

Lecture 5

1. Dates and times in SAS
2. Macros
3. Arrays
4. Calculating change from baseline

1

Working with dates in SAS (*LSB* §3.8–3.9)

SAS converts a calendar date to an integer = number of days from 1 January 1960 to the date.

SAS maps time to the real number line by setting midnight, 1 January 1960, to zero.

Example: 11 June 1995 is represented as 12945. Units are days.

2

Getting dates in and out of SAS

Inputting data directly in SAS code:

Data one;

```
INPUT list of variables ;  
CARDS ;  
  data lines  
;
```

INPUT statement must identify all non-numeric variables (*LSB §2.8*)

INFORMAT for character variables: \$w.

(w = length of character string; optional)

3

Here is the data:

```
data one ;  
  input ID gender $ birthdate ;  
  cards ;  
  4833 F 5/16/1978  
  4834 F 7/4/1980  
  4855 M 12/14/1988  
;
```

\$ identifies gender as character, with default length up to 8 characters, defined by first value (F)

Need **INFORMAT** for birthdate (*LSB §2.8*)

4

`INFORMAT` for birthdate `MMDDYY10.` (see *LSB §2.8*)

10 characters wide: *mm/dd/yyyy* or *mm-dd-yyyy*

```
data one;
  input ID gender $ birthdate MMDDYY10. ;
  cards;
  4833 F 5/16/1978
  4834 F 7/4/1980
  4855 F 12/14/1988
  ;
```

Period at the end of `MMDDYY10.` is required, otherwise SAS thinks it's another variable.

5

Here's what we have now:

```
Proc Print data = one;
```

Obs	ID	gender	birthdate
1	4833	F	6710
2	4834	F	7490
3	4855	F	10575

Values of birthdate are number of days since 1/1/1960.

6

To **OUTPUT** birthdate as an understandable date, we need a **FORMAT** statement.
(LSB §3.9)

```
data one;
  input ID gender $ birthdate MMDDYY10. ;
  birthdate1 = birthdate; new variable is copy of birthdate
  birthdate2 = birthdate;
  format birthdate1 MMDDYY10. ;
  format birthdate2 WORDDATE. ;
cards;
4833 F 5/16/1978
4834 F 7/4/1980
4855 F 12/14/1988
;
```

7

Here are the two date formats:

ID	gender	birthdate	birthdate1	birthdate2
4833	F	6710	05/16/1978	May 16, 1978
4834	F	7490	07/04/1980	July 4, 1980
4855	F	10575	12/14/1988	December 14, 1988

8

Here is what the Import Wizard does for you:

```
data one;
  input ID gender $ birthdate MMDDYY10. ;
  format birthdate MMDDYY10. ;
  cards;
  4833 F 5/16/1978
  4834 F 7/4/1980
  4855 F 12/14/1988
  ;
```

ID	gender	birthdate
4833	F	05/16/1978
4834	F	07/04/1980
4855	F	12/14/1988

9

Calculating with dates

The difference between two dates in SAS: `interval = date2 - date1;`
is the number of days between the two dates.

Find a person's age at a clinic visit from `visit_date` and `birth_date`

`(visit_date - birth_date)` (unit is *days*)

convert to years:

```
age1 = (visit_date - birth_date) / 365.25;
```

```
age2 = floor(age1);    floor(x) gives the largest integer  $\leq x$ 
```

```

data one;
  input ID gender $ birthdate MMDDYY10. ;
  format birthdate MMDDYY10. ;
  now = TODAY(); today as SAS date
  current_age = (today() - birthdate)/365.25;
  current_age2 = floor(current_age);
  cards;
4833 F 5/16/1978
4834 F 7/4/1980
4855 F 12/14/1988
  ;

```

11

ID	gender	birthdate	now	current_ age	current_ age2
4833	F	05/16/1978	18528	32.3559	32
4834	F	07/04/1980	18528	30.2204	30
4855	F	12/14/1988	18528	21.7741	21

Time within days: SAS converts times to number of seconds since midnight.

