NOTE: The data set WORK.FOO has 2 observations and 2 variables.

NOTE: DATA statement used:
 real time 0.78 seconds
 cpu time 0.03 seconds.

Test: Read your permanent SAS data set, insects.sas7bdat, into a temporary data set, foo.

Answer:

See what's in the SAS data set:

PROC CONTENTS: This PROC lists documentation stored with a temporary or permanent SAS data set.

Example: * what's in the SAS data set?;
 libname alvin 'p:\bio113';
 proc contents data=alvin.dogs;
 run;

 /* here is another proc contents statement */
 /* it's commented out but we can try it in class */

 /* proc contents position data=alvin.dogs; run; */

Result:

The CONTENTS Procedure

Data Set Name:	ALVIN.DOGS	Observations:	2
Member Type:	DATA	Variables:	2
Engine:	V8	Indexes:	0
Created:	16:07 Thursday, August 26, 2004	Observation Length:	16
Last Modified:	16:07 Thursday, August 26, 2004	Deleted Observations:	0
Protection:		Compressed:	NO
Data Set Type:		Sorted:	NO
Label:			

-----Engine/Host Dependent Information-----

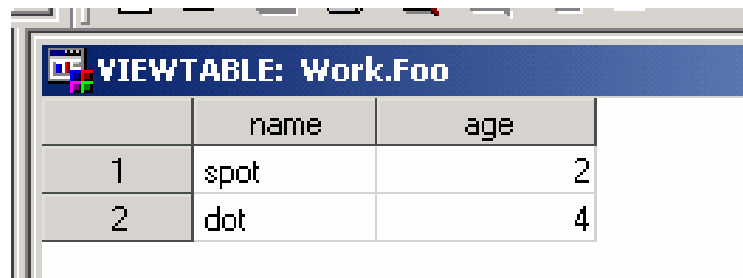
Data Set Page Size:	4096
Number of Data Set Pages:	1
First Data Page:	1
Max Obs per Page:	252
Obs in First Data Page:	2
Number of Data Set Repairs:	0
File Name:	p:\bio113\dogs.sas7bdat
Release Created:	8.0202M0
Host Created:	WIN_PRO

-----Alphabetic List of Variables and Attributes-----

#	Variable	Type	Len	Pos
2	age	Num	8	0
1	name	Char	8	8

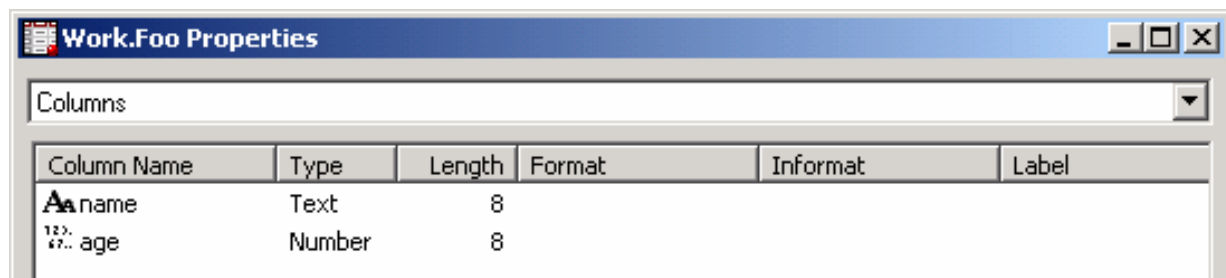
View the data: You may also view the data contained in the SAS data set. We've been doing that using PROC PRINT but it can also be done from SAS's Explorer. Click Explorer, Libraries and then click the active library WORK. Then double click Foo. (Note that if you right click on the file and choose View Columns from the pull-down menu, you get an abbreviated 'proc contents'.)

Here's what you see.



	name	age
1	spot	2
2	dot	4

Here's what you see with 'View Columns'.



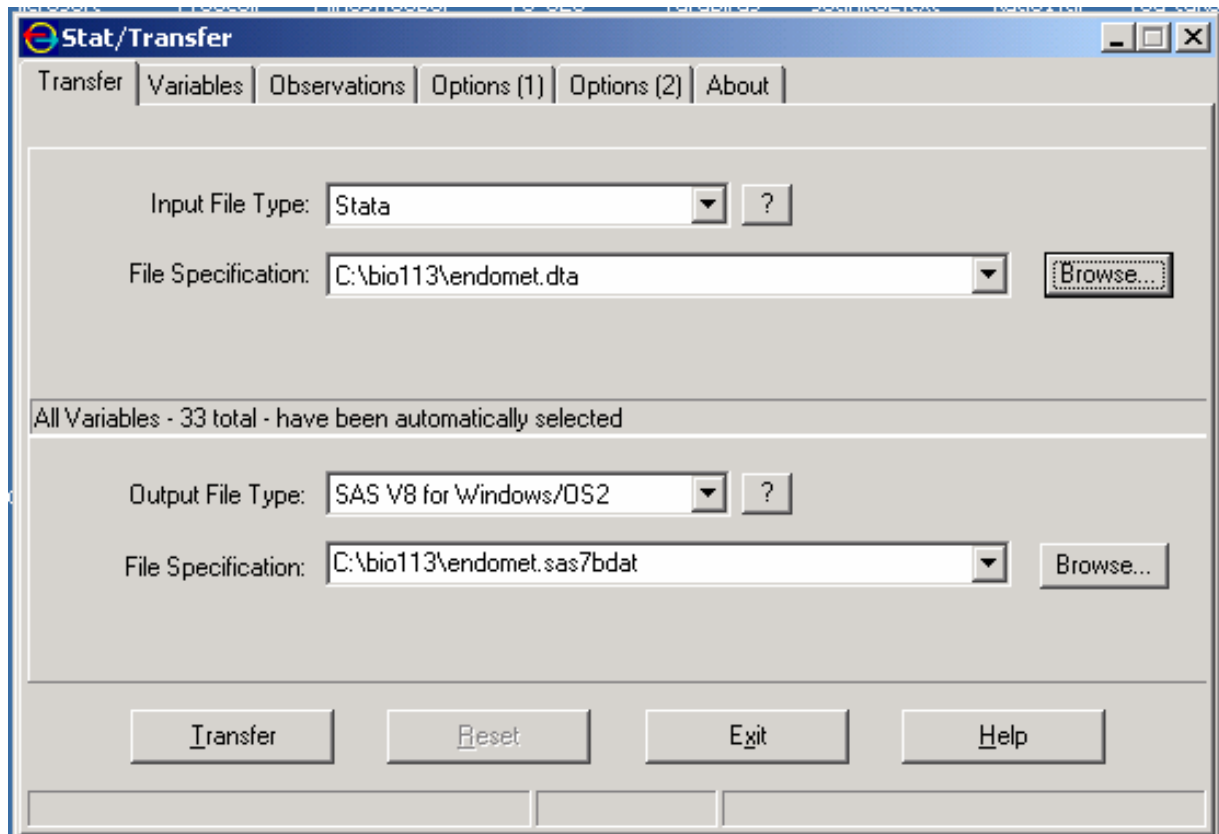
Column Name	Type	Length	Format	Informat	Label
name	Text	8			
age	Number	8			

Click on the ▼ to the right of **Columns** to choose to see the data set properties or engine and host information.

INPUT FROM OTHER PROGRAMS: STAT/TRANSFER™

Stat/Transfer is a separate, easy to use, Windows program that transfers data among several statistical, database and spreadsheet programs. Among the input and output files Stat/Transfer works with are ASCII, Access, dBase Excel, Lotus 1-2-3, Paradox, SAS, S-plus, SPSS, Stata and Systat.

Start Stat/Transfer by clicking Start, Miscellaneous and StatTransfer. You will see this dialogue box.



Select an input file type—I've indicated a Stata data set—and click Browse to locate the file. My Stata file is c:\bio113\endomet.dta. The suffix, .dta, indicates this is a Stata data set. Next, select the output file type—I've indicated a SAS for Windows file. Stat/Transfer gave the output file the same name as the input file except the suffix is .sas7bdat, which indicates a Version 7/8 SAS data set is to be created. You may change any part of this name (use Browse) but **do not change** the suffix. Finally, click Transfer and the SAS data set will be created from the Stata data set. The tabs--Variables, Observations and Options--permit you to transfer subsets of variables and observations and to select various transfer options. When you create a SAS data set with Stat/Transfer you avoid the INPUT issue entirely! You may immediately access the data set with SAS's SET statement. We will demonstrate this handy program in class.

CREATE NEW VARIABLES & CHANGE EXISTING VARIABLES (LSB 3.1-3.3, 3.7-3.8, 8.7-8.9, 8.12)

Assignment statements: New variables are created and existing ones modified with assignment statements. Note that these SAS statements **do not** begin with SAS keywords.

Create new variables: Assign a constant value or the result of an expression to a new variable. The new variable appears to the left of the equals sign. If any variable used in an expression is missing (.), the new variable will also be missing.

Example program:

```
libname in 'p:\bio113';
data one; set in.fool;

name='Molly'; * note quotes on char var;
age=5.2; * note NO quotes on num var;
age_mons=age*12;
tempc=5/9*(temp-32);
x=12.5+7.8+2.4;
y=a+b+c;
run;

proc print; run;
```

Values of variables in foo1.sas7bdat

id	temp	a	b	c
1	98.6	1	.	3
2	99.1	4	5	6

Result:

Obs	id	temp	a	b	c	name	age	age_mons	tempc	x	y
1	1	98.6	1	.	3	Molly	5.2	62.4	37.0000	22.7	.
2	2	99.1	4	5	6	Molly	5.2	62.4	37.2778	22.7	15

Change existing variables: Replace the value of a variable with a new value.

Example program:

```
libname in 'p:\bio113';
data one; set in.foo2;

age=age*12;
temp=9/5*temp+32;
avg1=(a+b+c)/3;
a=a+b+c;
avg2=(a+b+c)/3;
run;
proc print; run;
```

Values of variables in foo2.sas7bdat

id	age	temp	a	b	c
1	5.2	100	1	2	3
2	3.6	37	5	2	.

Result:

Obs	id	age	temp	a	b	c	avg1	avg2
1	1	62.4	212.0	6	2	3	2	3.66667
2	2	43.2	98.6	.	2	.	.	.

SAS functions: SAS has a library of functions that perform arithmetic, string and other operations on **constants** and **variables**. Use these functions in expressions to create new variables or replace existing ones. *Unlike the assignment statements above, missing values are not always generated by missing data when you use SAS functions.* We will find a listing of built-in SAS functions using SAS on-line Help (Click in succession: Help, Getting Started with SAS Software, Help on SAS software products, Base software, Using base SAS software, Working with the SAS language, SAS Functions, Function Categories).

Example program:

Values of variables in foo3.sas7bdat

```
libname in 'p:\bio113';
data one; set in.foo3;

xbp=mean(101,107,118);
ybp=mean(bp1,bp2,bp3);
zbp=mean(of bp1-bp3);

name=trim(last)||', '||trim(first);
init=substr(first,1,1);

tot1=round(sum(2.2,1.3,6.8));
tot2=round((sum(2.2,1.3,6.8)),.5);

true=sum(of bp1-bp3) > 350;
run;

proc print; run;
```

Result:

Obs	id	bp1	bp2	bp3	first	last	xbp	ybp
1	1	116	122	129	yogi	bear	108.667	122.333
2	2	111	.	114	rocky	squirrel	108.667	112.500
3	3	.	.	.	micky	mouse	108.667	.

Obs	zbp	name	init	tot1	tot2	true
1	122.333	bear, yogi	y	10	10.5	1
2	112.500	squirrel, rocky	r	10	10.5	0
3	.	mouse, micky	m	10	10.5	0

Date functions: SAS performs date/time arithmetic on dates expressed as number of days before or since January 1, 1960 (1/1/1960 has the value 0) and time expressed as seconds since midnight (midnight has the value 0). Date-time is expressed as seconds before or since midnight, January 1, 1960. Earlier, we used the date informat *mmddyy8.* to input a date, 01161960, which was then represented in the SAS data set as 15. SAS date and time functions create SAS dates from variables containing values for month, day, year, hours, minutes, seconds, etc, or from constants. More date/time functions are listed in SAS on-line help.

