



UNITED NATIONS
UNIVERSITY

UNU-WIDER

World Institute for Development
Economics Research

Introduction to data analysis using STATA

Miguel Niño-Zarazúa

World Institute for Development Economics Research

United Nations University

Background

- STATA is powerful command driven package for statistical analyses, data management and graphics
- STATA provides commands to conduct statistical tests, and econometric analysis including panel data analysis (cross-sectional time-series, longitudinal, repeated-measures), cross-sectional data, time-series, survival-time data, cohort analysis, etc
- STATA is user friendly, it has an extensive library of tools and internet capabilities, which install and update new features regularly

Introduction

- Stata /IC (or Intercooled Stata) can handle up to 2,047 variables. There is a special edition, Stata/SE that can handle up to 32,766 variables (and also allows longer string variables and larger matrices), and a version for multicore/multiprocessor computers called Stata/MP, which has the same limits but is substantially faster
- These three versions of STATA are available both for 32-bit and 64-bit computers; the latter can handle more memory (and hence more observations) and tend to be faster

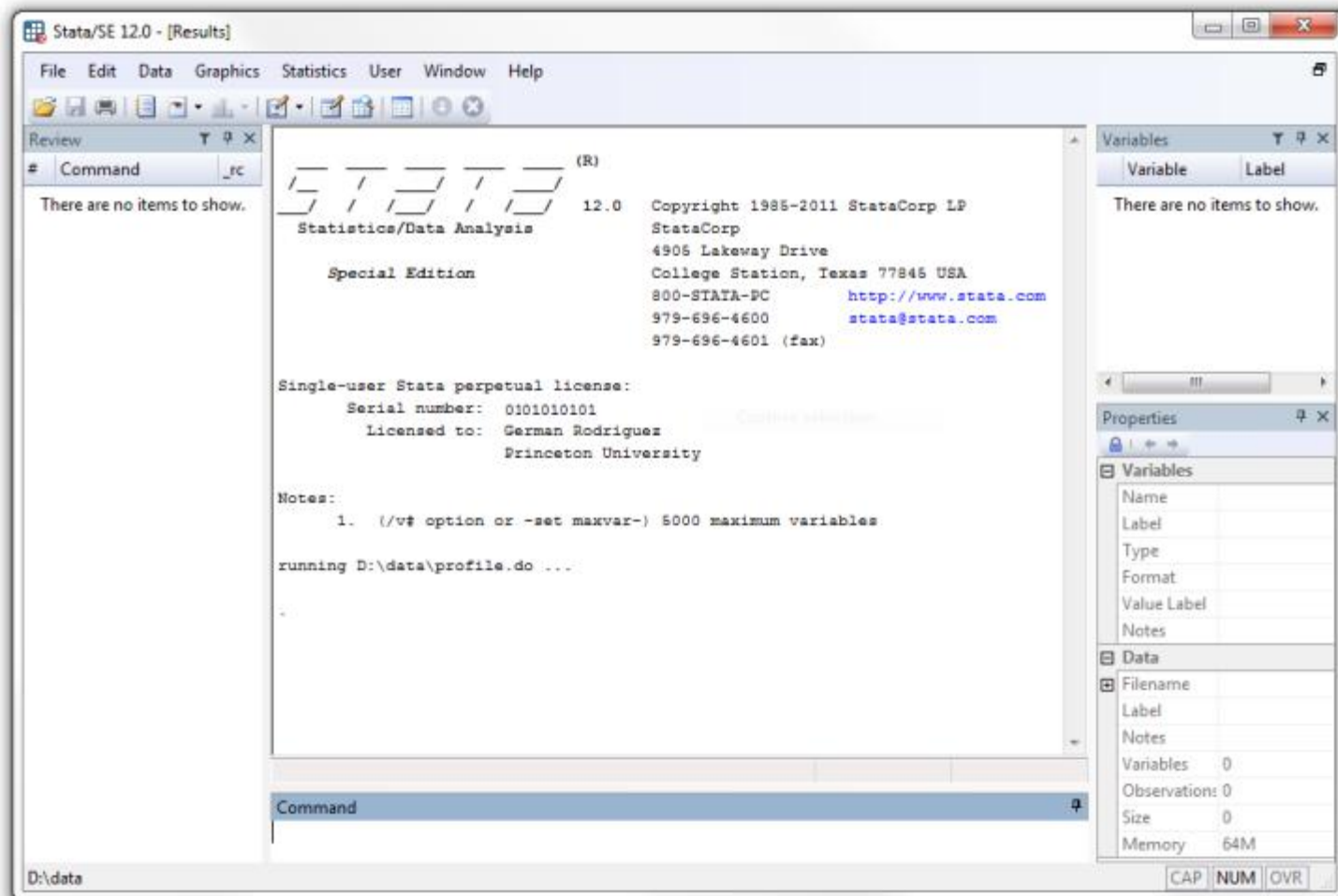
Transferring other files into Stata format

