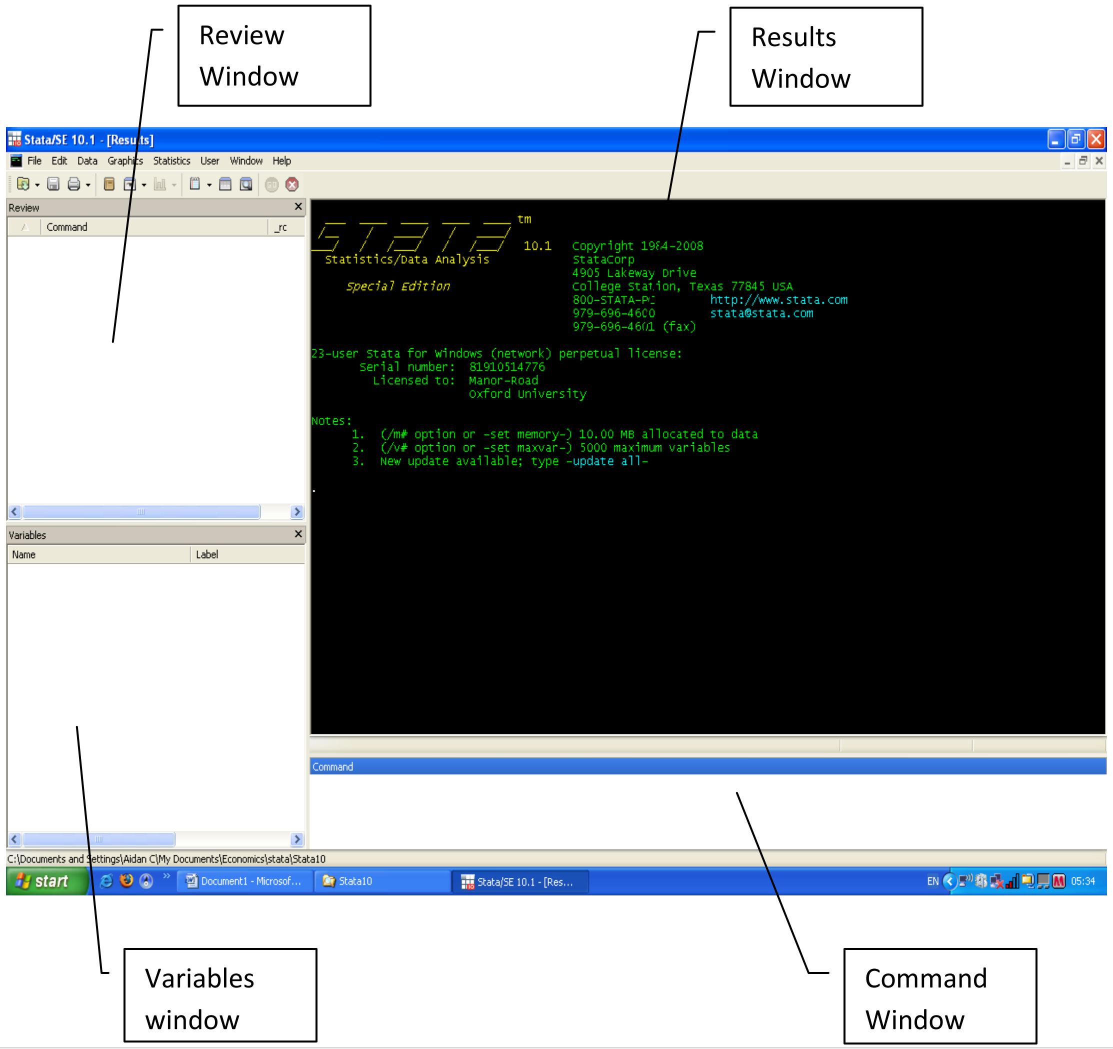
Getting Started with STATA

STATA is a statistical package with both a graphic user interface for 'point and click' analysis and a command window that allows direct coding for analysis. It is a powerful programme that is commonly used by economists and is especially well-suited to deal with all of the data analysis requirements of an impact evaluation. This document is to be used as a starter guide to help familiarise you with the layout and main functions available in STATA. This starter pack will help you navigate through the main elements required for data analysis in STATA including:

1. Using the 'Help' directory
2. How to load a dataset
3. Opening log and do files
4. The basic commands required for data analysis
5. More sophisticated commands required for treatment-control data analysis

Good Luck!

