More about the SAS language

Use of SAS
March 2009

Lists of variables

- Sometimes you need to refer to many variables at once
- E.g., if you have repeated measurements or just many similar variables

```
proc freq;
  tables spm1-spm392;
run;
```

- similar names `x1-x20`
- All character variables: `_CHARACTER_`
- All numerical variables: `_NUMERIC_`
- All variables: `_ALL_`

Labels

```
libname juul 'p:\sas\data\juul';
proc freq data=juul.juul2;
  table tanner;
run;
data hope;
  set juul.juul2;
  label tanner="Tanner stage";
run;
proc freq data=hope;
  table tanner;
run;
```

Formats

- Information about how to read or print variable
- Built-in formats (Numerical, dates, character)
- User defined formats (1=Male 2=Female)
- Pretty printouts
- Grouping in tables and analyses

Formats, cont.

- Standard formats: `10.3`, `best12.`, `E12.`, `$10.`, `date10.`, `yymmdd10.`.
- Always contain a dot (Do not forget it!)
- Can be associated permanently with variable in DATA step, or:
- Specified ad hoc with FORMAT statement in PROC steps.
- User defined formats are created and listed by PROC FORMAT

Example with proc format

```
proc format;
   value sexfor 1="male" 2="female";
run;
data a;
   input sex;
   datalines;
   1
   2
   3
   .
   ;
run;
proc print data=a;
run;
proc print data=a;
  format sex sexfor.;
run;
```

Exercise using formats

1. Get the bissau data into SAS using a libname statement
2. Use bissau data, generate a sas program creating formats for variables
   - `dead`: 1=died 2=Survived
   - `bcg`: 1=yes 2=No
   - `dtp`: 1=yes 2=No

   Help for the first one:

```
proc format;
  value deadfmt 1=Died 2=Survived;
run;
```

3. Make a `proc freq` of the three variables using your formats.

Using formats

Generate data

```
data test;
reply=1; do i=1 to 25; output; end;
reply=2; do i=1 to 21; output; end;
reply=3; do i=1 to 35; output; end;
run;
proc print data=test;
run;
```

Generate format for the variable reply

```
proc format;
value replyfor 1='Yes     '
               2='No      '
               3='Maybe   ';
run;
```

Distribution of the variable reply

```
proc freq data=test;
table reply; run;
```

Formats in PROC and DATA steps

The format used in a proc step

```
proc freq data=test;
table reply;
format reply replyfor.;
run;
```

or the format can be associated with the variable in a data step

```
data testfor;
set test;
format reply replyfor.;
run;
```

now the format will be used every time we use the data: testfor

```
proc freq data=testfor;
  table reply;
run;
```

Save data in a permanent data set

```
libname pdrev 'p:\';
data pdrev.testfor;
set testfor;
run;
```

Restart SAS and run the program

```
libname pdrev 'p:\';
proc freq data=pdrev.testfor;
table reply;
run;
```

SAS cannot find the format!

```
proc freq data=pdrev.testfor;
table reply;
format reply; *format _all_;
run;
```

`format _all_;` removes the formats. Can be very useful

A more advanced PROC FORMAT example:

```
proc format;
  value agegrpfmt 0-1="0-1" 2-4="2-4" 5-6="5-6";
run;
proc format fmtlib;run;
proc freq data=afrika.bissau;
  table agemm;
proc freq data=afrika.bissau;
  table agemm;
  format agemm agegrpfmt.;
run;
```

NB: To be able to use your created formats next time you start SAS, you can save the SAS code in a file, and run this the next time you will use the data set.

Date formats

- Actual value stored is the number of days since 1 January 1960.

```
data hope;
  input x;
datalines;
-1
0
1
;
run;
proc print data=hope;
   format x ddmmyy10.;
run;
```

- function `mdy()` month-day-year:

```
data a;
  x=mdy(1,17,2006);
run;
proc print data=a; run;
proc print data=a;  format x ddmmyyd.;run;
```

- Final `d` in the output format indicates "dash". Other possibility `c,s,n,p` (colon, slash, none, period)

Working with dates

As dates internally are stored as days since 1 Jan 1960, one can add and subtract dates and constants:

```
data prv;
  input nr dead DDMMYY10.;
  datalines;
  1 9-01-1975
  2 12-12-1956
run;

proc print data=prv;
run;

data prv;
  set prv;
  thisday=today();
  days=thisday-doed;
  years=days/365.25;
run;

proc print data=prv;
run;
```

Exercise with dates

In the SAS data set `bissau2.sas7bdat` (in the africa directory) the first 200 observations are from the original Bissau data. Variables are:

```
id        = ID of child
dob       = Date of birth
visitdate = Date of visit
agedays   = Age in days at visit
```

Please, check that the variable `agedays` was correctly calculated.

Appending data sets: SET

more cases, same variables

```
data group0;
  set sasuser.fitness;
  where group=0;
  comment1="Data1";
  keep group comment1;
run;
data group1;
  set sasuser.fitness;
  where group=1;
  comment2="Data2";
  keep group comment2;
run;
data group2;
  set sasuser.fitness;
  where group=2;
  comment3="Data3";
  keep group comment3;
run;
data all;
  set group0 group1 group2;
run;
proc print;
run;
```

Obs	group	comment1	comment2	comment3
1	0	Data1		
2	0	Data1		
3	0	Data1		
4	0	Data1		
5	0	Data1		
6	0	Data1		
7	0	Data1		
8	0	Data1		
9	0	Data1		
10	0	Data1		
11	1		Data2	
12	1		Data2	
13	1		Data2	
14	1		Data2	
15	1		Data2	
16	1		Data2	
17	1		Data2	
18	1		Data2	
19	1		Data2	
20	1		Data2	
21	2			Data3
22	2			Data3
23	2			Data3
24	2			Data3
25	2			Data3
26	2			Data3
27	2			Data3
28	2			Data3
29	2			Data3
30	2			Data3
31	2			Data3

Merging data set: MERGE

new variables, same cases. Normally there is a key, say `id`, and all data sets must be sorted by `id`

```
data name;
  input id name $6.;
datalines;
1 Henrik
2 Esben
3 Peter
;
data surname;
  input id sname $15.;
datalines;
2 Budtz-Jørgensen
1 Jensen
;
proc sort data=surname;
  by id;
data fullname;
  merge name surname;
  by id;
proc print data=fullname;
run;
```

Obs	id	name	sname
1	1	Henrik	Jensen
2	2	Esben	Budtz-Jørgensen
3	3	Peter	

Exercise: More about MERGE

In the library `'p:\sas\prg'` you will find the file `'exercise_merge.sas'`.Run the first part of the code.

```
data name;
  input fam name $6.;
datalines;
1 Henrik
1 Gustav
2 Esben
run;
data surname;
  input fam sname $15.;
datalines;
2 Budtz-Jørgensen
1 Jensen
run;

proc print data=name;
run;
```

```
proc print data=surname;
run;
```

This gives the output:

Obs	fam	name
1	1	Henrik
2	1	Gustav
3	2	Esben

Obs	fam	sname
1	2	Budtz-Jørgensen
2	1	Jensen

We want to merge the two data sets so that the names and surnames are correctly matched (Henrik and Gustav are called Jensen while Esben is called Budtz-Jørgensen). The following code gives to possible solustions. Run the code and explain the differences in the solutions.

```
*solution 1;
data fullname1;
  merge name surname;
run;
proc print data=fullname1;
run;
*solution 2;
proc sort data=surname;
  by fam;
run;
data fullname2;
  merge name surname;
  by fam;
run;
proc print data=fullname2;
run;
```