

*Introduction*

## What is SAS?

SAS is a software package for managing large amounts of data and performing statistical analyses.

It was created in the early 1960s by the Statistical Department at North Carolina State University. Today SAS is developed and marketed by SAS Institute Inc. with head office in Cary, North Carolina, U.S.A.

1

*Introduction (cont.)*

## The SAS System

The SAS System is mainly used for

- Data Management (about 80% of all users)
- Statistical Analysis (about 20% of all users)

The power of SAS lies in its ability to manage large data sets. It is fast and has many 2statistical and non-statistical features.

The disadvantage of SAS is its steep learning curve. It takes quite a bit of an effort to get started. User-friendly interfaces do exist, though.

2

*Introduction (cont.)*

## SAS Programming

SAS programming works in two steps:

### Data Step

1. reads data from file
2. makes transformations and adds new variables
3. creates SAS Data Set

### Proc Step

4. uses the SAS Data Set
5. produces the information we want, such as tables, statistics, graphs, web pages

3

*Introduction (cont.)*

## Data and Proc Steps

Example of a SAS program:

```
data work.main;
  set work.original;
  age=1997-birthyr;
  bmi=weight/(height*height);
run;

proc print data=work.main;
  var id age bmi;
run;

proc means data=work.main;
  var age bmi;
run;
```

} Data Step

} Proc Steps

4

## Windows

The main feature of SAS is its division of the main window into two halves. The left part is a navigator of SAS libraries and Results (from the Output window).

The right part is divided into three separate windows:

- Program window or Enhanced Editor
- Log window
- Output window

## Output and Log Windows

The result of a program execution is printed to the Output window. There you will find the prints, tables and reports, etc.

A log file is printed to the Log window.

The log file contains information about the execution, whether it was successful or not. It usually points out your mistakes with warning and error messages so that you can correct them.

## Example: SAS Log

```
65 proc gplot data=work.influnce;
66   plot di*pred / vaxis=axis1 haxis=axis1;
ERROR: Variable DI not found.
NOTE: The previous statement has been deleted.
67 run;
```

Make a habit of checking the Log window after every execution.

Even if SAS has accepted and executed the program, you may have made a methodological error. Check the note on how many observations were read, and if there were any missing values.

## Example: SAS Output

		patientens alder	
ALDER	Frequency	Cumulative Frequency	
0 - 24	41	41	
25 - 44	176	217	
45 - 64	77	294	
65 -	25	319	

*The SAS Environment (cont.)*

## File Types

These files are created by SAS:

- .sas file (SAS program)
- .log file (Log)
- .lst file (Output)

The SAS data sets are saved as .sd7 or .sas7bdat files.

(Other file types, e.g. catalogs, are also used and created by SAS, but we will not pursue this any further.)

9

*The SAS Environment (cont.)*

## Using the SAS System

You work with SAS using

- Menus and Toolbar
- Command Line
- Key Functions F1-F12

10

*The SAS Environment (cont.)*

## Example

Three different ways to Open a File in the Enhanced Editor:

1. Menus: choose File + Open
2. Toolbar: press the icon for “Open”
3. Command line: write  
include 'N:\temp\bp.sas'  
and press Enter.

11

*The SAS Environment (cont.)*

## Commands and Keys

	<b>Command</b>	<b>Key</b>
Open File	include 'P:\catalogue\filename.sas'	Ctrl-O
Save File	file 'P:\catalogue\filename.sas'	Ctrl-S
Clear Window	clear	F12
Zoom On/Off Window	zoom	F11
Recall Submitted Programs	recall	F4
Go To Enhanced Editor	wpgm	F5
Go To Log Window	log	F6
Go To Output Window	output	F7
Submit a Program	submit	F8
Change Key Definitions	keys	F9

12

*The SAS Environment (cont.)*

## Write and Read

In the Enhanced Editor you can

- create new, or edit existing, programs
- submit programs
- save programs (an unsaved file is marked with \* after the file name)

You can NOT edit the log file or the output file in their windows. They are only readable. If you wish to edit these files, save them and use the Enhanced Editor or Word.

13

*SAS syntax*

## Statements

The SAS code (syntax) consists of statements. Statements mostly begin with a **keyword**, and they ALWAYS end with a SEMICOLON.

```
data work.cohort;  
  set course.males98;  
run;  
  
proc print data=work.cohort;  
run;
```

Examples of keywords: data, set, run, proc.

14

*SAS syntax (cont.)*

## Statements

SAS statements can begin and end anywhere on a line.

```
data work.cohort;
```

One or several blanks can be used between words.

```
data    work.cohort;
```

One or several semicolons can be used between statements.

```
data work.cohort;;;  
;
```

15

*SAS syntax (cont.)*

## Statements

The statement can begin and end on different lines.

```
data  
    work.cohort;
```

SAS will not object to several statements on the same line. However, it is not considered good programming to have more than one statement per line. It makes the code difficult to read. Avoid this!

```
data work.cohort; set course.males98; run;
```

16

*SAS syntax (cont.)*

## Indenting to improve readability

Improve the readability of your program by adding more space to the code (= indenting).

Begin data steps and proc steps in the first position, as far left as possible. The ending run statement should also be in the first position.

All statements in between should start a few blanks in from the left margin.

This creates blocks of data steps and proc steps, and you can easily see where one ends and another begins.

17

*SAS syntax (cont.)*

## Example of Indenting

```
data work.height;
  infile 'h:\mep\rawdata_height.txt';
  input name $ 1-20
         kon    21
         alder  22-23
         height 24-30;

  if kon=0 and (height ne .) then
  do;
    if 0<height<81.75 then lnapprx=50;
    else
      if 81.75<=height then lnapprx=100;
    end;
  else lnapprx=.;

run;
```

18

*SAS syntax (cont.)*

## Indenting

Within statements it is also VERY useful to use indenting.  
Put similar syntactic words in the same position below each other.

Use blank lines a lot!

Markers of blocks should be placed in the same position below one another (e.g. data-run, proc-run, if-else, do-end).

19

*SAS Data Sets*

## What is a SAS Data Set?

A SAS data set is a special file type (.sas7bdat) which consists of a descriptive part and a data part.

The DESCRIPTIVE part includes

- general information, such as data set name, date of creation, number of observations and variables etc.
- variable information, such as variable name, type (character or numeric), format, length, label etc.

