Barbara Sianesi
Institute for Fiscal Studies

December, 2003

An introductory guide to

OVERVIEW

Stata resources

- For installed commands:
  `help` *command*
  `view help` *command* ‖ displays the help in the Viewer window
- To look for user-written Stata programs over the Internet:
  `net search` *keywords*
- Powerful search to find information on Stata material, both installed and over the Internet:
  `findit` *keywords*

General syntax

[`by` *varlist*`:`] *command* [*varlist*] [=*exp*] [*weight*] [`if` *exp*] [`in` *range*] [`,` *options*]

the square brackets denote optional elements, *varlist* a list of variables, *command* a Stata command, *exp* an algebraic expression, *range* an observation range and *options* a list of options.

- Stata's syntax is case sensitive.
- *varlist*

  If no variables are specified, the command is applied to all the variables in the dataset, equivalent to `_all`

  varB-varF → all the variables stored in between (to see: `desc`; to change: `order`)
  `*` → 0 or more characters go here
  `edu*` → all variables whose names start with edu
  `*78` → all variables whose names end with 78
  `?` → 1 character goes here
  `?*` → 1 or more characters etc.

- **`if`** and **`in`** allow to restrict the command to a specific subset of the data:
  - `in` *range* specifies the observations numbers. Note: as presently sorted!

  Examples:

`in 1`	the 1st observation	`in 5/10`	5th through 10th observation
`in -1`	the last observation	`in –3`	the 3rd-from-last observation
`in 5`	only the 5th observation	`in –9/-1`	the last 9 observations

  - `if` *exp* selects observations based on specific variable values, which must satisfy the if condition(s)

==	!=	>	>=	<	<=	&	|	!
equal	unequal	larger than	larger or equal	smaller than	smaller or equal	and	or	not

Examples:
```
if wage<1000
if place=="Canada" & age!=.
```

- **by** *varlist*: *command*
  repeat the command for each subset of the data for which the values of the variables in *varlist* are equal
  If data not sorted by *varlist* use either:
  `by` *varlist*`, sort`
  `bysort` *varlist.*

  Example:
  ```
  bysort foreign: summarize wage
  ```

- Subscripts:
  var[2] → the 2nd observation on var
  `_n` → the number of the current observation
  `_N` → the total number of observations
  Very handy if combined with the **by** *varlist*: prefix. More on this below.

- A dot (`.`) denotes a missing value.
  Note: a missing variable is always considered larger than any other value.

- Stata commands and variable names can be abbreviated, as long as no confusion arises.

- accessing Stata output
  From the last-run model:
  `_b`[*varname*] → the coefficient of *varname*
  `_se`[*varname*] → the std error of the coefficient of *varname*
  From the last-run command:
  Estimation command: `estimates list`
  `e(`*name*`)`
  General command: `return list`
  `r(`*name*`)`

- <u>list of numbers</u>
  1/3 → 1, 2, 3 5(10)35 → 5, 15, 25, 35
  1 2 3 → 1, 2, 3 8(-2)2 → 8, 6, 4, 2
  and combinations thereof

- quotes " " are used for strings, also for names of paths if they contain spaces
- to stop what Stata is doing: press the Break button
- to retrieve previous commands typed in: PgUp
- to delete a full line of a command: Esc

- Note: to handle weights (not considered in this handout): see `help weights`

Printing and preserving output

- just a table, list of data, etc.
  select with mouse in the Results Window and Edit/Copy or Edit/Copy Table to put it into the Windows Clipboard. Can then Edit/Paste it e.g. in Excel.

- whole sessions: Using Log files

  [`capture`] `log using` *filename*[`.log`]
  [`capture`] `log using` *filename*[`.log`]`, append`
  [`capture`] `log using` *filename*[`.log`]`, replace`
  `log off` ‖ to temporarily suspend the recording of the session
  `log on` ‖ to resume
  [`capture`] `log close`

  Note: If the `.log` extension is specified, the corresponding log-files can be opened, viewed, *edited* and printed from *any* text editor.
  The default otherwise is to create a `.smcl` file, which can be opened, viewed and printed in the Stata viewer: File/View…

PROGRAMMING A SEQUENCE OF TASKS: DO- FILES

instead of typing commands at the keyboard:
place all the commands you want to perform in a file