1- Orientation:



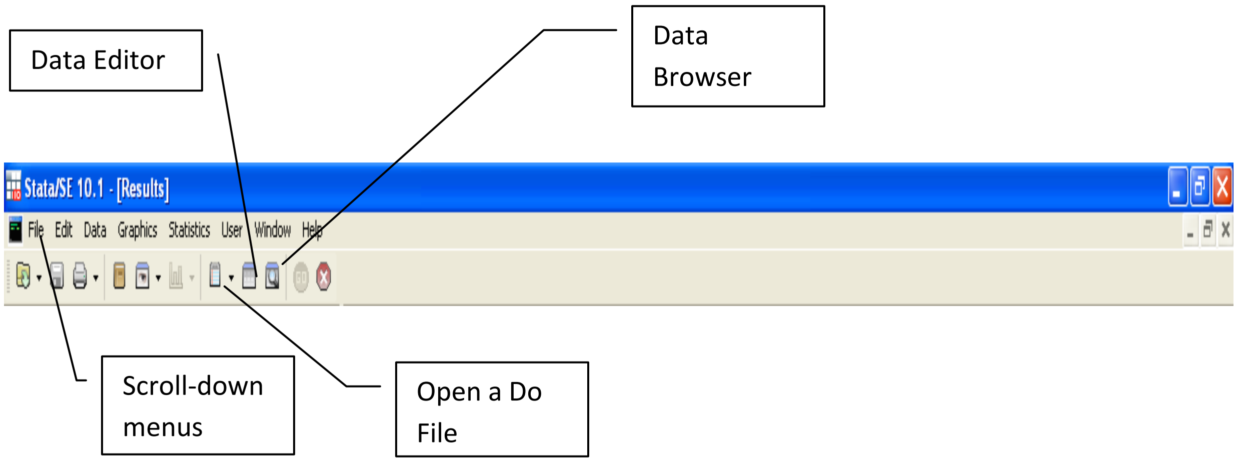
```
                                                   tm
  ___  ____  ____  ____  ____
 /__    /   ____/   /   ____/     10.1   Copyright 1984-2008
___/   /   /___/   /   /___/             StataCorp
  Statistics/Data Analysis               4905 Lakeway Drive
                                         College Station, Texas 77845 USA
     Special Edition                     800-STATA-PC        http://www.stata.com
                                         979-696-4600        stata@stata.com
                                         979-696-4601 (fax)

23-user Stata for Windows (network) perpetual license:
       Serial number:  81910514776
         Licensed to:  Manor-Road
                       Oxford University

Notes:
      1.  (/m# option or -set memory-) 10.00 MB allocated to data
      2.  (/v# option or -set maxvar-) 5000 maximum variables
      3.  New update available; type -update all-

.
```

- The variables window shows you what variables are in the database. When a dataset is loaded onto STATA you will find the list of variable names and descriptions in this window
- The command window is where you type commands into STATA in order to obtain an output
- The results window is where any results from the analysis requested from the command window shows up
- The review window keeps a history of all the commands you have made.


Data Editor
Data Browser
Stata/SE 10.1 - [Results]
File Edit Data Graphics Statistics User Window Help
Scroll-down menus
Open a Do File

- Scroll-down menus provide all ‘point and click’ options for many commands in STATA
- Data Browser allows you to look at the dataset loaded onto STATA, but in this mode no changes can be made to any cells/variables
- Data Editor allows one to view and change the data in the dataset
- Opening a Do File opens a text file that can be used to enter an entire set of STATA commands in one go.

2- Getting Help:

STATA has a built-in help directory to provide support when you know the name of a command but need to find out more information about how to use this. In order to access help, type in the command window:

help *command*

For instance, if you want to know what the command *summarize* is used for, type in:

help summarize

If, on the other hand, you know what you want to do but do not know the name of the command to use, you can click on the scroll-down menus option ‘help’ in the top left hand corner and choose ‘search’.

Try searching for a command that will help describe the dataset and see what results come up.

3- Opening a Data File:

First you need to tell STATA which folder you want to look in for your data file (in this case evaluation.dta). You can do this by assigning the file path to the name 'data' as follows:

gl data = "C:\Documents and Settings\DoH\Desktop\STATA workshop\evaluation_dta" (or the appropriate file path in which your data file has been saved).

now, whenever we use the command $data, STATA will know that you are referring to this file path.