- There are various ways to enter data into Stata:
 1. Manual entry by typing or pasting data into data editor
 2. Inputting ASCII files using `infile`, `insheet` or `infix`
 - i. If using text editing package to assemble dataset, save as text (.txt) file, not default (*e.g.* .xlsx)
 - ii. Free format data (*i.e.* excel columns separated by space, tab or comma *etc.*): use `infile` or `insheet`, for example: `insheet using filename`
 - iii. Fixed format data (*i.e.* data in fixed columns): use `infix`.
 3. If data in another format (*e.g.* SAS, SPSS), **Stat/Transfer** can be used to create a Stata dataset directly
- Stat/Transfer is able to optimise the size of the file (in terms of the memory required for each variable)

Bonus for the session: You will get a copy of Stat transfer

Stata windows


- When Stata starts up you will see five docked windows, initially arranged as shown below



Stata windows

- In the *Command window* you can type the commands. Stata shows the results in the larger window immediately above, called *Results*
- The history of command operations is listed in the window *Review* on the left, so you can keep track of the commands you have used.
- The *Variables window*, on the top right, lists the variables in the dataset
- The *Properties* window immediately below that (new in version 12), displays properties of the variables and datasets
- There are other windows that are useful, namely the *Graph*, *Viewer*, *Variables Manager*, *Data Editor*, and *Do file Editor*.
- Stata's graphical user interface allows selecting commands and options from a menu and dialog system. I strongly recommend to use the command language, and specifically *do.files* as a way to ensure replicability of the analysis

Exercise 1

1. Open Stata
2. Identify the Results window, Command window, Review window, Variables window
3. Type use "C:\Documents and Settings\Miguel Zarazua\My Documents\My documents\UNU-WIDER\GAPP project\STATA course\stata files Zambia HH survey 1998\HHINCOME.DTA", clear
4. Open the data editor () and inspect the data. What do you observe?
5. Exit the data editor and then clear the memory by typing `clear` in Command window
6. Look at Help Menu: Help → Contents . Inspect the links

Variable types

- STATA can handle numbers or strings. Numeric variables can be stored as integers (bytes, integers, or longs) or floating point (float or double).
Note: Stata does all calculations using doubles, and the *compress* command finds the most economical way to store each variable in your dataset
- Strings have varying lengths up to 244 characters. Strings are ideally suited for id variables
- You can convert between numeric and string variables. If a variable has been read as a string but really contains numbers you can use the command *destring*. Otherwise, you can use *encode* to convert string data into a numeric variable or *decode* to convert numeric variables to strings
- To inspect the type of variables, look at the “Type” column in the Variables window or type:
`describe [varlist]`

Getting started

- Stata syntax *is* case sensitive. All Stata command names must be in lower case
- Stata commands can often be abbreviated
 - ✓ Look for underlined letters in “Help”)
- With large datasets, it may be necessary to increase the memory limit in Stata from the default of 1 megabyte. Type:
`set memory #`
represents a number of kilobytes (k), megabytes (m) or gigabytes (g)