- can write it in any text editor (save with `.do` extension)
- Stata's do-file editor is very handy; click the button or type `doedit`.
- can save Review contents as a do-file
- Do-file editor: can do just a selection, even without saving
- launch do-file (from Do-file editor or within Stata with: `do` *filename*) and minimise Stata

Some useful commands & features

in creating, debugging and using do-files

- `version` 7

→ Stata is continually being developed; this ensures that your program will continue to work under future releases

- `set more off`

→ preventing Stata to stop and wait for key to be hit

- `clear`

→ to start from a clean slate
Note: can also be used as an option in `use` *newdata*`, clear`

- `capture` *command*

→ perform the command if it can, if not, then just moves on to the next instruction. E.g.:

`cap log close` before `log using` …
`cap drop` *varname* before `generate` *varname* `=` …

- Comments

`*` this line is not executed (a line commencing with * is ignored)

`/*` from here to below
these lines
are not executed `*/`

- Long lines

Option (a): comment out the carriage return (line break):

```
quietly replace lnf = theta2-  /*
   */   ln(1+exp(theta1)+exp(theta2)+exp(theta3)+  /*
   */   exp(theta4)) if treatment==3
```

Option (b): change the end-of-line delimiter from carriage return (cr) to `;`

```
use mydata
#delimit ;
quietly replace lnf = theta2-
     ln(1+exp(theta1)+exp(theta2)+exp(theta3)+
     exp(theta4)) if treatment==3 ;
sum lnf ;
#delimit cr
tab treatment
```

- `quietly` *command*

→ to suppress output

- `display`

→ can be use as calculator, e.g. `di 3*6.8`
→ can be used to re-display specific results, e.g. `summarize` *xvar*
`di r(mean)*r(N)`

- logging the output of a do-file

can either open a log, launch the do-file and then close the log
or can incorporate these commands in the do-file itself

- calling other do-files

do-files can call other do-files, which in turn can call other do-files and so on
Need to be careful with location of the necessary do-files;

Macros

are names that can stand for expressions, strings, variables, numbers, results from the program or results defined by the user

local macros

- are local to the program, i.e. exist only within the program that defines them
- created by: `local` *name* = *exp*
  `local` *name* "*string*"
- to refer to their content: `*name*'

global macros

- once defined, they remain in memory and can be used by other programs
- created by: `global` *name* = *exp*
  `global` *name* "*string*"
- to refer to their content: $*name*

A useful use for global macros: to store lists of regressors

```
global Xreg "age age2 sex edu2-edu4"
```

then at any time in the do-file/session:

```
regress logw $Xreg
```

then do other things, then:

```
probit group $Xreg duration
```

A useful use for local macros: as temporary variables

- will not clash with other variables with the same name
- automatically dropped when the program is terminated

`tempvar` *varname*

then refer to it as `*varname*'

ACCESSING DATA

Opening and saving Stata files

- opening a Stata file:
  `use` *filename* [`, clear`]
- reading a subset of the data:
  `use` *varlist* [`if` *exp*] [`in` *range*] `using` *filename* [`, clear`]
- saving a Stata file:
  `save` *filename*
  `save` *filename* `, replace`
  `save, replace`

Increasing memory

```
set memory #
```

Looking at the dataset

- Describing the contents of the data
  `describe`
  `describe using` *filename* → for data on disk

- Counting observations
  `count [if` *exp*`]`

E.g. to count the number of individuals in a datset with >1 observation per individual:

```
sort persid
count if persid==persid[_n-1]
```

- Listing data
  `list` [*varlist*] [`in` *range*] [`if` *exp*]

DATA MANAGEMENT

Renaming, dropping and documenting variables

Renaming a variable

`rename` *oldname newname*

Documenting

- a dataset:
  `label data` "*data label*"
- a variable:
  `label variable` *varname* "*varlabel*"
- the values of a categorical variable:
  `label define` g*lbl 0* "*male*" *1* "*female*"
  `label values` *gender glbl*

Dropping variables

`drop` *varlist*
`drop` [*varlist*] `in` *range*
`drop` [*varlist*] `if` *exp*
[`by` *varlist*`:`] `drop` *varlist*

Sometimes it's simpler to specify which ones to keep:

[`by` *varlist*`:`] `keep` *varlist* [`in` *range*] [`if` *exp*]

Generating and replacing variables

to modify the values of an existing variable:

`replace` *oldvar=exp* [`if` *exp*] [`in` *range*]

to create new variables:

`generate` [*type*] *newvar=exp* [`if` *exp*] [`in` *range*]

type: storage type of the (numerical) variable being created:

	Bytes	Min	Max
byte	1	-127	126
int	2	-32,767	32,766
long	4	-2,147,483,647	2,147,483,646
float	4	1E+36	10^36
double	8	1E+308	10^308

After having generated variables: `compress`

Examples (note the abbreviations):

```
replace rate = rate*100
replace age=25 if age==250

g constant=5
g logw = log(wage)
g age2 = age*age    /* or:  g age2 = age^2  */

sort idcode year
by id: g ustate = sum(union)
lab var ustate “cumulative periods of union membership”
drop constant ustate
```

Useful functions (see `help functions`):

```
log(), abs(), int(), round(), sqrt(), min(), max(), sum()
```

statistical functions
string functions (to manipulate strings and to convert between strings and numbers)
date functions
and more

Accessing Stata output (see above, general syntax):

```
summarize wage if sex==1
g maxincmale=r(max)

count if female==1
g number_fem=r(N)
```

Subscripts (see above, general syntax)

```
by id: g unionlag = union[_n-1]
by id: g dxvar = xvar-xvar[_n-1]

sort id year
by id: g entryage = age[1]
by id: g exitage  = age[_N]
```

Extended generate (see `help egen`):

```
egen meangrade = mean(grade), by(id)
egen income85  = pctile(income), p(85) by(region)
```

Recoding variables:

`recode` *varname rule* [`if` *exp*] [`in` *range*]
(see `help recode` for examples)

Dealing with categorical and dummy variables

Creating dummy (0-1) variables:
`g` *varname* `=` *exp*
→ dummy varname = 1 if *exp* is true and = 0 otherwise

```
g wagehigh = wage>=1000 if wage!=.
g age30=age==30
```

From continuous to categorical variables:

```
g age_gr = 1+(age>35)+(age>45)
```

– `recode(`oldvar, x1, x2,…,xk`)`

```
g age_gr1 = recode(age, 35, 45, 55)
```

– `autocode(`oldvar, #groups, xmin, xmax`)`

```
g age_gr2 = autocode(age, 3, 25, 55)
```

– `group(#)`

```
g age_gr3 = group(3)
```

From categorical to dummy:

`tab` *varname*, `g(`*varname2*`)`
`xi` *varlist_with*`_i.` (see p.15)

Dealing with string variables
`encode`, `recode`

Dealing with date variables
`date`, `mdy`

Combining and reshaping datasets

Combining: `append`, `merge`

Reshaping: `stack`, `xpose`, `reshape`, `collapse`

- sorting data

`sort` *varlist*
→ in ascending order of varlist – NB: missing values last!

`gsort` *–varname1 varname2*
→ if `-`, then in descending order

TAKING A FIRST LOOK AT THE DATA

Summarising the data

`summarize` [*varlist*] [`in` *range*] [`if` *exp*]
→ no. of non-missing obs, mean, std deviation, min and max

`, detail`
→ quantiles , 4 smallest and largest values, variance, mean, skewness and kurtosis

Exploratory data analysis

- Means

`means` [*varlist*] [`in` *range*] [`if` *exp*]
→ arithmetic, geometric and harmonic means and corresponding confidence intervals

- Centiles

`centile` [*varlist*] [`in` *range*] [`if` *exp*], `c(`*numlist*`)`
e.g. `c(5)` → the 5th centile
`c(10(10)90)` → the 10th, 20th, …, 80th and 90th centile
→ centiles and confidence intervals

- Correlations

`correlate` [*varlist*] [`in` *range*] [`if` *exp*]
`[,covariance]` → instead of correlation coefficients

`pwcorr` [*varlist*] [`in` *range*] [`if` *exp*]
→ all the pairwise correlation coefficients between the variables in *varlist*
`[,sig` → include significance level
`star(#)]` → star all the coefficients significant at the #*100% or more

- Tables

1. One-way tables: frequencies

`tabulate` *varname* [`in` *range*] [`if` *exp*]
`[,missing` → include missing values
`nolab` → numeric codes instead of labels
`plot ]` → bar chart of relative frequencies

```
sum wage
tab age if wage>r(mean)
```
→ age distribution for above mean-wage earners

