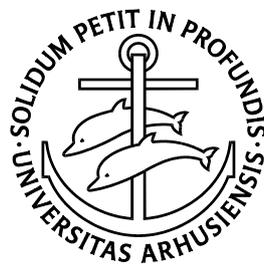


Introduction to Stata 8

Svend Juul



Contents

1. Installing, customizing and updating Stata	3
2. Windows in Stata	5
3. Suggested mode of operation	7
4. Getting help	9
5. Stata file types and names	10
6. Variables and observations	11
6.1. Variable names	11
6.2. Numeric variables	11
6.3. Missing values	12
7. Command syntax	13
8. Getting data into Stata	16
9. Documentation commands	18
10. Modifying data	20
10.1. Calculations	20
10.2. Selections	21
10.3. Renaming and reordering variables	23
10.4. Sorting data	24
10.5. Numbering observations	24
10.6. Combining files	25
10.7. Reshaping data	26
11. Description and analysis	27
11.1. Categorical data	27
11.2. Continuous data	30
12. Regression models	32
12.1. Linear regression	32
12.2. Logistic regression	33
13. Survival and related analyses	34
14. Graphs	38
15. Miscellaneous	54
15.1. Memory considerations	54
15.2. String variables	55
15.3. Dates. Danish CPR numbers	57
15.4. Random samples, simulations	59
15.5. Immediate commands	60
15.6. Sample size and power estimation	61
15.7. Ado-files	62
15.8. Exchange of data with other programs	63
15.9. For old SPSS users	63
16. Do-file examples	65
Appendix 1: Purchasing Stata and manuals	67
Appendix 2: Entering data with EpiData	68
Appendix 3: NoteTab Light: a text editor	70
Alphabetic index	71

Preface

Stata is a software package designed for data management and statistical analysis; the primary target group is academia.

This booklet is intended mainly for the beginner, but knowledge of fundamental Windows functions is necessary. Only basic commands and a few more advanced analyses are shown. You will find a few examples of output, and you can find more in the manuals. The booklet does not replace the manuals.

You communicate with Stata by commands, either by typing them or by using the menus to create them. Every action is elicited by a command, giving you a straightforward documentation of what you did. This mode of operation is a contrast to working with spreadsheets where you can do many things almost by intuition, but where the documentation of what you did – including the sequence of actions – may be extremely difficult to reconstruct.

Managing and analysing data is more than issuing commands, one at a time. In my booklet *Take good care of your data*¹ I give advice on documentation and safe data management, with SPSS and Stata examples. The main message is: *Keep the audit trail*. Primarily for your own sake, secondarily to enable external audit or monitoring. These considerations are also reflected in this booklet.

To users with Stata 7 experience: You can go on working as usual, and your version 7 do-files should work as usual. However, most users will love the new menu system and the much-improved graphics.

To users with SPSS experience: By design Stata is lean compared to SPSS. Concerning statistical capabilities Stata can do a lot more. Stata has a vivid exchange of ideas and experiences with the academic users while SPSS increasingly targets the business world. Just compare the home pages www.spss.com and www.stata.com. Or try to ask a question or give a suggestion to each of the companies. In section 15.9 I show some SPSS commands and their Stata counterparts.

Vince Wiggins of Stata Corporation has given helpful advice on the graphics section.

I welcome any comments and suggestions. My e-mail address is: sj@soci.au.dk.

Aarhus, September 2004

Svend Juul

1) Juul S. *Take good care of your data*. Aarhus, 2003. (download from www.biostat.au.dk/teaching/software).

Notation in this booklet

Stata commands are shown like this:

```
tabulate agegr sex , chi2
```

tabulate and *chi2* are Stata words, shown with italics, *agegr* and *sex* is variable information, shown with ordinary typeface.

In the output examples you will often see the commands preceded by a period:

```
. tabulate agegr sex , chi2
```

This is how commands look in output, but you should not type the period yourself when entering a command.

Optional parts of commands are shown in light typeface and enclosed in light typeface square brackets. Square brackets may also be part of a Stata command; in that case they are shown in the usual bold typeface. Comments are shown with light typeface:

```
save c:\dokumenter\proj1\alfa1.dta [ , replace]  
summarize bmi [fweight=pop] // Weights in square brackets
```

In the examples I use **c:\tmp** as my folder for temporary files.

In the Stata manuals it is assumed that you use **c:\data** for all of your Stata files. I strongly discourage that suggestion. I am convinced that files should be stored in folders reflecting the subject, not the program used; otherwise you could easily end up confused. You will therefore see that I always give a full path in the examples when opening (*use*) or saving files (*save*).

Throughout the text I use Stata's style to refer to the manuals:

[GSW] Getting Started with Stata for Windows
[U] User's Guide
[R] Base Reference manual (4 volumes)
[G] Graphics manual
[ST] Survival analysis and epidemiological tables

See more on manuals in appendix 1.

1. Installing, customizing and updating Stata

1.1. Installing Stata

[GSW] 1

No big deal, just insert the CD and follow the instructions. By default Stata will be installed in the `c:\Stata` folder. 'Official' ado-files will be put in `c:\Stata\ado`, 'unofficial' in `c:\ado`. To get information about the locations, enter the Stata command `sysdir`. The folders created typically are:

<code>c:\Stata</code>	the main program
<code>c:\Stata\ado\base</code>	'official' ado-files as shipped with Stata
<code>c:\Stata\ado\updates</code>	'official' ado-file updates
<code>c:\ado\plus</code>	downloaded 'unofficial' ado-files
<code>c:\ado\personal</code>	for your own creations (ado-files, profile.do etc.)

A must: Update now – and do it regularly

Right after installing Stata, in the menu bar click:

Help ► Official updates

and select <http://www.stata.com> to get the most recent modifications and corrections of bugs.

Do this regularly to keep Stata updated.

Recommended: Also install NoteTab Light

As described in section 2, Stata has some shortcomings in handling output, and you will benefit a lot from a good text editor. I recommend NoteTab Light; see appendix 3.

1.2. Customizing Stata

Create desktop shortcut icon

In the Start Menu you find the Stata shortcut icon. Right-click it and drag it to the desktop to make a copy. Next right-click the desktop shortcut icon and select Properties (Egenskaber).

In the Path field you see e.g. `c:\stata\wstata.exe /m1`, meaning that 1 MB of memory is reserved for Stata. You might increase this, see section 15.1 on memory considerations.

As the start folder you probably see `c:\data`. I strongly suggest to change the start folder to `c:\dokumenter` or whatever your personal main folder is. The reasons for this suggestion:

- You should put your own text-, graph-, data- and do-files in folders organised and named by subject, not by program, otherwise you will end up confused.
- All of your 'own' folders should be sub-folders under one personal main folder, e.g. `c:\dokumenter`. This has at least two advantages:
 - You avoid mixing your 'own' files with program files
 - You can set up a consistent backup strategy.

The profile.do file

[GSW] A7

If you put a **profile.do** file in the `c:\ado\personal` folder² the commands will be executed automatically each time you open Stata. Write your **profile.do** using any text-editor, e.g. Stata's Do-file editor or NoteTab (see appendix 3) – but not Word or WordPerfect.

```
// c:\ado\personal\profile.do
set logtype text // simple text output log
log using c:\tmp\stata.log , replace // open output log
cmdlog using c:\tmp\cmdlog.txt , append // open command log

// See a more elaborate profile.do in section 16
```

set logtype text writes plain ASCII text (not SMCL) in the output log, to enable displaying it in e.g. NoteTab.

log opens Stata's log file (**stata.log**) to receive the full output; the **replace** option overwrites old output. The folder `c:\tmp` must exist beforehand.

cmdlog opens Stata's command log file (**cmdlog.txt**); the **append** option keeps the command log from previous sessions and lets you examine and re-use past commands.

Fonts. Window sizes and locations

[GSW] 18, 20

Start by:

Prefs ► General Preferences ► Windowing

and let the Review and Variables windows *not* be floating. Next, maximize the main Stata window (click the upper right [□] button). Arrange windows as shown in section 2, and:

Prefs ► Save Windowing Preferences

If you somehow lost the setting, you can easily recreate it by:

Prefs ► Load windowing preferences

By default the Results window displays a lot of colours; to me they generate more smoke than light. I chose to highlight error messages only and to underline links:

Prefs ► General preferences ► Result colors ► Color Scheme: Custom 1

Result, Standard, Input:	White
Errors:	Strong yellow, bold
Hilite:	White, bold
Link:	Light blue, underlined
Background:	Dark blue or black

In each window (see section 2) you may click the upper left window button; one option is to select font for that type of window. Select a fixed width font, e.g. Courier New 9 pt.

Windows 98 and ME users only:

These Windows versions have restricted capacity for dialogs. To avoid crashes:

```
set smalldlg on , permanently
```

2) [GSW] appendix A7 recommends otherwise. I stick to my recommendation; this place is safer.

2. Windows in Stata

[GSW] 2

I suggest arranging the main windows as shown below. Once you made your choices:

Prefs ► Save Windowing Preferences

The screenshot shows the Stata 8.2 interface with three windows open:

- Review window:** Contains the command history:

```
sysuse auto.dta
summarize
generate gp100m = 100/mpg
label variable gp100m "Gallo
regress gp100m weight
```
- Variables window:** Lists variables in memory:

make	Make and M
price	Price
mpg	Mileage (mp
rep78	Repair recor
headroom	Headroom (i
trunk	Trunk space
weight	Weight (lbs.)
length	Length (in.)
turn	Turn Circle (
displacement	Displaceme
gear_ratio	Gear Ratio
foreign	Car type
gp100m	Gallons per
- Stata Results window:** Shows the output of the `regress` command:

```
. generate gp100m = 100/mpg
. label variable gp100m "Gallons per 100 miles"
. regress gp100m weight
```

Source	SS	df	MS			
Model	87.2964969	1	87.2964969	Number of obs =	74	
Residual	32.2797639	72	.448330054	F(1, 72) =	194.71	
Total	119.576261	73	1.63803097	Prob > F =	0.0000	

	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]	
weight	.001407	.0001008	13.95	0.000	.001206	.0016081
_cons	.7707669	.3142571	2.45	0.017	.1443069	1.397227

-more-

Command line window

In this one line window you may enter single commands and execute them by [Enter].

Review window

This window displays the most recent commands. Click a command in the Review window to paste it to the Command line window where you may edit and execute it. You may also scroll through past commands using the PgUp and PgDn keys.

Save past commands to a do-file by clicking the upper left Review window button and: Save Review Contents. Another option is to use the command log file (`cmdlog.txt`; section 1.2).

Variables window

You see a list of the variables in memory. Paste a variable name to the Command line window by clicking it. You may also paste a variable name to an active dialog field.

Results window

This is the primary display of the output. Its size is limited, and you can't edit it. You may print the contents, but in general my suggestion to let NoteTab print the log-file is handier.

Viewer window

[GSW] 3

The main use of this window is viewing help files (see *help* and *search*, section 4). You may select part of the viewer window for printing, using the mouse – but unfortunately not the keyboard – to highlight it.

Stata Viewer [Advice on Help]							
Back	Refresh	Search	Help!	Contents	What's New	News	
Command:	help viewer						
Using the Viewer				manual: [GS] 3 Using the Viewer			
In the Viewer, you can							
see help for contents or <u>help for any Stata command</u>							
<u>search help</u> files, documentation, and FAQs (<u>advice</u> on using search)							
<u>find and install</u> SJ, STB, and user-written programs from the net							
<u>review, manage, and uninstall</u> user-written programs							
check for and optionally install <u>official updates</u>							
<u>view</u> your logs or any file							
<u>launch</u> your browser							
see the <u>latest news</u> from www.stata.com							

Stata also suggests that you use the Viewer window for viewing and printing output (the log file), but it does not work well, and I find it much handier to use a general text editor (see section 3) for examining, editing and printing output.

Data window

[GSW] 9

The Data window looks like a spreadsheet, and you may use it for entering small amounts of data, but I don't recommend that, see section 8. A data entry program proper should be preferred, e.g. EpiData, see appendix 2.

In [GSW] 9 it is demonstrated how to use the *edit* command to make corrections directly in the data window. For reasons of safety and documentation I strongly discourage that. Modifications to data should be made with do-files; see *Take good care of your data*.

The *browse* command enables you to see but not modify data in the Data window. To see a specific cell issue the command:

```
browse age in 23 // the variable age in the 23rd observation
```

Do-file editor

[GSW] 15

This is a standard text editor used for writing do-files (see section 3.1). The do-file editor has the special feature that you may request Stata to execute all or a selection of the commands by clicking the Do-button. I prefer NoteTab Light to the do-file editor; see section 3.

3. Suggested mode of operation

The recommendations in this section only partly follow what is recommended in the [GSW] manual; I try to explain why, when my recommendations differ.

3.1. Issuing commands

The command line window

In the Command line window you may enter commands, one at a time, and execute them. This works well in simple tasks, but for complex tasks it is a lot safer and more efficient to write the commands in a do-file before execution.

The dialogs (menu system) [GSW] 2

The menu system enables you to generate quite complex commands without looking them up in the manuals. You may need to do some editing if you are not quite satisfied with the result.

For graphs and many analyses the menus are a blessing. But for documentation commands (section 9) and calculation commands (section 10) it is a lot easier to write the commands directly in the command window or in a do-file than to generate them by the menus.

Do-files [U] 19

See examples of do-files in section 16. A do-file is a series of commands to be executed in sequence. For any major tasks this is preferable to entering single commands because:

- You make sure that the commands are executed in the sequence intended.
- If you discover an error you can easily correct it and re-run the do-file.
- The do-file serves as documentation of what you did.

Use the do-file editor or any text editor like NoteTab (see appendix 3) to enter the commands. I prefer NoteTab to the do-file editor, because I have direct access to several do-files and the output in one place, although I cannot execute commands directly from NoteTab.

You may, after having issued a number of more or less successful commands, click the upper left Review window button to save the contents of the Review window as a do-file. The command log file (`cmdlog.txt`) may be used for the same purpose.

The `do` command

Find and execute a do-file by clicking: File ► Do...

or by entering path and filename in the command window:

```
do c:\dokumenter\proj1\alpha.do
```

You may also from the do-file editor execute the entire or a highlighted part of the current file by clicking the do-button (number two from the right). The disadvantage of this method is that the name of your do-file is not reflected in the output. I recommend issuing a `do` command with full path and name of the do-file, for reasons of documentation.

3.2. Handling output

Stata's output facilities are less than optimal. In this section I show how you can use the third-party program NoteTab to handle output for editing and printing.

The Results window

The output immediately appears in the Results window. You may print all of the Results window (be careful!) or a selected part of it. However, manoeuvring is more restricted than in a text editor, and you can use only the mouse, not the keyboard, to highlight a section. You cannot edit its contents. If you, like me, dislike the output interruptions by **-more-** you can:

```
set more off [, permanently]
```

The size of the Results window buffer is restricted, and you only have access to the last few pages of output. To increase the buffer size (default 32,000 bytes) permanently:

```
set scrollbufsize 200000
```

The Viewer window

This window is excellent to examine help files, see section 4. [GSW] section 3 and 17 also describe how to use it to examine and print output, but it is much too inflexible. The SMCL-formatted output is a mixed blessing, with parts of a table in bold, parts in plain text. And actually only the [GSW] manual uses SMCL-formatted output, the others use plain text.

The log files and NoteTab

You may generate two log files, a full log and a command log. [GSW] section 17 tells how to open and close log files. That didn't work well for me; I often forgot to open a log file. Now I let the start-up file **profile.do** create the log files automatically, see section 1.2.

The full log (**c:\tmp\stata.log**) is a copy of what you saw in the Results window. I use it to inspect, edit and print output in NoteTab. I selected plain ASCII text for the full log; it is overwritten next time you start Stata or when you issue the **newlog** command.

The **nt** command gives you rapid access to your output in NoteTab. See appendix 3 on how to create both commands.

The command log (**c:\tmp\cmdlog.txt**) includes all commands issued. It is cumulative, i.e. new commands are added to the file, which is not overwritten next time Stata is opened. You may use it instead of the Review window to see and recover previous commands.

Copying a table to a word processor document.

You might want to use a Stata table for publication. Copying a table directly to a word processor document does not work well, but you may use Excel as an intermediary:

1. Highlight the table in the Results window. Right-click it and select Copy Table
2. Open Excel and paste the table to it ([Ctrl]+V). Edit the table if needed.
3. Copy-and-paste the table from Excel to your document.

To do the trick correctly, Windows must be set to display decimal periods (unless you in Stata chose **set dp comma**).

4. Getting help

[GSW] 4, 19, 20; [U] 2, 8

4.1. The manuals

See recommendations in Appendix 1.

4.2. Online help

[GSW] 4; [U] 8, 32

Online help is best displayed in the Viewer window (see section 2 and 3). Issue **help** and **search** from the Viewer's command line, **whelp** and **findit** from the Stata command line. Keep your Stata updated – then your online help is updated too.

help and **whelp**

If you know the command name (e.g. **tabulate**) see the help file (tabulate.hlp) by:

help tabulate from the Viewer command line: *or*
whelp tabulate from Stata's command line window

The help file is displayed in the Viewer window, and from here you may print it. You may also use the links included. Try it.

search and **findit**. Using the menus

You need not know the command name. Get information about nonparametric tests by:

search nonparametric from the Viewer command line

To search Stata and the net for information on goodness-of-fit tests with poisson regression:

findit poisson goodness from Stata's command line window

You may also use the menus to locate a command:

Statistics ► Summaries, tables & tests ► Nonparametric tests
and find quite a few procedures.

FAQs (Frequently asked questions)

Try www.stata.com/support/faqs. This site includes a lot of advice on various topics.

Statalist and the Danish Statanewcomerlist

Statalist is a forum for questions from users; see www.stata.com/support/statalist.