For example:

```
set memory 100m
```

- By default, Stata assumes all files are in `c:\data`. To change this working directory, type:
`cd foldername`

Note: If the folder name contains blanks, it must be enclosed in quotation marks

Getting started

- Stata datasets always have the extension `.dta`
- Access existing Stata dataset *filename.dta* by selecting File → Open or by typing:
`use filename [, clear]`
- If the file name contains blanks, the address must be enclosed in quotation marks
- *filename* can also be a Stata file stored on the internet
- If a dataset is already in memory (and is not required to be saved), empty memory with `clear` option
- To save a dataset, click  or type:
`save filename [, replace]`
- Use `replace` option when overwriting an existing Stata (`.dta`) dataset

The use of *commands*

- To obtain help on a command (or function) type `help command_name`, which displays the help on a separate window called *the Viewer*
 - Note: Stata commands are case-sensitive. They can also be abbreviated. The documentation and online help underlines the shortest legal abbreviation
- If you don't know the name of the command you need you can search for it. Stata has a *search* command with a few options. You can also type *findit*, which searches the Internet as well as your local machine and shows results in the *Viewer*.
 - Try *search generate* (the command used to generate new variables) or *findit generate*

Operators and Expressions

These are key arithmetic, logical and relational operators you need to keep in mind:

Arithmetic

+ add

- subtract

* multiply

/ divide

^ raise to power

+ string concatenation

Logical

! not (also ~)

| or

& and

Relational

== equal

!= not equal (also ~=)

< less than

<= less than or equal

> greater than

>= greater than or equal

Examples:

```
gen tothhincsq = tothhinc^2 /* generates HH income squared */
```

```
gen Intothhinc = log(tothhinc) /* generates HH income in log form */
```

Useful commands: variable transformations

- *gen* command creates a new variable using an expression that may combine constants, variables, functions, and arithmetic and logical operators

```
gen lnhhinc = log(tothhinc) /* generates HH income in log form */
```

```
gen hhincsq = tothhinc^2 /* squared of hh income */
```

```
gen ten=10 /* constant value of 10 */
```

```
gen id=_n /* id number of observation */
```

```
gen total=_N /* total number of observations */
```

```
gen byte yr=year-1900 /* generates 50,51,..., n instead of 1950,1951,...,n */
```

```
gen rich=tothhinc if tothhinc > 1452500
```

Variable transformations

- The *egen* command creates new variables based on summary measures, such as sum, mean, min and max. For example:

```
egen mhincp=mean(tothhinc), by(province) /*average hincome  
by province */
```

```
egen maxhinc=max(tothhinc) /* largest hh income value */
```

```
egen counthinc=count(tothhinc) /* counts non-missing hh income  
obs */
```

```
egen float tothinc= rowtotal(totfdinc totnfinc), missing
```

```
egen double idi= concat(hid pid) , format(%20.0g) punct(.) /*
```

```
creates string for individual identifiers using the 1998 Zambian Living  
Conditions Monitoring Survey */
```

Variable transformations

`rename` command allows for changing the names of your variables:

```
rename tothinc thinc
```

`recode` allows to change the values that variables take. Suppose value 2020 of variable `year` which actually referred to 2002:

```
recode year 2020=2002
```

`replace` can be used to change the contents of an existing variable:

```
replace oldvar = exp1 [if exp2], e.g.
```

```
replace unemplrate=. if unemplrate==999
```

Note: Any functions that can be used with `generate` can be also used with `replace`. **if** can also be used to restrict the command to a desired subset of observations. The double equal sign `==` is used to test for equality, while the single equal sign `=` is used for assignment

Variable transformations

A label is a description of a variable in up to 80 characters. Useful when producing graphs etc. To create/modify labels type:

```
label variable varname "label"
```

- `rename` may be used to rename variables, as follows:
`rename oldvarname newvarname`

To drop a variable or variables, type:

```
drop varlist
```

Alternatively, `keep varlist` eliminates everything but *varlist*.