20

SAS Data Sets (cont.)

## The Data

The DATA part is the data values.

Data is organised with observations in the rows and variables in the columns.

Variables

Name	Age	Infarkt
Andersson	43	0
Bengtsson	45	2
Carlsson	42	4
Eriksson	45	2

Observations

Values

21

SAS Data Sets (cont.)

## Descriptive Part

Proc CONTENTS prints the descriptive part of a data set.

```
The CONTENTS Procedure

Data Set Name: PPT_EX8.MAIN          Observations: 64
Member Type:  DATA                 Variables: 9
Engine:       VB                    Indexes: 0
Created:      17:17 Tuesday, August 7, 2001  Observation Length: 72
Last Modified: 17:17 Tuesday, August 7, 2001  Deleted Observations: 0
Protection:                               Compressed: NO
Data Set Type:                             Sorted: NO
Label:

-----Alphabetic List of Variables and Attributes-----
#  Variable  Type  Len  Pos  Format  Informat
2  BIRTHYR   Num   8    0  BEST8.  FB.
5  CASE_1    Num   8    24  BEST8.  FB.
1  ID         Char  8    56  $8.     $8.
4  LENGTH    Num   8    16  BEST8.  FB.
3  WEIGHT    Num   8    8   BEST8.  FB.
6  age       Num   8    32  4.
8  bmi       Num   8    48
9  generatn  Char  5    64
7  height    Num   8    40
```

22

SAS Data Sets (cont.)

## Data Part

Proc PRINT prints the data part of a data set.

OBS	ID	BIRTHYR	WEIGHT	HEIGHT	AGE	BMI
1	001	1954	62	1.65	43	22.7732
2	002	1956	68	1.67	41	24.3824
3	003	1956	65	1.72	41	21.9713
4	004	1962	56	1.68	35	19.8413
5	005	1954	58	1.59	43	22.9421
6	006	1953	52	1.62	44	19.8141
7	007	1955	69	1.75	42	22.5306
8	008	1955	75	1.73	42	25.0593
9	009	1960	82	1.7	37	28.3737
10	010	1962	68	1.72	35	22.9854
11	011	1961	65	1.68	36	23.0300
12	012	1954	62	1.69	43	21.7079
13	013	1956	58	1.68	41	20.5499
14	014	1962	61	1.64	35	22.6800
15	015	1958	58	1.63	39	21.8300
16	016	1959	62	1.65	38	22.7732
17	017	1962	59	1.64	35	21.9363
18	018	1957	73	1.8	40	22.5309

23

SAS Data Sets (cont.)

## Create a Data Set

A SAS data set is created from

- SAS data set (.sas7bdat file)
- raw data file (.txt file)
- another external file through importing (EXCEL file, etc.)

or by

- manually entering the data

24

*SAS Data Sets (cont.)*

## Create a Data Set

To use an existing data set, a .sas7bdat file, is the most common way to create a SAS data set.

How to create a SAS data set from a raw data file is described in chapter Read Raw Data Into SAS.

Importing non-SAS data is not trivial. Use File + Import Data. Ask for help if you run into trouble.

- or use the program STAT-Transfer

25

*SAS Data Sets (cont.)*

## Create a Data Set

The easiest way to manually enter data into SAS is via the Viewtable facility

You can also use the CARDS or DATALINES statement

26

*SAS Data Sets (cont.)*

## Existing SAS Data Set (.sas7bdat)

Create a SAS data set from an existing SAS data set:

```
data work.main;  
  set work.original;  
  statements;  
run;
```

This will yield an exact copy of the old data set "original".  
The name of the copy is "main".

Usually we wish to change the new data set, by adding programming statements after the SET statement.

27

*SAS Data Sets (cont.)*

## Naming Data Sets

PLEASE, use descriptive names for your data sets.

It is not considered clever to name your data sets: final1, final2, final3, etc.

Other names to avoid are: new, old, mydata, analys, *your-name*, etc.

More on this topic in the chapter Naming Data Sets and Variables.

28

*SAS Data Sets (cont.)*

## Viewtable

The Viewtable facility is a user-friendly tool to look at your data set without using data steps or proc steps.

You enter the Viewtable window by issuing the “viewtable” command in the Command line.

This will yield a window very similar to EXCEL, with cells, rows and columns.

29

*SAS Data Sets (cont.)*

## Viewtable

It is very easy to create a data set in the Viewtable window. Just enter the data manually into the cells. The variable names are created by clicking on the column header and following the instructions.

When you click to save the data set, it is saved into a .sas7bdat file, which may then be used in any data step or proc steps in the Enhanced Editor.

30

*SAS Data Sets (cont.)*

## Viewtable

If you wish to open an existing data set into the Viewtable window, just issue the command

“viewtable *name-of-data-set*”

and it will open.

You can also open a data set from the Explorer window in the window area to the left. Just navigate to the right library and double click on the data set icon.

31

*SAS Data Sets (cont.)*

## Variables

There are two types of variables in SAS: character (char) and numerical (num).

The type refers to the values the variable have.

Examples of a variable called MONTH:

A character variable: MONTH with values ‘Jan’, ‘Feb’, ..., ‘Dec’.

A numerical variable: MONTH with values 1, 2, ..., 12.

32



*SAS Data Sets (cont.)*

## Variables

The values of a character variable are between quotes “”.

When the value is printed, all characters within the quotes are printed.

Typical character values are letters, while numerical values always are digits.

Character variables may include digits as well.

33

*SAS Data Sets (cont.)*

## Variables

Missing values of a character variable are represented by a blank, while a period “.” (punktum) denotes missing values of a numeric variable.

Character values can be 32767 characters long at most.  
(200 characters in version 6)

Good rule: Never use char variables to store numeric values.  
For example, always store Patient Number as a numeric.

34

*SAS Data Sets (cont.)*

## Naming Variables

Variable names (e.g. age, bmi) and data set names (e.g. main, original)

- can be 32 characters (letters, underscores and digits) long at most
- can be uppercase or lowercase or mixed (mAiN = MaIn)
- must start with a letter (A-Z) or an underscore (\_), not a digit

35

*SAS Data Libraries*

## What is a SAS Data Library?

A SAS data library is the catalogue where your data sets are stored.