Example program:

```
libname in 'p:\bio113';
data one; set in.foo4;

dob1=mdy(1,16,1960);
tob1=hms(2,10,0);

dob2=mdy(bm,bd,by);
tob2=hms(bh,bmn,0);

dob3=dhms(dob2,bh,bmn,0);

thisday=today();
now=time();
rightnow=24*60*60*thisday+now;
run;

proc print; run;
```

Values of variables in foo4.sas7bdat

id	bm	bd	by	bh	bmn
1	1	16	1960	2	10
2	12	25	1959	3	22

Result:

The SAS System

13:37 Wednesday, September 1, 2004

Obs	id	bm	bd	by	bh	bmn	dob1	tob1	dob2	tob2	dob3	thisday	now	rightnow
1	1	1	16	1960	2	10	15	7800	15	7800	1303800	16315	49296.16	1409665296.2
2	2	12	25	1959	3	22	15	7800	-7	12120	-592680	16315	49296.16	1409665296.2

Year 2K?

yearcutoff=nnnn where *nnnn* is the **first** year of the 100-year span. The yearcutoff option applies only to dates where year is described with 2 digits. Notice the effect of the two output formats, mmddyy8. and mmddyy10.

The default value for yearcutoff is 1920.

Program:

```
options nocenter yearcutoff=1940;
filename y2k 'p:\bio113\ex24.dat';
data one; infile y2k;
input id dob mmddyy6.;
dob1=dob;
interval=(today()-dob)/365.25;
run;

proc print; id id;
  var dob dob1 interval;
  format dob mmddyy8. dob1 mmddyy10.;
  title 'Dates with yearcutoff=1940';
run;
```

Data:

```
1 041500
2 041519
3 041520
4 041539
5 041540
6 041559
7 041560
8 041599
```

Result:

Dates with yearcutoff=1940

15:15 Sunday, August 15, 2004

id	dob	dob1	interval
1	04/15/00	04/15/2000	4.3340
2	04/15/19	04/15/2019	-14.6639
3	04/15/20	04/15/2020	-15.6660
4	04/15/39	04/15/2039	-34.6639
5	04/15/40	04/15/1940	64.3340
6	04/15/59	04/15/1959	45.3361
7	04/15/60	04/15/1960	44.3340
8	04/15/99	04/15/1999	5.3361

This is the output when you omit 'yearcutoff=1940' from the program:

Dates with yearcutoff=1920

15:15 Sunday, August 15, 2004

id	dob	dob1	interval
1	04/15/00	04/15/2000	4.3340
2	04/15/19	04/15/2019	-14.6639
3	04/15/20	04/15/1920	84.3340
4	04/15/39	04/15/1939	65.3361
5	04/15/40	04/15/1940	64.3340
6	04/15/59	04/15/1959	45.3361
7	04/15/60	04/15/1960	44.3340
8	04/15/99	04/15/1999	5.3361

Date difficulties

Occasionally you'll be given a SAS data set with dates that were read in without using SAS date functions. The variables may be stored as character or numeric but SAS will not have interpreted them as dates. You will need to disassemble the variables into months, days and years and then apply a SAS date function to get a proper SAS date. See if you can figure out how to make SAS times out of time information that was read in incorrectly.

```
* extract month, day, and year so you can use the mdy function when a date
  is read from raw data as a character or numeric variable instead of
  as a date variable;
```

```
data one;
input cdate $10. +1 ndate 8.;

mon1=1*substr(cdate,1,2);
day1=1*substr(cdate,4,2);
yr1 =1*substr(cdate,7,4);

mon2=int(ndate/1000000);
day2=int(ndate/10000)-(mon2*100);
yr2 =ndate-(mon2*1000000+day2*10000);

date1=mdy(mon1,day1,yr1);
/* or date2=mdy(mon2,day2,yr2); */

datalines;
09-30-2003 09302003
11/06/1970 11061970
run;
proc print; run;
```

The SAS System

Obs	cdate	ndate	mon1	day1	yr1	mon2	day2	yr2	date1
1	09-30-2003	9302003	9	30	2003	9	30	2003	15978
2	11/06/1970	11061970	11	6	1970	11	6	1970	3962

Useful SAS functions

These SAS functions were used in a program executed on 09/01/2004 at 1:37:03 PM.

Data:

x1	x2	x3	x4	x5	f	nm	a1	a2	mon	day	yr	hr	min	sec
10	5	23	.	37	21.655	-4	Muffy	st♦♦♦	9	01	2004	13	37	3

Function	SAS assignment statement	Answer (the value of 'var')
mean	var=mean(3,4,5);	4
	var=mean(x1,x2,x3,x4,x5);	18.75
	var=mean(of x1-x5);	18.75
sum	var=sum(3,4,5);	12
	var=sum(x1,x2,x3,x4,x5);	75
	var=sum(of x1-x5);	75
min	var=min(3,4,5);	3
	var=min(x1,x2,x3,x4,x5);	5
	var=min(of x1-x5);	5
max	var=max(3,4,5);	5
	var=max(x1,x2,x3,x4,x5);	37
	var=max(of x1-x5);	37
n	var=n(x1,x2,x3,x4,x5);	4
	var=n(of x1-x5);	4
nmiss	var=nmiss(x1,x2,x3,x4,x5);	1
	var=nmiss(of x1-x5);	1
round	var=round(11.336);	11
	var=round(f,.01);	21.66
abs	var=abs(nm);	4
int	var=int(f);	21
mod	var=mod(f,1);	0.655
	var=mod(x1,3);	1
exp	var=exp(x2);	148.413
log	var=log(x1);	2.30259
log10	var=log10(x1);	1
sqrt	var=sqrt(x1);	3.16228
**	var=x2**3;	125
mdy	date=mdy(9,30,2003);	16315 (01SEP2004)
	date=mdy(mon,day,yr);	16315 (01SEP2004)
hms	var=hms(hr,min,sec);	49023 (13:37:03)
dhms	var=dhms(date,hr,min,sec);	1409665023 (01SEP2004:13:37:03)
today	var=today();	16315 (01SEP2004)
time	var=time();	49023 (13:37:03)
substr	var=substr(a1,3,2);	ff
trim	var=trim(a2);	st
	var=trim(a2) a1;	stMuffy
upcase	var=upcase(a1);	MUFFY
lowcase	var=lowcase(a1);	muffy

CONDITIONAL PROCESSING (LSB 3.4-3.6)

Conditional assignment: IF-THEN, IF-THEN-DO-END

IF-THEN permits processing of subsets of observations. Only a single assignment or action may be made after **THEN**.

Program:

Values of variables in foo5.sas7bdat

```
libname in 'p:\bio113';
data one; set in.foo5;
if name='molly' then sex='f';
if age < 10 then agecat1=1;
* if . < age < 10 then agecat2=1;
if 30 <= wt < 40 and sex='m' then group=1;
if age=0 or age=99 then age=.;
if n(of x1-x4) = 4 then complete=1;
miss=nmiss(of x1-x4) > 0;
if name='mandy' then delete;
run;

proc print; run;
```

Result:

name	age	sex	wt	x1	x2	x3	x4	agecat1	agecat2	group	complete	miss
molly	9	f	30	1	.	3	3	1	1	.	.	1
max	12	m	54	5	3	2	7	.	.	.	1	0
mike	.	m	32	.	5	.	7	1	.	1	.	1
matt	.		62	3	4	1	6	1	1	.	1	0

* NOTE:

the statement: `if . < age <= 10 then agecat2=1;`

could be written: `if age > . & age <= 10 then agecat2=1;`

or: `if age ^= . & age <= 10 then agecat2=1;`

IF-THEN-DO-END: If you want to make **more than one** assignment, the syntax changes slightly.

IF condition THEN DO;

assignment 1

assignment 2 ...

END;

Program:

Values of variables in foo5.sas7bdat

name	age	sex	wt	x1	x2	x3	x4
molly	9	m	30	1	.	3	3
max	12	m	54	5	3	2	7
mandy	99	f	.	1	1	1	1
mike	.	m	32	.	5	.	7
matt	0		62	3	4	1	6

```
libname in 'p:\bio113';
```

```
data one; set in.foo5;
```

```
if name='molly' | . < age < 10 then do;
```

```
sex='f';
```

```
agecat=1;
```

```
end;
```

```
if nmiss(of x1-x4)=0 then do;
```

```
complete=1;
```

```
ratio=wt/age;
```

```
end;
```

```
run;
```

```
proc print; id name;
```

```
run;
```

Result:

Obs	name	age	sex	wt	x1	x2	x3	x4	agecat	complete	ratio
1	molly	9	f	30	1	.	3	3	1	.	.
2	max	12	m	54	5	3	2	7	.	1	4.5
3	mandy	99	f	.	1	1	1	1	.	1	.
4	mike	.	m	32	.	5	.	7	.	.	.
5	matt	0	f	62	3	4	1	6	1	1	.

Multiple choice processing: IF-THEN-ELSE

If the condition is not met, what value is assigned to the variable?

Unless you tell SAS otherwise, the value is missing. Use IF-THEN-ELSE to make different assignments when different conditions are met. Multiple assignments are made using DO...END as before.

Example program

```
libname in 'p:\bio113';
data one; set in.foo6;

if . < age <= 10 then agecat=1;
else if 10 < age <= 20 then agecat=2;
else if age > 20 then agecat=3;

if . < bp <= 90 then htn1=0;
else if bp > 90 then htn1=1;

htn2=(bp > 90);

if . < age <= 20 then do;
    if . < bp <= 80 then agehtn=0;
    else agehtn=1;
end;
else if age > 20 then do;
    if . < bp <= 90 then agehtn=0;
    else agehtn=1;
end;

if name='pam' | name='ann' then do;
    sex='f';
    group=5;
end;
else if name='sam' then do;
    sex='m';
    group=3;
end;

run;

proc print;
run;
```

Values of variables in foo6.sas7bdat

id	name	age	bp
1	ham	52	100
2	pam	20	70
3	nan	.	80
4	sam	8	78
4	dan	18	.
6	ann	26	.

Result:

Obs	id	name	age	bp	agecat	htn1	htn2	agehtn	sex	group
1	1	ham	52	100	3	1	1	1	.	.
2	2	pam	20	70	2	0	0	0	f	5
3	3	nan	.	80	.	0	0	.	.	.
4	4	sam	8	78	1	0	0	0	m	3
5	4	dan	18	.	2	.	0	1	.	.
6	6	ann	26	.	3	.	0	1	f	5

Subsetting IF: Send a subset of observations to the next DATA step

A subsetting IF statement of the form **IF condition;** is equivalent to these statements:

```
IF condition then OUTPUT;
IF not-condition THEN DELETE;
```

Obs	id	name	age	bp	agecat	htn1	htn2	agehtn	sex	group
1	1	ham	52	100	3	1	1	1		.
2	2	pam	20	70	2	0	0	0	f	5
3	3	nan	.	80	.	0	0	.		.
4	4	sam	8	78	1	0	0	0	m	3
5	4	dan	18	.	2	.	0	1		.
6	6	ann	26	.	3	.	0	1	f	5

Example program (make subsets of WORK.ONE, above, from previous page)

Result:

```
data two; set one;
if sex ^= 'f';
* if sex ^= 'f' then output;
* if sex = 'f' then delete;
run;
proc print;
  var name age sex;
run;
```

Obs	name	age	sex
1	ham	52	
2	nan	.	
3	sam	8	m
4	dan	18	

```
data two; set one;
if age < 20 & sex='m';
run;
proc print;
  var name age sex;
run;
```

Obs	name	age	sex
1	sam	8	m

```
data two; set one;
if age < 20 | sex='m';
run;
proc print;
  var name age sex;
run;
```

Obs	name	age	sex
1	nan	.	
2	sam	8	m
3	dan	18	

```
/* watch out on this one! */
data two; set one;
if age < 20;
if sex='m';
run;
proc print;
  var name age sex;
run;
```

Obs	name	age	sex
1	sam	8	m

ARRAYS (LSB 3.10)

Previously, we processed subsets of observations. ARRAY permits processing of subsets of *variables*. All variables in a particular array must be of the same data type. First you define the array and then you may process it.

Define arrays: It is my habit to use caps for array names--it makes them stand out from the variables! In any event, arrays are named using the rules for all SAS names. In an *explicit* array, the number of variables in the array or '*' appears in parentheses after the array name. In an *implicit* array, the number in parentheses is omitted. Arrays may be multidimensional but, if so, must be explicit. A \$ indicates that the array contains character variables.

Example:

```
array MISS(3) ga wt len;
array SS(6) score1-score6;
array SX(*) $ neuro cv gi ent;
array MULTI(3,2) a1 a2 b1 b2 c1 c2;

array MISS weight--age;
```

Process arrays: Arrays are usually processed in DO...END loops. In an explicit array, the number of times the loop is processed (usually it's the number of variables in the array) is specified by a subscript variable. You may process all variables in an array in the same way or you may address a particular variable by number (subscript). The first variable in the array has subscript, 1, the second, 2, and so forth. Several arrays may be processed in one DO...END if they contain the same number of variables.