To drop certain observations, use:

```
drop if exp
```

For example, `drop if unemplrate==.`

Exercise 2

- Open the dataset "HHROSTER.DTA"
- Use `describe` to ascertain which variables are in string format and which are in real format
- Rename `s1q5` as `sex`; `s1q3b` as `age`, and other variables
- Keep only those observations for adults age 18 and older
- Generate ids for each adult

Appending datasets

- To add another Stata dataset below the end of the dataset in memory, type:

```
append using filename
```

- Dataset in memory is called “master dataset”.
- Dataset *filename* is called “using dataset”.
- Variables (*i.e.* with same name) in both datasets will be combined
- Variables in only one dataset will have missing values for observations from the other dataset.

Appending datasets

Example

```
use even
```

```
list
```

number	even
6	12
7	14
8	16

```
use odd
```

```
list
```

```
number odd
```

1	1
2	3
3	5
4	7
5	9

You can type append using even:

```
append using even
```

```
list
```

Number	odd	even
1	1	.
2	3	.
3	5	.
4	7	.
5	9	.
6	.	12
7	.	14
8	.	16

Note that the missing values are forward-filled with (.)

Merging datasets

- To merge two datasets with identical identifiers (household ids) type:
`merge 1:1 varlist using filename`

For example:

```
use "E:\STATA course\stata\HHINCOME.DTA", clear
merge 1:1 hid using "E:\STATA
course\stata\TOTEXP.DTA"
```

Note, type:

```
duplicates report hid /* for a report on duplicated
observations */
```

```
duplicates examples hid /* lists one example for each group of
duplicated observations */
```

```
duplicates drop hid, force /* drops all but the first occurrence of
each group of duplicated observations. You can use if to restrict the
dropping */
```

Merging datasets

- Stata automatically creates a variable called `_merge` which indicates the results of the merge operation. The variable takes the values:

<code>_merge==1</code>	Observation from master data
<code>_merge==2</code>	Observation from using data
<code>_merge==3</code>	Observation from both master and using data (ideal)

- You can check for non-matching values by typing:

```
tabulate _merge
proportion _merge
drop _merge
```

Merging datasets

- If more than one observation has the same identifier in the master dataset, you can type:

```
merge m:1 varlist using filename
```

- Or if more than one observation has the same identifier in the *using* dataset, type:

```
merge 1:m varlist using filename
```

NOTE: you will need to sort the sample by the matching variable.
For example:

```
use "E:\STATA course\stata\HHINCOME.DTA", clear  
sort hid  
duplicates report hid
```

```
merge 1:m hid using "E:\STATA  
course\stata\HHROSTER.DTA"
```

Exercise 3

1. Open "HHINCOME.DTA" (as the master dataset) and merge it with "TOTEXP.DTA", using hid as the matching variable
 - **Should you use** merge 1:m, merge m:1, or merge 1:1?
 - What happens when you try to merge?
 - Look at the values that **_merge** takes: what does this indicate?

2. Open "HHINCOME.DTA" (as the master dataset) and merge it with "HHROSTER.DTA"
 - **Should you use** merge 1:m, merge m:1, or merge 1:1?
 - What happens when you try to merge?
 - Look at the values that **_merge** takes: what does this indicate?