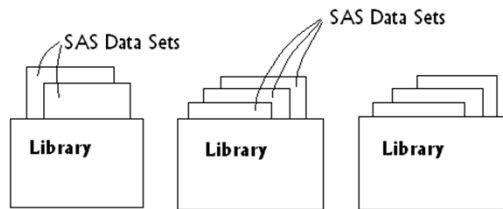
A data library is like a drawer in a filing cabinet. The cabinet may have several drawers representing several different libraries.

A data set is a file within a drawer. A drawer may contain several files.

36

*SAS Data Libraries (cont.)*

## Figure: SAS Data Libraries



37

*SAS Data Libraries (cont.)*

## WORK and SASUSER

Two data libraries are created automatically by SAS:  
WORK and SASUSER.

The WORK library is a temporary library. All its contents are deleted when you exit SAS. If you wish to keep your data sets, do not put them in the WORK library.

The SASUSER library is a permanent library. All its contents are kept when you exit SAS.

38

*SAS Data Libraries (cont.)*

## WORK and SASUSER

The physical location of the permanent SASUSER library is under 'C:\'.

It is not especially clever to save all the SAS programs and data sets in the same folder.

Generally we wish to store them in separate folders for separate projects or papers.

Therefore, it is possible to create your own permanent libraries.

39

*Submitting a SAS Program*

## Submit a Program

To submit (= execute) a SAS program:

- Menus: choose Run + Submit, or
- Toolbar: press the icon with "the running man", or
- Command line: issue "submit" command.

To halt a submitted program press CTRL + BREAK. You may have to press it a few times.

40

*Submitting a SAS Program (cont.)*

## Submit a Program

When a program is submitted each step (data or proc) is executed one at a time.

The contents of a data step is performed on each observation one at a time (i.e. creation of new variables etc.).

Each step generates log to the Log window. Output (if any) is generated to the Output window.

41

*Submitting a SAS Program (cont.)*

## Check the Log

**ALWAYS** browse the Log window after a submission!

If the submission is stopped by errors the data set is unchanged and you might do incorrect analyses on an old data set. You will only see it has been stopped by looking at the log.

(We emphasise this, since we are too well experienced with the consequences of the opposite.)

42

*Submitting a SAS Program (cont.)*

## Error Messages in the Log

Note (blue): information which do not indicate errors. They are usually informative to rule out any methodological errors in you programming.

Warning (green): points out errors which SAS could correct itself. The execution was performed with these changes. Still you should check whether it was done properly. Example: misspelled keywords.

43

*Submitting a SAS Program (cont.)*

## Error Messages in the Log

Error (red): serious errors which SAS could not handle. The execution was stopped. These errors must be corrected by you. Example: forgotten semicolons, invalid options, misspelled variable names.

Especially if you are updating data sets, be aware that red errors mean NO updating!

44

*Submitting a SAS Program (cont.)*

## Unbalanced Quotes

A special type of syntactic error is unbalanced quotes (“”).

Quotes must come in pairs. If they do not, the execution will keep on running forever. You halt it by submitting

```
' ;
```

45

*Submitting a SAS Program (cont.)*

## Enhanced Editor

When you press submit, all the code in the Enhanced Editor is executed.

If you only wish to submit a limited number of rows of the program code, mark it and press submit.

46

*The Data Step*

## Data Statement

Data sets are created through a data step. The data step begins with a DATA statement.

General form of the DATA statement:

```
data SAS-data-set;
```

The *SAS-data-set* is a name of the form

```
libref.filename
```

47

*The Data Step (cont.)*

## Data Statement

To create a data set ORIGINAL in the temporary library WORK:

```
data work.original;
```

When using the temporary WORK library it is possible to skip the work prefix and just write:

```
data original;
```

48

*The Data Step (cont.)*

## Create Variables

A new variable is created by:

```
variable=expression ;
```

The expression may consist of numbers, other variables and operators such as

- + addition
- subtraction
- \* multiplication
- / division
- \*\* exponentiation (potensopløftning)

49

*The Data Step (cont.)*

## Examples

```
data work.main;
  set work.original;

  age=1997-birthyr;

  height=height/100;

  bmi=weight/(height*height);
run;
```

50

*The Data Step (cont.)*

## Functions

Useful are the predefined functions in SAS, such as

```
exp(argument)    exponential function
log(argument)    natural logarithm
int(argument)    the integer part of a numeric argument
```

There are also non-mathematical SAS specific functions. A list of useful functions may be found in the SAS manual SAS Language (pp 521-616).

51

*The Data Step (cont.)*

## Examples

```
data work.main;
  set work.original;

  highest=max(height1, height2, height3);

  birthyr=year(brthdate);

  total=sum(x1, x2, x3, x4, x5);
run;
```

52

*The Data Step (cont.)*

## Variable Names

If you are creating a series of variables, such as repeated measurements, put the order number at the end of the name, e.g. x1, x2, x3, since

```
total=sum(x1,x2,x3,x4,x5) ↔ total=sum(of x1-x5)
```

The use of the notation x1-x5 is widely accepted in many expressions and procedures. It will not work if the order number is in the middle of the name.

Also see the chapter Naming Data Sets and Variables.

53

*The Proc Steps*

## Procedures

A procedure (proc) is a predefined function that operate on data sets. By specifying the predefined statements in the procedure you can adapt it to your needs and wishes.

Examples of procedures:

```
proc contents  prints the descriptive part of a data set
proc print    prints the data part of a data set
proc freq     creates frequency tables, etc.
proc means    calculates means and other statistics
```

54

*The Proc Steps (cont.)*

## Data Step Vs. Proc Step

Usually, for beginners as well as among advanced users, the data step is more comprehensible as a concept. Not seldom is extensive programming done in the data step, when the same result easily could have been obtained through a simple option in a procedure.

As a rule, operations on observations (within rows) are done in the data step, e.g. adding two variables together to make a third.

Operations on variables (within columns) are done in a proc step, e.g. taking the mean of a variable.

55

*The Proc Steps (cont.)*

## Proc CONTENTS

The CONTENTS procedure prints out the descriptive part of a data set.

The descriptive part includes:

- General information: data set name, number of observations, number of variables, etc.
- Variable information: variable name, type, length, position, format, label, etc.

56

*The Proc Steps (cont.)*

## Proc CONTENTS

The general form of the CONTENTS procedure:

```
proc contents data=SAS-data-set;  
run;
```

Example:

```
libname course 'h:\SasAtMEP\Course';  
  
proc contents data=course.main;  
run;
```

57

*The Proc Steps (cont.)*

## Proc PRINT

The PRINT procedure prints out the data part of a data set.