Example:

Program:

```
libname in 'p:\bio113';
data one; set in.foo7;
* drop i;
array MISS(3) ga wt len;
do i=1 to 3;
    if MISS(i)=9999 then MISS(i)=.;
end;
run;

proc print;
run;
```

Values of variables in foo7.sas7bdat

id	ga	wt	len
1	29	1150	9999
2	9999	764	18

Result:

Obs	id	ga	wt	len	i
1	1	29	1150	.	4
2	2	.	764	18	4

Program:

Values of variables in foo7.sas7bdat

```
* again with put _all_;
libname in 'p:\bio113';
data one; set in.foo7;
* drop i;
array MISS(3) ga wt len;
do i=1 to 3;
  put 'before: ' _all_;
  if MISS(i)=9999 then MISS(i)=.;
  put 'after: ' _all_;
end;
run;

proc print;
run;
```

id	ga	wt	len
1	29	1150	9999
2	9999	764	18

The output is identical but look at the SAS log to see how values are reassigned by the array.

```
213 * again with put _all_;
214 libname in 'c:\bio113\data';
NOTE: Libref IN was successfully assigned as follows:
      Engine:          V8
      Physical Name: c:\bio113\data
215 data one; set in.foo7;
216 * drop i;
217 array MISS(3) ga wt len;
218 do i=1 to 3;
219   put 'before: ' _all_;
220   if MISS(i)=9999 then MISS(i)=.;
221   put 'after: ' _all_;
222 end;
223 run;
```

```
before: id=1 ga=29 wt=1150 len=9999 i=1 _ERROR_=0 _N_=1
after:  id=1 ga=29 wt=1150 len=9999 i=1 _ERROR_=0 _N_=1
before: id=1 ga=29 wt=1150 len=9999 i=2 _ERROR_=0 _N_=1
after:  id=1 ga=29 wt=1150 len=9999 i=2 _ERROR_=0 _N_=1
before: id=1 ga=29 wt=1150 len=9999 i=3 _ERROR_=0 _N_=1
after:  id=1 ga=29 wt=1150 len=. i=3 _ERROR_=0 _N_=1
before: id=2 ga=9999 wt=764 len=18 i=1 _ERROR_=0 _N_=2
after:  id=2 ga=. wt=764 len=18 i=1 _ERROR_=0 _N_=2
before: id=2 ga=. wt=764 len=18 i=2 _ERROR_=0 _N_=2
after:  id=2 ga=. wt=764 len=18 i=2 _ERROR_=0 _N_=2
before: id=2 ga=. wt=764 len=18 i=3 _ERROR_=0 _N_=2
after:  id=2 ga=. wt=764 len=18 i=3 _ERROR_=0 _N_=2
```

Example:

Program:

```
libname in 'p:\bio113';
data one; set in.foo8;
* drop j;
array SX(*) $ resp cv gi heme;
count=0;
do j=1 to dim(SX);
    if SX(j)='y' then count=count+1;
end;
run;

proc print;
run;
```

Values of variables in foo8.sas7bdat

id	resp	cv	gi	heme
1	n	y	y	n
2	n	n	n	n
3	y	n	y	y

Result:

Obs	id	resp	cv	gi	heme	count	j
1	1	n	y	y	n	2	5
2	2	n	n	n	n	0	5
3	3	y	n	y	y	3	5

Program:

```
libname in 'p:\bio113';
data one; set in.foo9;
* drop rep;
array SS(3) sbp1 sbp2 sbp3;
array DD(3) dbp1 dbp2 dbp3;
array RR(3) rat1 rat2 rat3;
do rep=1 to 3;
    RR(rep)=SS(rep)/DD(rep);
end;
run;

proc print;
run;
```

Values of variables in foo9.sas7bdat

id	sbp1	sbp2	sbp3	dbp1	dbp2	dbp3
1	120	115	132	87	70	79
2	119	123	127	78	75	86

Result:

Obs	id	sbp1	sbp2	sbp3	dbp1	dbp2	dbp3	rat1	rat2	rat3	rep
1	1	120	115	132	87	70	79	1.37931	1.64286	1.67089	4
2	2	119	123	127	78	75	86	1.52564	1.64000	1.47674	4

Program:

Values of variables in foo9.sas7bdat

```
libname in 'p:\bio113';
data one; set in.foo9;
* drop rep;
array BB(2,3) sbp1-sbp3 dbp1-dbp3;
array RR(3) rat1-rat3;
do rep=1 to 3;
  RR(rep)=BB(1,rep)/BB(2,rep);
  put _all_;
end;
run;
proc print;
run;
```

id	sbp1	sbp2	sbp3	dbp1	dbp2	dbp3
1	120	115	132	87	70	79
2	119	123	127	78	75	86

Result:

```
139 libname in '.';
NOTE: Libref IN was successfully assigned as follows:
      Engine:          V8
      Physical Name: P:BIO113
```

```
140 data one; set in.foo9;
141 * drop rep;
142 array BB(2,3) sbp1-sbp3 dbp1-dbp3;
143 array RR(3) rat1-rat3;
144 do rep=1 to 3;
145   RR(rep)=BB(1,rep)/BB(2,rep);
146   put _all_;
147 end;
148 run;
```

```
id=1 sbp1=120 sbp2=115 sbp3=132 dbp1=87 dbp2=70 dbp3=79 rat1=1.3793103448 rat2=. rat3=.
rep=1 _ERROR_=0 _N_=1
id=1 sbp1=120 sbp2=115 sbp3=132 dbp1=87 dbp2=70 dbp3=79 rat1=1.3793103448
rat2=1.6428571429 rat3=. rep=2 _ERROR_=0 _N_=1
id=1 sbp1=120 sbp2=115 sbp3=132 dbp1=87 dbp2=70 dbp3=79 rat1=1.3793103448
rat2=1.6428571429 rat3=1.6708860759 rep=3 _ERROR_=0 _N_=1
id=2 sbp1=119 sbp2=123 sbp3=127 dbp1=78 dbp2=75 dbp3=86 rat1=1.5256410256 rat2=. rat3=.
rep=1 _ERROR_=0 _N_=2
id=2 sbp1=119 sbp2=123 sbp3=127 dbp1=78 dbp2=75 dbp3=86 rat1=1.5256410256 rat2=1.64
rat3=. rep=2 _ERROR_=0 _N_=2
id=2 sbp1=119 sbp2=123 sbp3=127 dbp1=78 dbp2=75 dbp3=86 rat1=1.5256410256 rat2=1.64
rat3=1.476744186 rep=3 _ERROR_=0 _N_=2
NOTE: There were 2 observations read from the dataset IN.FOO9.
NOTE: The data set WORK.ONE has 2 observations and 11 variables.
NOTE: DATA statement used:
      real time          0.05 seconds
```

```
149 proc print;
150 run;
```

```
NOTE: There were 2 observations read from the dataset WORK.ONE.
NOTE: PROCEDURE PRINT used:
      real time          0.00 seconds
```

Obs	id	sbp1	sbp2	sbp3	dbp1	dbp2	dbp3	rat1	rat2	rat3	rep
1	1	120	115	132	87	70	79	1.37931	1.64286	1.67089	4
2	2	119	123	127	78	75	86	1.52564	1.64000	1.47674	4

Program:

```

libname in 'p:bio113';
data one; set in.foo10;
* notice NO drop statement;
array MISS a--e;
do over MISS;
    if MISS=9999 then MISS=.;
end;
run;

proc print; run;

```

Values of variables in foo10.sas7bdat

id	a	b	c	d	e
1	229	9999	871	381	173
2	9999	637	377	9999	383

Result:

Obs	id	a	b	c	d	e
1	1	229	.	871	381	173
2	2	.	637	377	.	383

Program:

```

libname in 'p:\bio113';
data one; set in.foo9;
* notice NO drop statement;
array SS sbp1 sbp2 sbp3;
array DD dbp1 dbp2 dbp3;
array RR rat1 rat2 rat3;
do over RR; * or SS or DD;
    RR=SS/DD;
end;
run;

proc print;
run;

```

Values of variables in foo9.sas7bdat

id	sbp1	sbp2	sbp3	dbp1	dbp2	dbp3
1	120	115	132	87	70	79
2	119	123	127	78	75	86

Result:

Obs	id	sbp1	sbp2	sbp3	dbp1	dbp2	dbp3	rat1	rat2	rat3
1	1	120	115	132	87	70	79	1.37931	1.64286	1.67089
2	2	119	123	127	78	75	86	1.52564	1.64000	1.47674

Create observations: Up to now, we've read in a single observation and placed a single observation in a SAS data set or we've deleted it (subsetting IF). An OUTPUT statement *inside* the DO...END loop that processes an array, will cause an observation to be output *each time* the array is processed so that multiple observations are placed in a new SAS data set. This program *changes the unit of observation* in the data set from a subject to a subject-visit.

Example:

Program:

```
libname in 'p:\bio113';
data one; set in.foo11;

array WW(3) wt1-wt3;
do visit=1 to 3;
  wt=WW(visit);
  output;
end;
run;
proc print;
run;
```

Values of variables in foo11.sas7bdat

id	wt1	wt2	wt3
1	88	90	97
2	91	85	87

Result:

Obs	id	wt1	wt2	wt3	visit	wt
1	1	88	90	97	1	88
2	1	88	90	97	2	90
3	1	88	90	97	3	97
4	2	91	85	87	1	91
5	2	91	85	87	2	85
6	2	91	85	87	3	87

Program:

```
libname in 'p:\bio113';
data one; set in.foo11;
drop wt1-wt3; * but don't drop visit;
array WW(3) wt1-wt3;
do visit=1 to 3;
  wt=WW(visit);
  output;
end;
run;
proc print;
run;
```

Result:

Obs	id	visit	wt
1	1	1	88
2	1	2	90
3	1	3	97
4	2	1	91
5	2	2	85
6	2	3	87

Program:

```
libname in 'p:\bio113';
data one; set in.fool1;
drop wt1-wt3;
array WW wt1-wt3;
do over WW;
    wt=WW;
    visit=_i_;
    output;
end;
run;
proc print;
run;
```

Result:

Obs	id	visit	wt
1	1	1	88
2	1	2	90
3	1	3	97
4	2	1	91
5	2	2	85
6	2	3	87

Program:

```
* SAS log with put _all_;
libname in 'p:\bio113';
data one; set in.fool1;
drop wt1-wt3;
array WW wt1-wt3;
do over WW;
    wt=WW;
    visit=_i_;
    put _all_;
    output;
end;
run;
```

From the SAS log:

```
id=1 wt1=88 wt2=90 wt3=97 _I_=1 wt=88 visit=1 _ERROR_=0 _N_=1
id=1 wt1=88 wt2=90 wt3=97 _I_=2 wt=90 visit=2 _ERROR_=0 _N_=1
id=1 wt1=88 wt2=90 wt3=97 _I_=3 wt=97 visit=3 _ERROR_=0 _N_=1
id=2 wt1=91 wt2=85 wt3=87 _I_=1 wt=91 visit=1 _ERROR_=0 _N_=2
id=2 wt1=91 wt2=85 wt3=87 _I_=2 wt=85 visit=2 _ERROR_=0 _N_=2
id=2 wt1=91 wt2=85 wt3=87 _I_=3 wt=87 visit=3 _ERROR_=0 _N_=2
```

SORTING AND PRINTING DATA WITH PROCS: (LSB 4.1-4.7)

Basics: Procedures process the *most recently created* SAS data set unless you explicitly name another one
 Most procedures produce some sort of a report in the OUTPUT window but SORT doesn't so we must print to see what sort has done
 SORT *modifies* the data set—it changes the order of observations in the data set (verify this by looking at the data with VIEW between sorts).
 Process a subset (without changing the SAS data set) with WHERE

Example: Create a SAS data set then PROC it (this is one SAS session)

```
* create a SAS data set and run PROCS;
libname in 'p:\bio113';
data in.moredogs;
infile 'p:\bio113\ex25.dat';
input name $ age sex ribbons;
run;
```

Data:

```
spot 4 1 5
dot 2 0 3
cleo 1 0 8
oreo 1 1 4
```

PROCedure statements

Result:

```
proc sort data=in.moredogs;
  by name;
run;
proc print;
run;
```

Obs	name	age	sex	ribbons
1	cleo	1	0	8
2	dot	2	0	3
3	oreo	1	1	4
4	spot	4	1	5

```
proc sort; by descending ribbons;
run;
proc print;
  var name age ribbons;
run;
```