Creating a dummy variable

- To create a dummy variable, you can do the following:

```
gen poor=0
replace poor=1 if tothhinc<= 1452500 & tothhinc~=.
gen poor=0
replace poor=1 if tothhinc<= 1452500 & tothhinc!=.
```

- A shorter alternative to the above code is:

```
gen poor=(incpc>=1.25 & incpc~=.)
```

- Imagine you have a variable province with categories for each province. You could generate a set of dummy for each province:

```
tab province, gen(prdum)
```

This will create a dummy prdum1 equal to 1 for province 1 and zero otherwise; a dummy prdum2 for province 2 and zero otherwise, etc. When using large numbers of dummies in regressions, useful to name with pattern, *e.g.* id1, id2... Then id* can be used to refer to all variables beginning with *

'By' group processing

To execute a Stata command separately for groups of observations for which the values of the variables in *varlist* are the same, type:

```
by varlist: command
```

Most commands allow the `by` prefix, but data should be sorted by *varlist* (precede command with `sort varlist` or use `bysort`):

```
bysort varlist: command
```

Examples (using data on education, try the following)

```
bysort province: summarize s4q8
```

```
bysort province: tabulate district
```

```
bysort province: ta s4q8 if s1q3b>=18
```

Inspecting the data

- All variables are formatted as either numeric (real) or alphanumeric (string). You can inspect the format by typing:

```
describe [varlist] /*'s' for string and 'g' for numeric */
```

- `codebook` is useful for checking for data errors. This gives information on each variable about data type, label, range, mean, standard deviation *etc*
- Alternatively, `list` simply prints out the data for inspect
 - ✓ `list varname 1/30`
- `tabulate` generates one or two-way tables of frequencies (also useful for checking data):

```
tabulate rowvar [colvar]
```

For example, to obtain a cross-tabulation of `sex` and `educ` type:

```
tab district s1q5
```

Collapsing datasets

- To create a dataset of means, medians and other statistics, we can use the collapse command:

```
collapse (stat) varlist1 (stat) ... [weight], by(varlist2)
```

where ***stat*** can be mean, sd, sum, median or any other statistics, and **by (*varlist2*)** specifies the groups over which the means *etc.* are to be calculated

There are 4 types of weights that can be used in Stata:

1. **fweight** (frequency weights): weights indicate the number of duplicated observations
2. **pweight** (sampling weights): weights denote the inverse of the probability that an observation is included in the sample
3. **aweight** (analytic weights): weights are inversely proportional to the variance of an observation to correct for heteroskedasticity. Often, observations represent averages and weights are number of elements that gave rise to the average.
4. **iweight** (importance weights): weights have no other interpretation


Exercise 4

- Collapse the dataset “HHINCOME.DTA” by province to produce a dataset containing the means of total household income in Zambia
- Use sampling weights so that the variables take into account the inverse probabilities of the observations to be included in the sample
- Which are the provinces with the highest and lowest average household income?
- Repeat the exercise but now including the sex of the household head. What do you observe?

How to deal with missing values

- Stata distinguishes *missing* values. The basic missing value for numeric variables is represented by a dot .
- To check for missing values you can write:
 - List hhn if missing(ageh) (here hhn is the HH number in the income file of the Living Conditions Monitoring Survey 1998)
- Household survey data often use codes such as 888 for *not applicable* and 999 for *not ascertained*. For example age may be coded as 888 for people who didn't know their age and 999 for those who didn't respond. You will need to distinguish these two cases using different kinds of missing value codes
- If you want to recode 888's to .n (for "na" or non-applicable) and 999's to .m (for "missing") you could write:
 - *replace* ageh = .n *if* ageh == 888
 - *replace* ageh = .m *if* ageh == 999

Working with do.files

- A great advantage of STATA is that you can estimate equations and save them into a 'do.file' for future tasks. You can open a do.file by clicking on the 'New Do.file Editor' icon  by using Ctrl-9 in version 12 (Ctrl-8 in earlier versions) or by selecting Window|Do-file Editor|New Do-file Editor in the menu system and then save it.
- Alternatively, you can use an editor such as Notepad. Save the file using extension .do and then execute it using the command `do filename`
- **To load a file** you will need to type `use` in the command window or preferably, by typing `use` in a do.file