A Danish list especially for newcomers has been established by University of Southern Denmark and Aarhus University; see www.biostat.sdu.dk/statalist.html.

Error messages

Stata's short error messages include a code, e.g. **r(131)**. The code is a link, and by clicking it you get more clues. Some error messages, however, are not very informative.

5. Stata file types and names

[U] 14.6

.dta files: Stata data

The extension for a Stata data set is **.dta**. Stata data sets can only be created and interpreted by Stata itself.

.do files: command files

A do-file with the extension **.do** is a number of commands to be executed in sequence. Do-files are in standard ASCII format and can be edited and displayed by any text editor.

You may issue single commands in the command line window, but if you are doing anything substantial you should do it with a do-file. You find examples in section 16 and some useful examples in *Take good care of your data*. In both places I emphasize the importance of a system for naming do-files.

.ado files: programs

An ado-file with the extension **.ado** is a program. Ado-files are in standard ASCII format. For more information see section 15.7.

.hlp files: Stata help

Stata's online documentation is kept in **.hlp** files, written in SMCL format (somewhat like HTML). SMCL-formatted files can be displayed in the Viewer window.

.gph files: graphs

[GSW] 16; [G] (Graphics manual)

Stata graphs can be saved as **.gph** files; see section 14.8.

.dct files: dictionary files

[U] 24; [R] **infile**

Fixed format ASCII data may be read with **infile** using a dictionary file. I don't demonstrate this option in section 8.

6. Variables

A Stata data set is rectangular; here is one with five observations and four variables:

	Variables			
	obsno	age	height	weight
Observations	1	27	178	74
	2	54	166	67
	3	63	173	85
	4	36	182	81
	5	57	165	90

6.1. Variable names

Variable names can be 1-32 characters, but Stata often abbreviates long variable names in output, so I recommend to use only 8 characters. The letters **a-z** (but not **æøå**), the numbers **0-9** and **_** (underscore) are valid characters. Names must start with a letter (or an underscore, but this is discouraged because many Stata-generated variables start with an underscore). These are valid variable names:

```
a q17 q_17 pregnant sex
```

Stata is case-sensitive

Variable names may include lowercase and uppercase letters, but Stata is case-sensitive: **sex** and **Sex** are two different variable names. Throughout this booklet I use lowercase variable names; anything else would be confusing.

6.2. Numeric variables

[U] 15.2

Most often you don't need worry about numeric types, but if you encounter memory problems, you should know this (see section 15.1 on Memory considerations):

Type		Bytes	Precision (digits)	Range (approx.)
Integer	byte	1	2	± 100
	int	2	4	$\pm 32,000$
	long	4	9	$\pm 2 \times 10^9$
Floating point	float	4	7	$\pm 10^{36}$
	double	8	16	$\pm 10^{308}$

compress can reduce the size of your dataset considerably by finding the most economical way of storage.

Numeric formats

[U] 15.5.1

The default is General format, presenting values as readable and precisely as possible. In most cases you need not bother with numeric formats, but you may specify:

```
format dollars kroner %6.2f
```

Format	Formula	Example	$\sqrt{2}$	1,000	10,000,000
General	<code>%w.dg</code>	<code>%9.0g</code>	1.414214	1000	1.00e+07
Fixed	<code>%w.df</code>	<code>%9.0f</code>	1	1000	10000000
		<code>%9.2f</code>	1.41	1000.00	1.00e+07
Exponential	<code>%w.de</code>	<code>%10.3e</code>	1.414e+00	1.000e+03	1.000e+07

w: The total width, including period and decimals.

d: Number of decimals

You may use comma formats with Stata, but there are confusing limitations, and I don't recommend it. To display commas rather than periods (probably most relevant for graphs):

```
set dp comma
```

6.3. Missing values

[U] 15.2.1

Missing values are omitted from calculations. There are two types of missing values:

The *system missing value* is shown as a . (period). It is created in input when a numeric field is empty; by invalid calculations, e.g. division by 0, or calculations involving a missing value.

User-defined missing values are .a, .b, .c,z. It is a good idea to use a general principle consistently, e.g.:

- .a Question not asked (complications to an operation not performed)
- .b Question asked, no response
- .c Response: Don't know

Unfortunately no data entry program accepts .a in a numeric field. In EpiData you might choose the codes -1 to -3 (provided, of course, that they could not be valid codes) and let Stata recode them:

```
recode _a11 (-1=.a) (-2=.b) (-3=.c)
```

Missing values are high-end numbers; the ordering is:

All valid numbers < . < .a < .b < ... < .z

You need not bother about the actual numerical values behind the missing values, but you need to know the logics to avoid mistakes. In calculations missing values behave as expected, but not so in conditions. The following command initially surprised me by listing all whose age was > 65, and those with missing age:

```
list age if age > 65
```

To exclude the missing:

```
list age if age > 65 & age < .
```

To list the missing only, including the user-defined missing values:

```
list id age if age >= .           or  
list id age if missing(age)
```

7. Command syntax

[U] 14.1

Stata is case-sensitive, and all Stata commands are lowercase. Variable names may include lowercase and uppercase letters, but **sex** and **Sex** are two different variable names. Throughout this booklet I use lowercase variable names; anything else would be confusing. Uppercase variable names are sometimes used within programs (ado-files, see section 15.7) to avoid confusion with the variable names in the data set.

The general syntax of Stata commands can be written like this:

```
[prefix:] command [varlist] [if expression] [in range] [weight] [, options]
```

Qualifiers and options

Qualifiers are general to many commands. See below on **if**, **in** and weights.

Options are specific to a command. A comma precedes the option list. Missing or misplacing a comma is a frequent cause of error messages.

Command examples

Here are examples with the command **summarize** (mean, minimum, maximum etc.):

Prefix	Command	Varlist	Qualifiers	Options	Comments
	summarize				No varlist: All variables
	summarize	_all			_all : all variables
	summarize	sex age			Two variables
	summarize	sex-weight			Variables in sequence
	summarize	pro*			All variables starting with pro
	summarize	age	if sex==1		Males only
	summarize	bmi	in 1/10		First 10 observations
	summarize	bmi	[fweight=n]		Weighted observations
	sort	sex			Separate table for each sex.
by sex:	summarize	bmi			Data must be sorted first.
	summarize	bmi		, detail	Option: detail

Variable lists

[U] 14.1.1

A variable list (*varlist*) calls one or more variables to be processed. Examples:

(nothing)	sometimes the same as _all
_all	all variables in the data set
sex age pregnant	three variables
pregnant sex-weight	pregnant and the consecutive variables sex to weight
pro*	all variables starting with pro

In commands that have a dependent variable, it is the first in the *varlist*:

<code>oneway bmi sex</code>	<code>bmi</code> is the dependent variable
<code>regression bmi sex age</code>	<code>bmi</code> is the dependent variable
<code>scatter weight height</code>	scatterplot, weight is y-axis
<code>tabulate expos case</code>	The first variable defines the rows

Conditional commands. The `if` qualifier [U] 14.1.3.

The operators used in conditions are defined in section 10.1. Here are a few examples:

<code>summarize age if sex==1</code>	statistics for males only
<code>list id age if age < 35</code>	list only if <code>age < 35</code>
<code>replace npreg=. if sex==1</code>	set <code>npreg</code> to missing for males

Numeric ranges. [U] 14.1.4

Numeric ranges are marked by a slash:

```
recode age (1/9.999=1) (10/19.999=2) , generate(agegrp)
list sex age weight in 1/10 // observations 1 to 10
```

Number lists. The `in` qualifier [U] 14.1.8

A number list (*numlist*) is a list of numbers; there are some shorthand possibilities:

<code>1(3)11</code>	means	1 4 7 10
<code>1(1)4 4.5(0.5)6</code>	means	1 2 3 4 4.5 5 5.5 6
<code>4 3 2 7(-1)1</code>	means	4 3 2 7 6 5 4 3 2 1 (Danish CPR number test)
<code>1/5</code>	means	1 2 3 4 5
<code>4/2 7/1</code>	means	4 3 2 7 6 5 4 3 2 1 (Danish CPR number test)

Example:

```
list sex age weight in 1/10 // observations 1 to 10
tway line mort year , xlabel(1900(20)2000) // x-axis labels
```

Weighting observations [U] 14.1.6, [U] 23.13

A typical use is to 'multiply' observations when the input is tabular:

	Cases	Controls
Exposed	21	30
Unexposed	23	100
Total	44	130

```
. input expos case pop // see section 8
  1 1 21
  1 0 30
  0 1 23
  0 0 100
. end
. tabulate expos case [fweight=pop] , chi2 // see section 11.1
. cc expos case [fweight=pop] // see section 11.1
```

by and bysort prefix

[U] 14.5

Makes a command display results for subgroups of the data. Data must be pre-sorted:

```
sort sex
by sex: summarize age height weight
```

or, in one line:

```
bysort sex: summarize age height weight
```

Text strings, quotes

Stata requires *double* quotes around text strings, but you may omit quotes *unless* the string has embedded blanks or commas:

```
label define sex 1 male 2 female 9 "sex unknown"
```

You need not use quotes around filenames:

```
save c:\dokumenter\proj1\alfa1.dta
```

unless they include blank space:

```
save "c:\dokumenter\project 1\alfa1.dta"
```

Comments

[U] 19.1.2

The following is interpreted as comments, to include short explanations in a do-file:

- Lines that begin with *****
- text surrounded by **/*** and ***/**
- Text following **//** (the easy way; used in this booklet)

The purpose of comments is to make do-files more readable *to yourself* – Stata does not care whatever you write.

```
// C:\DOKUMENTER\PROFILE.DO executes when opening Stata
summarize bmi , detail // Body mass index
```

Long command lines

[U] 19.1.3

By default a command ends when the line ends (carriage return), and no special delimiter terminates commands. However, command lines in do- and ado-files should be no longer than 80 characters. The problem is solved by **///** telling Stata that the following line is a continuation.

```
infix str10 cprstr 1-10 bday 1-2 bmon 3-4 byear 5-6 ///
control 7-10 using c:\dokumenter\p1\datefile.txt
```

Another option is to define **;** (semicolon) as the future command delimiter:

```
#delimit ; // Semicolon delimits future commands
infix str10 cprstr 1-10 bday 1-2 bmon 3-4 byear 5-6
control 7-10 using c:\dokumenter\p1\datefile.txt ;
tab1 opagr ;
#delimit cr // Back to normal: Carriage return delimiter
```

8. Getting data into Stata

[U] 24; [GSW] 7

On exchange of data with other programs, see section 15.8.

Open Stata data

[R] **save**

Read an existing Stata data set from disk into memory by:

```
use c:\dokumenter\p1\a.dta [ , clear]
```

If there are data in memory, **use** will be rejected unless you specify the **clear** option.

You may also issue a **clear** command before the **use** command:

```
clear
```

If you want only observations that meet a condition:

```
use c:\dokumenter\p1\a.dta if sex==1
```

If you want the first 100 observations only:

```
use c:\dokumenter\p1\a.dta in 1/100
```

If you want to work with only a subset of variables:

```
use age sex q1-q17 using c:\dokumenter\p1\a.dta
```

Save Stata data

[R] **save**

Save the data in memory to a disk file by:

```
save c:\dokumenter\p1\a.dta [ , replace]
```

If you already have a disk file with this name, your request will be rejected unless you specify the **replace** option. *Only use the **replace** option if you really want to overwrite data.*

Enter data with EpiData

To enter data I recommend EpiData, available for free from www.epidata.dk. This easy-to-use program has all the facilities needed. Further information in appendix 2.

Enter data as commands or in a do-file

[R] **input**

Very small data sets. Define the variables with the **input** command and enter the values.

Finish with **end**. It can be done interactively from the command line or in a do-file. See more examples in section 14.7 (Graph examples).

```
. input case expos pop
  0 0 100
  0 1  30
  1 0  23
  1 1  21
. end
```

You may also enter data directly in Stata's data window (not recommended; see section 2 and [GSW] 6, 9).

Reading ASCII data

[U] 24

Reading tab- or comma-separated data

[R] **insheet**

In tab-separated data the tabulator character, here displayed as `<T>`, separates the values. A tab-separated ASCII file is created e.g. if you save an Excel worksheet as a text (.txt) file. If row 1 is variable names, Stata will find out and use them. In this and the following examples the value of **type** in observation 2 is missing.

```
id <T> type <T> sold <T> price
1 <T> 2 <T> 47 <T> 51.23
2 <T> <T> 793 <T> 199.70
```

You may read a tab-separated ASCII file with variable names in row 1 by the command:

```
insheet using c:\dokumenter\p1\a.txt , tab
```

In comma-separated data a comma separates each value:

```
1, 2, 47, 51.23
2, , 793, 199.70
```

If you have a comma-separated file without variable names in row 1 the command is:

```
insheet id type sold price using c:\dokumenter\p1\a.txt , comma
```

insheet assumes that all data belonging to one observation are in one line.

Reading freefield data

[R] **infile** (free format)

In freefield data commas or blanks separate each value:

```
1 2 47 51.23
2 . 793 199.70
```

If you have freefield data the command is

```
infile id type sold price using c:\dokumenter\p1\a.txt
```

infile does *not* assume that data belonging to one observation are in one line, and the following data are the same as the data above:

```
1 2 47 51.23 2 . 793 199.70
```

Reading fixed format data

[R] **infix**; [R] **infile** (fixed format)

In fixed format data the information on each variable is determined by the position in the line.

The blank **type** in observation 2 will be read as missing.

```
1 2 47 51.23
2 793 199.70
```

```
infix id 1 type 2-3 sold 4-7 price 8-14 using c:\dokumenter\p1\a.txt
```

Fixed format data can also be read by **infile**; to do this a dictionary file must be created, specifying variable names and positions etc. See [R] **infile** (fixed format).

9. Documentation commands

[GSW] 8

Stata does not need documentation commands; *you need the documentation yourself*. The output becomes more legible, and the risk of errors when interpreting the output is reduced.

Data set label [U] 15.6.1; [R] **label**

You can give a short description of your data, to be displayed every time you open (**use**) data.

```
label data "Fertility data Denmark 1997-99. ver 2.5, 19.9.2002"
```

It is wise to include the creation date, to ensure that you analyse the most recent version.

Variable labels [U] 15.6.2; [R] **label**

You can attach an explanatory text to a variable name.

```
label variable q6 "Ever itchy skin rash?"
```

Use informative labels, but make them short; they are sometimes abbreviated in output.

Value labels [U] 15.6.3; [R] **label**

You can attach an explanatory text to each code for a variable. This is a two-step procedure.

First define the label (double quotes around text with embedded blanks):

```
label define sexlbl 1 male 2 female 9 "sex unknown"
```

Next associate the label **sexlbl** with the variable **sex**:

```
label values sex sexlbl
```

Use informative labels, but make them short; value labels are often abbreviated to 12 characters in output.

Most often you will use the same name for the variable and its label:

```
label define sex 1 male 2 female  
label values sex sex
```

but the separate definition of the label enables you to reuse it:

```
label define yesno 1 yes 2 no  
label values q1 yesno  
label values q2 yesno
```

If you want to correct a label definition or add new labels, use the **modify** option:

```
label define sexlbl 9 "unknown sex" , modify
```

adds the label for code 9 to the existing label definition.

In output Stata unfortunately displays either the codes *or* the value labels, and you often need to see them both, to avoid mistakes. You may solve this by including the codes in the labels; this happens automatically with:

```
numlabel _all , add
```

See label definitions

See the value label definitions by:

label list or
labelbook

See the variable label definitions by:

describe

See a full codebook by:

codebook

Notes

[R] **notes**

You may add notes to your data set:

note: 19.9.2000. Corrections made after proof-reading

and to single variables:

note age: 20.9.2000. Ages > 120 and < 0 recoded to missing

The notes are kept in the data set and can be seen by:

notes

Notes are cumulative; old notes are not discarded (and that is nice)

10. Modifying data

Don't misinterpret the title of this section: Never modify your original data, but add modifications by generating new variables from the original data. Not documenting modifications may lead to serious trouble. Therefore modifications:

- should always be made with a do-file with a name reflecting what it does: **gen.alfa2.do** generates **alfa2.dta**.
- The first command in the do-file reads data (eg. *use*, *infix*).
- The last command saves the modified data set with a *new* name (*save*).
- The do-file should be 'clean', ie. not include commands irrelevant to the modifications.

See examples of modifying do-files in section 16 and in *Take good care of your data*.

10.1. Calculations

Operators in expressions [GSW] 12; [U] 16.2

Arithmetic	Relational	Logical
^ power	> greater than	! not
* multiplication	< less than	~ not
/ division	>= > or equal	 or
+ addition	<= < or equal	& and
- subtraction	== equal	
	!= not equal	
	~= not equal	

Arithmetic operators

```
generate alcohol = beers+wines+spirits
generate bmi = weight/(height^2)
```

The precedence order of arithmetic operators are as shown in the table; power before multiplication and division, before addition and subtraction. Control the order by parentheses; however the parentheses in the last command were not necessary since power takes precedence over division – but they didn't harm either.