*15:45:10 is represented as $56710 = 15*60*60 + 45*60 + 10$. Units are seconds.*

SAS converts date-times to the number of seconds since midnight,

1 January 1960.

12

SAS Macros (LSB, Ch 7)

A SAS macro is a program written in **macro language**, similar to regular SAS but different.

Macros allow you to:

- create a global variable
- simplify code and create flexible and generalizable code
- use SAS procedures and data steps inside DO-loops or conditionally

Macros can help you make code simpler.

Many published SAS papers provide a macro.

13

Key fact about macros

The SAS program you submit is compiled: translated into machine statements for execution.

There is one compiler for regular SAS code, another for macro code.

Macro texts start with `%` or `&` .

SAS picks out the macros and compiles them first.

After this, SAS compiles the regular SAS code.

14

Names of macro programs start with `%` .

SAS has dozens of built-in macros, listed and defined in SAS Help > Base SAS

One SAS macro is `%LET` This macro assigns a value to a macro variable:

```
%LET name = value ;
```

When `&name` appears in regular SAS code, it is replaced with *value*

value is treated as *character string*.

`&` (ampersand) starts the name in calls, not the definition

15

```
%LET detection_limit = 0.25;
```

In the SAS code:

```
Data two;  
  set one;  
  C1 = max(C, &detection_limit ) ;
```

Use `&` before the LET-variable to call it. The code that runs is:

```
Data two;  
  set one;  
  C1 = max(C, 0.25 ) ;
```

16

SAS does not actually change the code in the program:

```
200 %LET detection_limit=0.25;
    . . .
208 data two;
209     set one;
210     C1 = max(c, &detection_limit);
211
```

NOTE: There were 1019 observations read from the data set WORK.ONE.NOTE: The data se

17

Writing a macro

The format to create a macro is

```
%MACRO macro-name
    statements
%MEND macro-name
```

The simplest macro contains only a character string:

```
%macro iterations
    500
%mend iterations
```

The name of the macro is `%iterations`. Use the name to call the macro.

18

```
%macro iterations
  500
%mend iterations
```

. . .

```
Data three;
  set two;
  DO j = 1 to %iterations ;
  . . .
```

This DO-loop would execute 500 times.

19

```
%macro iterations
  500
%mend iterations
```

is equivalent to

```
%LET iterations = 500 ;
```

20

Nested comments

Two ways to “comment out” parts of SAS code:

```
* statement ;
```

everything between * and ; is comment.

```
/*  
  statement1;  
  statement2;  
  statement3;  
*/
```

everything between /* and */ is comment (green in editor).

21

What happens here?

```
/*  
  statement1;  
    /*  
      statement2;  
      statement3;  
    */  
  statement4;  
*/  
statement 5;
```

Which statements are commented out?

22

SKIP macro

Recall: **SAS picks out the macros and compiles them first.**

Macros are compiled separately from regular SAS code.

```
%macro SKIP;  
    statement1;  
    statement2;  
    statement3;  
%mend SKIP;
```

This effectively comments out the statements.

SKIP macro also works correctly when nested.

See P Grant: “The SKIP Statement,” on the course website.

23

Adding arguments to a macro

A macro is a function, and functions are more useful if they are not constant.

To include arguments in your macro, list them in parentheses in the %MACRO statement. SAS calls these **macro parameters**.

```
%macro typewriterplot(yvar=, xvar=, data=); arguments don't start with &  
  
    proc plot data= &data ; within the macro, argument variables start with &  
  
        plot &yvar * &xvar;  
  
    run;  
%mend typewriterplot;
```

24

You call the macro by providing values for the parameters, as follows:

```
%typewriterplot(yvar=child_IQ, xvar=mom_IQ, data= ph6470.child_iq);
```

To see how SAS has interpreted your macro arguments:

`options MPRINT` at the beginning of your program

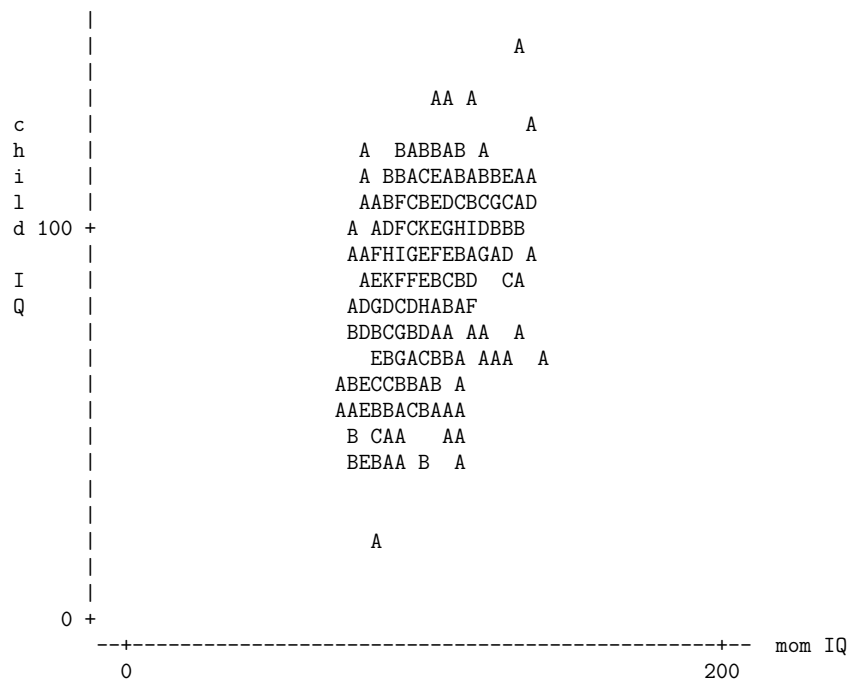
25

SAS log:

```
1878  options MPRINT;
1879
1880  %macro typewriterplot(yvar=, xvar=, data=);
1881      proc plot data=&data;
1882          plot &yvar*&xvar /vspace=25 hspace=50 ;
1883      run;
1884  %mend typewriterplot;
1885
1886  %typewriterplot(yvar=child_IQ, xvar=mom_IQ, data= ph6470.child_
MPRINT(TYPEWRITERPLOT):  proc plot data=ph6470.child_iq;
MPRINT(TYPEWRITERPLOT):  plot child_IQ*mom_IQ /vspace=25 hspace=50 ;
MPRINT(TYPEWRITERPLOT):  run;
```

26

Plot of child_IQ*mom_IQ. Legend: A = 1 obs, B = 2 obs, etc.



27

Data step: IF-THEN-ELSE (LSB §3.5–3.6)

We have already used the **subsetting IF**:

```
IF (condition);
```

Observation is included in the data set only if (condition) is true for that observation.

Second part of IF is THEN

```
IF (condition) THEN statement;
```

```
IF (educ_years GE 12) THEN education = "high school graduate";
```

statement is executed only if (condition) is true for that observation.

28

For more than one conditional statements, use DO; . . . ; END;

```
IF (id = 1207645) THEN DO;  
  age = 46;  
  BMI = 32.4;  
  clinic = "Fairview";  
END;
```

statements between DO and END are executed only if (condition) is true for that observation.

SAS will write an error message in the log file for a DO-loop without an END.

IF - THEN - ELSE

Can also specify statements for both results of testing a condition:

```
IF (condition) THEN statement_A;  
  ELSE statement_B;
```

```
IF (educ_years GE 12) THEN education = "high school graduate";  
  ELSE education = "dropped out";
```

Either THEN or ELSE can start a DO-loop.