Using annotations in do.files

- It is always good to make annotations to your do files with explanatory comments that provide information of what you are trying to do
- In a do file you can use two types of comments: `//` and `/* */`
 - `//` is used to indicate that everything that follows to the end of the line is a comment and should be ignored by Stata. For example:
`gen one= 1 // this is a constant equal to one for a the model`
 - `/* */` is used to indicate that all the text between the opening `/*` and the closing `*/`, is a comment to be ignored by Stata
- It is a good practice to start a do file with comments that include at least a title, and the analysis the researcher is seeking to undertake. Assumptions and notes about required files could also be noted

Using annotations in do.files

- In a do-file you will probably want to break long commands into lines to improve readability. You can use `///`, which allows you to continue on the next line. For example:

```
graph twoway (scatter hhsize sexh) ///  
             (lfit hhsize sexh)
```

- You can also write

```
graph twoway (scatter hhsize sexh) /*  
*/           (lfit hhsize sexh)
```

- An alternative is to use a semi-colon instead through the command `#delimit`:
`#delimit ;`
`graph twoway (scatter hhsize sexh)`
`(lfit hhsize sexh) ;`

To return to using `_carriage_return` as the delimiter use `#delimit cr`

Using Log Files

- To keep a *permanent* record of your results, you could use a log file. Stata writes all results to the log file, so you keep track of your work:

```
log using filename, replace (or text)
```

where *filename* is the name of your log file

- By default the log is written using SMCL, Stata Markup and Control Language, which can only be viewed using Stata's Viewer. The *text* option creates logs in plain text (ASCII) format, which can be viewed in Notepad or a word processor
- The *replace* option specifies that the file is to be overwritten if it already exists or append if you want to append chronologically your work. This is useful if you need to run your commands several times

Exercise 6

Open the do-file editor in Stata. Run commands using throughout the exercises using the dataset “HHINCOME.DTA” and save it as a different file:

```
use "E:\STATA course\stata\HHINCOME.DTA", clear  
save "E:\STATA course\stata\HHINCOME_1.DTA", replace  
Clear  
use "E:\STATA course\stata\HHINCOME_1.DTA", clear
```

Use some of the commands you have learned to transform and generate new variables, including dummies

Basic statistics

`tabstat` produces a table of summary statistics:

```
tabstat varlist [, statistics(statlist) ]
```

Example:

```
tabstat hhsz sexh, stats(mean sd sdmean n)
```

`summarize` displays a variety of univariate summary statistics (number of non-missing observations, mean, standard deviation, minimum, maximum):

```
summarize [varlist]
```

Basic statistics

- `table` displays table of statistics:

```
table rowvar [colvar] [, contents (clist varname)]
```

clist can be `freq`, `mean`, `sum` *etc.*

- *rowvar* and *colvar* may be numeric or string variables.

Example:

```
table sexh, c(mean tothhinc)
```

Note: Missing values are excluded from tables by default. To include them as a group, use the `missing` option with `table`.

Correlations

To obtain the correlation between a set of variables, type:

```
correlate [varlist] [[weight]] [, covariance]
```

`covariance` option displays the covariances rather than the correlation coefficients

`pwcorr` displays all the pairwise correlation coefficients between the variables in *varlist*:

```
pwcorr [varlist] [[weight]] [, sig]
```

```
pwcorr [varlist] [[weight]] [, print(#) star(#)]
```

`sig` adds a line to each row of matrix reporting the significance level of each correlation coefficient. `print(#)` specifies the significance level of coefficients. `star(#)` specifies the significance level of correlation coefficients to be starred

Difference between `correlate` and `pwcorr` is that the former performs listwise deletion of missing observations while the latter performs pairwise deletion

Cross-tabulations

- `tabulate` produces two-way tables of frequency counts, along with various measures of association, including the common Pearson's chi-squared, the likelihood-ratio chi-squared, Cramér's V, Fisher's exact test, Goodman and Kruskal's gamma, and Kendall's tau-b

```
tabulate sexh marith, chi2
```

or

```
tabulate sexh marith, all
```

Linear regression

To perform a linear regression of *depvar* on *varlist*, type:

```
regress depvar varlist [[weight]] [if  
exp] [, noconstant robust]
```