So, to open our database, we need to enter the command:

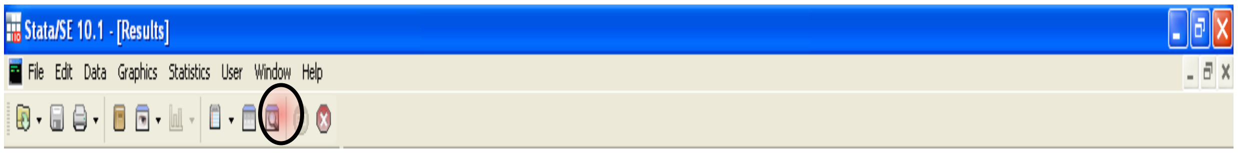
use "$data\evaluation.dta"

The dataset should be opened and the variables window in the bottom left hand corner should now be populated with the variable names in the dataset.

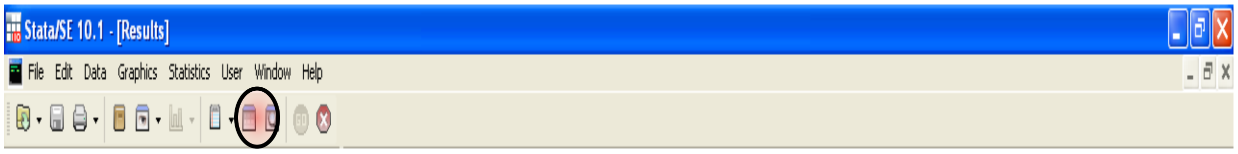
4- Exploring the data:

Now that we have loaded up a dataset, the first thing we want to be able to do is get an overview of the data.

The variable names are in the bottom left variables window, but in order to see the actual data, we need to type into the command window '**browse**', or click on the icon in the top left corner of a table with a magnifying glass called 'data browser'.


Stata/SE 10.1 - [Results]
File Edit Data Graphics Statistics User Window Help

In this mode we cannot change any of the values inside the dataset. If we want to edit the dataset (change a cell value or column name etc.) we must click on the icon labelled 'data editor' (picture of a table with no magnifying glass), or enter the command '**edit**' in the command window.


Stata/SE 10.1 - [Results]
File Edit Data Graphics Statistics User Window Help

Now the dataset appears and, by highlighting a cell, we are able to change any values we like.

Take note that nothing can be written in the command window while the data table is open for viewing or editing. You must close the table before resuming with commands.

Since many datasets contain thousands, or millions of entries, it doesn't make sense to go through it line by line. We can get a much better picture of all of the data by summarising the information in one table.

Firstly, if we want to get information on all of the variables included in the dataset, we use the command '**describe**'. Type this in the command window to see the details of each variable.

If we want to get descriptive information about the actual information contained in the dataset (not just variable names) we can use the command '**summarize**' to help us do this.

Try typing in the following:

summarize

A table of information describing the number of observations, mean, standard deviation, minimum and maximum values should appear.

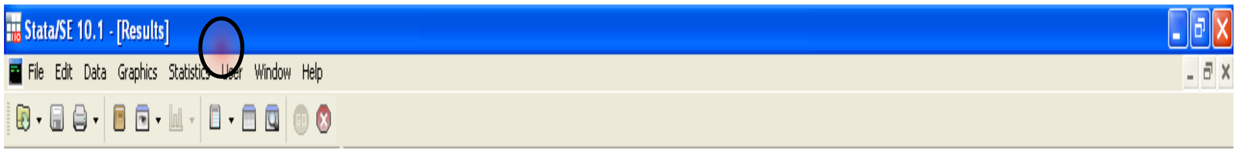
If we want to focus in on a particular set of variables we can restrict this and type the following:

summarize ophe round

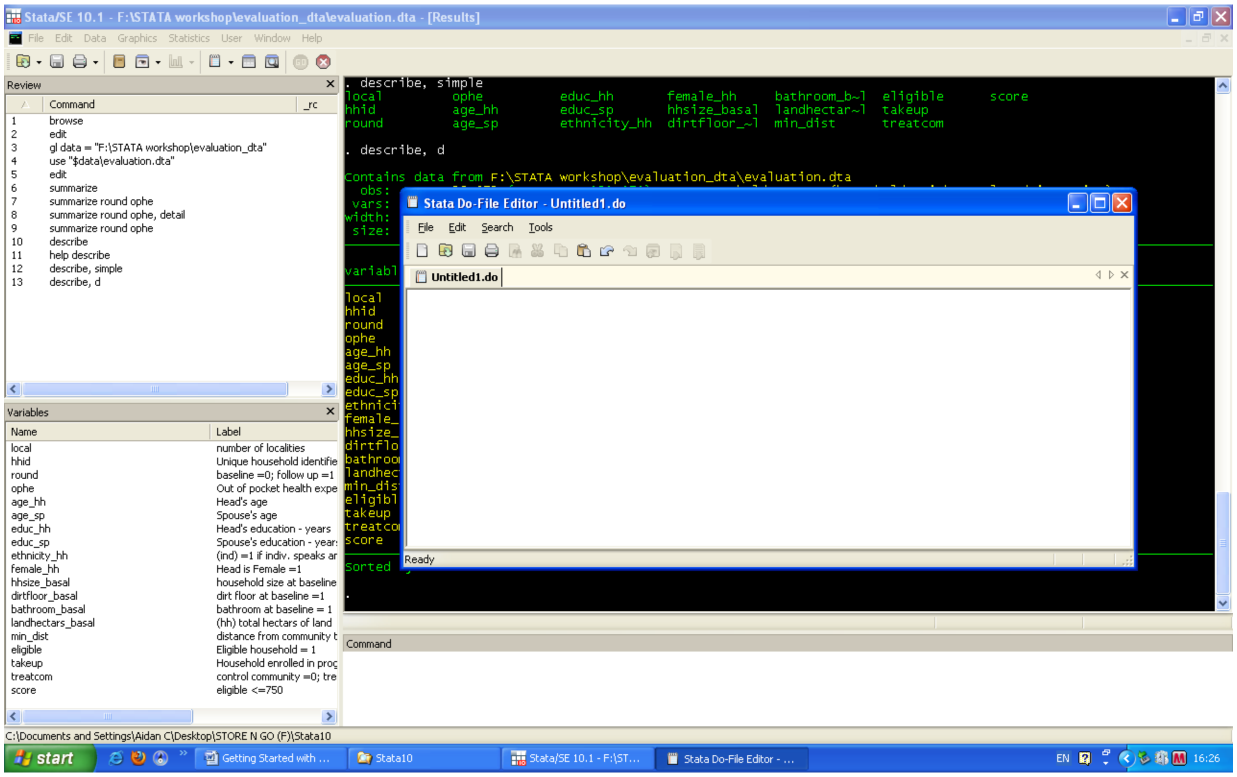
(we can replace ophe and round with any variables we are interested in).

5- Using a Do File:

Rather than entering one command at a time, it is best practice to open a text document where you can enter all of the commands that you want to use in one go. This way you always have a record of your activities and if you make a big mistake with the data (such as accidentally deleting some important information) you can always go back to the beginning. To do this we go to the icons in the top left corner and click on the 'New Do-File Editor' icon.


Stata/SE 10.1 - [Results]
File Edit Data Graphics Statistics User Window Help

This will open up a screen that will allow you to enter text.



Stata/SE 10.1 - F:\STATA workshop\evaluation_dta\evaluation.dta - [Results]

Review

	Command	_rc
1	browse	
2	edit	
3	gl data = "F:\STATA workshop\evaluation_dta"	
4	use "$data\evaluation.dta"	
5	edit	
6	summarize	
7	summarize round ophe	
8	summarize round ophe, detail	
9	summarize round ophe	
10	describe	
11	help describe	
12	describe, simple	
13	describe, d	

Variables

Name	Label
local	number of localities
hhid	Unique household identifie
round	baseline =0; follow up =1
ophe	Out of pocket health expe
age_hh	Head's age
age_sp	Spouse's age
educ_hh	Head's education - years
educ_sp	Spouse's education - year:
ethnicity_hh	(ind) =1 if indiv. speaks ar
female_hh	Head is Female =1
hhsize_basal	household size at baseline
dirtfloor_basal	dirt floor at baseline =1
bathroom_basal	bathroom at baseline = 1
landhectars_basal	(hh) total hectars of land
min_dist	distance from community t
eligible	Eligible household = 1
takeup	Household enrolled in prog
treatcom	control community =0; tre
score	eligible <=750

```
. describe, simple
local          ophe           educ_hh        female_hh      bathroom_b~l   eligible       score
hhid           age_hh         educ_sp        hhsize_basal   landhectar~l   takeup
round          age_sp         ethnicity_hh   dirtfloor_~l   min_dist       treatcom

. describe, d

Contains data from F:\STATA workshop\evaluation_dta\evaluation.dta
  obs:
 vars:
width:
 size:

variabl

local
hhid
round
ophe
age_hh
age_sp
educ_hh
educ_sp
ethnici
female_
hhsize_
dirtflo
bathroo
landhec
min_dis
eligibl
takeup
treatco
score

Sorted

.
```