Relational and logical operators

```
replace salary = . if age<16 // salary set to missing
summarize age if sex==1
list sex age weight height if sex==1 & age <= 25
keep if sex==1 | age <= 25
```

Logical expressions can be true or false; a value of 0 means false, any other value (including missing values) means true. This means that with **sex** coded 1 for males and 0 for females and no missing values, the second command could have been written as:

```
summarize age if sex
```

generate; replace

[R] generate

Generate a new variable by:

```
generate bmi=weight/(height^2)
```

If the target variable (**bmi**) already exists in the data set, use **replace**:

```
replace bmi=weight/(height^2)
```

Do a conditional calculation (males only):

```
generate mbmi=1.1*bmi if sex==1
```

Besides the standard operators there are a number of functions:

[R] functions

<code>generate y=abs(x)</code>	absolute value <code> x </code>
<code>gen y=exp(x)</code>	exponential, e^x
<code>gen y=ln(x)</code>	natural logarithm
<code>gen y=log10(x)</code>	base 10 logarithm
<code>gen y=sqrt(x)</code>	square root
<code>gen y=int(x)</code>	integer part of x . <code>int(5.8) = 5</code>
<code>gen y=round(x)</code>	nearest integer. <code>round(5.8) = 6</code>
<code>gen y=round(x, 0.25)</code>	<code>round(5.8, 0.25) = 5.75</code>
<code>gen y=mod(x1,x2)</code>	modulus; the remainder after dividing x1 by x2
<code>gen y=max(x1,...xn)</code>	maximum value of arguments
<code>gen y=min(x1,...xn)</code>	minimum value of arguments
<code>gen y=sum(x)</code>	cumulative sum across observations, from first to current obs.
<code>gen y=_n</code>	<code>_n</code> is the observation number
<code>gen y=_N</code>	<code>_N</code> is the number of observations in the data set

egen

[R] egen

egen (extensions to generate) gives some more useful functions. It may be confusing that functions working differently with **generate** and **egen** have the same names, so take care. Here are some examples:

Generating the same value for all observations

<code>egen meange=mean(age)</code>	Mean age across observations
<code>by sex: egen meange=mean(age)</code>	Mean age across observation, for each sex
<code>egen sumage=sum(age)</code>	Sum of age across all observations (unlike generate's sum)
<code>egen maxage=max(age)</code>	Maximum value of age across observations
<code>egen minage=min(age)</code>	Minimum value of age across observations

Generating individual values for each observation (each row)

<code>egen minq=rmin(q1-q17)</code>	Min. value of q1-q17 for this observation
<code>egen maxq=rmax(q1-q17)</code>	Max. value of q1-q17 for this observation
<code>egen meanq=rmean(q1-q17)</code>	Mean value of q1-q17 for this observation
<code>egen sumq=rsum(q1-q17)</code>	Sum of q1-q17 for this observation
<code>egen valq=robs(q1-q17)</code>	Number of non-missing q1-q17 for this observation

recode

[R] **recode**

Changes a variable's values, e.g. for grouping a continuous variable into few groups. The inverted sequence ensures that age 55.00 (at the birthday) goes to category 4:

```
recode age (55/max=4) (35/55=3) (15/35=2) (min/15=1) , generate(agegr)
```

Value labels for the new variable may be included at once:

```
recode age (55/max=4 "55+") (35/55=3 "35-54") (15/35=2 "15-34") ///  
(min/15=1 "-14") , generate(agegr)
```

Very important: The *generate* option creates a new variable with the recoded information; without *generate* the original information in *age* will be destroyed.

Other examples:

<code>recode expos (2=0)</code>	Leave other values unchanged
<code>recode expos (2=0) , gen(exp2)</code>	Values not recoded transferred unchanged
<code>recode expos (2=0) if sex==1</code>	Values not recoded (<code>sex != 1</code>) set to missing
<code>recode expos (2=0) if sex==1 , copy</code>	Values not recoded (<code>sex != 1</code>) unchanged
<code>recode expos (2=0) (1=1) (else=.)</code>	Recode remaining values to missing (.)
<code>recode expos (missing=9)</code>	Recode any missing (., .a, .b etc.) to 9

Another way to recode continuous data into groups:

[R] **egen**

```
egen agegrp=cut(age) , at (0 5(10)85 120)
```

age	agegrp
$0 \leq \text{age} < 5$	0
$5 \leq \text{age} < 15$	5
$15 \leq \text{age} < 25$	15
..	..
$85 \leq \text{age} < 120$	85

for

Enables you with few command lines to repeat a command. To do the modulus 11 test for Danish CPR numbers (see section 15.3) first multiply the digits by 4,3,2,7,6,5,4,3,2,1; next sum these products; finally check whether the sum can be divided by 11. The CPR numbers were split into 10 one-digit numbers *c1-c10*:

```
generate test = 0  
for C in varlist c1-c10 \ x in numlist 4/2 7/1 : ///  
  replace test = test + C*X  
  replace test = mod(test,11) // Remainder after division by 11  
  list id cpr test if test !=0
```

C and *x* are stand-in variables (names to be chosen by yourself; note the use of capital letters to distinguish from existing variables), to be sequentially substituted by the elements in the corresponding list. Each list must be declared by type; there are four types: *newlist* (list of new variables), *varlist* list of existing variables, *numlist* (list of numbers), *anylist* (list of words).

for is not documented in the manuals any more. The *foreach* and *forvalues* commands partially replace it, but they don't handle parallel lists as shown above. See section 15.7.

10.2. Selections

Selecting observations

[GSW] 13; [R] **drop**

You may remove observations from the data in memory by:

```
keep if sex == 1           or, with the same effect:  
drop if sex != 1
```

You may restrict the data in memory to the first 10 observations:

```
keep in 1/10
```

A selection may be requested already when opening a data set:

```
use c:\dokumenter\p1\a.dta if sex == 1
```

Observations dropped can only be returned to memory with a new **use** command. However, **preserve** and **restore** (documented in [P]) let you obtain a temporary selection:

```
preserve    // preserve a copy of the data currently in memory  
keep if sex == 1  
    calculations  
    analyses  
restore    // reload the preserved dataset
```

Selecting variables

[GSW] 13; [R] **drop**

You may remove variables from the data in memory by:

```
keep sex age-weight       and by:  
drop sex age-weight
```

A selection may be requested already when opening a data set:

```
use sex age-weight using c:\dokumenter\p1\a.dta
```

Sampling

[R] **sample**

Keep a 10% random sample of the observations:

```
sample 10
```

To obtain a sample of exactly 57 observations:

```
sample 57 , count
```

10.3. Renaming and reordering variables

Renaming variables

[R] **rename**

```
rename koen sex
```

The variable name **koen** is changed to **sex**. Contents and labels are unchanged.

Reordering variables

[R] **order**

To change the sequence of variables specify:

```
order id age-weight
```

The new sequence of variables will be as defined. Any variables not mentioned will follow after the variables mentioned.

10.4. Sorting data

sort

[R] **sort**, [R] **gsort**

To sort your data according to **mpg** (primary key) and **weight** (secondary key):

```
sort mpg weight
```

sort only sorts in ascending order; **gsort** is more flexible, but slower. To sort by **mpg** (ascending) and **weight** (descending) the command is:

```
gsort mpg -weight
```

10.5. Numbering observations

[U] 16.7

The variable **age** in the third observation can be referred to as **age[3]**. The principle is:

First observation	age[1]
Last observation	age[_N]
Current observation	age[_n]
Previous (lag) observation	age[_n-1]
Next (lead) observation	age[_n+1]
Observation 27	age[27]

From a patient register you have information about hospital admissions, one or more per person, identified by **cpr** and **admdate** (admission date). You want to construct the following variables: **obsno** (observation number), **persno** (internal person ID), **admno** (admission number), **admtot** (patient's total number of admissions).

```
. use c:\dokumenter\proj1\alfa1.dta
. sort cpr admdate
. gen obsno=_n // _n is the observation number
. by cpr: gen admno=_n // _n is obs. number within each cpr
. by cpr: gen admtot=_N // _N is total obs. within each cpr
. sort admno cpr // all admno==1 first
. gen persno=_n if admno==1 // give each person number if admno==1
. sort obsno // original sort order
. replace persno=persno[_n-1] if persno >=. // replace missing persno
. save c:\dokumenter\proj1\alfa2.dta
. list cpr admdate obsno persno admno admtot in 1/7

      cpr      admdate  obsno  persno  admno  admtot
1. 0605401234 01.05.1970      1      1      1      3
2. 0605401234 06.05.1970      2      1      2      3
3. 0605401234 06.05.1971      3      1      3      3
4. 0705401234 01.01.1970      4      2      1      1
5. 0705401235 01.01.1970      5      3      1      1
6. 0805402345 01.01.1970      6      4      1      2
7. 0805402345 10.01.1970      7      4      2      2

. summarize persno // number of persons (max persno)
. anycommand if admno==1 // first admissions
. anycommand if admno==admtot // last admissions
. tab1 admtot if admno==1 // distribution of n of admissions
```

You may also create a keyfile linking **cpr** and **persno**, and remove **cpr** from your analysis file. See example 16b in *Take good care of your data*.

10.6. Combining files

[U] 25

Appending files

[R] **append**

To combine the information from two files with the same variables, but different persons:

```
// c:\dokumenter\proj1\gen.filab.do
use c:\dokumenter\proj1\fila.dta , clear
append using c:\dokumenter\proj1\filb.dta
save c:\dokumenter\proj1\filab.dta
```

Merging files

[R] **merge**

To combine the information from two files with different information about the same persons:

```
// c:\dokumenter\proj1\gen.filab.do
use c:\dokumenter\proj1\fila.dta , clear
merge lbnr using c:\dokumenter\proj1\filb.dta
save c:\dokumenter\proj1\filab.dta
```

Both files must be sorted beforehand by the matching key (**lbnr** in the example above), and the matching key must have the same name in both data sets. Apart from the matching key the variable names should be different. Below A and B symbolize the variable set in the input files, and numbers represent the matching key. Missing information is shown by . (period):

fila	filb	filab	_merge
1A	1B	1AB	3
2A		2A.	1
	3B	3.B	2
4A1	4B	4A1B	3
4A2		4A2B	3

Stata creates the variable **_merge** which takes the value 1 if only data set 1 (**fila**) contributes, 2 if only data set 2 (**filb**) contributes, and 3 if both sets contribute. Check for mismatches by:

```
tab1 _merge
list lbnr _merge if _merge < 3
```

For **lbnr** 4 there were two observations in **fila**, but only one in **filb**. The result was two observations with the information from **filb** assigned to both of them. This enables to distribute information eg. about doctors to each of their patients – if that is what you desire. But what if the duplicate **lbnr** 4 was an error? To check for duplicate id's before merging, sort and compare with the previous observation:

```
sort lbnr
list lbnr if lbnr==lbnr[_n-1]
```

Another way to check for and list observations with duplicate id's is:

```
duplicates report lbnr
duplicates list lbnr
```

merge is a lot more flexible than described here; see [R] **merge**.

10.7. Reshaping data

collapse

[R] **collapse**

You want to create an aggregated data set, not with the characteristics of each individual, but of groups of individuals. One situation might be to characterize physicians by number of patient contacts, another to make a reduced data set for Poisson regression (see section 13):

```
. // gen.stcollaps.cancer2.do
. use c:\dokumenter\proj1\stsplitt.cancer2.dta , clear
. collapse (sum) risktime died , by(agegr drug)
. save c:\dokumenter\proj1\stcollaps.cancer2.dta
. summarize
```

Variable	Obs	Mean	Std. Dev.	Min	Max
drug	15	2	.8451543	1	3
agegr	15	55	7.319251	45	65
risktime	15	4.133334	3.01067	.3581161	10.88906
died	15	2.066667	2.374467	0	8

reshape

[R] **reshape**

E.g. with repeated measurements some analyses require a 'wide', some a 'long' data structure:

Long structure

id	time	sbp
1	1	140
1	2	120
1	3	130
1	4	135
2	1	155
etc.		

Wide structure

id	sbp1	sbp2	sbp3	sbp4
1	140	120	130	135
2	155	etc.		

Switch between structures by:

```
reshape wide sbp , i(id) j(time) // Restructures long to wide
reshape long sbp , i(id) j(time) // Restructures wide to long
```

xpose

[R] **xpose**

You may transpose observations and variables, i.e. let observations become variables and variables become observations. This may be useful e.g. for restructuring data for a graph:

```
xpose , clear // clear is not optional
```

expand

[R] **expand, contract**

You may duplicate observations according to a weighting variable. Look at the example in section 7 about weighting observations. You can obtain the same result by expanding the four observations to 130:

```
expand pop
tabulate expos case , chi
```

contract does the opposite of **expand**.

11. Description and analysis

This section gives information on the simpler statistical commands with examples of output.

summarize

[R] **summarize**

summarize gives an overview of variables. It is useful for an initial screening of the data, especially the Obs column giving the number of non-missing observations, and the Min and Max columns.

```
. summarize
```

Variable	Obs	Mean	Std. Dev.	Min	Max
id	37	19	10.82436	1	37
type	37	1.891892	.9656254	1	4
price	35	46.58	16.3041	11.95	78.95
rating	35	2.514286	.9194445	1	4

Obtain detailed information on the distribution of selected variables by the **detail** option:

```
summarize price , detail
```

Find the nice modification **summv1**, also displaying variable labels, by: **findit summv1**.

```
. summv1
```

Variable	Obs	Mean	Std.Dev	Min	Max	Label
id	37	19	10.8244	1	37	identification number
type	37	1.89189	.965625	1	4	type of wine
price	35	46.58	16.3041	11.95	78.95	price per 75 cl bottle
rating	35	2.51429	.919444	1	4	quality rating

list

[GSW] 11; [R] **list**

Case listings are useful to examine data, to check the result of calculations, and to locate errors. The following lists sex-age for the first 10 observations. Codes rather than value labels are displayed.

```
list sex-age in 1/10 , nolabel
```

Stata's listing facilities are clumsy when you want to list many variables simultaneously. Find and install the useful alternative **slist** by: **findit slist**.

11.1. Categorical data

tab1 (simple frequency tables) and **tab2** (crosstables) are both described in [R] **tabulate**.

tab1

tab1 gives one-way tables (frequency tables) for one or more variables:

```
. tab1 rating type          (table for type not shown)
-> tabulation of rating
      quality |
      rating |      Freq.      Percent      Cum.
-----+-----
```

quality rating	Freq.	Percent	Cum.
1. poor	6	17.14	17.14
2. acceptable	9	25.71	42.86
3. good	16	45.71	88.57
4. excellent	4	11.43	100.00
-----+-----			
Total	35	100.00	

tab2

[R] **tabulate**

tab2 with two variables give a two-way table (crosstable). In the following you see the use of three options. **column** requests percentage distributions by column; **chi2** a χ^2 test, and **exact** a Fisher's exact test:

```
. tab2 rating type , column chi2 exact
-> tabulation of rating by type
      quality |
      rating |      1 red      2 white      3 rosé      4 undeter |      Total
-----+-----
```

quality rating	1 red	2 white	3 rosé	4 undeter	Total
1. poor	4	1	1	0	6
	25.00	9.09	20.00	0.00	17.14
(Part of output omitted)					
4. excellent	2	1	0	1	4
	12.50	9.09	0.00	33.33	11.43
-----+-----					
Total	16	11	5	3	35
	100.00	100.00	100.00	100.00	100.00
-----+-----					
Pearson chi2(9) =		6.9131	Pr = 0.646		
Fisher's exact =			0.653		

You can request a three-way table (a two-way table for each value of **nation**) with:

```
bysort nation: tabulate rating type
```

The command:

```
tab2 rating type nation
```

gives three two-way tables, one for each of the combinations of the variables. But beware: you can easily produce a huge number of tables.

[P] **foreach**

Imagine that you want 10 two-way tables: each of the variables **q1-q10** by **sex**. With **tabulate** you must issue 10 commands to obtain the result desired. If you call **tab2** with 11 variables you get 55 two-way tables: all possible combinations of the 11 variables. The **foreach** command (see section 15.7) lets you circumvent the problem:

```
foreach Q of varlist q1-q10 {
  tabulate `Q' sex
}
```

The local macro **Q** is a stand-in for **q1** to **q10**, and the commands generate 10 commands:

```
tabulate q1 sex
tabulate q2 sex  etc.
```

tabi

[R] tabulate

tabi is an 'immediate' command (see section 15.5) enabling you to analyse a table without first creating a data set. Just enter the cell contents, delimiting the rows by \ (backslash):

```
. tabi 10 20 \ 17 9 , chi exact
```

row	col		Total
	1	2	
1	10	20	30
2	17	9	26
Total	27	29	56

Pearson chi2(1) = 5.7308 Pr = 0.017
 Fisher's exact = 0.031
 1-sided Fisher's exact = 0.016

epitab

[ST] epitab

The commands in the **epitab** family perform stratified analysis. Here I show **cc**.

```
. cc case exposed , by(age) woolf
```

Maternal age	OR	[95% Conf. Interval]		M-H Weight
<35	3.394231	.9048403	12.73242	.7965957 (Woolf)
35+	5.733333	.5016418	65.52706	.1578947 (Woolf)
Crude	3.501529	1.110362	11.04208	(Woolf)
M-H combined	3.781172	1.18734	12.04142	

Test of homogeneity (M-H) chi2(1) = 0.14 Pr>chi2 = 0.7105
 Test that combined OR = 1:
 Mantel-Haenszel chi2(1) = 5.81
 Pr>chi2 = 0.0159

All procedures perform stratified analysis (Mantel-Haenszel). **cc** gives odds ratios for each stratum and the Mantel-Haenszel estimate of the common odds ratio. The test of homogeneity tests whether the odds ratio estimates could reflect a common odds ratio.