It is possible to choose which variables to print. If none are chosen, all variables will be printed.

The first column is the OBS column, which indicates observation. If there are more variables than will fit into the output window, the output is split and the exceeding variables printed on the following page. The OBS column is reprinted to indicate observation.

58

*The Proc Steps (cont.)*

## Proc PRINT

The general form of the PRINT procedure:

```
proc print data=SAS-data-set;  
run;
```

OR to print a specified list of variables from the data set

```
proc print data=SAS-data-set;  
  var variable1 variable2 variable3 ... ;  
run;
```

59

*The Proc Steps (cont.)*

## Proc PRINT

Examples:

```
proc print data=course.main;  
run;
```

```
proc print data=course.main;  
  var age bmi;  
run;
```

60

*The Proc Steps (cont.)*

## Proc FREQ

The FREQ procedure is mainly used to create frequency tables, although it has a wide range of statistical features as well.

It creates both one-way and multiple-way tables.

(FREQ is pronounced “frek” in Danish and “freek” in English.)

61

*The Proc Steps (cont.)*

## Proc FREQ

The general form of FREQ procedure:

```
proc freq data=SAS-data-set;  
  tables var1;  
run;
```

OR for a two-way table

```
proc freq data=SAS-data-set;  
  tables var1 * var2 / nopercnt norow nocol;  
run;
```

62

*The Proc Steps (cont.)*

## Proc FREQ

Example one-way table:

```
proc freq data=course.main;  
  tables age;  
run;
```

The SAS System				
AGE	Frequency	Percent	Cumulative Frequency	Cumulative Percent
35	4	22.2	4	22.2
36	1	5.6	5	27.8
37	1	5.6	6	33.3
...				
43	3	16.7	17	94.4
44	1	5.6	18	100.0

63

*The Proc Steps (cont.)*

## Proc FREQ

Example two-way table:

```
proc freq data=course.main;  
  tables case*age/nopercnt nocol norow;  
run;
```

The nopercnt option suppresses the printing of cell percentages.

Nocol and norow suppress column and row cell percentages respectively.

64



*The Proc Steps (cont.)*

## Proc FREQ

The SAS System

TABLE OF AGE BY CASE\_1

AGE	CASE_1		Total
Frequency	0	1	
35	3	1	4
36	1	0	1
37	0	1	1
...			
44	0	1	1
Total	9	9	18

65

*The Proc Steps (cont.)*

## Proc SORT

The SORT procedure sorts the data set according to a chosen variable. The sorted data set replaces the unsorted data set, unless you define an OUT data set.

The SORT procedure can sort by several variables, in ascending (default) or descending (option) order.

Missing values are defined as minus infinity, i.e. less than all other numeric values.

66

*The Proc Steps (cont.)*

## Proc SORT

The general form of the SORT procedure:

```
proc sort data=SAS-data-set out=SAS-data-set;  
  by variables;  
run;
```

OR if you want descending order

```
proc sort data=SAS-data-set out=SAS-data-set;  
  by descending variables;  
run;
```

67

*The Proc Steps (cont.)*

## Proc SORT

Example:

```
proc sort data=course.main out=course.sortage;  
  by case_1 age;  
run;
```

This will yield a data set called *course.sortage*, where the observations are sorted by *case\_1* and within each category of *case\_1* by age.

68

*The Proc Steps (cont.)*

## Proc SORT

There is no result in the Output window from proc SORT, but a proc PRINT of data set course.sortage gives:

OBS	ID	BIRTHYR	WEIGHT	HEIGHT	AGE	BMI	CASE_1
1	010	1962	68	1.72	35	22.9854	0
2	014	1962	61	1.64	35	22.6800	0
3	017	1962	59	1.64	35	21.9363	0
4	011	1961	65	1.68	36	23.0300	0
...							
9	012	1954	62	1.69	43	21.7079	0
10	004	1962	56	1.68	35	19.8413	1
11	009	1960	82	1.7	37	28.3737	1
12	002	1956	68	1.67	41	24.3824	1
13	003	1956	65	1.72	41	21.9713	1
14	007	1955	69	1.75	42	22.5306	1
...							
17	005	1954	58	1.59	43	22.9421	1
18	006	1953	52	1.62	44	19.8141	1

69

*The Proc Steps (cont.)*

## Proc MEANS

The MEANS procedure calculates basic statistics. By default the statistics are

N = number of non-missing observations

Mean = mean value, average

Std Dev = standard deviation

Minimum = minimum value

Maximum = maximum value

Optional statistics include Nmiss (number of missing observations), range (maximum-minimum), etc.

70

*The Proc Steps (cont.)*

## Proc MEANS

The general form of MEANS procedure:

```
proc means data=SAS-data-set;  
run;
```

This will yield summary statistics on all variables in the data set.

Missing values are excluded from the analysis.

71

*The Proc Steps (cont.)*

## Proc MEANS

Example:

```
proc means data=course.main;  
run;
```

Variable	N	The MEANS Procedure			
		Mean	Std Dev	Minimum	Maximum
BIRTHYR	62	1959.35	3.3294354	1953.00	1967.00
WEIGHT	63	61.7460317	6.5129356	47.0000000	80.0000000
LENGTH	64	1.6675000	0.0617213	1.4800000	1.8000000
CASE_1	64	0.4687500	0.5029674	0	1.0000000
age	62	37.6451613	3.3294354	30.0000000	44.0000000
height	64	1.6675000	0.0617213	1.4800000	1.8000000
bmi	63	22.1982718	1.9262282	17.9591837	29.3847567

Naturally, the character variable ID is not displayed.

72

*The Proc Steps (cont.)*

## Proc MEANS

You can modify the proc MEANS code to suit your wishes:

- To specify the number of decimals used in the printout add the option MAXDEC.
- If you are only interested in a selection of variables, use a VAR statement.

73

*The Proc Steps (cont.)*

## Proc MEANS

Example:

```
proc means data=course.main maxdec=2;  
  var age bmi;  
run;
```

Variable	N	Mean	Std Dev	Minimum	Maximum
AGE	18	39.44	3.24	35.00	44.00
BMI	18	22.65	1.95	19.81	28.37

74

*The Proc Steps (cont.)*

## Proc MEANS

In proc MEANS it is possible to calculate statistics on subgroups of the data set, e.g. the mean bmi and age for cases and controls separately.