Obs	name	age	ribbons
1	cleo	1	8
2	spot	4	5
3	oreo	1	4
4	dot	2	3

```
proc sort; by age descending ribbons;
run;
proc print;
  var name age ribbons;
run;
```

Obs	name	age	ribbons
1	cleo	1	8
2	oreo	1	4
3	dot	2	3
4	spot	4	5

PROCedure statements

Result:

```
proc sort; by sex name;
run;
proc print; by sex; id name;
  sum ribbons;
  var age ribbons;
run;
```

```
sex=0
name    age    ribbons
cleo    1      8
dot     2      3
-----
sex                    11
```

```
sex=1
name    age    ribbons
oreo    1      4
spot    4      5
-----
sex                    9
=====
                    20
```

```
proc print; where sex=0;
  var name ribbons;
run;
```

```
Obs    name    ribbons
1      cleo    8
2      dot     3
```

```
proc sort nodupkey out=one;
  by age;
run;
proc print;
run;
```

```
Obs    name    age    sex    ribbons
1      cleo    1      0      8
2      dot     2      0      3
3      spot    4      1      5
```

Dress up the output:

Label: Fuller description of variables
 Each label may be up to 256 characters in length including blanks
 Must be enclosed in single or double quotes
 Remains with variable if stated in the DATA step
 Disappears after step if stated in the PROC step

Example: label ribbons='Number of first place ribbons';
 label age="Dog's age" sex='Dog''s sex';

Title: Label top of page

Footnote: Label bottom of page

Each of 10 titles and footnotes may be up to 256 characters including blanks
 Must be enclosed in matched single or double quotes
 Appear anywhere in program and remain until replaced by another

To return to SAS defaults: TITLE;
 FOOTNOTE;

Example: title 'Ourtown Dog Show';
 title1 'Ourtown Dog Show';
 title2 'Winner''s Summary';
 footnote 'Stats compiled by judges';

Program (continue with moredogs.sas7bdat. Note: data= is required):

```
* print in.moredogs (previous page) with titles, footnote, labels;
proc print data=in.moredogs label; id name;
  label ribbons='Number of first place ribbons'
        age="Dog's age" sex='Dog''s sex';
  title1 'Ourtown Dog Show';
  title2 'Winner''s Summary';
  footnote 'Stats compiled by judges';
run;
```

Result:

Ourtown Dog Show
 Winner's Summary

name	Dog's age	Dog's sex	Number of first place ribbons
cleo	1	0	8
dot	2	0	3
oreo	1	1	4
spot	4	1	5

Stats compiled by judges

Format: Label values of variables

Use SAS built-in formats (date, time, dollars, etc)

Create your own with PROC FORMAT (Up to 200 characters and spaces enclosed in quotes although truncated in many PROCs)

Remains with variable and *change how it's stored* if stated in the DATA step

Disappears after step if stated in the PROC step

Example: Use SAS formats

Data

Program:

```
data one;
infile 'p:\bio113\ex26.dat';
input name $5. weight 6. dob date9.;
run;
proc print;
    var name weight dob;
    format weight 4.1 dob mmddyy10.;
run;
```

```
harry3.565 9jun1996
mary 10.33122nov1994
```

Result:

Obs	name	weight	dob
1	harry	3.6	06/09/1996
2	mary	10.3	11/22/1994

Example: Create your own formats

Format names, which appear after the key word VALUE, may not exceed 8 characters and may not end with a number.

Program:

```
* print variables in in.moredogs;
libname in 'p:\bio113';
proc format;
    value sex 0='female' 1='male';
    value age 0-1='puppy' 2-high='adult';
run;
proc print data=in.moredogs;
    var name sex age;
    format sex sex. age age.;
run;
proc print data=in.moredogs;
    var name sex age;
    format sex sex1. age age3.;
run;
```

Result:

Obs	name	sex	age
1	cleo	female	puppy
2	dot	female	adult
3	oreo	male	puppy
4	spot	male	adult

Obs	name	sex	age
1	cleo	f	pup
2	dot	f	adu
3	oreo	m	pup
4	spot	m	adu

Create your own permanent format library:

If you find yourself using the same formats over and over again, you might want to put them into your own private format library. The libname statement tells SAS where to put the format library. The format library will always have the name formats.sas7bcat (the suffix is sas7bcat, not sas7bdat, which reminds SAS that this special data set contains formats not data).

Example: Create a *permanent* format library

Program:

```
libname mylib 'p:\bio113';
proc format library=mylib;
    value sex 0='female' 1='male';
    value age 0-1='puppy' 2-high='adult';
run;
```

Example: Return later and print variables using the format library. When you create the format library you may choose the *libref* (I used mylib). However, when you use a format library, its ***libref must be library***.

Program:

```
* print variables in in.moredogs;
libname in 'p:\bio113';
libname library 'p:\bio113';
data one; set in.moredogs;
run;

proc print;
    var name sex age;
    format sex sex. age age.;
run;
```

Result:

Obs	name	sex	age
1	cleo	female	puppy
2	dot	female	adult
3	oreo	male	puppy
4	spot	male	adult

SIMPLE DATA SUMMARIZATION: (LSB 4.9-4.11, 4.14, 7.2-7.3)

First, create a SAS data set:

```
* create a data set for summarization;
data one;
infile 'p:\bio113\ex27.dat';

input id 1-2 sex $3 ga 4-5 bw 6-9 (cs ivh)(1.);

label ga='gestational age'
      bw='birth weight'
      cs='Cesarian delivery'
      ivh='intraventricular hemorrhage';
run;

proc print label; id id;
  title 'data set for summarization';
run;
```

Data:

```
1 m31146200
2 f25 56411
3 f27 93101
4 f28100210
5 m28106711
6 f32132700
7 m26 88001
8 f33165000
9 m29125010
10m26 73601
11f30121200
```

Result:

data set for summarization

id	sex	gestational age	birth weight	Cesarian delivery	intraventricular hemorrhage
1	m	31	1462	0	0
2	f	25	564	1	1
3	f	27	931	0	1
4	f	28	1002	1	0
5	m	28	1067	1	1
6	f	32	1327	0	0
7	m	26	880	0	1
8	f	33	1650	0	0
9	m	29	1250	1	0
10	m	26	736	0	1
11	f	30	1212	0	0

Summarize categorical data - PROC FREQ:

Create univariate, bivariate, multivariate tables

Statistics include Fisher's exact test, X^2 , Mantel-Haenszel X^2 , Likelihood Ratio X^2

Specify [page-variable] * row-variable * column-variable

Appearance is greatly improved by labels and formats.

Program continues:

```
proc format;
    value yn 0='No' 1='Yes';
run;

proc freq;                                * NOTE: using work.one *;
    tables cs sex ivh;
    tables sex*ivh / chisq;
    tables sex*cs*ivh / expected;
format ivh cs yn.;
run;
```

Result:

The FREQ Procedure

Cesarian delivery

cs	Frequency	Percent	Cumulative Frequency	Cumulative Percent
No	7	63.64	7	63.64
Yes	4	36.36	11	100.00

sex	Frequency	Percent	Cumulative Frequency	Cumulative Percent
f	6	54.55	6	54.55
m	5	45.45	11	100.00

intraventricular hemorrhage

ivh	Frequency	Percent	Cumulative Frequency	Cumulative Percent
No	6	54.55	6	54.55
Yes	5	45.45	11	100.00

Table of sex by ivh

sex ivh(intraventricular hemorrhage)

Frequency				
Percent				
Row Pct				
Col Pct	No	Yes		Total
-----+-----+-----+-----+-----				
f	4	2		6
	36.36	18.18		54.55
	66.67	33.33		
	66.67	40.00		
-----+-----+-----+-----+-----				
m	2	3		5
	18.18	27.27		45.45
	40.00	60.00		
	33.33	60.00		
-----+-----+-----+-----+-----				
Total	6	5		11
	54.55	45.45		100.00

Statistics for Table of sex by ivh

Statistic	DF	Value	Prob
Chi-Square	1	0.7822	0.3765
Likelihood Ratio Chi-Square	1	0.7899	0.3741
Continuity Adj. Chi-Square	1	0.0764	0.7823
Mantel-Haenszel Chi-Square	1	0.7111	0.3991
Phi Coefficient		0.2667	
Contingency Coefficient		0.2577	
Cramer's V		0.2667	

WARNING: 100% of the cells have expected counts less than 5. Chi-Square may not be a valid test.

Fisher's Exact Test

Cell (1,1) Frequency (F)	4
Left-sided Pr <= F	0.9329
Right-sided Pr >= F	0.3918
Table Probability (P)	0.3247
Two-sided Pr <= P	0.5671

Sample Size = 11

The FREQ Procedure

Table 1 of cs by ivh
Controlling for sex=f

cs(Cesarian delivery)		ivh(intraventricular hemorrhage)		
Frequency	Expected	Percent	Row Pct	Col Pct
			No	Yes
			Total	
No	3	1	4	
	2.6667	1.3333		
	50.00	16.67	66.67	
	75.00	25.00		
	75.00	50.00		
Yes	1	1	2	
	1.3333	0.6667		
	16.67	16.67	33.33	
	50.00	50.00		
	25.00	50.00		
Total	4	2	6	
	66.67	33.33	100.00	

The FREQ Procedure

Table 2 of cs by ivh
Controlling for sex=m

cs(Cesarian delivery)		ivh(intraventricular hemorrhage)		
Frequency	Expected	Percent	Row Pct	Col Pct
			No	Yes
			Total	
No	1	2	3	
	1.2	1.8		
	20.00	40.00	60.00	
	33.33	66.67		
	50.00	66.67		
Yes	1	1	2	
	0.8	1.2		
	20.00	20.00	40.00	
	50.00	50.00		
	50.00	33.33		
Total	2	3	5	
	40.00	60.00	100.00	

Summarize continuous data - PROC MEANS:

Calculates mean, standard deviation, variance, etc. for one or more variables

Use with BY-variable if data set is sorted by BY-variable

Perform paired t-test

Summaries may be output to a new SAS data set

Program:

```
proc means maxdec=1;          *NOTE: still using work.one *;
    var ga bw;
run;
```

Result:

Variable	Label	N	Mean	Std Dev	Minimum
ga	gestational age	11	28.6	2.6	25.0
bw	birth weight	11	1098.3	320.6	564.0

Variable	Label	Maximum
ga	gestational age	33.0
bw	birth weight	1650.0

Program:

```
proc sort; by ivh;
run;
proc means maxdec=1; by ivh;    * with by variable *;
    var ga bw;
    format ivh yn.;
run;
```

Result:

intraventricular hemorrhage=No

Variable	Label	N	Mean	Std Dev	Minimum
ga	gestational age	6	30.5	1.9	28.0
bw	birth weight	6	1317.2	222.1	1002.0

Variable	Label	Maximum
ga	gestational age	33.0
bw	birth weight	1650.0

intraventricular hemorrhage=Yes

Variable	Label	N	Mean	Std Dev	Minimum
ga	gestational age	5	26.4	1.1	25.0
bw	birth weight	5	835.6	192.6	564.0

Variable	Label	Maximum
ga	gestational age	28.0
bw	birth weight	1067.0

Program:

```
proc means maxdec=1; class ivh;    * with class variable (no sort) *;
    var ga bw;
    format ivh yn.;
run;
```

Result:

intraventricular hemorrhage	N Obs	Variable	Label	N	Mean	Std Dev
No	6	ga	gestational age	6	30.5	1.9
		bw	birth weight	6	1317.2	222.1
Yes	5	ga	gestational age	5	26.4	1.1
		bw	birth weight	5	835.6	192.6

intraventricular hemorrhage	N Obs	Variable	Label	Minimum	Maximum
No	6	ga	gestational age	28.0	33.0
		bw	birth weight	1002.0	1650.0
Yes	5	ga	gestational age	25.0	28.0
		bw	birth weight	564.0	1067.0

Program:

```
* with output of specific statistics to a new data set (two);
proc means noprint mean std; by ivh;
    var ga bw;
    output out=two
        mean=ga bw std=ga_s bw_s;
run;
proc print data=two;
run;
```

Result:

Obs	ivh	_TYPE_	_FREQ_	ga	bw	ga_s	bw_s
1	0	0	6	30.5	1317.17	1.87083	222.150
2	1	0	5	26.4	835.60	1.14018	192.552