Data step: Arrays (LSB §3.11)

An **array** is a vector or matrix of variables

In a data step, to create a vector array:

```
ARRAY array_name [number of variables] variable list ;
```

In a data step, to create a matrix array:

```
ARRAY array_name [number of rows, number of columns]  
list of row 1 variables, row 2 variables, etc ;
```

Whenever you are repeating the same calculation more than twice, consider using arrays.

31

Change-from-baseline using arrays

It is common to use change from baseline (starting point) in an outcome:

$$\Delta y = \text{change in } y = \text{final value } y_F - \text{baseline value } y_0$$

so that positive Δy means an increase, negative Δy means a decrease.

Example. A nutrition clinical study compared three fiber supplements—barley, oats, control—in a parallel-arm design. Primary outcomes were changes from baseline to study midpoint and endpoint in blood lipids, stress markers, glucose and insulin.

For the analysis, subtract baseline values from final values and midpoint values for each of 8 responses.

32

```

S
c
r      B      B      B      B      B      M      M      M      F
e      a      B      B      a      a      s      i      M      M      i      M      i      i
e      B      s      a      a      a      s      s      i      M      M      i      M      i      i
n      a      e      s      s      s      e      e      d      i      i      d      i      d      n
-      s      -      e      e      e      -      -      -      d      d      -      d      -      a
-      C      e      -      -      -      H      G      C      -      -      H      -      G      l
O      S      A      h      -      h      H      L      -      C      C      l      h      H      C      C      I      l      -
b      I      e      g      o      T      o      D      D      R      Y      u      o      D      R      Y      n      u      T
s      D      x      e      l      G      l      L      L      P      S      c      l      L      P      S      s      c      G

1 1 F 22 223 127 189 41 123 1.5 3.9 98 186 46 1.8 4.0 5 94 150
2 2 M 26 264 48 280 52 218 0.1 7.3 92 271 51 0.1 8.9 4 93 48
3 3 F 54 . 96 209 59 131 0.3 8.7 89 218 66 0.2 9.9 5 92 64

```

```

F      F      F      F      F      F
i      F      F      F      i      i      f
n      i      i      i      n      n      b      i
a      n      n      n      a      a      a      n      m
l      a      a      a      l      l      s      a      i      m
-      l      l      l      -      -      w      h      e      l      d      i
-      C      -      -      H      G      t      t      -      -      -      d
O      h      H      L      -      C      C      l      -      -      i      i      t      l      -
b      o      D      D      R      Y      u      k      -      -      n      n      r      d      t
s      l      L      L      P      S      c      g      m      s      s      t      l      g

1 201 43 128 0.9 3.5 95 82.2 167.64 10 11 Barley 116 118
2 295 53 232 0.1 9.2 90 72.6 175.26 5 8 Barley 209 56
3 185 63 109 0.3 9.5 92 56.8 160.02 8 6 Barley 138 71

```

A simple direct approach is to write down all the subtractions:

```

data one;
  set pubh.barley;
  dm_chol = Mid_Chol - base_chol;

```

dm: d for delta, m for midpoint

```

dm_TG = Mid_TG - Base_TG;
dm_HDL = Mid_HDL - Base_HDL;
dm_LDL = Mid_LDL - Base_LDL;
dm_CRP = Mid_CRP - Base_CRP;
dm_HCYS = Mid_HCYS - Base_HCYS;
dm_Ins = Mid_Ins - Base_Ins;
dm_Gluc = Mid_Gluc - Base_Gluc;

```

Differences with array

```
data two;
  set pubh.barley;
  array base[8] base_chol Base_TG Base_HDL Base_LDL
    Base_CRP Base_HCYS Base_Ins Base_Gluc;

  array mid[8] mid_chol mid_TG mid_HDL mid_LDL
    mid_CRP mid_HCYS mid_Ins mid_Gluc;

  array diff[8] dm_chol dm_TG dm_HDL dm_LDL
    dm_CRP dm_HCYS dm_Ins dm_Gluc;

  DO i=1 to 8 ;    DO-loop does all the subtractions
    diff[i] = mid[i] -base[i];
  end;
```