There are two different ways to deal with subgroup statistics, depending on what output you are interested in:

- BY statement
- CLASS statement

75

*The Proc Steps (cont.)*

## Proc MEANS

The BY statement:

```
proc means data=course.main maxdec=2;  
  var age bmi;  
  by case_1;  
run;
```

The BY statement requires that the data set has previously been sorted according to the BY variable.

76

*The Proc Steps (cont.)*

## Proc MEANS

Result from BY statement:

CASE\_1=0

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
age	32	37.50	3.08	33.00	44.00
bmi	34	22.09	1.95	17.96	29.38

CASE\_1=1

Variable	N	Mean	Std Dev	Minimum	Maximum
age	30	37.80	3.62	30.00	44.00
bmi	29	22.33	1.92	19.61	27.34

77

*The Proc Steps (cont.)*

## Proc MEANS

The CLASS statement:

```
proc means data=course.main maxdec=2;  
  var age bmi;  
  class case_1;  
run;
```

The CLASS statement does NOT require any sorting.

78

*The Proc Steps (cont.)*

## Proc MEANS

Result from CLASS statement:

The MEANS Procedure

CASE_1	N	Variable	N	Mean	Std Dev	Minimum	Maximum
0	34	age	32	37.50	3.08	33.00	44.00
		bmi	34	22.09	1.95	17.96	29.38
1	30	age	30	37.80	3.62	30.00	44.00
		bmi	29	22.33	1.92	19.61	27.34

79

*The Online HELP*

## Quick Help on Syntax

SAS has a very good ONLINE HELP. In this help you can get full information on syntax. For more theoretical issues you should use the paperback manuals or the Online Documentation (see later on how to use them).

The online help is accessed through the Command line and command "help".

```
help print  
help means
```

(Do not write "proc" before the process name.)

80

*The Online HELP (cont.)*

## Using the Online Help

When a help command is issued the HELP Window will open with topics:

Introduction

Syntax

Additional Topics (occasionally)

To get full information on how to write the code, choose SYNTAX.

81

*The Online HELP (cont.)*

## Example

As an example, access online help for proc MEANS.

“help means” + choose SYNTAX

These are all the possible statements that proc MEANS accept. If you want to know more about a specific statement just click on it and read:

82

*The Online HELP (cont.)*

## Example

PROC MEANS: Syntax

```
PROC MEANS <option(s)> <statistic-keyword(s)>; BY  
<DESCENDING> variable-1 <... <DESCENDING> variable-  
n><NOTSORTED>;
```

```
CLASS variable(s) </ option(s)>;  
FREQ variable;  
ID variable(s);  
OUTPUT <OUT=SAS-data-set> <output-statistic-specification(s)>  
<id-group-specification(s)> <maximum-id-specification(s)>  
<minimum-id-specification(s)> </ option(s)> ;  
TYPES request(s);  
VAR variable(s) < / WEIGHT=weight-variable>;  
WAYS list;  
WEIGHT variable;
```

83

*The Online HELP (cont.)*

## Explanation to the Online Help Text

- underlined word = keyword referring to a statement (statements within a procedure are optional, the PROC and the RUN statements are required)
- black word = required if the corresponding keyword is used
- words within “<>” = optional, not required
- words separated by “|” = possible choices of values for a specific option

84

*The Online HELP (cont.)*

## Example

If you click on the “PROC MEANS”, a list of possible options will be displayed.

Among them is the MAXDEC= option which we have already used. The equal sign is required. Next to MAXDEC= is the black word “number”. If you use the MAXDEC option you are required to fill in a number corresponding to the maximum number of decimals to be displayed.

(The exact conventions depend on which version of the help you use.... -pd)

85

*Subsetting a Data Set*

## Subsets of Data

Often one wants to use only a subset of a data set, e.g. persons older than 60 years, women, cases etc.

This is particularly useful when performing data cleaning, and you only want to print the observations with extreme values of a variable, say blood pressure > 200.

86

*Subsetting a Data Set (cont.)*

## WHERE option

In procedures you use the WHERE data set option to subset the data set.

```
proc print data=SAS-data-set (where= (expression));  
run;
```

The WHERE data set option may be used in any procedure. It can also be used in data steps, although it is less usual.

```
data course.cases;  
  set course.main (where= (case_1=1));  
run;
```

87

*Subsetting a Data Set (cont.)*

## WHERE option

The *expression* must be a logical one, resulting in true or false.

Only observations for which the expression is true will be used in the proc step.

Examples of expressions are:

```
where=(birthyr gt 1950)  
where=(1947<birthyr<1950)  
where=(birthyr=1947)
```

88

*Subsetting a Data Set (cont.)*

## Conditional Operators

Possible conditional operators (use sign or abbreviation):

=	eq	equal to
^=	ne	not equal to
>	gt	greater than
<	lt	less than
>=	ge	greater than or equal to
<=	le	less than or equal to

89

*Subsetting a Data Set (cont.)*

## Logical Operators

You can also combine several expressions by using logical operators:

AND if all expressions are true then the compound expression will be true, else it is false

OR if any of the expressions are true, then the compound expression will be true, else it is false

90

*Subsetting a Data Set (cont.)*

## Examples

```
proc freq data=course.main(wher=(birthyr<1950 and case_1=1));  
  ... ;  
run;
```

```
proc means data=course.main(wher=((birthyr<1950 or  
  birthyr>1953)  
  and  
  (case_1 ne .)));  
  ... ;  
run;
```

We recommend that you use parentheses as much as you can.  
It will enhance the reading as well as reduce the errors.  
There is no limit to how many pairs you are allowed to use.

91

*Subsetting a Data Set (cont.)*

## Special Operators

Special operators to use with the WHERE option:

IS MISSING	true if value is missing
BETWEEN - AND	true if value falls within the range

Examples:

```
wher=(birthyr is missing) ↔ wher=(birthyr=.)  
wher=(birthyr between 1950 and 1952) ↔  
  ↔ wher=(1950<=birthyr<=1952)
```

92

*Subsetting a Data Set (cont.)*

## Special Operators

Special character operators to use with the WHERE option:

LIKE true if value is identical to the argument

```
where=(lastname like 'Johnsson');
```

93

*Subsetting a Data Set (cont.)*

## Special Operators

The power of the LIKE operator is that you can use different types of matching arguments:

'Smi%' matches all character values that begin with "Smi"

'%th' matches all character values that end with "th"

94

*Subsetting a Data Set (cont.)*

## Special Operators

'Smi\_' matches all character values that are four characters long and begin with "Smi"

'Smi\_\_' matches all character values that are five characters long and begin with "Smi"

95

*Subsetting a Data Set (cont.)*

## Special Operators

Another special operator is CONTAIN.