- where *depvar* is the dependent variable;
- *varlist* is the set of independent variables (regressors); The `noconstant` option excludes the constant;
- `robust` specifies that Stata report the Huber-White standard errors, which account for heteroskedasticity;
- With weights, you can obtain weighted least squares (GLS)

Exercise 7

1. Compute the correlation coefficients between these two variables:

```
pwcorr tothhinc expfdhc [aweight=weight], sig
```

Are they significantly correlated at 5% level?

2. Cross-tabulate the following two variables: sexh and marith. Compute the Chi2 and find out whether they are statistically significantly correlated
3. Estimate a heteroskedastic-robust GLS model for Inhhinc on sexh, ageh, ageh2 hhsiz. What do you observe?

DASP poverty toolkit

Go to <http://dasp.ecn.ulaval.ca/modules.htm>

Click on [***Register and download it***](#)

Downloading DASP Package The Stata package DASP is freely distributed. The user should be registered before downloading DASP

To register, choose the item "Registration" and submit your request after completing the required information. You will receive your username and password instantly by email

To download DASP, connect to the intranet service by choosing the item "Login" and then select the item "Download DASP: Distributive Analysis Stata Package"

DASP poverty toolkit

1. Installing and updating the DASP package

In general, the *.ado files are saved in the following main directories:

Priority Directory Sources

1 UPDATES: Official updates of Stata *.ado files

2 BASE: *.ado files that come with the installed Stata software

3 SITE: *.ado files downloaded from the net

4 PLUS: ..

5 PERSONAL: Personal *.ado files

2. Installing DASP modules.

a. Unzip the file dasp.zip in the directory c:

b. Make sure that you have c:/dasp/dasp.pkg or c:/dasp/stata.toc

c. In the Stata command windows, type the syntax

net from c:/dasp

DASP poverty toolkit

d. Type the syntax

net install dasp_p1.pkg, force replace

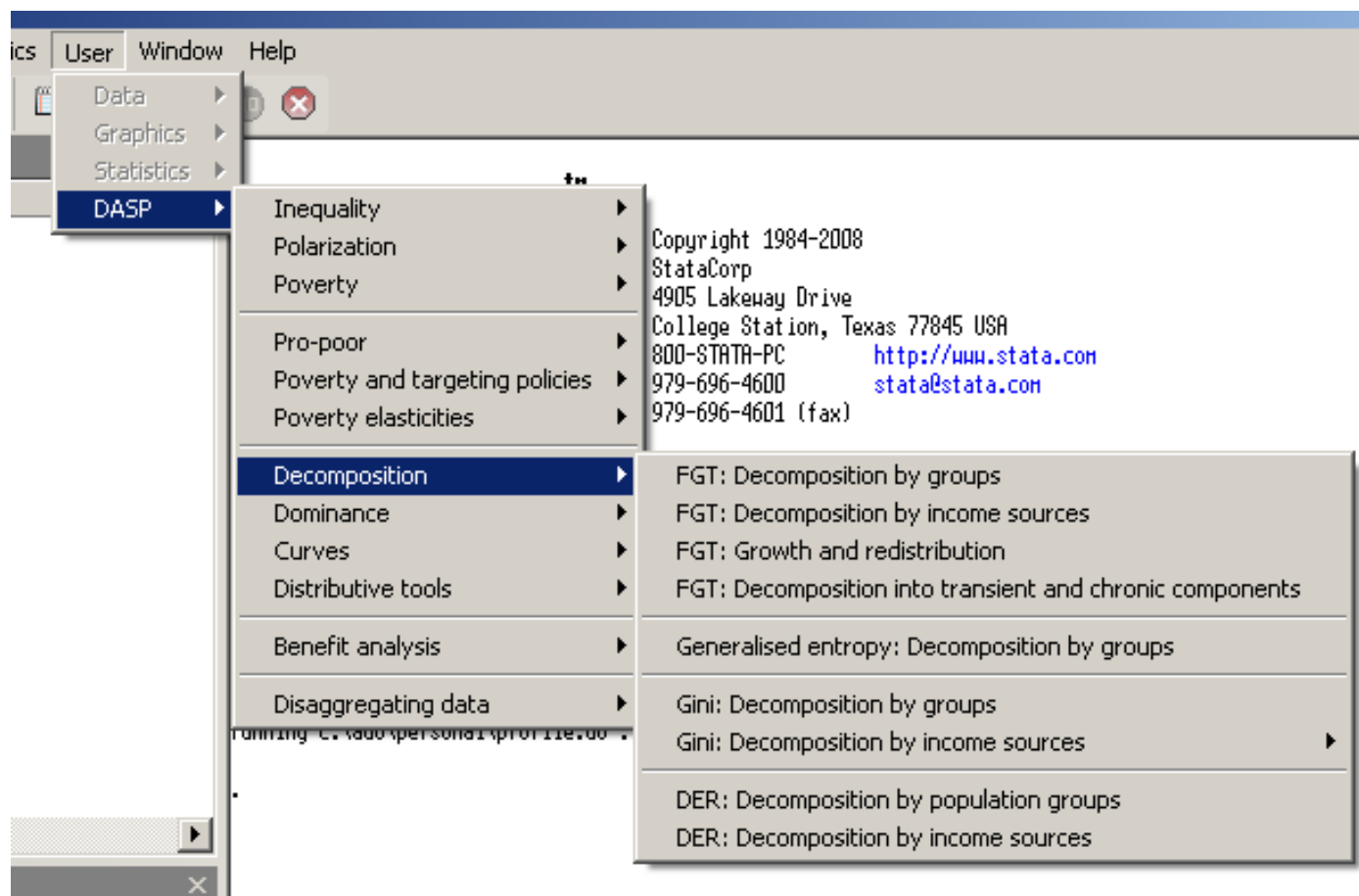
net install dasp_p2.pkg, force replace

net install dasp_p3.pkg, force replace

- To add the *DASP* sub menus, the file ***profile.do*** (which is provided with the *DASP* package) must be copied into the ***PERSONAL*** directory
- If the file ***profile.do*** already exists, add the contents of the *DASP* –provided ***profile.do*** file into that existing file and save it.
- To check if the file ***profile.do*** already exists, type the command:

findfile profile.do

DASP submenu



Estimating poverty (a hypothetical case!)

```
/* deriving income per capita per month */
```

```
gen incpc= tothinc /hhsiz
```

```
summarize incpc, d
```

```
inspect incpc
```

```
iqr incpc
```

```
iv incpc
```

```
/* Dealing with outliers at 5% */
```

```
hadimvo incpc, generate(outliers Doutliers) p(.05)
```

```
drop if outliers==1
```

```
iqr incpc
```

```
lv incpc, generate
```

Estimating poverty (a hypothetical case!)

```
/* let us assume we want to estimate the FGT family of poverty  
measures based on a poverty line set at per capita daily incomes <=2000  
kwacha at prices of 1998 */
```

```
gen dayincpc = incpc/30.415
```

```
ifgt dayincpc, alpha(0) pline(2000)
```

```
ifgt dayincpc, alpha(1) pline(2000)
```

```
ifgt dayincpc, alpha(2) pline(2000)
```

```
/* Poverty estimates based on a relativist measures)*/
```

```
ifgt dayincpc, alpha(0) opl(mean) prop(50)
```

```
ifgt dayincpc, alpha(1) opl(mean) prop(50)
```

```
ifgt dayincpc, alpha(2) opl(mean) prop(50)
```