2. Two-way tables: frequencies and measures of association

`tabulate` *var1 var2* [`in` *range*] [`if` *exp*]
`[,missing` → include missing values
`nolab` → numeric codes instead of labels
`row` → relative frequency of that cell within its row
`col` → relative frequency of that cell within its column
`nofreq` → frequencies not displayed
`all ]` → display all measures of association:
Pearson chi2, likelihood-ratio chi2, Cramer's V, gamma, Kendall's tau-b
(tests of the hyp that row and col variables are independent)

3. Summary statistics

`tabulate` *var1* [*var2*] [`in` *range*] [`if` *exp*]`, sum(`*var3*`)`
→ summaries of *var3* – mean, std dev and frequency – by (i.e. conditional on) *var1* (and *var2*)

Are there differences in wage and wage dispersion by county?

```
tab county, sum(wage) nofreq
```

`table` *rowvar* [*colvar* [*supercolvar*]] [`in` *range*] [`if` *exp*]
`, c(`*clist*`)` → mean/sd/count/max/min/med/sum/p# varname,
`row` → total across rows
`col` → total across columns
`by(`*superrowvar*`)`

```
table  edcat, c(count  wage mean  wage sd wage)
table  edcat foreign, c(mean  wage) row col
table  foreign, c(mean  wage) by(edcat) row
```

ESTIMATION

Overview

[by *varlist*:] *command* *yvar* *xvarlist* [if *exp*] [in *range*] [, *options*]

if and in define the estimation sub-sample
Note: in order not to clutter notation, in the following, they are omitted.

Useful options:

, robust	→ robust standard errors (White correction for heteroskedasticity)
cluster(*persid*)	→ if repeated obs per individual, with robust
level(#)	→ set significance level for confidence intervals. Default = 95

- To replay the last results (at any time before a new estimation or a clear):
  command

- To display the V/Cov matrix of $\hat{\beta}$ after estimation:
  vce[, corr]

- To retrieve the

V/Cov matrix	→ e(V)
coeff on var	→ _b[*var*]
std err of coeff on *var*	→ _se[*var*]

Regression analysis

OLS

regress *yvar* *xvarlist*

Dummy variables

xi : *command varlist*
with *varlist* of the form:

i.*var*	→ created dummies for categorical var
i.*var1**i.*var2*	→ creates dummies for categorical var1 and var2 plus all interactions
i.*var1***var3*	→ creates dummies for categorical var1 and continuous var3 plus all interactions

Manually:

```
g foreign_2 = foreign*(agecat==2)
g foreign_3 = foreign*(agecat==3)
```

Predicted values and residuals

`predict` *newvar* `[,` *statistic*: in particular
`xb` → default: predicted value of dependent variable
`residual]` → the residuals

```
xi: regr logw age age2 i.sex*i.edcat
predict fitted
predict resid, residual
graph resid fitted, yline(0) ylabel xlabel
graph logw fitted age, by(sex) c(.l) s(oi) sort
```

Hypothesis testing

Note: `regress` already provides overall F test and individual t tests

1) linear hypothesis (Wald test)

`test` *exp=exp*
`test` *coefficientlist*
`, accumulate` → jointly with previous test

Note: for `test`, both *varname* and `_b[`*varname*`]` denote the coefficient on *varname*.

Examples:

```
regr logw age group sex edu2-edu4
test age group sex edu2-edu4
test group
test age=1
test 2*(age+sex) = -3*(edu2-(edu3+1))

test _b[x1]=0
test _b[x2]=0, acc

test x1 x2 x3
```

2) non-linear hypothesis (Wald test)

`testnl` *exp=exp*

```
testnl (3*_b[age]^2=_b[sex]) (_b[sex]/_b[foreign]=4)
```

→ testing two hypotheses jointly

Robust regression

```
xi: regr logw group age age2 sex i.edcat, robust
```

Instrumental Variables

`ivreg` *depvar* [*exogvarlist*] (*endogvarlist* = *IVvarlist*)

→ estimates a linear regression model of *depvar* on *exogvarlist* and *endogvarlist* using *IVvarlist* (along with *exogvarlist*) as instruments for *endogvarlist*.

Binary qualitative outcome model

`probit` *depvar indepvarlist* [`, robust`]
→ estimate maximum-likelihood probit models

`dprobit`
→ same as `probit` but instead of reporting coefficients, it reports the change in the probability for an infinitesimal change in each independent, continuous variable and, by default, the discrete change in the probability for dummy variables

`logit` *depvar indepvarlist*
[`, robust`
`or`] → report coefficients β transformed to odds ratios exp(β)

Note: `logistic` is identical command, with some minor differences