CONTAIN yields true if any string of the value matches the argument.

```
where=(lastname contains 'nss');
```

All last names with the letters "nss" in them will be selected.

96



*Subsetting a Data Set (cont.)*

## Create a Subset

Sometimes you want to make a new data set from a subset.

These are two ways to accomplish this:

```
data course.cases(where=(case_1=1));
  set course.main;
run;
```

which will delete the redundant observations at the end of the data step.

97

*Subsetting a Data Set (cont.)*

## Create a Subset

```
data course.cases;
  set course.main(where=(case_1=1));
run;
```

which will delete the redundant observations at the beginning of the data set.

However, a good advice is to minimise the number of data sets as much as possible. The fewer data sets the better! There is nearly always a procedure to use on the main data set instead of creating a new data set.

98

*Subsetting a Data Set (cont.)*

## WHERE statement

There is a third way too, the WHERE statement:

```
data course.cases;
  set course.main; where (case_1=1);
run;
```

Note that no equal sign is used in the WHERE statement! Apart from the equal sign, the WHERE statement works as the WHERE option.

May also be written *if case\_1=1;*

99

*Subsetting a Data Set (cont.)*

## OBS and FIRSTOBS

In both data and proc steps you can use the OBS data option to subset a specific range of observations.

To subset the ten first observations:

```
proc print data=course.main(obs=10);
```

To subset observations 5 to 10:

```
proc print data=course.main(firstobs=5 obs=10);
```

Note that no WHERE is used!

100

*Subsetting a Data Set (cont.)*

## Subsetting IF

There is one occasion when a WHERE statement will not work, namely if you wish to condition on a variable which you create in that same data step.

Instead you can use a subsetting IF statement:

```
if (your-expression);
```

101

*Subsetting a Data Set (cont.)*

## Example

```
data course.less20yr;  
  set course.main;  
  
  age75=1975-birthyr;  
  if (age75 < 20);  
  
run;
```

This will yield a data set where all the observations were younger than 20 yrs in 1975.

102

*Subsetting a Data Set (cont.)*

## Delete Observations

It is possible to delete both observations and variables from a data set.

Observations are deleted through the DELETE statement:

```
if expression then delete;
```

(The IF-THEN statement is explained in detail later on.)

103

*Subsetting a Data Set (cont.)*

## Example

```
data course.after65;  
  set course.main;  
  
  if birthyr<1965 then delete;  
  
run;
```

This will yield a data set where all observations with birthyr<1965 have been deleted.

104

*Subsetting a Data Set (cont.)*

## DELETE vs. WHERE vs. IF

The same result could have been obtained through a WHERE statement, or by a subsetting IF

```
if birthyr>=1965;
```

However, if variable BIRTHYR is created in the same data step, then you need to use IF or a WHERE data set option on the target data set:

```
data course.after65(where=(birthyr>65));  
  ...;  
run;
```

105

*Subsetting a Data Set (cont.)*

## Delete Variables

To delete variables use either of the KEEP and DROP statements.

```
keep variable1 variable2 ... ;  
drop variable1 variable2 ... ;
```

The KEEP statement will yield a data set where all variables NOT listed in the KEEP statement list are deleted.

The DROP statement will yield a data set where all variables listed in it are dropped.

106

*Subsetting a Data Set (cont.)*

## Delete Variables

You use KEEP and DROP in the same way as WHERE, either as a data set option, or as a statement in a data step.

Thus KEEP and DROP data set options can be used in procedures as well.

Be aware that you can NOT use both KEEP and DROP in the same data step.

107

*Subsetting a Data Set (cont.)*

## Examples

KEEP statement:

```
data course.half;  
  set course.main;  
  keep birthyr age case_1;  
run;
```

108

*Subsetting a Data Set (cont.)*

## Examples

KEEP data set option:

```
data course.half(keep=birthyr age case_1);  
  set course.main;  
run;
```

or

```
data course.half;  
  set course.main(keep=birthyr age case_1);  
run;
```

109

*IF-THEN-ELSE*

## IF-THEN-ELSE

Quite often one wishes to create a variable of the form

$$X = \begin{cases} 1 & \text{if } \text{age} < 30 \\ 0 & \text{else} \end{cases}$$

The value of X depends on the value of variable age.

110

*IF-THEN-ELSE (cont.)*

## IF-THEN-ELSE Statement

To define the X variable in SAS is simple. The code is as follows:

```
data course.groups;  
  set course.main;  
  
  if (age<30) then X=1;  
  else X=0;  
run;
```

Note, that since a missing value is interpreted as negative infinity, it will fall under X=1. More about this problem shortly.

111

*IF-THEN-ELSE (cont.)*

## IF-THEN-ELSE Statement

The general form of the IF-THEN-ELSE statement is

```
if expression1 then expression2 ;  
else expression3 ;
```

Expression1 must be a logical expression yielding either true or false.

If expression1 is true than expression2 is calculated.

If expression1 is false then expression3 is calculated.

112

*IF-THEN-ELSE (cont.)*

## Example

It is very useful to use several IF-THEN-ELSE statements in a series.

```
if (age<30) then generatn=1;
else
  if (30<=age<60) then generatn=2;
else
  if (60<=age) then generatn=3;
else generatn=.;
```

113

*IF-THEN-ELSE (cont.)*

## Example

Say that the age=67, then (age<30) will yield false, and we will go right to the ELSE line.

There the expression is a new IF-THEN-ELSE statement where we continue our execution.

(30<=age<60) is false too, and so we continue at the next ELSE line.

There the expression (60<=age) is true, and consequently we calculate the THEN-expression generatn=3.

114

*Set and Merge*

## Joining Data Sets

Previously we have taken a subset of a data set. It is also possible to join (concatenate and merge) data sets together.