Program:

```
proc means noprint mean std data=one; class ivh;
    var ga bw;
    output out=two
        mean=ga bw std=ga_s bw_s;
run;
proc print data=two;
run;
```

Result:

Obs	ivh	_TYPE_	_FREQ_	ga	bw	ga_s	bw_s
1	.	0	11	28.6364	1098.27	2.61812	320.551
2	0	1	6	30.5000	1317.17	1.87083	222.150
3	1	1	5	26.4000	835.60	1.14018	192.552

Display relationships between variables - PROC PLOT:

Produce Y by X scatter plots

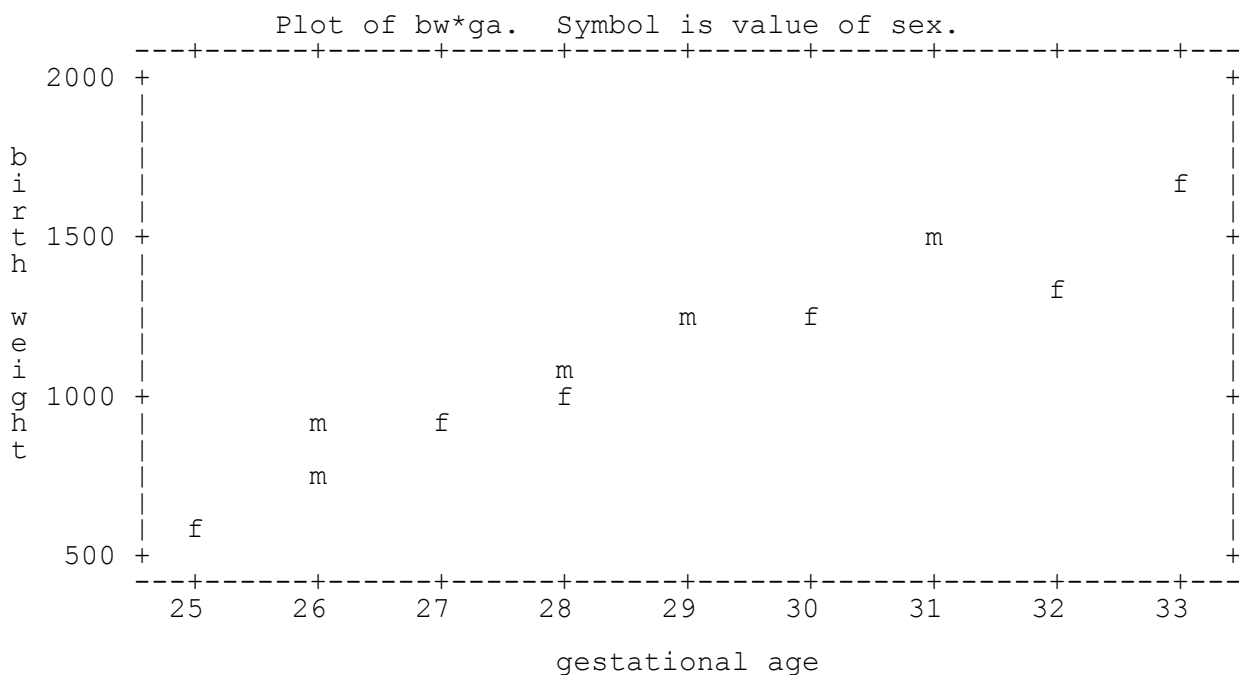
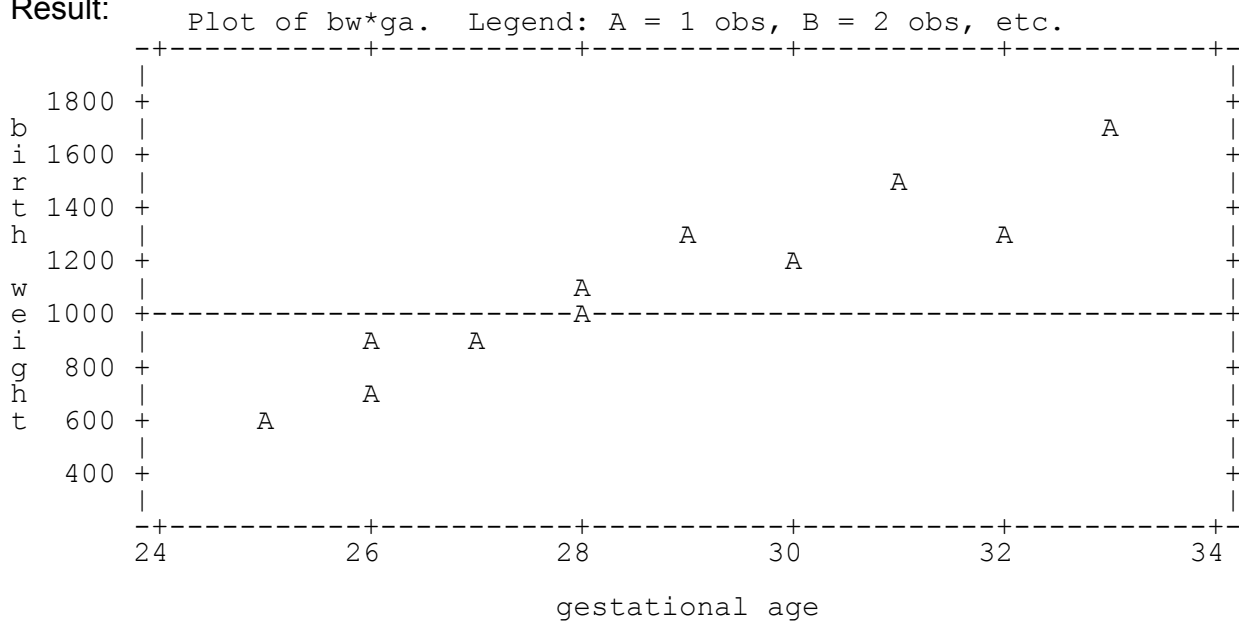
Specify plotting symbol(s)

Specify axis range and intervals (tick marks)

Program:

```
proc plot data=one; * NOTE still using work.one *;
  plot bw*ga / haxis=24 to 34 by 2
    vaxis=400 to 1800 by 200 vref=1000;
run;
proc plot data=one;
  plot bw*ga = sex;
run;
```

Result:



PROC GPLOT: High resolution plotting in SAS:

GPLOT (graphic plot) makes plots in which points are not limited to locations determined by dot matrix printers. The plots also can be in color--you can see color on the screen but, obviously, you need a color printer to get a hard copy. What you get, in spite of being 'graphic', is not publication quality. Notice that SAS opens a fourth window, GRAPH, to display the results from gplot.

Examples:

```
libname in 'p:\bio113';

data one; set in.ivh1;
run;

* simple plot - gestational age vs birth weight - See top next page for output;
proc gplot;
  plot ga*bw;
run;

* HIGHLIGHT SEX;
proc gplot;
  plot ga*bw=sex;
run;

* CHANGE SYMBOL COLORS;
proc gplot;
  plot ga*bw=sex;
  symbol1 color=red;
  symbol2 color=blue;
run;

* CHANGE SYMBOL VALUES;
proc gplot;
  plot ga*bw=sex;
  symbol1 color=red v='f';
  symbol2 color=blue v='m';
run;

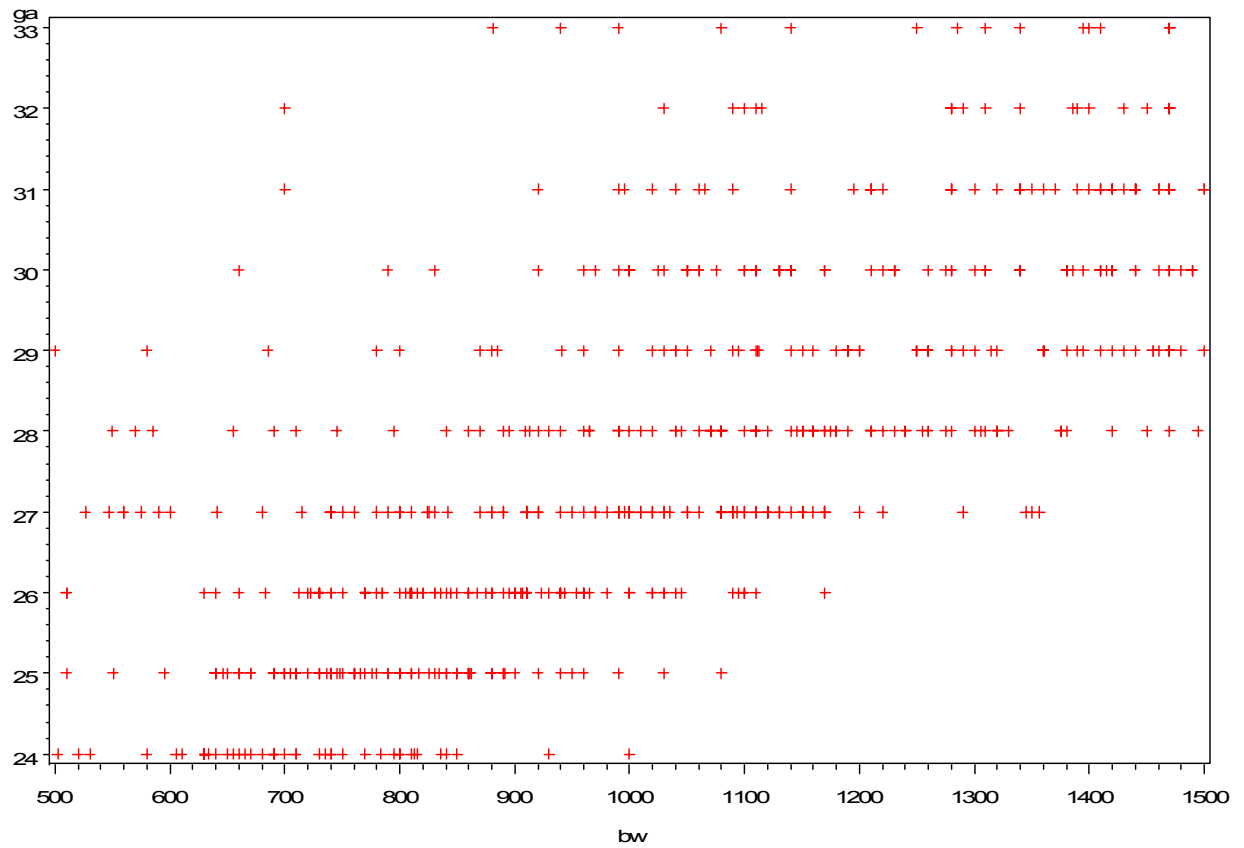
proc gplot;
  plot ga*bw=sex;
  symbol1 v=circle;
  symbol2 v=plus;
run;

* SPRUCE IT UP SOME MORE;
proc gplot;
  plot ga*bw=sex / haxis=axis2 vaxis=axis1;
  title 'Plot of Gestational Age versus Birth Weight by Sex';
  footnote justify=right 'IVH data set';
  axis1 label=('Gestational Age');
  axis2 label=('Birth Weight');
run;

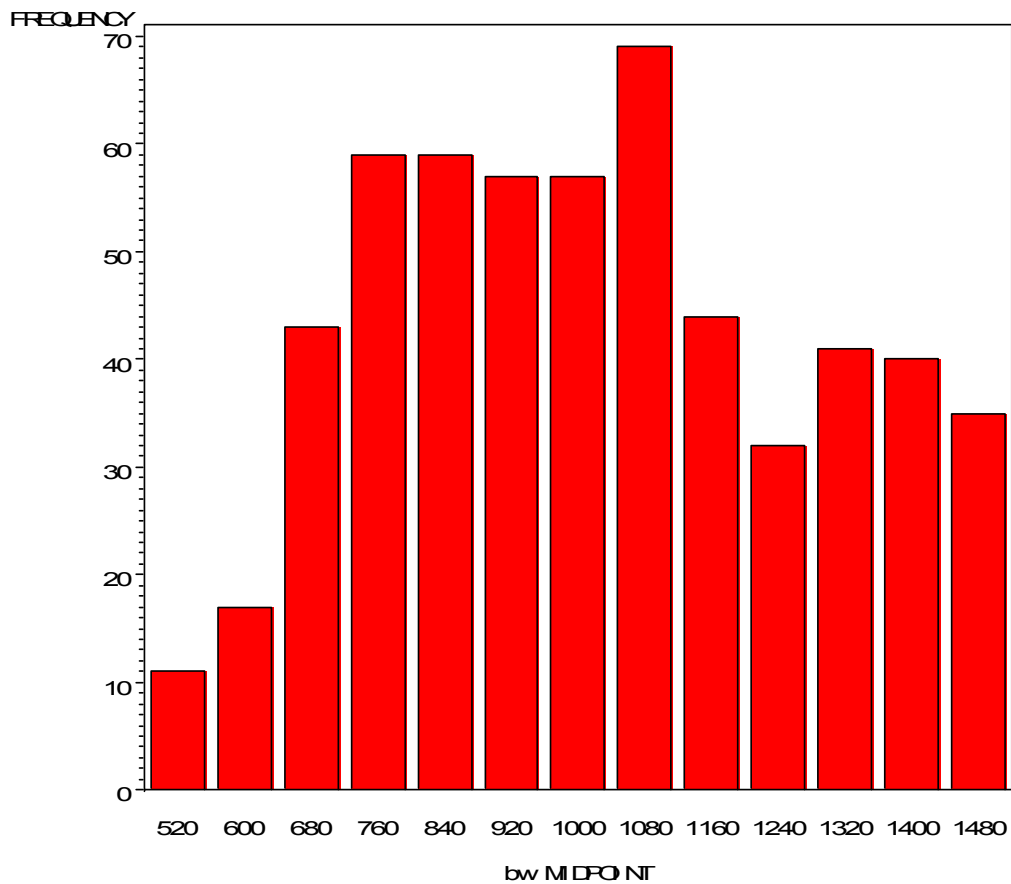
* ROTATE THE Y-AXIS LABEL;
proc gplot;
  plot ga*bw=sex / haxis=axis2 vaxis=axis1;
  axis1 label=(a=90 'Gestational Age');
  title;
  footnote;
run;

* HISTOGRAM OF BIRTH WEIGHT - See bottom next page for output;
proc gchart;
  vbar bw;
  title 'Histogram of Birth Weight';
```

run;