Stata Do-File Editor - Untitled1.do

Untitled1.do

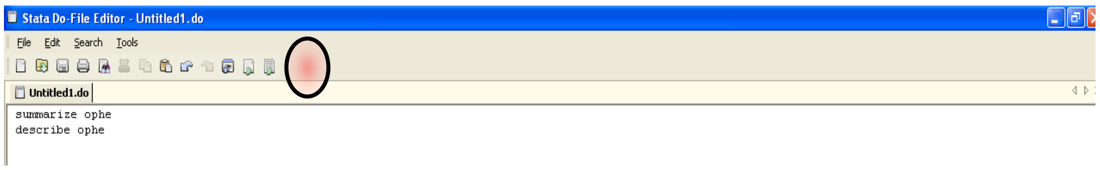
C:\Documents and Settings\Aidan C\Desktop\STORE N GO (F)\Stata10

Try type in the text box:

summarize ophe

describe ophe

Now click on the 'do' icon at near the top and look at your STATA results window:



Stata Do-File Editor - Untitled1.do

Untitled1.do

```
summarize ophe
describe ophe
```

What has happened?

Now, at the beginning of each line, insert a * so that the document reads:

***summarize ophe**

***describe ophe**

What has happened now?

Whenever we place a * at the beginning of the line, STATA will ignore this. This is useful if you want to give certain sections headings etc.

For instance, we could write:

***Summary for the variable on out of pocket expenditures (ophe)**

summarize ophe

describe ophe

A useful command when you write do-files is **#delimit**. It allows you to decide whether the end of a line should be the end of your command, or whether a semi-colon should mark the end of your command. When you write long commands, you may want to be able to write it on several lines, without Stata taking each line for one command. In this case, you should write:

#delimit ;

Then you should end each command with a semi-colon. To go back to end of line=end of command, you should use **#delimit cr**. So we could write:

```
#delimit ;
summarize ophe;
describe ophe;
describe
ophe;                    (see it doesn't matter if you write your command on two lines here)
#delimit cr
summarize ophe
describe ophe
```

6- Keeping a Log File:

While a do file keeps a record of all of your commands so that you can run the exact same analysis at any time, a log file keeps a record of all of the output created as a result of the commands used. In essence, it stores your entire statistical analysis, from which you can use for any reports you wish to produce from the data.

To make sure that we keep a log file, we first need to determine which file path the log will be saved. For ease of use we can use the same file path that the data was stored in:

Eg. "C:\Documents and Settings\DoH\Desktop\STATA workshop\evaluation_dta"

But remember, we have previously called this file path 'data', so, rather than typing out the whole file path again, we can follow the shortcut:

log using "$data\myfirstlog.txt"

Type this into your do file so that you should have the following:

capture log close (closes any log file currently open)

log using "$data\myfirstlog.txt", replace t (replace t makes sure that the log file overwrites any previous version of the log file named 'myfirstlog.txt'.

***Summary for the variable on out of pocket expenditures (ophe)**

summarize ophe

describe ophe

capture log close (closes current log file)

So what has this done?

Seemingly nothing, but now go to the directory in which you have your data and you should find a text file by the name 'mylog'. Open it up and see what's there...

Always remember to use the command **capture log close** before and after using the log file to make sure that you always close the log file for use.

7- More STATA Commands:

So now that you have the basic tools to start analysing data in STATA, here is a list of useful commands that will help with doing some real number crunching (and point you in the right direction for completing the worksheet). Remember, you can always use the **help** command to explain more detail about each command:

7-1- GET STATA COMMANDS

ssc install Name_of_Stata_command: installs a specific stata command in Stata when it is not installed already. Some commands have been developed by Stata users and are not directly integrated in the Stata package. This command allows you to get those commands For example, if we wanted to use the outreg2 command (see below), we would have to install it ourselves by typing: **ssc install outreg2**.

7-2- MACROS

gl label1 = "...": assigns whatever is between the inverted commas "..." to a given label.

global label1 variable1 variable2 variable3...: Assigns all of the variables requested to a given label or name. In this case, rather than writing out all of the variables (1,2,3...) every time, we can just use the shortcut 'label1'. For example, if we want to summarise variable1, variable2 and variable3 we could write this as: **summarize variable1 variable2 variable3** or we could use the shortcut **summarize $label1.** Please note that to 'call' the global macro in our do-file, we must use a $ sign before the name of the macro.

local label1 variable2 variable2 variable3…: the local command does the exact same thing as the global command. There are two main differences though. First, when you 'call' the local macro in your do-file, you will not use the $ sign, but specific quotes. If we want to summarize variable1, variable2 and variable3, we will write: **summarize `label1'** – note the two quotes are different. Second, if you have defined a global macro in a do-file, you can still call this macro after your do-file has been executed, Stata will recognize it. This is not the case for local macros: they are called local macros because they only work locally, in the current do-file execution. Once your do-file has been executed, Stata does not remember what `label1' was.