Command	Measure of association	Immediate command
ir	Incidence rate ratio, incidence rate difference	iri
cs	Cohort studies: Risk ratio, risk difference	csi
cc	Case-control studies: Odds ratio	cci
tabodds	Odds ratio, several exposure levels. Trend test	
mhodds	Odds ratio, several exposure levels. Trend test	
mcc	Odds ratio (matched case-control data)	mcci

If you want to stratify by more than one variable, the following command is useful:

```
egen racesex=group(race sex)
cc case exposed , by(racesex)
```

The immediate commands do not perform stratified analysis; an example with **cci**. Just enter the four cells (a b c d) of the 2×2 table:

```
cci 10 20 17 9 , woolf
```

11.2. Continuous variables

oneway

[R] **oneway**

compares means between two or more groups (analysis of variance):

```
oneway price type [ , tabulate noanova]
```

```
. oneway price type , tabulate
```

type of wine	Summary of price per 75 cl bottle		
	Mean	Std. Dev.	Freq.
1. red	48.15	12.650239	15
2. white	42.590909	20.016952	11
3. rosé	43.45	17.375268	6
4. undeterm	59.616666	15.821924	3
Total	46.58	16.304104	35


```
Analysis of Variance
```

Source	SS	df	MS	F	Prob > F
Between groups	780.660217	3	260.220072	0.98	0.4162
Within groups	8257.34954	31	266.366114		
Total	9038.00975	34	265.823816		

```
Bartlett's test for equal variances: chi2(3) = 2.3311 Prob>chi2 = 0.507
```

The table, but not the test, could also be obtained by;

```
tabulate type , summarize(price)
```

[R] **tabsum**

anova

[R] **anova**

Similar to **oneway**, but handles a lot of complex situations.

tabstat

[R] **tabstat**

tabstat is a flexible tool for displaying several types of tables, but includes no tests. To obtain a number of descriptive statistics; here the mean, standard deviation, and 25, 50, and 75 percentiles:

```
. tabstat price mpg weight , stat(n mean sd p25 p50 p75) col(stat) format(%8.2f)
```

variable	N	mean	sd	p25	p50	p75
price	74.00	6165.26	2949.50	4195.00	5006.50	6342.00
mpg	74.00	21.30	5.79	18.00	20.00	25.00
weight	74.00	3019.46	777.19	2240.00	3190.00	3600.00

The **col(stat)** option let the statistics form the columns; without it the statistics would have formed the rows. The **format()** option lets you decide the display format. The statistics are:

n, **mean**, **sum**, **min**, **max**, **range**, **sd**, **var**, **cv** (coefficient of variation), **semean**, **skew** (skewness), **kurt** (kurtosis), **p1**, **p5**, **p10**, **p25**, **p50** (or median), **p75**, **p90**, **p95**, **p99**, **q** (quartiles: **p25**, **p50**, **p75**), and **iqr** (interquartile range).

ttest

[R] **ttest**

T-test for comparison of means for continuous normally distributed variables:

<code>ttest bmi , by(sex)</code>	Standard t-test, equal variances assumed
<code>ttest bmi , by(sex) unequal</code>	Unequal variances (see <code>sdtest</code>)
<code>ttest prebmi==postbmi</code>	Paired comparison of two variables
<code>ttest prebmi==postbmi , unpaired</code>	Unpaired comparison of two variables
<code>ttest bmidiff==0</code>	One-sample t-test
<code>ttesti 32 1.35 .27 50 1.77 .33</code> n1 m1 sd1 n2 m2 sd2	Immediate command. Input n, mean and SD for each group

Distribution diagnostics

Diagnostic plots:

[R] **diagplots**

<code>pnorm bmi</code>	Normal distribution (P-P plot)
<code>qnorm bmi</code>	Normal distribution (Q-Q plot)

Formal test for normal distribution:

[R] **swilk**

<code>swilk bmi</code>	Test for normal distribution
------------------------	------------------------------

Test for equal variances:

[R] **sdtest**

<code>sdtest bmi , by(sex)</code>	Compare SD between two groups
<code>sdtest prebmi==postbmi</code>	Compare two variables

Bartlett's test for equal variances is displayed by `oneway`, see above.

Non-parametric tests

For an overview of tests available, in the Viewer window command line enter:

```
search nonparametric
```

Here you see e.g.

<code>kwallis</code>	Kruskall-Wallis equality of populations rank test
<code>signrank</code>	Sign, rank, and median tests (Wilcoxon, Mann-Whitney)

Another way to look for nonparametric tests is via the menu system:

Statistics ► Summaries, tables & tests ► Nonparametric tests

12. Regression analysis

Performing regression analysis with Stata is easy. Defining regression models that give sense is more complex. Especially consider:

- If you look for causes, make sure your model is meaningful. Don't include independent variables that represent steps in the causal pathway; it may create more confounding than it prevents. Automatic selection procedures are available in Stata (see [R] **sw**), but they may seduce the user to non-thinking. I will not describe them.
- If your hypothesis is non-causal and you only look for predictors, logical requirements are more relaxed. But make sure you really are looking at *predictors*, not consequences of the outcome.
- Take care with closely associated independent variables, e.g. education and social class. Including both may obscure more than illuminate.

12.1. Linear regression

regress [R] **regress**

A standard linear regression with **bmi** as the dependent variable:

```
regress bmi sex age
```

xi: [R] **xi**

The **xi:** prefix handles categorical variables in regression models. From a five-level categorical variable **xi:** generates four indicator variables; in the regression model they are referred to by the **i.** prefix to the original variable name:

```
xi: regress bmi sex i.agegrp
```

You may also use **xi:** to include interaction terms:

```
xi: regress bmi age i.sex i.treat i.treat*i.sex
```

By default the first (lowest) category will be omitted, i.e. be the reference group. You may, before the analysis, select **agegrp 3** to be the reference by defining a 'characteristic':

```
char agegrp[omit] 3
```

predict [R] **predict**

After a regression analysis you may generate predicted values from the regression coefficients, and this may be used for studying residuals:

```
regress bmi sex age
predict pbmi
generate rbmi = bmi-pbmi
scatter rbmi pbmi
```

or use **rvfplot**, see below

Regression diagnostics [R] **Regression diagnostics**

The chapter is very instructive. Get a residual plot with a horizontal reference line by:

```
rvfplot , yline(0)
```

12.2. Logistic regression

logistic

[R] **logistic**

A standard logistic regression with **ck** as the dependent variable:

```
logistic ck sex smoke speed alc
```

The dependent variable (**ck**) must be coded 0/1 (no/yes). If the independent variables are also coded 0/1 the interpretation of odds ratios is straightforward, otherwise the odds ratios must be interpreted per unit change in the independent variable.

The **xi:** prefix applies as described in section 12.1:

```
xi: logistic ck sex i.agegrp i.smoke  
xi: logistic ck i.sex i.agegrp i.smoke i.sex*i.smoke
```

After running **logistic**, use **predict** as described in section 12.1:

```
predict
```

After running **logistic** obtain Hosmer-Lemeshow's goodness-of-fit test with 10 groups:

```
lfit , group(10)
```

After running **logistic** obtain a classification table, including sensitivity and specificity with a cut-off point of your choice:

```
lstat , cutoff(0.3)
```

Repeat **lstat** with varying cut-off points or, smarter, use **lsens** to see sensitivity and specificity with varying cutoff points:

```
lsens
```

See a ROC curve:

```
lroc
```

13. Survival analysis and related issues

st

[ST] manual

The **st** family of commands includes a number of facilities, described in the Survival Analysis manual [ST]. Here I describe the **stset** and **stsplit** commands and give a few examples. The data is **cancer1.dta**, a modification of the **cancer.dta** sample data accompanying Stata.

The observation starts at randomization (**agein**), the data set includes these variables:

```
. summv1           // summv1 is a summarize displaying variable labels.
                    // Get it by: findit summv1
```

Variable	Obs	Mean	Std.Dev	Min	Max	Label
lbnr	48	24.5	14	1	48	Patient ID
drug	48	1.875	.841099	1	3	Drug type (1=placebo)
drug01	48	.583333	.498224	0	1	Drug: placebo or active
agein	48	56.398	5.6763	47.0955	67.8746	Age at randomization
ageout	48	57.6896	5.45418	49.0122	68.8737	Age at death or cens.
risktime	48	1.29167	.854691	.083333	3.25	Years to death or cens.
died	48	.645833	.483321	0	1	1 if patient died

stset

[ST] **stset**

stset declares the data in memory to be survival time (st) data. I create two versions: In **st.cancer1.dta** time simply is **risktime**, age not taken into consideration. In **st.cancer2.dta** time at risk is defined by age at entry (**agein**) and exit (**ageout**) enabling to study and control for the effect of age.

Simple analysis – age not included

stset data with **risktime** as the time-of-exit variable:

```
. // c:\dokumenter\proj1\gen.st.cancer1.do
. use c:\dokumenter\proj1\cancer1.dta , clear
. stset risktime , failure(died==1) id(lbnr)

           id: lbnr
    failure event: died == 1
obs. time interval: (risktime[_n-1], risktime]
exit on or before: failure

-----
    48 total obs.
     0 exclusions

-----
    48 obs. remaining, representing
    48 subjects
    31 failures in single failure-per-subject data
    62 total analysis time at risk, at risk from t =           0
           earliest observed entry t =           0
           last observed exit t =           3.25
```

```
. summarize
-----+-----
Variable | Obs      Mean      Std. Dev.      Min      Max
-----+-----
  lbnr   |    48     24.5       14           1       48
  ....   |
 risktime |    48     1.291667   .8546908   .0833333   3.25
   died   |    48     .6458333   .4833211     0         1
   _st    |    48         1         0           1         1
   _d     |    48     .6458333   .4833211     0         1
   _t     |    48     1.291667   .8546908   .0833333   3.25
   _t0    |    48         0         0           0         0
. save c:\dokumenter\proj1\st.cancer1.dta
```

Four new variables were created, and the `st`'ed data set is prepared for a number of incidence rate and survival analyses:

- `_st` 1 if the observation includes valid survival time information, otherwise 0
- `_d` 1 if the event occurred, otherwise 0 (censoring)
- `_t` time or age at observation end (here: `risktime`)
- `_t0` time or age at observation start (here: 0)

```
Summary of time at risk and incidence rates [ST] stptime
stptime , by(drug) per(1000) dd(4) // rates x 1000, 4 decimals
stptime , at(0(1)5) by(drug) // 1-year intervals
```

```
A table of the survivor function: [ST] sts list
sts list , by(drug) compare at(0(0.5)5) // ½ year intervals
```

```
The corresponding graph: [ST] sts graph
sts graph , by(drug)
```

```
To obtain a cumulative incidence (1-S) graph displaying the interval 0-0.25 at the y-axis:
sts graph , by(drug) failure ylabel(0(0.05)0.25)
```

```
A logrank test comparing two or more groups: [ST] sts test
sts test drug
```

```
Cox proportional hazards regression analysis: [ST] stcox
stcox drug01 // drug dichotomized
xi: stcox i.drug // 3 drugs
```

Including age in the analysis

stset data with **ageout** as the time-of-exit variable, **agein** as the time-of-entry variable:

```
. // c:\dokumenter\proj1\gen.st.cancer2.do
. use c:\dokumenter\proj1\cancer1.dta , clear
. stset ageout , enter(time agein) failure(died==1) id(lbnr)
. summarize
  Variable |      Obs      Mean   Std. Dev.      Min      Max
-----+-----
      lbnr |       48      24.5      14          1       48
      ...  |
      _st  |       48         1         0          1         1
      _d  |       48      .6458333   .4833211         0         1
      _t  |       48      57.61966   5.444583   49.87939   68.70284
      _t0 |       48      56.328    5.659862   47.97637   67.73915
. save c:\dokumenter\proj1\st.cancer2.dta
```

The **sts** and **stcox** analyses as shown above now must be interpreted as age-adjusted (delayed entry analysis). Summary of time at risk and age-specific incidence rates:

```
stptime , at(45(5)70) by(drug) // 5 year age intervals
```

stsplit

[ST] stsplit

To look at the influence of age at incidence or survival, **stsplit** the data, expanding each observation to an observation for each age interval:

```
. // c:\dokumenter\proj1\gen.stsplit.cancer2.do
. use c:\dokumenter\proj1\st.cancer2.dta , clear
. stsplit agegr , at(45(5)70)
. summarize
  Variable |      Obs      Mean   Std. Dev.      Min      Max
-----+-----
      lbnr |       61      26.37705   14.03586         1       48
      drug |       61      1.967213   .8557105         1         3
      drug01 |       61      .6229508   .4886694         0         1
      agein |       61      55.58628   5.651005   47.09552   67.87458
      ageout |       61      56.87054   5.563221   49.01218   68.8737
      risktime |       61      1.412568   .8856398   .0833333         3.25
      died  |       48      .6458333   .4833211         0         1
      _st  |       61         1         0          1         1
      _d  |       61      .5081967   .5040817         0         1
      _t  |       61      56.87054   5.563221   49.01218   68.8737
      _t0 |       61      55.85415   5.610502   47.09552   67.87458
      agegr |       61      54.01639   5.832357         45         65
. save c:\dokumenter\proj1\stsplit.cancer2.dta
```

The data now has 61 observations with events and risktime distributed to the proper age intervals. Describe risktime etc. by:

```
bysort drug: stsum , by(agegr)
```

poisson

[R] poisson

The `stsplit.cancer2.dta` data set above can be used for Poisson regression with a little more preparation. `died` and `risktime` *must* be replaced as shown. You also *may* collapse the file to a table with one observation for each age group and drug (see section 10.7):

```
. // c:\dokumenter\proj1\gen.stcollaps.cancer2.do
. use c:\dokumenter\proj1\stsplit.cancer2.dta , clear
. replace died = _d
. replace risktime = _t - _t0
. summarize
-----+-----
Variable |      Obs      Mean   Std. Dev.   Min       Max
-----+-----
.... |
risktime |       61   1.016394   .7343081   .0833321     2.75
died |       61   .5081967   .5040817         0         1
  _st |       61         1         0         1         1
  _d |       61   .5081967   .5040817         0         1
  _t |       61   56.87054   5.563221   49.01218   68.8737
  _t0 |       61   55.85415   5.610502   47.09552   67.87458
agegr |       61   54.01639   5.832357         45         65
. collapse (sum) risktime died , by(agegr drug)
. summarize
-----+-----
Variable |      Obs      Mean   Std. Dev.   Min       Max
-----+-----
drug |       15         2   .8451543         1         3
agegr |       15        55   7.319251         45         65
risktime |       15   4.133334   3.01067   .3581161   10.88906
died |       15   2.066667   2.374467         0         8
. save c:\dokumenter\proj1\stcollaps.cancer2.dta
```

These data are ready for a Poisson regression:

```
. xi: poisson died i.drug i.agegr if risktime>0 , exposure(risktime) irr
i.drug      _Idrug_1-3      (naturally coded; _Idrug_1 omitted)
i.agegr     _Iagegr_45-65  (naturally coded; _Iagegr_45 omitted)
Poisson regression                                Number of obs =       15
                                                LR chi2(6) =       29.20
                                                Prob > chi2 =       0.0001
Log likelihood = -19.558255                    Pseudo R2 =       0.4274
-----+-----
died |      IRR   Std. Err.      z    P>|z|   [95% Conf. Interval]
-----+-----
  _Idrug_2 | .2125451   .1044107   -3.15  0.002   .0811534   .5566669
  _Idrug_3 | .1434259   .068503   -4.07  0.000   .0562441   .3657449
  _Iagegr_50 | 2.427286   2.576403    0.84  0.403   .3031284   19.43637
  _Iagegr_55 | 3.892978   4.067407    1.30  0.193   .5022751   30.17327
  _Iagegr_60 | 6.20448    6.644274    1.70  0.088   .7606201   50.61077
  _Iagegr_65 | 11.05612   12.48911    2.13  0.033   1.208027   101.1879
  risktime | (exposure)
```

After running `poisson`, test goodness-of-fit by:

`poisgof`

14. Graphs

14.1. Introduction

The purpose of this section is to help you understand the fundamentals of Stata 8 graphs, and to enable you to create and modify them.

With Stata's dialogs you can easily define a graph. Once you made your choices, press [Submit] rather than [OK]; this gives the opportunity to modify your choices after having looked at the result.

This was the easy part. For the purpose of analysis you can do most things with the dialogs. Look at the illustrations in this section to get some ideas of the types and names of graphs. At www.ats.ucla.edu/stat/stata/Library/GraphExamples/default.htm you find a number of graph examples with the commands used.

The following more complex stuff illustrates how to make graphs ready for publication.

– o –

Stata can produce high-quality graphs, suited for publication. However, the first edition of the Graphics manual is complicated to use, to say the least; don't feel inferior if you get lost in the maze while looking up information. The on-line help works better, once you understand the general principles.

Use the Graphics manual to see examples of graphs, but skip the style and options specifications unless you are very dedicated.

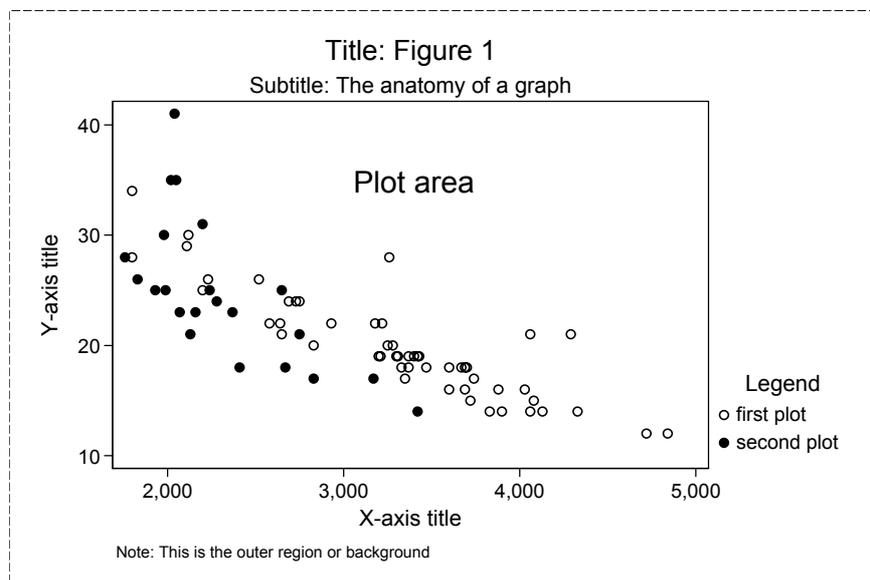
The style of the graphs presented here is different from the manual style; I attempted to hit a leaner mainstream style used in most scientific journals. The graphs are based upon my schemes **lean1** with a framed plot area and no gridlines and **lean2** with no frame but with gridlines. Find the schemes used by *findit lean schemes*. See more on this issue under Schemes, section 14.9.