Histogram of Birth Weight



DATA MANAGEMENT I - SET: (LSB 5.1-5.3)

Basics: SET reads *existing* SAS data sets *not* 'raw' data
 Reads in observations, sequentially, one at a time
 May read and modify a single data set
 May read, combine (concatenate) and modify multiple data sets

First, create 2 SAS data sets (work.one and work.two):

```
* create two SAS data sets for data management tasks;
```

```
data one;
infile 'p:\bio113\ex28.dat';
input id 1-2 ga 3-4 bw 5-8 cs 9;
run;

data two;
infile 'p:\bio113\ex29.dat';
input id ga bw cs acs;
run;

proc print data=one; title 'one'; run;
proc print data=two; title 'two'; run;
```

Data:

```
1 3114621
8 16500
3 25 931
6 3213270

7 26 880 0 1
10 26 736 . 1
2 25 564 1 0
11 30 1212 1 0
```

Result:

one

Obs	id	ga	bw	cs
1	1	31	1462	1
2	8	.	1650	0
3	3	25	931	.
4	6	32	1327	0

two

Obs	id	ga	bw	cs	acs
1	7	26	880	0	1
2	10	26	736	.	1
3	2	25	564	1	0
4	11	30	1212	1	0

Concatenate 2 SAS data sets:

Example: Concatenate SAS data sets, work.one and work.two, with SET.

```
libname in 'p:\bio113';
data in.babyl; set one two; * creates babyl *;
run;
proc print; title 'one and two concatenated'; run;
```

Result:

one and two concatenated

Obs	id	ga	bw	cs	acs
1	1	31	1462	1	.
2	8	.	1650	0	.
3	3	25	931	.	.
4	6	32	1327	0	.
5	7	26	880	0	1
6	10	26	736	.	1
7	2	25	564	1	0
8	11	30	1212	1	0

Interleave 2 SAS data sets:

Example: Repeat, but this time interleave the 2 SAS data sets by ID and create the new variable, gacat. Both data sets need to be sorted before interleaving.

```
proc sort data=one; by id; run;
proc sort data=two; by id; run;

data in.babyl; set one two; by id; * replaces old babyl with new one *;

if . < ga < 30 then gacat='younger';
else if ga >= 30 then gacat='older';

run;
proc print; title 'one and two interleaved'; run;
```

Result:

one and two interleaved

Obs	id	ga	bw	cs	acs	gacat
1	1	31	1462	1	.	older
2	2	25	564	1	0	younger
3	3	25	931	.	.	younger
4	6	32	1327	0	.	older
5	7	26	880	0	1	younger
6	8	.	1650	0	.	
7	10	26	736	.	1	younger
8	11	30	1212	1	0	older

Create a subset:

Example: Now let's create a new temporary data set, `work.older`, that's a subset of the data set `in.baby1` created in the previous step. The subsetting IF statement tells SET to bring in a subset of observations, those with `gacat='older'`.

```
data older; set in.baby1;
if gacat='older';
run;

proc print; title 'older babies';
run;
```

Result:

older babies

Obs	id	ga	bw	cs	acs	gacat
1	1	31	1462	1	.	older
2	6	32	1327	0	.	older
3	11	30	1212	1	0	older

Example: Now let's create a new temporary data set, `work.cs`, that's a different subset of `work.baby1`. The subsetting IF statement tells SET to bring in the subset of observations with non-missing `cs`.

```
data cs; set in.baby1;
if cs > .; *** could be written: if cs ^= .; ***;
run;

proc print; title 'babies not missing on cs';
run;
```

Result:

babies not missing on cs

Obs	id	ga	bw	cs	acs	gacat
1	1	31	1462	1	.	older
2	2	25	564	1	0	younger
3	6	32	1327	0	.	older
4	7	26	880	0	1	younger
5	8	.	1650	0	.	.
6	11	30	1212	1	0	older

DATA MANAGEMENT II - MERGE: (LSB 5.4-5.7, 5.9-5.10, 5.14)

Basics: MERGE reads *existing* SAS data sets *not* 'raw' data
 Reads in observations, sequentially, one at a time
 Reads, combines (joins) and modifies multiple data sets
 Meaningful joining requires one or more BY variables
 Joining is one-to-one, one-to-many or many-to one but *never* many-to-many

First use SET to get a different subset of observations and variables from work.baby1 and then create a second SAS data set. Notice KEEP= option and DROP statement.

```
* make data sets for merge examples;

data one; set in.baby1 (keep=id bw ga acs);
drop bw;
if . < bw < 1000;
run;

data two;
infile 'p:\bio113\ex30.dat';
input id momid colloid1-colloid3 acs;
run;

proc print data=one; title 'old data'; run;
proc print data=two; title 'new data'; run;
```

Data:

```
2 2 9 15 16 1
10 8 22 11 19 .
3 2 4 . 9 1
```

Result:

old data

Obs	id	ga	acs
1	2	25	0
2	3	25	.
3	7	26	1
4	10	26	1

new data

Obs	id	momid	colloid1	colloid2	colloid3	acs
1	2	2	9	15	16	1
2	10	8	22	11	19	.
3	3	2	4	.	9	1

One-to-one join:

Example: Perform a one-to-one merge, matching on ID. Data sets must be sorted prior to merge. Note **RENAME** statement (rename old-varname=new-varname...;).

```
proc sort data=one; by id; run;
proc sort data=two; by id; run;

data in.baby2; merge one two; by id;
rename colloid1=coll colloid2=col2 colloid3=col3;
run;
proc print; title 'baby2 - merged data set'; run;
```

Result:

baby2 - merged data set

Obs	id	ga	acs	momid	coll	col2	col3
1	2	25	1	2	9	15	16
2	3	25	1	2	4	.	9
3	7	26	1
4	10	26	.	8	22	11	19

One-to-many join:

Example: Create a SAS data set with information about mothers, sort it and in.baby2 by MOMID, and join the two by MOMID.

```
data mom;
infile 'p:\bio113\ex31.dat';
input momid age;
run;

proc sort data=in.baby2; by momid; run;
proc sort data=mom; by momid; run;

proc print data=in.baby2; title 'baby2'; run;
proc print data=mom; title 'mom'; run;

data in.baby3; merge in.baby2 mom; by momid;
run;
proc print; title 'baby3 - merge baby2 and mom data'; run;
```

Data:

```
2 38
5 34
8 29
```

Result:

baby2

Obs	id	ga	acs	momid	coll	col2	col3
1	7	26	1
2	2	25	1	2	9	15	16
3	3	25	1	2	4	.	9
4	10	26	.	8	22	11	19

mom

Obs	momid	age
1	2	38
2	5	34
3	8	29

baby3 - merge baby2 and mom data

Obs	id	ga	acs	momid	coll	col2	col3	age
1	7	26	1
2	2	25	1	2	9	15	16	38
3	3	25	1	2	4	.	9	38
4	.	.	.	5	.	.	.	34
5	10	26	.	8	22	11	19	29

Join with IN= option: Creates a special **temporary** SAS variable that indicates whether the data set contributed data to the current observation. In the data step, the value of the variable is 1 if the data set contributed data to the current observation, and 0 otherwise. The variable is available during the data step but is not placed in the output data set.

Example: Notice that obs 1 in in.baby3 has no data from work.mom and obs 4 has no data from in.baby2. Now repeat the merge with the IN= option.

```
* Merge with in= option;
data in.baby3; merge in.baby2 (in=in1) mom (in=in2); by momid;
if in1 and in2;
run;

proc print; title 'babies have moms and moms have babies'; run;
```

Result:

babies have moms and moms have babies

Obs	id	ga	acs	momid	col1	col2	col3	age
1	2	25	1	2	9	15	16	38
2	3	25	1	2	4	.	9	38
3	10	26	.	8	22	11	19	29

Look at what IN= is doing by creating variables that hold the values of the IN= variables:

```
* merge DOES NOT make use of IN=;
data in.baby3; merge in.baby2 (in=in1) mom (in=in2); by momid;
baby=in1;
mom=in2;
run;
proc print; title; run;
```

Obs	id	ga	acs	momid	col1	col2	col3	age	baby	mom
1	7	26	1	1	0
2	2	25	1	2	9	15	16	38	1	1
3	3	25	1	2	4	.	9	38	1	1
4	.	.	.	5	.	.	.	34	0	1
5	10	26	.	8	22	11	19	29	1	1

```
* merge uses IN= for BOTH data sets;
data in.baby3; merge in.baby2 (in=in1) mom (in=in2); by momid;
if in1 and in2;
baby=in1;
mom=in2;
run;
proc print; run;
```

Obs	id	ga	acs	momid	col1	col2	col3	age	baby	mom
1	2	25	1	2	9	15	16	38	1	1
2	3	25	1	2	4	.	9	38	1	1
3	10	26	.	8	22	11	19	29	1	1

```
* merge uses IN= for baby2 data set only;
data in.baby3; merge in.baby2 (in=in1) mom (in=in2); by momid;
if in1;
baby=in1;
mom=in2;
run;
proc print; run;
```

Obs	id	ga	acs	momid	col1	col2	col3	age	baby	mom
1	7	26	1	1	0
2	2	25	1	2	9	15	16	38	1	1
3	3	25	1	2	4	.	9	38	1	1
4	10	26	.	8	22	11	19	29	1	1

"Many-to-one join" (doesn't use MERGE!):

Example: Find how GA and BW of the CS babies differs from the average for the group. Use SET, not MERGE, and a SAS automatic variable, `_n_`. Like the variable created with `IN=`, `_n_` is **temporary** and does not appear in your SAS data set.

babyl

Obs	id	ga	bw	cs	acs	gacat
1	1	31	1462	1	.	older
2	2	25	564	1	0	younger
3	3	25	931	.	.	younger
4	6	32	1327	0	.	older
5	7	26	880	0	1	younger
6	8	.	1650	0	.	
7	10	26	736	.	1	younger
8	11	30	1212	1	0	older

Program:

```
data diffs; set in.babyl;
keep id ga bw;
if cs > .;
run;

proc means noprint mean;
  var ga bw;
  output out=avg mean=xga xbw;
run;

data diffs; set diffs;
drop _type_ _freq_;

if _n_=1 then set avg;

gadiff=xga-ga;
bwdiff=xbw-bw;

run;

proc print; title 'ga and bw differences'; run;
```

Result:

ga and bw differences

Obs	id	ga	bw	xga	xbw	gadiff	bwdiff
1	1	31	1462	28.8	1182.5	-2.2	-279.5
2	2	25	564	28.8	1182.5	3.8	618.5
3	6	32	1327	28.8	1182.5	-3.2	-144.5
4	7	26	880	28.8	1182.5	2.8	302.5
5	8	.	1650	28.8	1182.5	.	-467.5
6	11	30	1212	28.8	1182.5	-1.2	-29.5

DATA MANAGEMENT III - OUTPUT: (LSB 5.11)

Creates one or more SAS data sets from a single SAS data set

Overrides SAS implicit output

Example: Return to baby1 and create separate SAS data sets for babies who were delivered by C-section and those who were not.
Notice where we use the KEEP= (or DROP=) option to keep **different** variables in the two datasets.