Concatenate= adding observations

Merge = adding variables, match data sets

115

*Set and Merge (cont.)*

## SET

To concatenate two or several data sets, use the SET statement.

```
data SAS-data-set;
  set data-set1 data-set2 data-set3 ... ;
run;
```

The top observations in the SAS-data-set will be from data-set1, the next observations from data-set2 and so on.

116

Set and Merge (cont.)

## SET

If there are different variables in the data sets, missing values will be generated for the observations which did not have the variables before the concatenation..

Variables not present in all the data sets will be found to the far right of the concatenated data set.

117

Set and Merge (cont.)

## Example

We wish to add three observations to our data set course.main. The three observations are stored in data set course.extraobs.

```
data course.concatenate;  
  set course.main course.extraobs;  
run;
```

118

Set and Merge (cont.)

## Example

Concatenate.sas7bdat:

OBS	ID	BIRTHYR	WEIGHT	AGE	BMI	ALDER	
1	001	1954	62	43	22.77319	.	}
2	002	1956	68	41	24.38237	.	
3	003	1956	65	41	21.97134	.	
4	004	1962	56	35	19.84127	.	
5	005	1954	58	43	22.94213	.	
6	006	1953	52	44	19.81405	.	
7	007	1955	69	42	22.53061	.	
8	008	1955	75	42	25.05931	.	
9	009	1960	82	37	28.3737	.	
10	010	1962	68	35	22.9854	.	
11	011	1961	65	36	23.03005	.	
12	012	1954	62	43	21.70792	.	
13	013	1956	58	41	20.54989	.	
14	014	1962	61	35	22.67995	.	
15	015	1958	58	39	21.82995	.	
16	016	1959	62	38	22.77319	.	
17	017	1962	59	35	21.93635	.	
18	018	1957	73	40	22.53086	.	
19	029	.	.	.	28.59	48	}
20	030	.	.	.	22.14	51	
21	031	.	.	.	23.85	42	

119

Set and Merge (cont.)

## Example

Since alder = age it would be nice to transform the three alder values to age values. Use a RENAME data set option.

```
rename=(old-var-name=new-var-name);
```

In the example:

```
data course.concatenate;  
  set course.main  
      course.extraobs(rename=(alder=age));  
run;
```

120

Set and Merge (cont.)

## Example

OBS	ID	BIRTHYR	WEIGHT	AGE	BMI
1	001	1954	62	43	22.77319
2	002	1956	68	41	24.38237
3	003	1956	65	41	21.97134
4	004	1962	56	35	19.84127
5	005	1954	58	43	22.94213
6	006	1953	52	44	19.81405
7	007	1955	69	42	22.53061
8	008	1955	75	42	25.05931
9	009	1960	82	37	28.3737
10	010	1962	68	35	22.9854
11	011	1961	65	36	23.03005
12	012	1954	62	43	21.70792
13	013	1956	58	41	20.54989
14	014	1962	61	35	22.67995
15	015	1958	58	39	21.82995
16	016	1959	62	38	22.77319
17	017	1962	59	35	21.93635
18	018	1957	73	40	22.53086
19	029	.	.	48	28.59
20	030	.	.	51	22.14
21	031	.	.	42	23.85

course.main

course.extraobs

121

Set and Merge (cont.)

## MERGE

To merge or match two or several data sets, use the MERGE statement:

```
data SAS-data-set;  
  merge data-set1 data-set2 data-set3 ... ;  
  by variable;  
run;
```

The first variables in the SAS-data-set will be from data-set1, the next variables from data-set2 etc.

122

Set and Merge (cont.)

## MERGE

The BY variable (matching variable) is usually an identification variable unique to all observations and present in all data sets to be merged.

The data sets must be sorted according to the matching variable before MERGE is used.

If there are different observations in the data sets, missing values will be generated for observations not present in other data sets. These observations will be found at the end of the merged data set.

123

Set and Merge (cont.)

## Example

We wish to add a variable to our data set course.main.

The new variable is the number of children each woman has, variable CHILD.

This variable is stored together with the variable ID in a data set called course.extravar. ID is our matching variable, unique for all observations.

124

*Set and Merge (cont.)*

## Example

```
proc sort data=course.main course.extravar;
  by id;
run;

proc sort data=course.extravar;
  by id;
run;

data course.merged;
  merge course.main course.extravar;
  by id;
run;
```

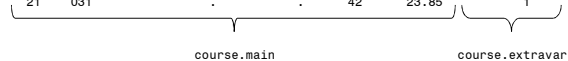
125

*Set and Merge (cont.)*

## Example

merged.sas7bdat:

OBS	ID	BIRTHYR	WEIGHT	AGE	BMI	CHLD
1	001	1954	62	43	22.7732	1
2	002	1956	68	41	24.3824	0
3	003	1956	65	41	21.9713	2
4	004	1962	56	35	19.8413	2
5	005	1954	58	43	22.9421	0
6	006	1953	52	44	19.8141	3
7	007	1955	69	42	22.5306	1
8	008	1955	75	42	25.0593	2
9	009	1960	82	37	28.3737	4
10	010	1962	68	35	22.9854	.
11	011	1961	65	36	23.0300	2
...						
17	017	1962	59	35	21.9363	2
18	018	1957	73	40	22.5309	0
19	029	.	.	48	28.59	0
20	030	.	.	51	22.14	3
21	031	.	.	42	23.85	1



126

*Read Raw Data Into SAS*

## Raw Data

Sometimes you have a raw data file you want to read into SAS. The data are not stored in a SAS data set, or .sas7bdat file.

Perhaps data are of the form:

```
001 Anderson 1954 62 43
002 Askelund 1956 68 41
003 Bengtson 1956 65 41
004 Carner 1962 56 35
005 Holmfors 1954 58 43
006 Johnsson 1953 52 44
...
016 Torberg 1959 62 38
017 Turner 1962 59 35
018 Öhlund 1957 73 40
```

127

*Read Raw Data Into SAS (cont.)*

## List Input Format

This form of data is called list input format. It means that each data value is separated by a blank.

List input formatted data have the following characteristics:

- all values contain eight or fewer characters
- values are separated by one or several blanks
- missing values (both numeric and character) must be written as a single period (.)
- numerical values must include all necessary decimal points

128



*Read Raw Data Into SAS (cont.)*

## INPUT and CARDS

To create a data set using list input formatted data, write

```
data course.nameincl;
  input id $ name $ birthyr weight age;
  cards;
001 Anderson 1954 62 43
002 Askelund 1956 68 41
003 Bengtson 1956 65 41
...
016 Torberg 1959 62 38
017 Turner 1962 59 35
018 Öhlund 1957 73 40
;
run;
```

129

*Read Raw Data Into SAS (cont.)*

## INPUT and CARDS

In the INPUT statement you list the variables in the same order as they are presented.