You will meet some critical remarks in this section. However:

- Stata's graphics is a very versatile system; you can create almost whatever you want, except (fortunately) 3-D effects.
- The Stata people are very open to criticism and suggestions, and the users' input no doubt will give inspiration to improved design, accessibility and documentation.

14.2. The anatomy of graphs

Figure 1 shows the most important elements of a graph. The *graph area* is the entire figure, including everything, while the *plot area* is the central part, defined by the axes.



A *graph* consists of several elements: Title, legend, axes, and one or more *plots*, e.g. two scatterplots within the same plot area; Figure 1 includes two scatterplots.

Below is the command that generated Figure 1 (except the dashed outer frame). The elements of the command will be explained later.

```
sysuse auto.dta           // open auto.dta accompanying Stata
set scheme lean1
twoway (scatter mpg weight if foreign==0)           ///
      (scatter mpg weight if foreign==1)           ///
      ,                                             ///
      title("Title: Figure 1")                     ///
      subtitle("Subtitle: The anatomy of a graph")  ///
      ytitle("Y-axis title")  xtitle("X-axis title")  ///
      note("Note: This is the outer region or background")  ///
      legend(title("Legend") ,                    ///
              label(1 "first plot") label(2 "second plot"))  ///
      text(35 3400 "Plot area")
```

14.3. The anatomy of graph commands

The overall syntax of graph commands is:

```
graph-command (plot-command , plot-options) (plot-command , plot-options) , graph-options
```

This is the syntax style generated by the dialogs, and I will stick to it.

Unfortunately the Graphics manual frequently uses another, less transparent style:

```
graph-command plot-command , plot-options || plot-command , plot-options || , graph-options
```

Clue: Put a || where the standard syntax has a) parenthesis closing a plot specification.

When letting the dialog generate a simple scatterplot command, the result is like this:

```
twoway (scatter mpg weight)
```

twoway defines the graph type; *scatter* defines a plot in the graph. You could enter the same in the command window, but Stata also understands this short version:

```
scatter mpg weight
```

The variable list (e.g. *mpg weight*) in most graph commands may have one or more dependent (y-) variables, and one independent (x-) variable, which comes last.

Graph commands may have options; as in other Stata commands a comma precedes the options. *title()* is an option to the *twoway* graph command:

```
twoway (scatter mpg weight) , title("74 car makes")
```

Plot specifications may have options. *msymbol()* is an option to *scatter*; it is located within the parentheses delimiting the plot specification. *msymbol()* lets you select the marker symbol (a hollow circle) to be used in the scatterplot:

```
twoway (scatter mpg weight , msymbol(Oh))
```

Options may have sub-options. *size()* is a sub-option to the *title()* option; here it lets the title text size be 80% of the default size:

```
twoway (scatter mpg weight) , title("74 car makes" , size(*0.8))
```

Warning: Options don't tolerate a space between the option keyword and the parenthesis, like the following (□ denotes a blank character):

```
title□("74 car makes")
```

The error message may be confusing, e.g. 'Unmatched quotes' or 'Option not allowed'.

Advice: Graph commands tend to include a lot of nested parentheses, and you may make errors (I often do). In the Do-file editor, place the cursor after an opening parenthesis and enter [Ctrl]+B, to see the balancing closing parenthesis. In NoteTab you can use [Ctrl]+M (match) in the same way.

14.4. Axis options

Axis lengths

Unfortunately axis lengths cannot be controlled directly, only the entire graph size. By trial and error you may then obtain the desired axis lengths. To make a graph 3 by 4 inches:

```
twoway (scatter mpg weight) , ysize(3) xsize(4)
```

You can, however, determine the aspect ratio of the plot area (the y/x axis ratio) by the *aspect()* option. To obtain a square plot area:

```
twoway (scatter mpg weight) , ysize(3) xsize(4) aspect(1)
```

Ticks, labels and gridlines

Stata sets reasonable ticks and labels at the axes; you may also define them yourself. The following command sets a tick and a label for every 20 years at the x-axis, minor ticks divide each major interval in two. The y-axis has a log scale; tick marks are defined.

```
twoway (line incidence year) , ///
      xlabel(1900(20)2000) xmtick(##2) ///
      yscale(log) ylabel(1 2 5 10 20 50 100)
```

You may define maximum and minimum values at the axes:

```
... , yscale(log range(0.8 150))
```

If you use the `s2color`, `s2mono` or `lean2` scheme, the default is horizontal gridlines and no vertical gridlines. To drop horizontal and include vertical gridlines (hardly a good idea in this case):

```
... , xlabel( , grid) ylabel(1 2 5 10 20 50 100 , nogrid)
```

If you want to display decimal commas rather than periods, give the Stata command:

```
set dp comma
```

Plotregion margin

By default twoway graphs include a margin between the extreme plot values and the axes, to avoid symbols touching axes. If you want a zero margin – as in the `twoway line` plot, section 14.7 – include:

```
... , plotregion(margin(zero))
```

14.5. Placing graph elements

The placement of graph elements, e.g. the legend, is defined by location relative to the plot area (*ring* position) and a direction (*clock* position). The placement of elements in Figure 1 was determined by the scheme applied (see section 14.9); the placements were:

Element	Ring position <i>ring()</i>	Clock position <i>pos()</i>	Position can be modified
Plot area	0	...	No
Y-axis title	1	9	No
X-axis title	1	6	No
Subtitle	6	12	Yes
Title	7	12	Yes
Legend	3	4	Yes
Note	4	7	Yes

The `twoway line` plot, section 14.7, illustrates an alternative placement of the legend:

```
... , legend(label(1 "Males") label(2 "Females") ring(0) pos(8))
```

A text block is placed in the plot area by giving its y and x coordinates; `place(c)` (the default) means that the coordinates apply to the center of the text block; `place(se)` that they apply to the block's southeast corner. See example in Figure 1 and the *twoway line* plot, section 14.7:

```
... , text(90 69 "1999-2000")
```

14.6. Appearance of markers, lines, etc.

Options and their arguments for defining the appearance of lines, bars and markers:

Element	Color	Lines		Markers	
		Pattern	Width	Symbol	Size
Legend etc. fill outline	<code>color()</code> <code>fcolor()</code> <code>lcolor()</code>	<code>lpattern()</code>	<code>lwidth()</code>		
Bars, areas fill outline	<code>bcolor()</code> <code>bfcolor()</code> <code>blcolor()</code>	<code>blpattern()</code>	<code>blwidth()</code>		
Markers fill outline	<code>mcolor()</code> <code>mfcolor()</code> <code>mlcolor()</code>		<code>mlwidth()</code>	<code>msymbol()</code>	<code>msize()</code>
Connecting lines	<code>clcolor()</code>	<code>clpattern()</code>	<code>clwidth()</code>		
Arguments:	<code>none</code> Grayscale: <code>black</code> <code>gs0</code> (black) <code>..</code> <code>gs16</code> (white) <code>white</code>	<code>blank</code> <code>1</code> or <code>solid</code> <code>-</code> or <code>dash</code> <code>_</code> or <code>longdash</code> <code>shortdash</code> <code>dot</code> <code>dash_dot</code> Formula, e.g. <code>"-."</code> <code>"--.."</code>	<code>none</code> <code>*1.3</code> 130% of default	<code>i</code> invisible <code>O</code> circle <code>D</code> diamond <code>S</code> square <code>T</code> triangle <code>p</code> point <code>+</code> plus <code>X</code> cross Small: <code>o d s t x</code> Hollow: <code>Oh</code>	<code>*0.7</code> 70% of default

Marker symbols

Markers are defined by symbol: `msymbol()`, outline colour: `mlcolor()`, fill colour `mfcolor()` and size `msize()`. To define a hollow circle:

```
twoway (scatter mpg weight , msymbol(Oh))
```

A hollow circle (`Oh`) is transparent. Obtain a circle with a non-transparent white fill by:

```
twoway (scatter mpg weight , msymbol(O) mfcolor(white))
```

Connecting lines

The *twoway line* and *twoway connected* examples, section 14.7, use *connecting lines*; here the `clpattern()` and `clwidth()` options apply:

```
twoway (line m1840-k1999 age , clpattern( - 1 - 1 - 1 ))
```

The default connect-style is a straight line. Obtain a step-curve like a Kaplan-Meier plot by:

```
twoway (line cum time , connect(J))
```

Bars

Bar graphs (*twoway bar*) and range plots use *bar outlines*; here the *blpattern()* and *blwidth()* options apply. The colour of the *bar fill* is defined by the *bfcolor()* option:

```
... , bar(1, bfcolor(gs9)) bar(2, bfcolor(gs14))
```

14.7. Examples

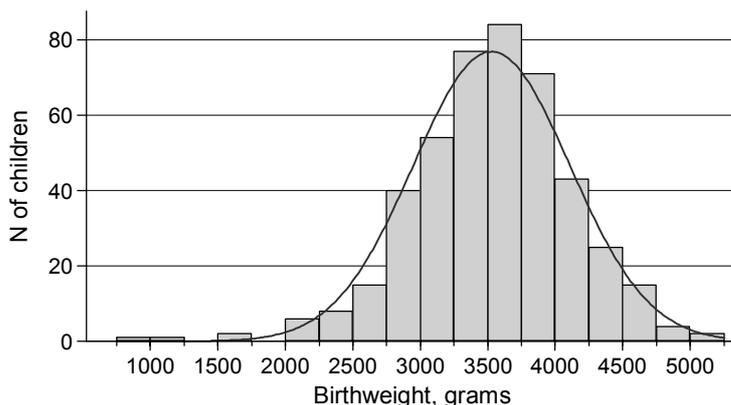
On the following pages you find illustrations of some important graph types, including the commands that generated the graphs. The appearance is different from the manual's graphs; it was determined by my *schemes lean1* and *lean2*, described in section 14.9.

For each graph you see the do-file that made it, including the data for the graph or a *use* command. I suggest letting do-files generating graphs always start with a *gph.* prefix, for easy identification.

In the illustrations I reduced the graph size by the *xsize()* and *ysize()* options. This, however, leads to too small text and symbols, and I enlarged them by the *scale()* option.

twoway graphs have continuous x- and y-axes. Many plot-types fit in twoway graphs; exceptions are graph bar, graph box and graph pie.

histogram

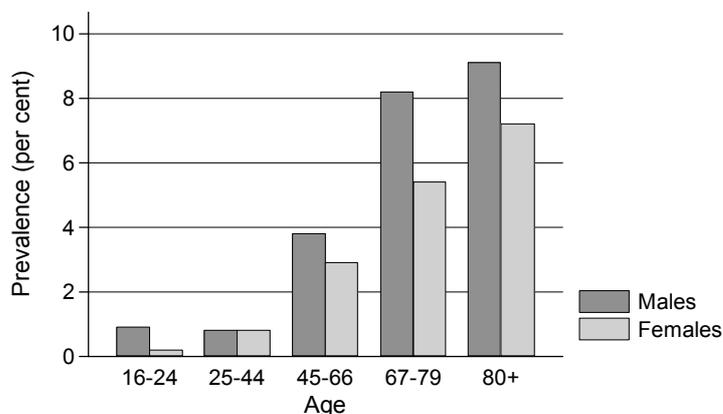


A histogram depicts the distribution of a continuous variable. The y-axis may reflect a count (frequency), a density or a percentage; the corresponding normal curve may be overlaid.

Histograms are documented in [R] *histogram* and in [G] *graph twoway histogram*.

```
// c:\dokumenter\...\gph.birthweight.do
use "C:\dokumenter\...\newborns.dta" , clear
set scheme lean2
histogram bweight ///
, ///
frequency ///
normal ///
start(750) width(250) ///
xlabel(1000(500)5000) ///
xmticks(##2) ///
xtitle("Birthweight, grams") ///
yttitle("N of children") ///
plotregion(margin(b=0)) ///
xsize(4) ysize(2.3) scale(1.4)
```

graph bar



```
// c:\dokumenter\...\gph.diabetes prevalence.do
clear
input str5 age m f
16-24 .9 .2
25-44 .8 .8
45-66 3.8 2.9
67-79 8.2 5.4
80+ 9.1 7.2
end
set scheme lean2
graph bar m f ///
, ///
over(age) ///
btitle("Age") ///
yttitle("Prevalence (per cent)") ///
legend( label(1 "Males") label(2 "Females") ) ///
xsize(4) ysize(2.3) scale(1.4)
```

For some reason the `xtitle()` option is not valid for bar graphs. To generate an x-axis title you may, however, use `btitle()` instead.

Bar fill colours are assigned automatically according to the scheme. This option would generate a very dark fill for females:

```
... , bar(2 , bfcOLOR(gs3))
```

In bar graphs the x-axis is categorical, the y-axis continuous. In the example variables `m` and `f` defined the heights of the bars, but actually `graph bar` used the default `mean` function, as if the command were (with one observation per bar the result is the same):

```
graph bar (mean) m f , over(age)
```

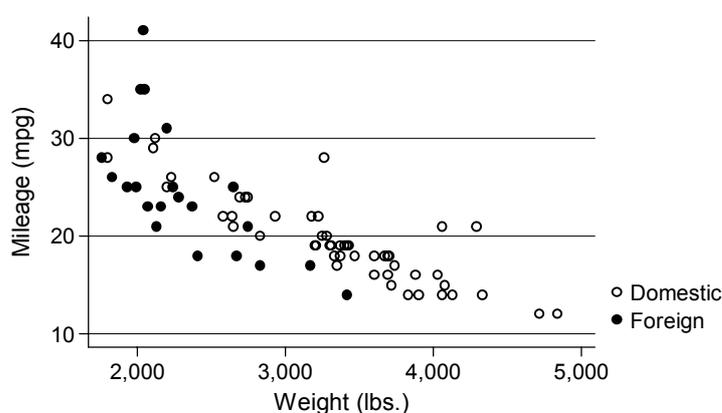
With the `auto.dta` data you could generate bars for the number of domestic and foreign cars by:

```
graph bar (count) mpg , over(foreign)
```

Actually what is counted is the number of non-missing values of `mpg`.

Bar graphs are documented in [G] `graph bar` and [G] `graph twoway bar`.

twoway scatter



```
// c:\dokumenter\...\gph.mpg_weight.do
clear
sysuse auto
set scheme lean2

twoway
  (scatter mpg weight if foreign==0, msymbol(Oh)) ///
  (scatter mpg weight if foreign==1, msymbol(O))   ///
  ,
  legend(label(1 "Domestic") label(2 "Foreign"))   ///
  xsize(4) ysize(2.3) scale(1.4)
```

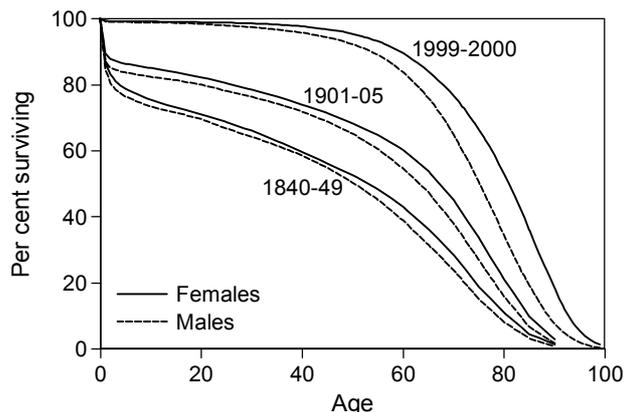
Twoway graphs have continuous x- and y-axes; scatter is the "mother" of twoway graphs.

A graph with one plot has no legend; this one with two plots has. The default legend texts often need to be replaced by short, distinct texts, like here. Since the `xtitle()` and `yttitle()` options were not specified, Stata used the variable labels as axis titles.

`msymbol` may be abbreviated to `ms`; I chose to avoid abbreviations for readability; it is much more important that commands are easy to read than easy to write. In this case the `msymbol()` options were not necessary since `(Oh)` and `(O)` are the default first symbols under the lean schemes.

The graph displays the same data as the initial Figure 1, this time using the scheme `lean2`: The plot area has horizontal grid-lines, but no frame.

twoway line



```
// c:\dokumenter\...\gph.DKsurvival.do
use c:\dokumenter\...\DKsurvival.dta , clear
sort age // Data must be sorted by the x-axis variable
list in 1/3, clean // List to show the data structure
      age  m1840  k1840  m1901  k1901  m1999  k1999
1.    0  100.00  100.00  100.00  100.00  100.00  100.00
2.    1   84.47   86.76   86.93   89.59   99.16   99.37
3.    2   80.58   83.11   85.22   87.89   99.08   99.32
set scheme lean1
twoway //
      (line m1840-k1999 age , cllpattern( - 1 - 1 - 1 )) //
      , //
      plotregion(margin(zero)) //
      xtitle("Age") //
      ytitle("Per cent surviving") //
      legend(label(1 "Males") label(2 "Females") order(2 1) //
      ring(0) pos(8)) //
      text(91 72 "1999-2000") //
      text(77 48 "1901-05") //
      text(49 40 "1840-49") //
      xsize(3.3) ysize(2.3) scale(1.4) //
```

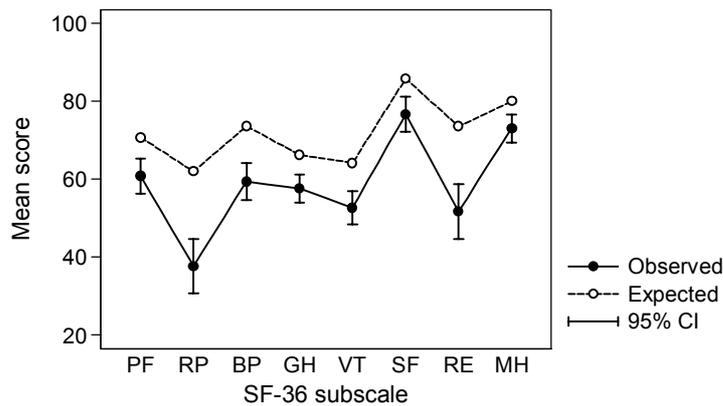
A line plot is a variation of scatterplot without markers, but with connecting lines. This graph includes six line plots, required by one plot-specification with six y- and one x-variable. The `cllpattern()` option defines the six connected-line patterns.