Program:

```
proc print data=in.baby1; title 'baby1'; run;

data cs0 (keep=id ga acs) cs1 (keep=id bw acs); set in.baby1;

if cs=0 then output cs0;
else if cs=1 then output cs1;
else delete;
run;

proc print data=cs0; title 'cs no'; run;
proc print data=cs1; title 'cs yes'; run;
```

Result:

baby1

Obs	id	ga	bw	cs	acs	gacat
1	1	31	1462	1	.	older
2	2	25	564	1	0	younger
3	3	25	931	.	.	younger
4	6	32	1327	0	.	older
5	7	26	880	0	1	younger
6	8	.	1650	0	.	
7	10	26	736	.	1	younger
8	11	30	1212	1	0	olde

cs no

Obs	id	ga	acs
1	6	32	.
2	7	26	1
3	8	.	.

cs yes

Obs	id	bw	acs
1	1	1462	.
2	2	564	0
3	11	1212	0

DATA MANAGEMENT IV– CHANGE THE UNIT OF OBSERVATION (LSB 3.9-3.10, 5.12-5.14)

Make several observations from one:

Example: Each baby has 3 colloid measures. Make an observation for each non-missing colloid value for a baby by using OUTPUT inside the DO...END processing of an array. This changes the **unit of observation** from a baby to a baby-measurement time.

```
data one; set in.baby3 (keep=id momid col1-col3); if momid=2;
drop momid;
run;
```

```
proc print; title 'short fat data set'; run;
```

```
data two; set one;
keep id col time;
```

```
array CC(3) col1-col3;
do time=1 to 3;
    col=CC(time);
    if col > . then output;
end;
```

```
run;
```

```
proc print; title 'long skinny data set'; run;
```

Result:

short wide data set

Obs	id	col1	col2	col3
1	2	9	15	16
2	3	4	.	9

long skinny data set

Obs	id	time	col
1	2	1	9
2	2	2	15
3	2	3	16
4	3	1	4
5	3	3	9

Make one observation from several:

Example: Undo what was just done! Data set must be sorted by the id variable (ID) and the indexing variable (TIME).

```
* make short data set from long one;
* notice: SET with BY;
*         RETAIN;
*         ARRAY processed without DO...END;
*         LAST.id;

proc sort data=two; by id time;

data three; set two; by id time;

drop col time;

retain col1 col2; * NOTE: col3 does NOT have to be retained *;

array DD(3) col1-col3;

DD(time)=col;

if last.id then do;
    output;
    do time=1 to 3;
        DD(time)=.;
    end;
end;

run;

proc print; title 'short wide data set again'; run;
```

Result:

short wide data set again

Obs	id	col1	col2	col3
1	2	9	15	16
2	3	4	.	9

DATA ANALYSIS - PROC STEP (LSB 7.1, 7.4-7.8, SAS User's Guide: Statistics)

PROCs work on *existing* SAS data sets *not* 'raw' data

Some PROCs (e.g., print, freq, means, univariate) will PROC the entire data set (actually, only numeric variables) unless you supply a list of variables with VAR or TABLES statements.

Other PROCs will fail (e.g., ttest, anova, glm, plot) unless you specify variables to use (PLOT) and/or how to use them (CLASS, MODEL).

Univariate: Examine statistics and distribution of a continuous variable

Example: Are babies receiving more fluids/kg of birth weight on day 4 than on day 1. Is the change normally distributed? Is it significantly different from 0?

```
Program:  * univariate example;
          libname in 'p:\bio113';
          data one; set in.ivh1;
          delta=1000*((fluid4/bw)-(fluid1/bw));
          run;
          proc univariate plot; id id;
              var delta;
          run;
```

Result:

The UNIVARIATE Procedure

Variable: delta

Moments

N	553	Sum Weights	553
Mean	42.9247996	Sum Observations	23737.4142
Std Deviation	30.6322999	Variance	938.337796
Skewness	-0.4271125	Kurtosis	1.15482643
Uncorrected SS	1536886.21	Corrected SS	517962.463
Coeff Variation	71.362709	Std Error Mean	1.30261789

Basic Statistical Measures

Location		Variability	
Mean	42.92480	Std Deviation	30.63230
Median	45.39877	Variance	938.33780
Mode	44.44444	Range	229.25737
		Interquartile Range	36.62313

NOTE: The mode displayed is the smallest of 2 modes with a count of 3.

Tests for Location: Mu0=0

Test	-Statistic-	-----p Value-----	
Student's t	t 32.95272	Pr > t	<.0001
Sign	M 232.5	Pr >= M	<.0001
Signed Rank	S 71756.5	Pr >= S	<.0001

Quantiles (Definition 5)

Quantile	Estimate
100% Max	136.26374
99%	114.98258
95%	88.60759
90%	78.78788
75% Q3	62.26415
50% Median	45.39877
25% Q1	25.64103
10%	4.90196
5%	-14.28571
1%	-36.78161
0% Min	-92.99363

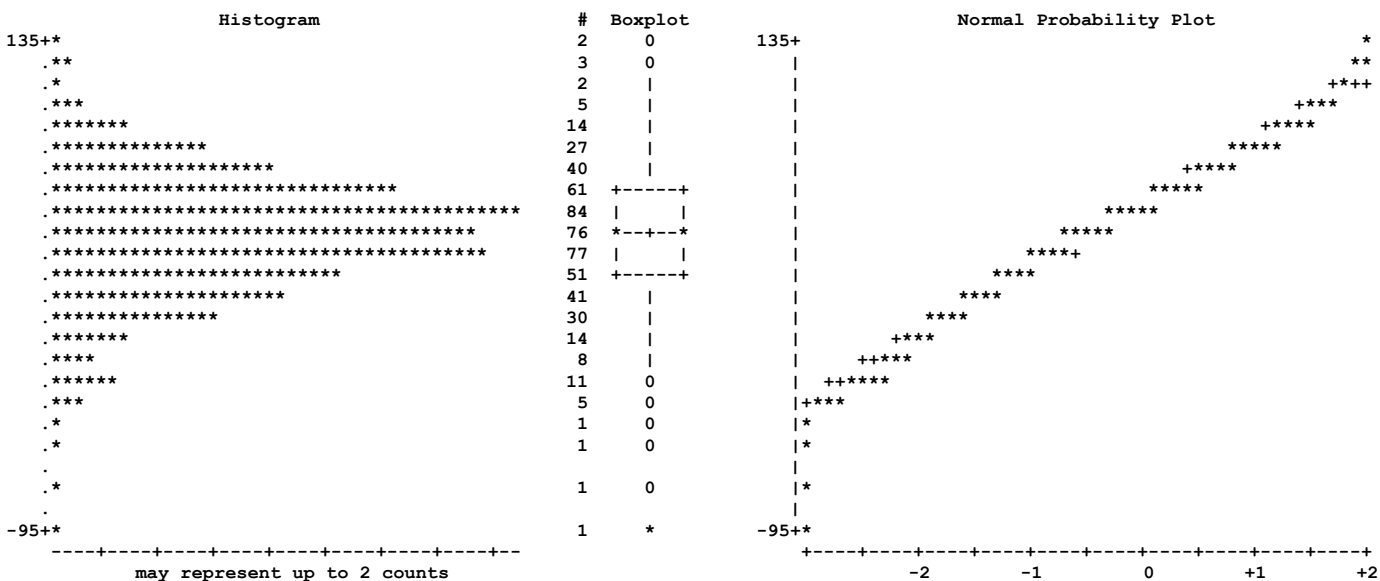
The UNIVARIATE Procedure
Variable: delta

Extreme Observations

-----Lowest-----			-----Highest-----		
Value	id	Obs	Value	id	Obs
-92.9936	110151	9	122.609	111531	110
-75.9398	136971	420	125.974	220151	493
-55.4348	111131	84	128.205	111051	77
-42.0455	110902	66	130.556	136482	389
-39.1892	136161	372	136.264	111521	109

Missing Values

-----Percent Of-----			
Missing Value	Count	All Obs	Missing Obs
.	11	1.94	100.00



T test: Compare mean value for 2 samples (independent t-test)

Example: Do babies with and without IVH have different pCO₂ values on day 1 or day 2?

```

Program:  * ttest example - use work.one from univariate example;
          proc format;
            value yn 0='No' 1='Yes';
          run;

          proc ttest;
            class ivh; format ivh yn.;
            var pco2_1 pco2_2;
          run;

```

Result:**The TTEST Procedure**

Statistics											
Variable	Class	N	Lower CL Mean	Mean	Upper CL Mean	Lower CL Std Dev	Std Dev	Upper CL Std Dev	Std Err	Minimum	Maximum
pco2_1	No	425	32.289	33.179	34.069	8.7435	9.3315	10.005	0.4526	6	68
pco2_1	Yes	110	29.173	30.873	32.572	7.9408	8.9925	10.368	0.8574	15	58
pco2_1	Diff (1-2)		0.3595	2.3061	4.2527	8.7389	9.2632	9.8548	0.9909		
pco2_2	No	297	35.321	36.212	37.104	7.2261	7.8076	8.4916	0.453	15	64
pco2_2	Yes	101	35.191	36.713	38.234	6.7714	7.7076	8.9465	0.7669	16	59
pco2_2	Diff (1-2)		-2.263	-0.501	1.2616	7.2761	7.7824	8.3651	0.8964		

T-Tests

Variable	Method	Variances	DF	t Value	Pr > t
pco2_1	Pooled	Equal	533	2.33	0.0203
pco2_1	Satterthwaite	Unequal	175	2.38	0.0185
pco2_2	Pooled	Equal	396	-0.56	0.5767
pco2_2	Satterthwaite	Unequal	175	-0.56	0.5747

Equality of Variances

Variable	Method	Num DF	Den DF	F Value	Pr > F
pco2_1	Folded F	424	109	1.08	0.6504
pco2_2	Folded F	296	100	1.03	0.8965

Npar1way: Compare medians for two or more groups with the Wilcoxon rank sums test or the Kruskal-Wallis test. Unfortunately, Npar1way does not display the medians only the mean of the ranks for each category (mean score).

Example: Is MAP (mean blood pressure) on day 1 different for babies whose mothers received no, partial or complete antenatal corticosteroid courses?

Program:

```
* npar1way example - use work.one again;
proc format;
    value acs 1="none" 2="partial" 3="complete";
run;

proc npar1way wilcoxon;
    class acs; format acs acs.;
    var map1;
run;
```

Result:

The NPAR1WAY Procedure

Wilcoxon Scores (Rank Sums) for Variable map1
Classified by Variable acs

acs	N	Sum of Scores	Expected Under H0	Std Dev Under H0	Mean Score
partial	130	35420.50	33215.00	1448.78971	272.465385
complete	207	55222.50	52888.50	1632.48186	266.775362
none	173	39662.00	44201.50	1573.90994	229.260116

Average scores were used for ties.

Kruskal-Wallis Test

Chi-Square	8.4380
DF	2
Pr > Chi-Square	0.0147

Reg: Perform univariate or multivariate linear regression. This is one of several SAS procedures for linear regression and is nice because diagnostic plots are built in.

Example: What's the relationship between birth weight and gestational age? Is the relationship linear?