7-3- SUMMARIES

summarize variable1 variable2: provides a summary of all variables requested

summarize variable1 variable2, detail: provides a summary of all variables requested with additional statistics

browse: opens the data table for inspection

edit: opens the data table for editing

describe: gives the names and descriptions of the variables in the dataset

tabulate variable1 variable2: cross-tabulates the variables requested

7-4- MANIPULATION OF DATA

keep/drop ...: keeps (or drops) only the observations that have been specified (see logical operators next to help explain this)

if, &(and)**, |**(or)**, ==** (equal to)**, >** (greater than)**, <** (less than)**:** these are logical operators that can be used in conjunction with many different commands. For instance, with the **keep** command, we may want only to keep observations in our dataset if they are from round 1. In this case we would write the command:

keep if round == 1

We may want to be more specific: we only want to keep observations that are from round 1 *and* if the poverty index (score) is less than 600. We can then write:

keep if round == 1 & score < 600

Alternatively, if we wanted to keep all observations if they were from round 1 *or* they have a score of less than 600, we can do the following:

keep if round == 1 | score < 600

So, in STATA: *and* = &, *or* = |

+, -, *, /: Usual mathematical operators that can be used to add, subtract, multiply and divide observations in the dataset

generate variableX: creates a new variable called variableX. This could be a combination of other variables in the dataset. For instance, if we wanted variableX to be the multiplication of variableY and variableZ we could write the following code: **generate variableX = variableY*variableZ.**

replace variableX = ...: If variableX already exists and we want to change some of the entries for this variable, we can do so with the **replace** command. For instance, if we now wanted all observations for variableX to be the sum of variableY and variableZ, we could use: **replace variableX=variableY+variableZ**

```
foreach x in element1 element2 element3 {
        Whatever we want to do to each element, referred to as `x’ with the specific quotes
}
```

This command allows us to manipulate the data in a systematic way: the command loops over each element of the list, here it loops over element1, then element2, then element3, and does whatever command we write between the braces. Note that the opening brace should be at the end of the foreach line, and the closing brace should be on a separate line after all the commands of the loop. Most of the time, we use two variations of this loop command, which, instead of looping over any element in a given loop, loops over a list of variables, or over numeric values:

```
foreach var in varlist variable1 variable2 variable3 {
        Whatever we want to do to each variable, referred to as `var’ with the specific quotes
}
```

In this command, Stata loops over each variable indicated between varlist and the opening brace. E.g. say we have four variables, *slept_untreat_bednet*, *slept_treat_bednet*, *went_ANC* and *weighted_at_birth*, which are all binary variables equal to 1 standing for Yes or 2 for No. If we want all these variables to equal 1 if Yes, and 0 if No, we can use a loop instead of writing four times the same command:

```
foreach var in varlist slept_untreat_bednet slept_treat_bednet went_ANC weighted_at_birth {
        replace `var’=0 if `var’==2
}
```

Stata will pick each variable one after the other and replace it by 0 when the variable equals 2.

```
forvalues val=x(y)z {
        Whatever we want to do with each value, referred to as `val’ with the specific quotes
}
```

This command loops over numeric values, from x to z, by steps of y. E.g., say we have a *group_code* variable equal to 1 for treatment group 1, to 2 for treatment group 2, to 3 for treatment group 3, and to 4 for the control group. We would like to create a *treatment_1* variable, equal to 1 when *group_code* equals 1, and 0 otherwise. We would also like to create a *treatment_2* variable, equal to 1 when *group_code* equals 2, 0 otherwise. In fact, we would like to do it for the two remaining treatment groups as well: we want to create *treatment_3* equal to 1 when *group_code* equals 3, 0 otherwise, and *treatment_4*, equal to 1 when *group_code* equals 4, and 0 otherwise. It is tedious to write each command for each variable, and as we'll program more, we may have to do this kind of repetitive task for more complicated commands. Therefore, the use of loops is extremely timesaving. For this example, we will write:

```
forvalues number=1(1)4 {
        generate treatment_`number'=group_code==`number'
        replace treatment_`number'=. If group_code==.
}
```

Note that the name of our numeric values could be anything: we could have written

```
forvalues blablabla=1(1)4 {
        generate treatment_`blablabla'=group_code==`blablabla'
        replace treatment_`blablabla'=. If group_code==.
}
```

What matters is that inside the loop, we refer to the appropriate name we picked in the forvalues command, with the appropriate quotes. It is the case for the foreach command as well.