Make sure data are sorted according to the x-axis variable; otherwise the result is nonsense.

The example shows how to include text in a graph and how to position the legend within the plot area (see section 14.5 on placement of graph elements).

Twoway graphs by default include "empty" space between the axes and the extreme plot values. The graph option `plotregion(margin(zero))` lets the plot start right at the axes.

twoway connected; twoway rcap



```
// c:\dokumenter\...\gph.SF36a.do

clear
input scale n obs sd norm
1 139 60.81 27.35 70.77
2 139 37.65 42.06 62.01
...
8 139 73.06 21.54 79.99
end

generate se=sd/sqrt(n)
generate cil=obs+1.96*se
generate ci2=obs-1.96*se

label define scale 1 "PF" 2 "RP" 3 "BP" 4 "GH" 5 "VT" 6 "SF" 7 "RE" 8 "MH"
label values scale scale

set scheme lean1
twoway                                     ///
  (connected obs scale , msymbol(O) clpattern(1))          ///
  (connected norm scale , msymbol(O) mfcolor(white) clpattern(-))  ///
  (rcap cil ci2 scale)                                     ///
  ,                                                       ///
  ytitle("Mean score")                                     ///
  xtitle("SF-36 subscale")                                 ///
  xlabel(1(1)8 , valuelabel noticks)                       ///
  xscale(range(0.5 8.5))                                   ///
  legend(label(1 "Observed") label(2 "Expected") label(3 "95% CI")) ///
  xsize(4) ysize(2.3) scale(1.4)
```

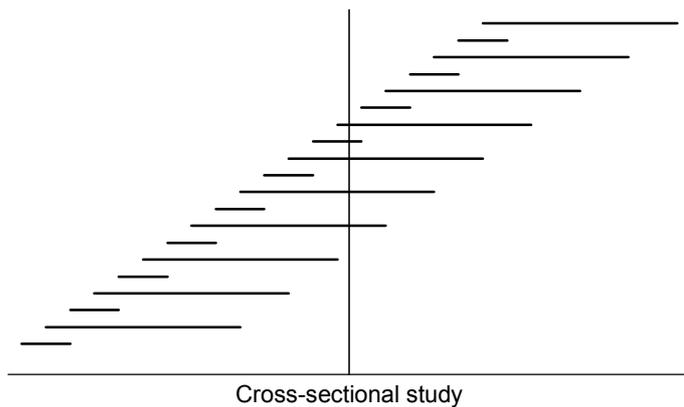
This graph includes three plots; two **connected** and one **rcap**. In twoway plots both axes are continuous, so you could not have a categorical variable (PF, RP etc.) at the x-axis.

Solution: use a numerical variable and use value labels to indicate the meaning. This graph style is frequently used to present SF-36 results, although connecting lines may be illogical when displaying eight qualitatively different scales.

xscale(range(0.5 8.5)) increased the distance between plot symbols and plot margin.

rcap does not calculate confidence intervals for you; it is up to you to provide two y- and one x-value for each confidence interval. **rspike** would have plotted intervals without caps.

twoway rspike



```
// c:\dokumenter\...\gph.length_bias.do
clear
set obs 20
gen x= n
gen y1=x
gen y2=y1+2
replace y2=y1+8 if mod(x,2)==0
set scheme lean2
twoway (rspike y1 y2 x , horizontal blwidth(*1.5))    ///
      ,                                              ///
      yscale(off)  ylabel(, nogrid)  ytitle("")      ///
      xlabel(none)  xtitle("Cross-sectional study")  ///
      xline(14.5)                                     ///
      xsize(3.7) ysize(2.3) scale(1.4)
```

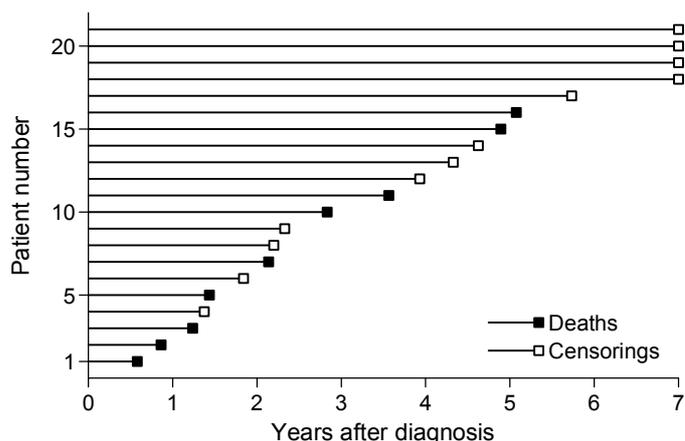
The purpose of this graph is to illustrate length bias: a cross-sectional (prevalence) study may mislead you. Cases with short duration (due to successful treatment or high case fatality) are underrepresented in a cross-sectional sample.

rspike is in the **twoway r*** family: range plots, like **rcap** shown before; this time it is horizontal.

In range plots and droplines (next page) the lines technically are bar outlines, and options are **blcolor()**, **blpattern()** etc.; hence the **blwidth(*1.5)** to make the spikes wider than the default.

It is easy to create one or more reference lines; use **xline()** and **yline()**.

twoway dropline



```
// c:\dokumenter\...\gph.obstime.do
use "c:\dokumenter\...\cohort1.dta" , clear
list patient time died in 1/5 , clean

  patient  time  died
1.      1   0.578    1
2.      2   0.867    1
3.      3   1.235    1
4.      4   1.374    0
5.      5   1.437    1

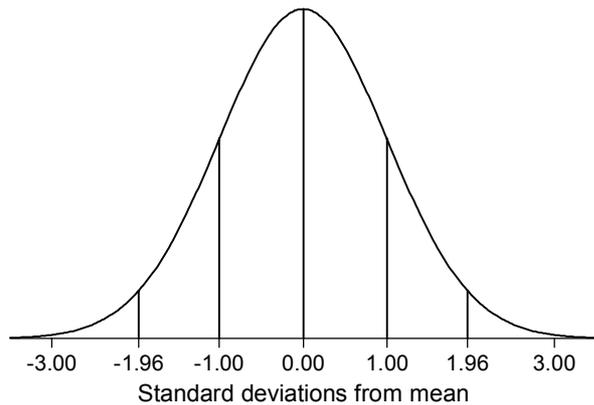
set scheme lean2

twoway                                     ///
  (dropline time patient if died==1, horizontal msymbol(S))  ///
  (dropline time patient if died==0, horizontal             ///
    msymbol(S) mfcolor(white))                               ///
  ,                                                                 ///
  plotregion(margin(zero))                                     ///
  ytitle("Patient number")                                    ///
  yscale(range(0 22))                                        ///
  ylabel(1 5 10 15 20 , nogrid)                             ///
  xtitle("Years after diagnosis")                           ///
  xlabel(0(1)7)                                             ///
  legend(label(1 "Deaths") label(2 "Censorings") ring(0))  ///
  xsize(3.7) ysize(2.5) scale(1.3)
```

In a dropline plot a line 'drops' from a marker perpendicularly to the x- or y-axis. Droplines technically are bar outlines, like range plots, and their appearance is controlled by `blpattern()`, `blcolor()` and `blwidth()`.

The marker for censorings is a square with white fill, not a hollow square, to avoid the dropline to be visible within the marker.

twoway function



```
// c:\dokumenter\...\gph.normal.do
set scheme lean2
twoway                                     ///
  (function y=normden(x) , range(-3.5 3.5)  ///
    droplines(-1.96 -1 0 1 1.96))          ///
  ,                                         ///
  plotregion(margin(zero))                 ///
  yscale(off) ylabel(, nogrid)             ///
  xlabel(-3 -1.96 -1 0 1 1.96 3 , format(%4.2f))  ///
  xtitle("Standard deviations from mean")      ///
  xsize(3) ysize(2.3) scale(1.4)
```

twoway function gives you the opportunity to visualize any mathematical function. The result has no relation to the actual data in memory. The **range()** option is necessary; it defines the x-axis range.

Other examples:

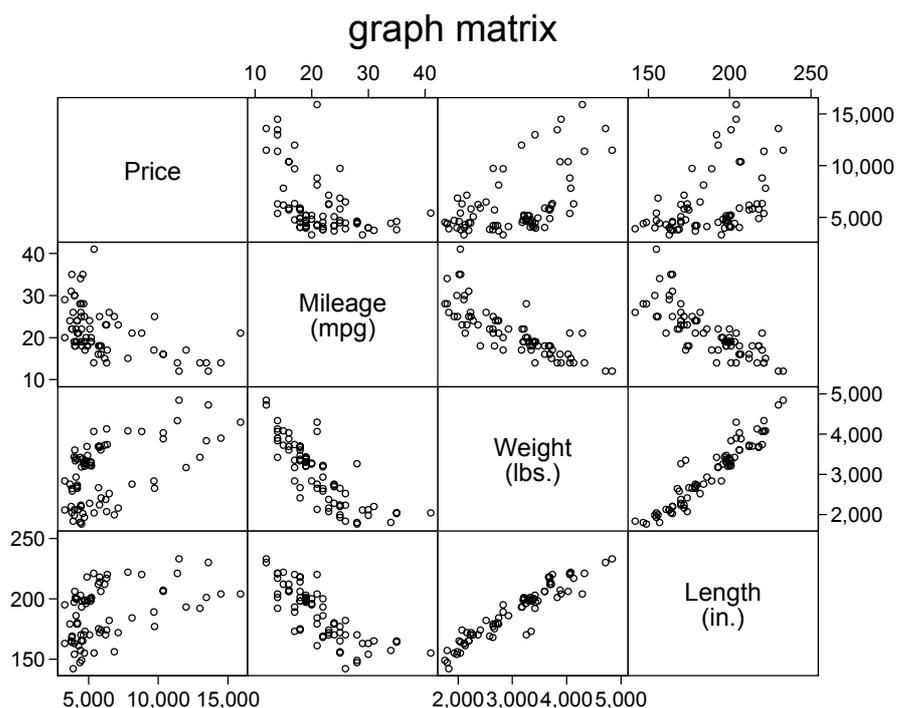
An identity line, to be overlaid in a scatterplot comparing two measurements:

```
twoway                                     ///
  (scatter sbp2 sbp1)                       ///
  (function y=x , range(sbp1))
```

A parabola:

```
twoway (function y=x^2 , range(-2 2))
```

graph matrix



```
// c:\dokumenter\...\gph.matrix.do
sysuse auto , clear
set scheme lean1
graph matrix price mpg weight length    ///
'                                       ///
title(graph matrix)                   ///
mlwidth(*0.7)                          ///
xsize(5) ysize(4)
```

Matrix scatterplots are useful for analysis, but are infrequently used for publication.

The `lean1` and `lean2` schemes by default use a small hollow circle as marker in matrix scatterplots. Here `mlwidth(*0.7)` made the marker outline thinner.

The upper right cells are redundant rotated images of the lower left cells; omit them by:

```
graph matrix price mpg weight length , half
```

14.8. Saving, displaying and printing graphs

Save a graph

The active graph can be saved as a `.gph` file:

```
graph save "c:\dokumenter\...\DKsurvival.gph" [, asis replace]
```

The ***asis*** option saves a 'frozen' graph, it is displayed as is, regardless of scheme settings. Without this option you save a 'live' graph: you may display it again, maybe using a different scheme or modifying its size. The manual states that you may edit it, but that is not the case.

My firm recommendation:

Rarely save graph files; always save a do-file for each graph with a name that tells what it does, e.g. **`gph.DKsurvival.do`**. Let all graph-defining do-files start with a **`gph.`** prefix, for easy identification. The do-file documents what you did, you can edit it to modify the graph, and you can modify the do-file to create another graph. Remember to include the data or a **`use`** command reading the data used. This advice also applies when you initially defined the graph command with a graph dialog.

Open a saved graph

A saved graph may be displayed by:

```
graph use "c:\dokumenter\...\DKsurvival.gph"
```

Display and print a graph

Re-display the current graph by:

```
graph display [, scale(1.2) ysize(3) xsize(5) scheme(lean2)]
```

The **`scale()`** option is useful to increase marker and text size, e.g. for a slide show. **`xsize()`** and **`ysize()`** modify the size of the graph area (arguments in inches), and **`scheme()`** lets you display a graph under a different scheme – but that sometimes fails.

Copying and printing 'smooth' coloured or gray areas sometimes give poor results, and a raster-pattern is preferable. This is a printer, not a Stata issue; in this respect modern printers are worse than older. At my old HP LaserJet 1100 printer the LaserJet III printing mode translates gray areas to raster-patterns, copying and printing nicely. You may need to experiment.

If you in the future don't want Stata's logo being printed on each graph:

```
graph set print logo off
```

Copy a graph to a document

Copy-and-paste a graph to another document using the standard [Ctrl]+C and [Ctrl]+V procedure. The graph will be transferred as a metafile; there are two types, which you may select via the Graph Preferences:

Prefs ► Graph Preferences ► Clipboard

Select Enhanced Metafile (EMF) or Windows Metafile (WMF); which one works best depends on your system and printer; take a critical look at the results.

Submitting graphs to journals etc.

The requirements of journals vary, but the best results are probably obtained by Windows metafiles (.wmf, .emf) and Encapsulated PostScript (.eps). When saving a graph, select that format. You may open an eps-file by Adobe Acrobat Reader to ensure that the result is satisfactory. Formats like Portable Network Graphics (.png) and TIFF (.tif) give unsatisfactory results.

For updated information on exporting graphs, see:

```
whelp graph_export
```

14.9. Schemes: Default appearance

A *scheme* is a collection of default styles; the Graphics Manual uses a scheme close to **s2mono**, while the Base Reference Manual uses **s1mono**. **s2mono** is itself a modification of the **s2color** scheme.

This note uses the schemes **lean1** and **lean2**; they are modifications to **s1mono** and **s2mono**. Most scientific journals use a lean graph style – or at least require that graphs submitted are lean and black-and-white. If you are interested, download and install both schemes (use the command **findit lean schemes**).³

The difference between the two is that **lean1** has a framed plot area, but no gridlines, while the opposite is the case for **lean2**. Section 14.7 includes examples using both schemes.

To select a scheme:

```
set scheme lean2
```

To make Stata remember **lean1** as your default scheme:

```
set scheme lean1 , permanently
```

To create a scheme with your own preferences use Stata's do-file editor or another text editor to enter the options you want in your own personal scheme (e.g. **myscheme**) and save it as **c:\ado\personal\myscheme.scheme**. Scheme terminology differs from graph command terminology; documentation is forthcoming.³

³ Juul S. Lean mainstream schemes for Stata 8 graphics. The Stata Journal 2003; 3: 295-301.

15. Miscellaneous

15.1. Memory considerations

[U] 7

In Intercooled Stata a data set can have a maximum of 2,000 variables (Stata/SE: 32,000). Stata keeps the entire data set in memory, and the number of observations is limited by the memory allocated. The memory must be allocated before you open (*use*) a data set.

As described in section 1.2 the initial default memory is defined in the Stata icon. To change this to 15 MB, right-click the icon, select Properties, and change the path field text to: `c:\stata\wstata.exe /m15`.

If the memory allocated is insufficient you get the message:

```
no room to add more observations
```

You may increase the current memory to 25 MB by:

```
clear           // You can't change memory size with data in memory  
set memory 25m
```

You can see the amount of used and free memory by:

```
memory
```

compress

[R] **compress**

To reduce the physical size of your data set – and the memory requirements – Stata can determine the fewest bytes needed for each variable (see section 6.2), and you can safely:

```
compress
```

and save the data again (*save... , replace*). This may reduce the memory need by 80%.

Handling huge data sets

If you are regularly handling huge data sets you may consider:

- to use **compress** to reduce memory requirements
- to increase your computer's RAM (not expensive)

For instance with 512 Mb RAM you could **set memory 400m** for Stata, and this would fit very large data sets, e.g. a million observations with 100 variables. Stata/SE can handle up to 32,000 variables, but memory restrictions otherwise don't differ from Intercooled Stata. SAS or SPSS might be alternatives.

You might not be able to create the entire Stata data set because of its hugeness. Try to read one part of the data, compress and save, read the next part of the data, compress and save, etc., and finally combine (**append**) the partial data sets (see section 10.6):

```
// c:\dokumenter\p1\gen.aa.do  
infix id 1 type 2 sold 4-7 using c:\dokumenter\p1\aa.txt in 1/10000  
compress  
save c:\tmp\aa1.dta  
infix id 1 type 2 sold 4-7 using c:\dokumenter\p1\aa.txt in 10001/20000  
compress  
append using c:\tmp\aa1.dta  
save c:\dokumenter\p1\aa.dta
```

15.2. String variables

[U] 15.4; [U] 26

Throughout this text I have demonstrated the use of numeric variables, but Stata also handles string (text) variables. It is almost always easier and more flexible to use numeric variables, but sometimes you might need string variables. String values must be enclosed in quotes:

```
replace ph=45 if nation == "Danish"
```

"Danish", "danish", and "DANISH" are different string values.