If a variable is a character variable, put a dollar sign (\$) after the name.

When there is no dollar sign, then SAS interprets the variable as a numeric.

130

*Read Raw Data Into SAS (cont.)*

## INPUT and CARDS

The CARDS statement indicates that data lines are to follow.

It is written below the INPUT statement.

Note that no semicolons are present at the end of each data line, **ONLY** at the end of all the data. It marks the end of data entry.

Equivalent to CARDS is the keyword DATALINES.

131

*Read Raw Data Into SAS (cont.)*

## Column Input Format

If the data you wish to read into a data set is specified in columns (= positions), you may use the column input format.

```
001 Andersson 1954 62 43
002 Askelund 1956 68 41
003 Bengtsson 1956 65 41
004 Carner 1962 56 35
005 Holmfors 1954 58 43
006 Johansson 1953 52 44
...
016 Torberg 1959 62 38
017 Wernersson 1962 59 35
018 Öhlund 1957 73 40
```

```
-----1-----2-----3
```

132

## Column Input Format

Each variable occupies a specified number of characters or positions (1 position = 1 column).

The first variable ID is in columns 1-3, NAME in 5-12 (including blanks at the end), BIRTHYR in 13-16 etc.

The ruler below the data is a SAS ruler. It is often used in the Log window.

Each column is indicated by “-”. The digit 1 indicates column 10, 2 column 20 etc. The + indicates columns 5, 15, 25 etc.

## Column Input Format

To create a data set using column input formatted data, write

```
data course.nameincl;
  input id $ 1-3 name $ 5-12
        birthyr 13-16 weight 18-19 age 21-22;
  cards;
001 Andersson  1954 62 43
002 Askelund   1956 68 41
003 Bengtsson  1956 65 41
...
016 Torberg    1959 62 38
017 Wernersson 1962 59 35
018 Öhlund     1957 73 40
;
run;
```

## Column vs. List Input Format

The only difference from the code for list input formatted data is that you must specify the columns in the INPUT statement. (ID \$ 1-3, name \$ 5-12, etc.)

Differences in data characteristics:

- values must not contain more characters than are specified in the input statement
- missing values are written as blank for character variable, period (.) for numerical

## Data from .TXT File

To read data (list or column formatted) from external text files (.txt) use an INFILE statement.

```
data course.nameincl;
  infile 'h:\SAS-folder\data-file.txt';
  input id      $ 1-3
        name    $ 5-12
        birthyr 13-16
        weight  18-19
        age     21-22
;
run;
```

*Read Raw Data Into SAS (cont.)*

## Data from .TXT File

The INPUT statement is written as before, depending on whether you have column or list formatted data in the text file.

No CARDS (or DATALINES) statement is used.

The INFILE statement must be written above the INPUT statement.

137

*Naming Data Sets and Variables*

## Data Set and Variable Names

The name of a data set or a variable should reflect its contents.

It is not obvious to an outsider that the variable “rel51yrd” refers to “relapse 5 years after first diagnosis”. A better name would perhaps be “relapse5”.

Add comments in the data step to clarify what the variables are!

138

*Naming Data Sets and Variables (cont.)*

## Data Set and Variable Names

- can be 32 characters (letters, underscores and digits) long at most
- can be uppercase (versaler) or lowercase (gemena) or mixed (OrIgInAl = oRiGiNaL)
- must start with a letter (A-Z) or an underscore (\_), not a digit

139

*Naming Data Sets and Variables (cont.)*

## Data Set and Variable Names

The increased character limit to 32 characters in version 8.0 is welcomed by many SAS users (formerly it was only 8 characters in version 6.12).

But a long name is not always to prefer either. It makes the code tiresome to read, not to mention write.

A good idea is therefore to try to keep to a maximum of 8 to 12 characters also in future. Use labels.

140

*Naming Programs*

## Program File Names

Contrary to the SAS data set and variable names, there is no limit to 32 characters when you name the program files.

Do not hesitate to use long names, although the same rules as for the variables names apply.

Be consistent, no cryptic abbreviations.

141

*Organise Your Programming*

## Structuring SAS Files

Here is a suggestion of how to organise your SAS programs.

It is not presumed to be The Ultimate Structure, but it is a good start to get organised.

All ideas to improve this structure are most welcome.

142

*Appendix*

## List of Useful Procedures

PROC APPEND = concatenates data sets (compare SET statement)

PROC CATALOG = administrates SAS Catalogues

PROC CHART = creates simple charts, bar charts, block charts, pie charts, etc.

PROC COMPARE = compares the contents of two data sets, or variables within a data set

PROC CONTENTS = prints the descriptive part of a data set

PROC COPY = copies a SAS Data Library or parts of it

PROC CORR = calculates correlation coefficients

PROC DATASETS = lists, renames, deletes data sets etc.

143

*Appendix (cont.)*

## List of Useful Procedures (cont.)

PROC FORMAT = defines formats and informats permanently or temporarily

PROC FREQ = creates frequency tables and basic statistical tests

PROC MEANS = creates descriptive statistics

PROC OPTIONS = lists all current SAS System options values

PROC PLOT = creates simple Y-X plots

PROC PRINT = prints the data part of a data set

PROC PRINTTO = defines destination of output and log, e.g. when you wish to use an output as input for another procedure

144

*Appendix (cont.)*

### List of Useful Procedures (cont.)

PROC RANK = calculates ranking of different sorts

PROC REPORT = creates better layout for output

PROC SORT = sorts data sets by one or several variables

PROC SQL = implementing query language SQL, used to retrieve data sets

PROC STANDARD = standardises variables to a given mean and variance

PROC SUMMARY = descriptive statistics in form of summations

PROC TABULATE = descriptive statistics in tables

PROC TRANSPOSE = transposes data sets, i.e. transforms rows to columns, and vice versa

145

*Appendix (cont.)*

### List of Useful Procedures (cont.)

PROC UNIVARIATE = creates descriptive distribution statistics and plots, histograms etc.

146