7-5- REGRESSIONS

regress variable1 variable2 variable3: runs a linear regression, where variable1 is the dependent (left-hand-side) variable and variable2, variable3 are the independent/explanatory (right-hand-side) variables

probit variable1 variable2 variable3..: this is a probit regression command where variable1 is the dependent (binary) variable and the other variables are explanatory. For example, if variable1 was participation in a programme (yes or no), variable2 was age and variable3 was income, this probit regression would compute the probability that an individual would participate in the programme based on his/her age and income level.

predict newvariable: This stores the predicted results from the previous regression analysis in a new variable for later use.

7-6- PLOTS

scatter variable1 variable2: Produces a scatter plot of the requested variables

twoway: allows the overlaying of multiple graphs.

kdensity variable1: calculates and graphs the kernel density of a variable. This is to say that the estimated distribution (eg. Bell curve shape for heights of children etc.) of the variable is computed and plotted.

hist variable1: Produces a histogram showing the distribution of a single variable.

7-7- TESTS OF DIFFERENCE BETWEEN MEANS

7-7-1- Mean tests between two groups

ttest outcome_var, by (group_var): conducts a difference in 2 means test, comparing the means of 2 groups of outcome_var, choosing the 2 groups based on group_var. Eg. If we wanted to see if there was a difference in the average height of a child in the control and treatment groups, and we know that *height* is the variable for child height and *treatment* is the binary (1 or 0) value for whether or not the child was given the treatment, we can compare the control and treatment groups by using the command: **ttest height, by (treatment)**. This will tell us if there is a significant difference in the average height of children between the treatment and control groups.

7-7-2- Mean tests between more than two groups

regress outcome_var group_var1 group_var2 group_var3 group_var4: conducts a test of difference in mean between more than 2 groups (here 4, but this could be any number of groups). E.g., say we have four different treatment groups (treatment 1, treatment 2, treatment 3 and control), which are defined by 4 different binary 0-1 variables: *treatment_1*, *treatment_2*, *treatment_3*, and *treatment_4*. If we want to see if there is a difference in the average height of a child between the four groups, and we know that *height* is the variable for child height, we can compare the four groups by using the command: **regress height treatment_1 treatment_2 treatment_3 treatment_4**. This will tell us if there is a significant difference in the mean height of children between the four groups through the F-test.

Note that since each observation of the dataset should be allocated to one of the four treatment groups, Stata will drop one of the groups in the regression (because of perfect colinearity between the group variables). It does not matter which variable is dropped for the result of the F-test. For example, you can choose which variable should be dropped yourself: try **regress height treatment_1 treatment_2 treatment_3**, or **regress height treatment_1 treatment_2 treatment_4**: you will see the F-stat does not change.

However, if there is a significant difference between the four groups, this regression will not tell us which groups significantly differ from each other. To know that, we want to perform mean tests between each of the four groups. This can be done with the 2 sequential commands:

regress outcome_var group_var1 group_var2 group_var3 group_var4
lincom group_var1-group_var3: calculates the difference in outcome_var obtained through the linear combination of group_var1 - group_var3, and displays the P-value of the T-test of difference between the means of group_var1 and group_var3.

This command could be applied to any treatment group. E.g., if we consider the example above, say we want to know the difference in children's height between treatment groups 2 and 4 and if the two mean heights are statistically different from each other, we will type the two commands sequentially:

regress height treatment_1 treatment_2 treatment_3 treatment_4
lincom treatment_2 - treatment_4

We will obtain the T-stat of the difference in means test between treatment groups 2 and 4. Please note that to be able to use lincom, we cannot drop the treatment group of our choice in the regress command, but should include all of the four treatment groups.

If we want to perform several mean tests with lincom, we will also have to run the regression again right before using each lincom command.

An alternative to **lincom** exists and makes outputting results easier: in our previous example, we could have used **lincomest** instead of **lincom** after **regress**. The advantage of this command is that it turns the results of lincom into estimates: with **lincomest**, Stata performs the same **lincom** command, but in addition stores the results of the command, making it possible to use and export the results of the mean tests. Our example would be:

```
regress height treatment_1 treatment_2 treatment_3 treatment_4
lincomest treatment_2 - treatment_4
```

7-8- MATRICES

matrix define A=(a,b,c\d,e,f): defines a 2x3 matrix A containing a, b and c in the first line, d, e and f in the second line. The main matrix operators are comma to add an element on the same line (i.e. create a new column), and backslash to add elements on a line below (i.e. create a new line). See **help matrix** for more details and matrix operators and matrix functions.

el(A, i, j): matrix function which takes the i, j element of matrix A (element located on line i, column j). This matrix function is to be used in a command, but is not a command per se.

e(b) and e(V): these are not Stata commands, but two system matrices obtained after a regression. After certain commands, Stata stores the results of these commands in what is called **return** or **ereturn** (see help return and help ereturn). After a regression, Stata stores some of the results in memory in ereturn. The ereturn results we are usually interested in are e(b) and e(V), two matrices which respectively store the vector of coefficients from the previous estimation (stored in a one-line matrix), and the vector of variance-covariance of these coefficients. To see what is stored in memory after a regression, we can use **ereturn list**.