A string can include any character, also numbers; however number strings are not interpreted by their numeric value, just as a sequence of characters. Strings are sorted in dictionary sequence, however all uppercase letters come before lowercase; numbers come before letters. This principle is also applies to relations: "12" < "2" < "A" < "AA" < "Z" < "a".

String formats

[U] 15.5.5

`%10s` displays a 10 character string, right-justified; `%-10s` displays it left-justified.

Reading string variables into Stata

In the commands reading ASCII data (see section 8) the default data type is numeric. String variables should be defined in the input command. `str5` means a 5 character text string:

```
infix id 1-4 str5 icd10 5-9 using c:\dokumenter\p1\a.txt
```

Generating new string variables

The *first* time a string variable is defined it must be declared by its length (`str10`):

```
generate str10 nation = "Danish" if ph==45  
replace nation = "Swedish" if ph==46
```

Conversion between string and numeric variables

Number strings to numbers

If a CPR number is recorded in `cprstr` (type string), no calculations can be performed.

Conversion to a numeric variable `cprnum` can be obtained by:

```
generate double cprnum = real(cprstr)  
format cprnum %10.0f
```

`cprnum` is a 10 digit number and must be declared `double` for sufficient precision (see section 6.2). Another option is `destring` (it automatically declares `cprnum double`):

```
destring cprstr , generate(cprnum)
```

Non-number strings to numbers

If a string variable `sex` is coded as eg. "M" and "F", convert to a numeric variable `gender` (with the original string codes as value labels) by:

```
encode sex , generate(gender)
```

[R] `encode`

Display the meaning of the numeric codes by:

```
label list gender
```

Numbers to strings

You want the numeric variable `cprnum` converted to a string variable `cprstr`:

```
generate str10 cprstr = string(cprnum , "%10.0f")
```

String manipulations

[U] 16.3.5; [GSW] 12

Strings can be combined by `+`:

```
generate str5 svar5 = svar3 + svar2
```

You may isolate part of a string variable by the `substr` function. The arguments are: source string, start position, length. In the following `a3` will be characters 2 to 4 of `strvar`:

```
generate str3 a3 = substr(strvar,2,3)
```

You may substitute characters within a string. In an example above the string variable `cprstr` was created from the numeric variable `cprnum`. However, for persons with a leading `0` in the CPR number the string will start with a blank, not a `0`. This can be remedied by:

```
replace cprstr = subinstr(cprstr," ","0",1)
```

The `upper` function converts lower case to upper case characters; the `lower` function does the opposite. Imagine that ICD-10 codes had been entered inconsistently, the same code sometimes as `E10.1`, sometimes as `e10.1`. These are different strings, and you want them to be the same (`E10.1`):

```
replace icd10 = upper(icd10)
```

Handling complex strings, eg. ICD-10 codes

[U] 26.4

In the ICD-10 classification of diseases all codes are a combination of letters and numbers (e.g. `E10.1` for insulin demanding diabetes with ketoacidosis). This is probably convenient for the person coding diagnoses (an extremely important consideration), but for the data handling it is quite inconvenient. I suggest to split a 5 character ICD-10 string variable (`scode`) into a one character string variable (`scode1`) and a four digit numeric variable (`ncode4`):

```
generate str1 scode1 = substr(scode,1,1)
generate ncode4 = real(substr(scode,2,4))
format ncode4 %4.1f
```

What did we obtain? Two variables: the string variable `scode1` with 26 values (A to Z) and a numeric variable `ncode4` (0.0-99.9). Now identify diabetes (E10.0-E14.9) by:

```
generate diab=0
replace diab=1 if scode1=="E" & ncode4>=10 & ncode4<15
```

If you received ASCII data, the same result could have been obtained by letting eg. the `infix` command read the same data twice as different types:

```
infix id 1-4 str5 scode 5-9 str1 scode1 5 ncode2 6-9 ///
using c:\dokumenter\...\list1.txt
```

The diabetics could also have been identified this way:

```
replace diab=1 if scode>="E10" & scode<"E15"
```

15.3. Dates. Danish CPR numbers

Date variables

[U] 15.5.2

Dates are numeric variables; the internal value is the number of days since 1 Jan 1960; dates before that are negative.

Date formats

[U] 15.5.3; [U] 27.2.3

Format specifications start with `%d`. Specifying `%d` only is equivalent to `%dD1CY` displaying a date as `28sep2000`. To display this date as `28.09.2000` specify the format `%dD.N.CY` (**D** for day, **N** for numeric month, **C** for century, **Y** for two-digit year). Example:

```
format bdate %dD.N.CY
```

Reading date variables

[U] 27.2.1

A date may be input as three variables: day, month, year and next transformed to a date variable:

```
infix bd 1-2 bm 3-4 by 5-8 using c:\dokumenter\p1\datefile.txt
generate bdate = mdy(bm,bd,by) // sequence must be m d y
format bdate %dD.N.CY
```

Another option is to enter the date as a string (`sdate`) and translate it to a date variable:

```
infix str10 sdate 1-10 using c:\dokumenter\p1\datefile.txt
generate bdate = date(sdate,"dmy") // "dmy" defines sequence
format bdate %dD.N.CY
```

The `date` function 'understands' most input formats: `17jan2001`, `17/1/2001`, `17.1.2001`, `17 01 2001`, but not `17012001`. However `todate`, a user-written function, handles this situation; find and download it by: `findit todate`.

In general: enter 4-digit years to avoid ambiguity on the century.

Calculations with dates

[U] 27.2.4

To express the length of a time interval in years you must:

```
generate opage = (opdate-bdate)/365.25
```

You may extract day, month and year from a date variable (`bdate`):

```
generate bday = day(bdate)
gen bmonth = month(bdate)
gen byear = year(bdate)
```

On Danish CPR numbers: extracting key information

Sometimes you get date information as a CPR number in an ASCII file. You can read the the CPR number as one string variable and the date information from the same columns:

```
infix str10 cprstr 1-10 bday 1-2 bmon 3-4 byear 5-6 ///
control 7-10 using c:\dokumenter\p1\datefile.txt
generate bdate = mdy(bmon,bday,byear)
```

Or you can extract key information from a CPR number read as one string variable (**cprstr**):

```
generate bday = real(substr(cprstr,1,2))
gen bmon = real(substr(cprstr,3,2))
gen byear = real(substr(cprstr,5,2))
gen control = real(substr(cprstr,7,4))
gen pos7 = real(substr(cprstr,7,1))           // to find century
```

Before creating **bdate** you must decide the century of birth; see the rules below:

```
generate century = 19
replace century = 20 if pos7 >= 4 & byear <= 36
replace century = 18 if pos7 >= 5 & pos7 <= 8 & byear >= 58
replace byear = 100*century + byear
generate bdate = mdy(bmon,bday,byear)
```

The information on sex can be extracted from **control**; the **mod** function calculates the remainder after division by 2 (male=1, female=0):

```
generate sex = mod(control,2)
```

Century information in Danish CPR numbers

The 7th digit (the first control digit) informs on the century of birth:

Pos. 7	Pos. 5-6 (year of birth)		
	00-36	37-57	58-99
0-3	19xx	19xx	19xx
4, 9	20xx	19xx	19xx
5-8	20xx	not used	18xx

Source: www.cpr.dk

Validation of Danish CPR numbers

To do the modulus 11 test for Danish CPR numbers first multiply the digits by 4, 3, 2, 7, 6, 5, 4, 3, 2, 1; next sum these products; finally check whether the sum can be divided by 11.

Assume that the CPR numbers were split into 10 one-digit numbers **c1-c10**. Explanation of **for**: see section 7.

```
generate test=0
for C in varlist c1-c10 \ X in numlist 4/2 7/1 :   ///
  replace test=test+C*X
replace test=mod(test,11)           // Remainder after division by 11
list id cpr test if test !=0
```

To extract **c1-c10** from the string **cprstr**:

```
for C in newlist c1-c10 \ X in numlist 1/10 :     ///
  gen C=real(substr(cprstr,X,1))
```

To extract **c1-c10** already when reading data:

```
infix str10 cprstr 1-10 c1-c10 1-10 using c:\...\dfile.txt
```

I developed an ado-file (cprcheck.ado) that extracts birth date and sex information and checks the validity of a CPR number. Find and download it by:

```
findit cprcheck
```

15.4. Random samples, simulations

Random number functions

[R] Functions

Stata can create 'pseudo-random' numbers:

```
gen y=uniform()           Uniformly distributed in the interval 0-1
gen y=invnorm(uniform())  Normal distribution, mean=0, SD=1
gen y=10+2*invnorm(uniform()) Normal distribution, mean=10, SD=2
```

If you run the same command twice it will yield *different* numbers. If you need to reproduce the same series of 'random' numbers, initialize the seed (a large integer used for the initial calculations):

```
set seed 654321
```

Random samples and randomization

You may use `sample` to select a random sample of your data set:

```
sample 10                Selects an approximately 10 percent random sample.
sample 53 , count        Selects exactly 53 observations at random.
```

You may assign observations randomly to two treatments:

```
generate y=uniform()
generate treat=1
replace treat=2 if y>0.5
```

And you may sort your observations in random sequence:

```
generate y=uniform()
sort y
```

Generating artificial data sets

You may use `set obs` to create empty observations. The following sequence defines a file with 10,000 observations, used to study the behaviour of the difference (`dif`) between two measurements (`x1`, `x2`), given information about components of variance (`sdwithin`, `sdbetw`).

```
set obs 10000
generate sdbetw = 20
generate sdwithin = 10
generate sdtotal = sqrt(sdbetw^2 + sdwithin^2)
generate x0 = 50 + sdbetw*invnorm(uniform())
generate x1 = x0 + sdwithin*invnorm(uniform())
generate x2 = x0 + sdwithin*invnorm(uniform())
generate dif = x2 - x1
summarize
```

See another example in section 14.7 (the `twoway rspike` graph)

Advanced simulations

[R] `simulate`

With `simulate` you may set up quite complex Monte Carlo simulations.

15.5. Immediate commands

[U] 22

An 'immediate' command requires tabular or aggregated input; data in memory are not affected. The immediate commands *tabi*, *cci*, *csi*, *iri* and *ttesti* are mentioned in section 11, and *sampsi* (sample size estimation) in section 15.6.

Confidence intervals

[R] *ci*

The general command *ci* and the 'immediate command' *cii* calculate confidence intervals. I here show the use of *cii*:

Normal distribution: *cii* 372 37.58 16.51
 N mean SD

Binomial distribution: *cii* 153 40
 N events

Poisson distribution: *cii* 247.1 40 , *poisson*
 time events

Stata as a pocket calculator

[R] *display*

The *display* command gives the opportunity to perform calculations not affecting the data in memory (*_pi* is a Stata constant):

```
. display 2*_pi*7  
43.982297
```

You may include an explanatory text:

```
. display "The circumference of a circle with radius 7 is " 2*_pi*7  
The circumference of a circle with radius 7 is 43.982297
```

15.6. Sample size and study power

sampsi

[R] **sampsi**

Sample size and study power estimation are pre-study activities: What are the consequences of different decisions and assumptions for sample size and study power?

You must make these *decisions*:

- The desired significance level (α). Default: 0.05.
- The minimum relevant contrast – expressed as study group means or proportions.
- Sample size estimation: The desired power ($1-\beta$). Default: 0.90.
- Power estimation: Sample sizes.

And with comparison of means you must make an *assumption*:

- The assumed standard deviation in each sample.

Here are short examples for the four main scenarios:

Comparison of:	Sample size estimation	Power estimation
Proportions	<code>sampsi 0.4 0.5</code>	<code>sampsi 0.4 0.5 , n(60)</code>
Means	<code>sampsi 50 60 , sd(8)</code>	<code>sampsi 50 60 , sd(8) n(60)</code>

Further options are available:

Situation	Option	Sample size		Power	
		prop.	mean	prop.	mean
Significance level; default: 0.05	<code>alpha(0.01)</code>	+	+	+	+
Power; default: 0.90	<code>power(0.95)</code>	+	+		
Unequal sample sizes; ratio=n2/n1	<code>ratio(2)</code>	+	+		
Unequal sample sizes	<code>n1(40) n2(80)</code>			+	+
Unequal SDs	<code>sd1(6) sd2(9)</code>		+		+

Example: Sample size estimation for comparison of means, unequal SDs and sample sizes:

```
sampsi 50 60 , sd1(14) sd2(10) ratio(2)
```

sampsi also handles trials with repeated measurements, see [R] **sampsi**.

15.7. ado-files

[U] 20-21, [P] (Programming manual)

An ado-file is a program. Most users will never write programs themselves, but just use existing programs. If you are a freak, read more in the User's Guide ([U] 20-21) and the programming manual [P]. Save user-written programs in `c:\ado\personal`. To see the locations of all ado-files issue the command `sysdir`.

The simplest form of an .ado file is a single command or a do-file with a leading `program define` command and a terminating `end` command. There must be a new line after the terminating `end`.

Here is an example to demonstrate that creating your own commands is not that impossible.

`datetime` displays date and time

```
program define datetime
// c:\ado\personal\datetime.ado.  Displays date and time.
  display "      $S_DATE $S_TIME "
end
```

Just enter `datetime` in the command window, and the date and time is displayed:

```
. datetime
      9 Feb 2003  16:54:15
```

Two ado-files useful for the interaction between Stata and NoteTab are shown in appendix 3.

`foreach` and `forvalues`

[P] `foreach`; `forvalues`

These commands are documented in the programming manual, and in the online help (`whelp foreach`). Also see the FAQ www.stata.com/support/faqs/data/foreach.html. They enable you to repeat a command for a number of variables or values. The commands can be used not only in ado-files, but also in do-files and even interactively. Look at the sequence in section 11.1:

```
foreach Q of varlist q1-q10 {
  tabulate `Q' sex
}
```

`Q` is a *local macro* (see [U] 21.3); `foreach` defines it as a stand-in for the variables `q1` to `q10`, and the sequence generates ten `tabulate` commands. The local macro is in effect only within the braces `{ }` which must be placed as shown.

When referring to the local macro `Q` it must be enclosed in single quotes: ``Q'`. In the manuals single quotes are shown differently; but the opening quote is ``` (accent grave), and the ending quote the simple `'`.

15.8. Exchange of data with other programs

Beware: Translation between programs may go wrong, and you should check carefully eg. by comparing the output from SPSS' **DESCRIPTIVES** and Stata's **summarize**. Especially compare the number of valid values for each variable and take care with missing values and date variables.

StatTransfer

[U] 24.4

StatTransfer is a reasonably priced program (purchase: see Appendix 1) that translates between a number of statistical packages, including Stata. Variable names, and variable and value labels are transferred too. StatTransfer 7 understands Stata 8 files, but StatTransfer 6 does not. To create a Stata 7 data set for conversion by StatTransfer 6:

```
saveold c:\dokumenter\proj1\alfa.dta
```

Transferring data to Excel and other spreadsheets

[R] **outsheet**

Many statistical packages read Excel data. To create a tab-separated file (see section 8) readable by Excel:

```
outsheet [varlist] using c:\dokumenter\proj1\alfa.txt , nolabel
```

In Excel open the file as a text file and follow the instructions. Variable names, but no labels are transferred.

[R] **outfile**

If you want the data written to a comma-separated ASCII file the command is:

```
outfile [varlist] using c:\...\alfa.txt , nolabel comma
```

Reading Excel data

[R] **insheet**

Many packages can create Excel data, and probably all can create text files similar to those created by Stata's **outsheet** command. From Excel save the file as a tab-separated text file (see section 8). Stata reads it by:

```
insheet using c:\dokumenter\p1\alfa.txt , tab
```

15.9. For old SPSS users

SPSS and Stata have similarities and differences. Among the differences are:

- While SPSS can define any numeric code as a missing value, Stata's user-defined missing values are special codes; see section 6.3.
- Stata's missing values are high-end numbers. This may complicate conditions; see section 6.3.
- While SPSS executes all transformation commands up to a procedure command one case at a time, Stata performs each command for the entire data set before proceeding to the next command. This leads to different behaviour when combining selections (**keep if**; **drop if**) with observation numbers (**[_n]**).

Frequently used SPSS commands and the similar Stata commands

SPSS command	Similar Stata command
Data in and out	
DATA LIST	<code>infile; infix; insheet</code>
GET FILE	<code>use</code>
SAVE OUTFILE	<code>save</code>
Documentation commands etc.	
VARIABLE LABELS	<code>label variable</code>
VALUE LABELS	<code>label define</code> followed by <code>label values</code>
FORMAT sex (F1.0) .	<code>format sex %1.0f</code>
MISSING VALUES	Missing values are special; see section 6.3.
COMMENT; *	<code>*</code> or <code>//</code>
DOCUMENT	<code>note</code>
DISPLAY DICTIONARY	<code>describe; codebook</code>
Calculations	
COMPUTE	<code>generate; replace; egen</code>
IF (sex=1) y=2.	<code>generate y=2 if sex==1</code>
RECODE a (5 thru 9=5) INTO agr.	<code>recode a (5/9=5) , generate(agr)</code>
DO REPEAT ... END REPEAT	<code>for; foreach; forvalues</code>
SELECT IF	<code>keep if; drop if</code>
TEMPORARY. SELECT IF (sex=1) .	<code>command if sex==1</code>
SAMPLE 0.1.	<code>sample 10</code>
SPLIT FILE	<code>by...:</code>
WEIGHT	Weights can be included in most commands; see section 7.
Analysis	
DESCRIPTIVES	<code>summarize</code>
FREQUENCIES	<code>tabulate; tab1</code>
CROSSTABS	<code>tabulate; tab2</code>
MEANS bmi BY agegrp.	<code>oneway bmi agegrp , tabulate</code>
T-TEST	<code>ttest</code>
LIST	<code>list</code>
WRITE	<code>outfile; outsheet</code>
Advanced	
SORT CASES BY	<code>sort</code>
AGGREGATE	<code>collapse</code>
ADD FILES	<code>append</code>
MATCH FILES	<code>merge</code>

16. Do-file examples

Here follow short examples of do-files doing typical things. Find more examples in *Take good care of your data*. All major work should be done with do-files rather than by entering single commands because:

1. The do-file serves as documentation for what you did.
2. If you discover an error, you can easily correct the do-file and re-run it.
3. You are certain that commands are executed in the sequence intended.