Program:

```
* reg example - use work.one again;
proc reg;
  model bw = ga;
  plot R. * ga;
run;
/* note: proc reg lineprinter; gives dot matrix plot */
```

Result:

The REG Procedure

Model: MODEL1

Dependent Variable: bw

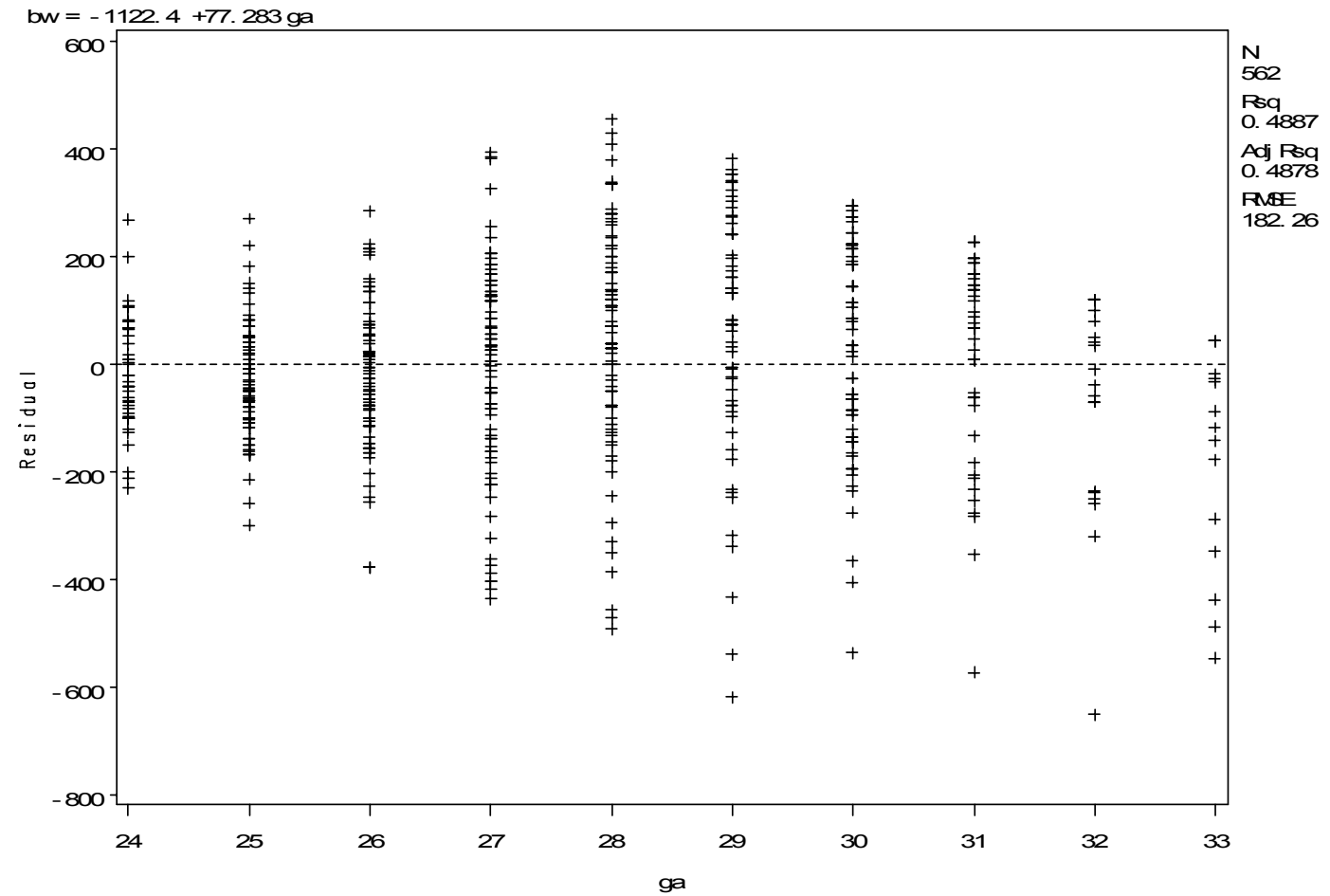
Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	1	17780974	17780974	535.30	<.0001
Error	560	18601510	33217		
Corrected Total	561	36382484			

Root MSE	182.25526	R-Square	0.4887
Dependent Mean	1016.53381	Adj R-Sq	0.4878
Coeff Var	17.92909		

Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > t
Intercept	1	-1122.36248	92.76595	-12.10	<.0001
ga	1	77.28300	3.34031	23.14	<.0001



ANOVA: Perform one-way analysis of variance--compare means when there are more than two groups. Make multiple comparisons with Bonferroni, Duncan, Tukey, Scheffe, etc.

Example: Do the babies receive different amounts of colloid (in cc/kg birth weight) on days 1 to 4? Note we need to make a 'long skinny' data set.

```
Program: * 1-way anova example after making long data set;
data two; set one (keep=bw coll-col4);
array CC(4) coll-col4;
do day=1 to 4;
    colloid=1000*(CC(day)/bw);
    output;
end;
run;

proc anova;
    class day;
    model colloid=day;
    means day / tukey bon;
run;
```

Result:

The ANOVA Procedure

Class Level Information

Class	Levels	Values
day	4	1 2 3 4

Number of observations 2264

NOTE: Due to missing values, only 657 observations can be used in this analysis.

The ANOVA Procedure

Dependent Variable: colloid

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	13052.13753	4350.71251	33.59	<.0001
Error	653	84573.99588	129.51607		
Corrected Total	656	97626.13341			

R-Square	Coeff Var	Root MSE	colloid Mean
0.133695	65.34327	11.38051	17.41650

Source	DF	Anova SS	Mean Square	F Value	Pr > F
--------	----	----------	-------------	---------	--------

day	3	13052.13753	4350.71251	33.59	<.0001
-----	---	-------------	------------	-------	--------

The ANOVA Procedure

Tukey's Studentized Range (HSD) Test for colloid

NOTE: This test controls the Type I experimentwise error rate.

Alpha 0.05
 Error Degrees of Freedom 653
 Error Mean Square 129.5161
 Critical Value of Studentized Range 3.64260

Comparisons significant at the 0.05 level are indicated by ***.

day		Difference	Simultaneous 95%		
Comparison		Between Means	Confidence Limits		
1	- 2	8.431	5.527	11.335	***
1	- 3	9.299	6.143	12.456	***
1	- 4	10.221	6.795	13.646	***
2	- 1	-8.431	-11.335	-5.527	***
2	- 3	0.868	-2.492	4.229	
2	- 4	1.790	-1.825	5.404	
3	- 1	-9.299	-12.456	-6.143	***
3	- 2	-0.868	-4.229	2.492	
3	- 4	0.922	-2.899	4.742	
4	- 1	-10.221	-13.646	-6.795	***
4	- 2	-1.790	-5.404	1.825	
4	- 3	-0.922	-4.742	2.899	

The ANOVA Procedure

Bonferroni (Dunn) t Tests for colloid

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than Tukey's for all pairwise comparisons.

Alpha 0.05
 Error Degrees of Freedom 653
 Error Mean Square 129.5161
 Critical Value of t 2.64632

Comparisons significant at the 0.05 level are indicated by ***.

day		Difference	Simultaneous 95%		
Comparison		Between Means	Confidence Limits		
1	- 2	8.431	5.448	11.415	***
1	- 3	9.299	6.056	12.542	***
1	- 4	10.221	6.702	13.740	***
2	- 1	-8.431	-11.415	-5.448	***
2	- 3	0.868	-2.585	4.321	
2	- 4	1.790	-1.924	5.504	
3	- 1	-9.299	-12.542	-6.056	***
3	- 2	-0.868	-4.321	2.585	
3	- 4	0.922	-3.004	4.847	
4	- 1	-10.221	-13.740	-6.702	***
4	- 2	-1.790	-5.504	1.924	

4	- 3	-0.922	-4.847	3.004
---	-----	--------	--------	-------

GLM: Perform 1- to N-way analysis of variance

Example: Is the pCO₂ on the first day of life associated with gestational age (categorized), birth hospital, an interaction between the two?

Program:

```
* 2-way anova with interaction - use work.one & create ga categories;
data one; set one;
if . < ga < 26 then gacat=1;
else if 26 <= ga < 29 then gacat=2;
else if ga >= 29 then gacat=3;
proc glm;
  class gacat hosp;
  model pco2_1=gacat hosp gacat*hosp;
run;
```

Result:

The SAS System

The GLM Procedure

Class Level Information

Class	Levels	Values
gacat	3	1 2 3
hosp	2	1 2

Number of observations 566

NOTE: Due to missing values, only 533 observations can be used in this analysis.

Dependent Variable: pco2_1

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	5	16328.03364	3265.60673	57.61	<.0001
Error	527	29870.72058	56.68068		
Corrected Total	532	46198.75422			

R-Square	Coeff Var	Root MSE	pco2_1 Mean
0.353430	23.01959	7.528658	32.70544

Source	DF	Type I SS	Mean Square	F Value	Pr > F
gacat	2	3494.42522	1747.21261	30.83	<.0001
hosp	1	12719.74043	12719.74043	224.41	<.0001
gacat*hosp	2	113.86799	56.93400	1.00	0.3669

Source	DF	Type III SS	Mean Square	F Value	Pr > F
gacat	2	2574.98973	1287.49486	22.71	<.0001
hosp	1	11786.47349	11786.47349	207.95	<.0001

gacat*hosp	2	113.86799	56.93400	1.00	0.3669
------------	---	-----------	----------	------	--------

Logist: Perform logistic regression (outcome variable is dichotomous).

Example: Is development of IVH related to characteristics of the babies and/or their hospital experience?

Program:

```
* logistic regression example - use work.one and create dummy variables;
data one; set one;

if ga > . then ga1=(. < ga < 26);
if ga > . then ga2=(26 <= ga < 29);

if rom > . then romcat=(rom >= 1);

if acs > . then complete=(acs=3);
run;

proc logistic descending;
    model ivh=ga1 ga2 pih romcat complete;
run;
```

Result:

The LOGISTIC Procedure

Model Information

Data Set	WORK.ONE
Response Variable	ivh
Number of Response Levels	2
Number of Observations	531
Model	binary logit
Optimization Technique	Fisher's scoring

Response Profile

Ordered Value	ivh	Total Frequency
1	1	105
2	0	426

Probability modeled is ivh=1.

NOTE: 35 observations were deleted due to missing values for the response or explanatory variables.

Model Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Intercept Only	Intercept and Covariates
AIC	530.083	496.101
SC	534.358	521.750
-2 Log L	528.083	484.101

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	43.9819	5	<.0001
Score	42.8291	5	<.0001
Wald	38.1241	5	<.0001

Analysis of Maximum Likelihood Estimates

Parameter	DF	Estimate	Standard Error	Wald Chi-Square	Pr > ChiSq
Intercept	1	-2.0982	0.3056	47.1438	<.0001
gal	1	1.6962	0.3398	24.9185	<.0001
ga2	1	0.9517	0.3105	9.3958	0.0022
pih	1	-0.5449	0.3898	1.9538	0.1622
romcat	1	0.2267	0.2380	0.9071	0.3409
complete	1	-0.6469	0.2461	6.9094	0.0086

Odds Ratio Estimates

Effect	Point Estimate	95% Wald Confidence Limits	
gal	5.453	2.802	10.614
ga2	2.590	1.409	4.760
pih	0.580	0.270	1.245
romcat	1.254	0.787	2.000
complete	0.524	0.323	0.848

Association of Predicted Probabilities and Observed Responses

Percent Concordant	66.4	Somers' D	0.400
Percent Discordant	26.5	Gamma	0.430
Percent Tied	7.1	Tau-a	0.127
Pairs	44730	c	0.700

Lifetest: Perform Kaplan-Meier (product limit) survival analysis.

Example: Is hospital length of stay related to gestational age (categorical). Deaths are censored

Program:

```
proc lifetest notable plots=(s) data=one;
    time los*dead(1);
    strata gacat;
run;
/* note: proc lifetest notable plots=(s) data=one lineprinter; for dot plot*/
```

Result:

The LIFETEST Procedure

Summary of the Number of Censored and Uncensored Values

Stratum	gacat	Total	Failed	Censored	Percent Censored
1	1	113	107	6	5.31
2	2	254	247	7	2.76
3	3	197	196	1	0.51

Total		564	550	14	2.48

NOTE: There were 2 observations with missing values, negative time values or frequency values less than 1.

Testing Homogeneity of Survival Curves for los over Strata

Rank Statistics

gacat	Log-Rank	Wilcoxon
1	-73.669	-26890
2	-19.678	-14009
3	93.346	40899

Covariance Matrix for the Log-Rank Statistics

gacat	1	2	3
1	114.047	-84.546	-29.500
2	-84.546	132.927	-48.381
3	-29.500	-48.381	77.881

Covariance Matrix for the Wilcoxon Statistics

gacat	1	2	3
1	10992531	-7436866	-3555664
2	-7436866	14619495	-7182629
3	-3555664	-7182629	10738293

Test of Equality over Strata

Test	Chi-Square	DF	Pr >
			Chi-Square
Log-Rank	126.1490	2	<.0001

Wilcoxon	173.9234	2	<.0001
-2Log(LR)	62.2093	2	<.0001

This graph would be in color on your screen. The line on the left is the babies in gacat=3 (29 weeks and above), the middle line is the babies in gacat=2 (26 to 28 weeks), and the line on the right is the babies in gacat=1 (less than 26 weeks).