For example, say we do the regression from our previous example: **regress height treatment_1 treatment_2 treatment_3 treatment_4.** We would like to extract the coefficient of treatment_2 (the second variable in our regression) and put it in a matrix called treatment_2_coeff. To do that, we should use the function el(). We want to pick the element 1,2 (meaning 1st line, 2nd column) of matrix e(b) to get the coefficient of treatment_2. We should do:

```
regress height treatment_1 treatment_2 treatment_3 treatment_4
mat list e(b)                  (just to check how the matrix looks like)
mat define treatment_2_coeff=(el(e(b), 1, 2))
```

mat list treatment_2_coeff
If we want to define a 1-column, 4-line matrix called all_treatment_coeff, containing the coefficients of the four treatment groups , we will write:
regress height treatment_1 treatment_2 treatment_3 treatment_4
mat define all_treatment_coeff=(el(e(b),1,1)\el(e(b),1,2)\el(e(b),1,3)\el(e(b),1,4))
We can list the content of the matrix: **mat list all_treatment_coeff**

7-9- OUTPUTTING COMMANDS

outreg2, using(File_path/Filename): exports estimation results into filename, located at a given file path. This command is very useful used after a regression to export regression results, and allows us to build nice-looking tables right from Stata. See help outreg2 for more details.

makematrix Matrix_name, from(…, …): Stata_command: builds a matrix of given 'return' or 'ereturn' elements indicated in the 'from()' parentheses, based on the results stored after Stata_command. The way makematrix works is similar to loops: Stata runs the command written in Stata_command, for several variables which are mentioned in the Stata command itself or in the lhs() or rhs() options (see help makematrix for more details on how to specify variables). Every time Stata performs Stata_command, it extracts the elements indicated in 'from()', and builds a matrix of these elements. At the end of the one makematrix command, we obtain one general matrix. The columns of the matrix are defined based on the elements indicated in the 'from()' option: each element of the 'from()' option represents a column of the matrix. Each variable we mentioned in the makematrix command represents a row of the matrix. This command is useful if we want to get specific outputs from a regression (or any command where results are stored in return or ereturn), and if more user-friendly commands such as outreg2 do not allow us to get these elements as easily or presented the way we want.

For example, say we would like to regress the three variables *height, weight, age*, representing respectively the height, weight and age of a child, on each treatment group (treatment_1 to treatment_4). For each outcome variable, we would like to take the coefficients of the four treatment groups and get them all on one line in a matrix A. The first row will gather the four coefficients for the first regression of *height*, the second line will gather the four coefficients for the second regression of *weight*, the third line will gather the four coefficients for the third regression of *age*. This is how we can do it in only one command:
makematrix A, from(el(e(b),1,1) el(e(b),1,2) el(e(b),1,3) el(e(b),1,4)) lhs(height weight age) listwise: regress treatment_1 treatment_2 treatment_3 treatment_4
Here, we are first telling Stata we want to run the regression **regress X treatment_1 treatment_2 treatment_3 treatment_4**, with X being the left-hand-side variable specified in the lhs() option: Stata will take each of the *height, weight* and *age* variables from the lhs() option one after the other, and run the regression (Stata is basically looping over *height, weight* and *age*). I have specified the option listwise, so that Stata takes the biggest number of observations possible for each regression.
Out of each regression, we are asking Stata to take 4 elements based on the results stored in ereturn: el(e(b),1,1), el(e(b),1,2), el(e(b),1,3) and el(e(b),1,4), and put them in one line in the matrix A. Stata will

repeat this action for each variable *height, weight* and *age*. In the end, we will obtain a 3-line, 4-column matrix. The row names will be height, weight and age, and the column names will be el(e(b),1,1), el(e(b),1,2), el(e(b),1,3) and el(e(b),1,4). This is something you could modify using the label option in the makematrix command, or using the **matrix rownames** and **matrix colnames** commands.

mat2txt, matrix(Matrix_name) saving(File_path/Filename): export matrix Matrix_name into Filename located at File_path.

outsheet using File_path/Filename: export data in memory to Filename located at File_path. The reverse command allowing us to import data from a given Filename to Stata is **insheet using File_path/Filename**. These commands are useful when we want to export Stata datasets into text files, or import data stored in a text file.