Example 1 generates the first Stata version of the data, and example 2 generates a modified version. I call both do-files *vital* in the sense that they document modifications to the data. Such do-files are part of the documentation and they should be stored safely. Safe storage also means safe retrieval, and they should have names telling what they do. My principle is this:

In example 1 `gen.wine.do` generates `wine.dta`. In example 2 `gen.visit12a.do` generates `visit12a.dta`. This is different from example 3 where no new data are generated, only output. This do-file is not vital in the same sense as example 1 and 2, and it should *not* have the `gen.` prefix (the Never Cry Wolf principle).

As mentioned in section 3 I prefer to use NoteTab rather than the do-file editor for creating do-files. The last command in a do-file must be terminated by a carriage return; otherwise Stata cannot 'see' the command.

Example 1. gen.wine.do generates Stata data set wine.dta from ASCII file

```
// gen.wine.do creates wine.dta      13.5.2001
infix id 1-3 type 4 price 5-10 rating 11      ///  
    using c:\dokumenter\wines\wine.txt  
  
// Add variable labels  
label variable id "Identification number"  
lab var type "Type of wine"  
lab var price "Price per 75 cl bottle"  
lab var rating "Quality rating"  
  
// Add value labels  
label define type 1 "red" 2 "white" 3 "rosé" 4 "undetermined"  
label values type type  
lab def rating 1 "poor" 2 "acceptable" 3 "good" 4 "excellent"  
lab val rating rating  
  
// Add data set label  
label data "wine.dta created from wine.txt, 13.5.2001"  
save c:\dokumenter\wines\wine.dta
```

Example 2. gen.visit12a.do generates visit12a.dta from visit12.dta

```
// gen.visit12a.do generates visit12a.dta with new variables.  
use c:\dokumenter\proj1\visit12.dta, clear  
  
// Calculate hrqol: quality of life score.  
egen hrqol=rsum(q1-q10)  
label variable hrqol "Quality of life score"  
  
label data "Visit12a.dta created by gen.visit12a.do, 02.01.2001"  
save c:\dokumenter\proj1\visit12a.dta
```

Example 3. Analyse Stata data

```
// winedes.do Descriptive analysis of the wine data 14.5.2001

use c:\dokumenter\wines\wine.dta
describe
codebook
summarize
tab1 type rating
tabulate type rating , chi2 exact
oneway price rating , tabulate
```

Note that in example 1 and 2 the structure was:

1. Read data (*infix*, *use*)
2. Calculation and documentation commands
3. Save data (*save*)

In these examples a new generation of the data was created; changes were documented with a do-file with a **gen.** prefix.

In example 3 the structure was:

1. Read data (*use*)
2. Analysis commands

No new data generation was created, and the **gen.** prefix does not belong to such do-files.

Example 4. Elaborate profile.do

```
// c:\ado\personal\profile.do executes automatically when opening Stata.
// Write session start time in time.txt.
set obs 1
gen time="***** Session started: `c(current_date)' `c(current_time)'"
outfile time using c:\tmp\time.txt, noquote replace
clear

// Copy session start time to the cmdlog (cmdlog.txt) and open it.
// ! means that a DOS command follows.
! copy /b c:\tmp\cmdlog.txt + c:\tmp\time.txt c:\tmp\cmdlog.txt /y
cmdlog using c:\tmp\cmdlog.txt , append

// Open the log (stata.log).
set logtype text
log using c:\tmp\stata.log , replace
```

Compared to the **profile.do** suggested in section 1.2, this version adds a time stamp to the command log file (**cmdlog.txt**). This means better possibilities to reconstruct previous work.

Appendix 1

Purchasing Stata and manuals

To most users the following manuals will suffice:

- [GSW] *Getting Started* manual
- [U] *User's Guide*
- [R] *Base Reference Manual* (four volumes)

but I wouldn't recommend less to anybody. This booklet does not intend to replace the manuals, but hopefully it can serve as a guide.

If you work with epidemiological studies and survival analysis, you also need:

- [ST] *Survival Analysis and Epidemiological Tables*

If you want to decide exactly how your graphs should look, you need:

- [G] *Graphics* manual

However, the Graphics manual is a difficult companion; it took quite some time for me to understand where to look for what.

If you want to write your own programs (ado-files), the User's Guide helps you some of the way, but you may need:

- [P] *Programming* manual

The Scandinavian sales agent for Stata and StatTransfer is Metrika (www.metrika.se).

Students and employees at University of Aarhus and Aarhus University Hospital can purchase Stata at a special discount rate. Other educational institutions may have similar arrangements.

Various local information concerning Stata and other software may be found at:

www.biostat.au.dk/teaching/software.

Appendix 2

EpiData 3.0

www.epidata.dk

EpiData is an easy-to-use program for entering data. It has the facilities needed, but nothing superfluous. Data entered can be saved as EpiInfo, Excel, DBase, SAS, SPSS and Stata files. EpiData with documentation is available for free from www.epidata.dk.

EpiData files

If your dataset has the name **first**, you will work with three files:

- first.qes** is the definition file where you define variable names and entry fields.
- first.rec** is the data file in EpiInfo 6 format.
- first.chk** is the checkfile defining variable labels, legal values and conditional jumps.

Suggested options

Before starting for the first time, set general preferences (File < Options). I recommend:

<p>[Show dataform]</p> <p>Font: Courier New bold 10pt. Background: White Field colour: Light blue Active field: Highlighted, yellow Entry field style: Flat with border Line height: 1</p>	<p>[Create datafile]</p> <p>IMPORTANT: First word in question is fieldname Lowercase</p>
--	---

Working with EpiData

EpiData's toolbar guides you through the process:

[Define data] = [Make datafile] = [Add checks] = [Enter data] = [Document] = [Export data]

[Define Data]: You get the EpiData editor where you define variable names, labels, and formats. If the name of your dataset is **first**, save the definition file as **first.qes**:

```
FIRST.QES   My first try with EpiData.

entrdate   Date entered           <today-dmy>
lbnr       Questionnaire number   ####
init       Initials               ____
sex        Sex                    #      (1 male  2 female)
npreg      Number of pregnancies  ##
=====
                                           Page 2
bdate      Date of birth           <dd/mm/yyyy>
occup      Occupation              ##      (see coding instruction OCCUP)
```

- The first word is the variable name, the following text becomes the variable label.
- **##** indicates a two-digit numeric field,
- **##.#** a four-digit numeric field with one decimal.
- **____** a three character string variable,
- **<dd/mm/yyyy>** a date,
- **<today-dmy>** an automatic variable: the date of entering the observation.
- Text not preceding a field definition ("**1 male 2 female**"; "**=====**"; "**Page 2**") are instructions etc. while entering data.

Variable names can have up to 8 characters **a-z** (but not æøå) and **0-9**; they must start with a letter. Avoid special characters, also avoid **_** (underscore). If you use Stata for analysis remember that Stata is case-sensitive (always use lowercase variable names).

[Make Datafile]: Save the empty data file **first.rec**.

[Add checks]: You do not have to write the actual code yourself, but may use the menu system. The information is stored in a checkfile (**first.chk**) which is structured as below.

```
* FIRST.CHK
LABELBLOCK
  LABEL sexlbl
    1 Male
    2 Female
  END
END
sex
  COMMENT LEGAL USE sexlbl
  JUMPS
    1 bdate
  END
END
```

Good idea to include the checkfile name as a comment.

Create the label definition **sexlbl**. You might give it the name **sex** – but e.g. a label definition **n0y1** (0 No; 1 Yes) might define a common label for many variables

Use the **sexlbl** label definition for **sex**. Other entries than 1, 2, and nothing will be rejected.

If you enter 1 for **sex**, you will jump to the variable **bdate**; see the menu below.

The meaning of the Menu dialog box is not obvious at first sight, and I will explain a little:

Checkfile name

Select the variable

Variable label and data type displayed

Define possible values. A range e.g. as: 0-10, 99

Jump to **bdate** if **sex** is 1

Skipping the field may be prevented

Same value in all records (eg operator ID)

[▼] Select among existing label definitions

[+] Define new value labels

Save Save variable definitions

Edit Edit variable definitions

[Enter data]: You see a data entry form as you defined it; it is straightforward. With the options suggested the active field shifts colour to yellow, making it easy for you to see where you are.

As an assurance against typing mistakes you may enter part or all of the data a second time in a second file and compare the contents of file1 and file2.

[Document] lets you create a codebook, including variable and value labels and checking rules. The codebook shown below displays structure only, to be compared with your primary codebook; you also have the option to display information about the data entered.

[Export]: Finally you can export your data to a statistical analysis programme. The .rec file is in EpiInfo 6 format, and EpiData creates dBase, Excel, Stata, SPSS and SAS files. Variable and value labels are transferred to Stata, SAS and SPSS files, but not to spreadsheets.

Appendix 3

NoteTab Light

www.notetab.com

Both the Results window and the Viewer window have limitations in the ability to handle output, and you will benefit from a good text editor. I use NoteTab Light, available for free from www.notetab.com. I find NoteTab superior to Stata's Do-file editor; however you cannot execute a do-file directly from NoteTab as you can from the Do-file editor. The use is straightforward, like a word processor. I recommend the following options:

View ► Printing Options	
Margins	Left 2 cm, Right 1 cm, Top 1 cm, Bottom 1 cm
Font	Courier New 9 pt
Other	
Page Numbers	Top, right
Number format	Page %d
Header	Date + Title
Footer	None
Date Filter	"your name" dd.mm.yyyy hh:nn

When you finished, save the settings by clicking the [Save] button.

Some NoteTab versions put a `.txt` extension to every file when saving it. To prevent this:

View ► Options ► File Filters
Default Extension: (nothing)

Making NoteTab work with Stata

In the following I assume that `profile.do` (see section 1.2) defined `c:\tmp\stata.log` as the full log, including both commands and results; it must be in simple text format, not SMCL format.

Open `c:\tmp\stata.log` in NoteTab to view, edit and print results. In NoteTab each file has its own tab; you need not close them at exit. If NoteTab was open while running Stata you might not see the latest output, but don't worry, just select **E**dit ► **R**eload (or [Alt]+[E] [R]), and you have access to the updated output. Or right-click the file's tab and select **R**eload.

I suggest that you create two ado-files (see section 15.7) to ease your work:

nt opens Stata's log in NoteTab

Enter `nt` in Stata's command line window, and the log (`c:\tmp\stata.log`) opens in NoteTab. `winexec` executes a Windows command:

```
program define nt
// c:\ado\personal\nt.ado.  Opens the Stata log file in NoteTab.
  winexec "C:\programmer\NoteTab Light\NoteTab.exe" c:\tmp\stata.log
end
```

newlog discards old log and opens new log

```
program define newlog
// c:\ado\personal\newlog.ado.  Discards old log and opens new log.
  log close
  log using c:\tmp\stata.log , replace
end
```

Index

- A**
- ado-files 62
 - Aggregating data 26
 - anova** 30
 - append** 25
 - Arithmetic operators 20
 - ASCII data 17
 - Axis labels (graphs) 40
 - Axis options (graphs) 40
- B**
- Bar graph options 42
 - Bar graphs 44
 - Bartlett's test 31
 - browse** 6
 - by:** prefix 15
 - bysort** prefix 15
- C**
- Calculations 20
 - cc** 29
 - char** 32
 - ci, cii** 60
 - clear** 16
 - codebook** 19
 - collapse** 26
 - Command line window 5;7
 - Command syntax 13
 - Comma-separated data 17
 - Comments 15
 - compress** 54
 - Conditional commands 14
 - Confidence interval (graphs) 47
 - Confidence intervals 60
 - Connecting lines (graphs) 42;47
 - Continuation lines 15
 - contract** 26
 - Cox regression 35
 - CPR numbers 57
 - cprcheck** 58
 - Customizing Stata 3
- D**
- Data entry 68
 - Data set label 18
 - Data window 6
 - Date formats 57
 - date** function 57
 - Date variables 57
 - describe** 19
 - destring** 55
 - display** 60
 - do** 7
 - Do-file editor 6
 - Do-files 7;10;65
 - drop** 23
 - dropline** (graphs) 49
- E**
- egen** 21
 - encode** 55
 - Entering data 68
 - EpiData 68
 - epitab** command family ...29
 - Error messages 9
 - Excel 63
 - expand** 26
- F**
- File names 10
 - findit** 9
 - Fixed format data 17
 - for** 22
 - foreach** 62
 - format** 12
 - Format, dates 57
 - Format, numeric data 12
 - Format, strings 55
 - forvalues** 62
 - Freefield data 17
 - function** (graphs) 50
- G**
- generate** 21
 - Goodness-of fit test 33;37
 - Graph area 39
 - graph bar** 44
 - Graph command syntax 39
 - graph display** 52
 - graph matrix** 51
 - Graph options 40
 - graph save** 52
 - graph twoway**
 - connected** 47
 - graph twoway dropline** 49
 - graph twoway function** 50
 - graph twoway line** 46
 - graph twoway rcap** 47
 - graph twoway rspike** 48
 - graph twoway scatter** 45
 - graph use** 52
 - Graphs 38
 - Grid lines (graphs) 40
- H**
- help** 9
 - histogram** 43
 - Hosmer-Lemeshow test 33
- I**
- ICD-10 codes 56
 - if** qualifier 14;20
 - Immediate commands 60
 - in** qualifier 14
 - infile** 17
 - infix** 17
 - input** 16
 - insheet** 17;63
 - Installing Stata 3
- K**
- Kaplan-Meier curve 35
 - keep** 23
 - Kruskall-Wallis test 31
- L**
- label** 18
 - Labels 18
 - lfit** 33
 - Line graphs 46
 - Linear regression 32
 - list** 27
 - Log files 8
 - Logical operators 20
 - logistic** 33
 - Logistic regression 33
 - Logrank test 35
 - Long command lines 15
 - lroc** 33
 - lsens** 33
 - lstat** 33
- M**
- Macro 62
 - Mann-Whitney test 31
 - Mantel-Haenszel analysis 29
 - Manuals 9;67
 - Markers (graphs) 42
 - Matrix scatterplot 51
 - mdy** function 57
 - Memory considerations 54
 - merge** 25
 - Missing values 12

N		recode 22	summv1 27
newlog.ado (user program)	70	Reference line (graphs) 48	Survival analysis 34
Non-parametric tests	31	regress 32	Syntax 13
Normal distribution	31	Regression analysis 32	
Notation in this booklet	2	Regression, Cox..... 35	
note	19	Regression, linear 32	
NoteTab Light	70	Regression, logistic 33	
nt.ado (user program)	70	Regression, Poisson..... 37	
Number lists	14	Relational operators..... 20	
Numbering observations	24	rename 23	
Numeric formats	12	Reordering variables..... 23	
Numeric ranges	14	replace 21	
Numeric variables	11	reshape 26	
numlabel	18	Results window 5;8	
		Review window 5	
		ROC curve..... 33	
		run 7	
O		S	
Observations	11	sample 23;59	
oneway	30	Sample size estimation 61	
Open a graph..... 52		sampsi 61	
Operators..... 20		save 16	
Options..... 13		Saving graphs 52	
order	23	Scatterplot, matrix 51	
outfile 63		Scatterplots..... 45	
Output	8	Schemes (graphs) 53	
outsheet	63	sdtest 31	
		search 9	
		Selecting observations..... 23	
		Selecting variables..... 23	
		Simulations..... 59	
		slist 27	
		sort 24	
		Spreadsheets 63	
		SPSS and Stata 63	
		st command family 34	
		Stata manuals..... 9	
		StatTransfer 63	
		stcox 35	
		stptime 35	
		Stratified analysis 29	
		String formats 55	
		string function..... 56	
		String variables..... 55	
		sts graph 35	
		sts list 35	
		sts test 35	
		stset 34	
		stsplot 36	
		stsum 36	
		Study power..... 61	
		substr function..... 56	
		summarize 27	
P		T	
Plot area	39	tab1 28	
pnorm	31	tab2 28	
poisgof 37		tabi 29	
poisson 37		table 30	
Poisson regression..... 37		Tab-separated data 17	
Power estimation..... 61		tabulate 27	
P-P plot 31		Ticks (graphs)..... 40	
predict 32		time.ado (user program) 62	
profile.do 4;66		Transposing data..... 26	
Programs	62	T-test..... 31	
Purchasing Stata..... 67		ttest, ttesti 31	
		twoway connected 47	
		twoway dropline 49	
		twoway function 50	
		twoway graphs 43	
		twoway line 46	
		twoway rcap 47	
		twoway rspike 48	
		twoway scatter 45	
		U	
		Updating Stata	3
		use 16	
		V	
		Value labels	18
		Variable labels	18
		Variable lists	13
		Variable names	11
		Variables..... 11	
		Variables window	5
		Variance homogeneity	31
		Viewer window..... 6;8	
		W	
		Weighting observations	14
		whelp 9	
		Wilcoxon test..... 31	
		Windows in Stata..... 5	
		X	
		xi: prefix..... 32	
		xpose 26	
Q		X	
qnorm	31		
Q-Q plot..... 31			
Qualifiers..... 13			
Quotes	15		
R			
Random numbers	59		
Random samples	23;59		
real function	55		