What is Stata?

UK Data Service

Author: UK Data Service
Updated: November 2015
Version: 8

Contents

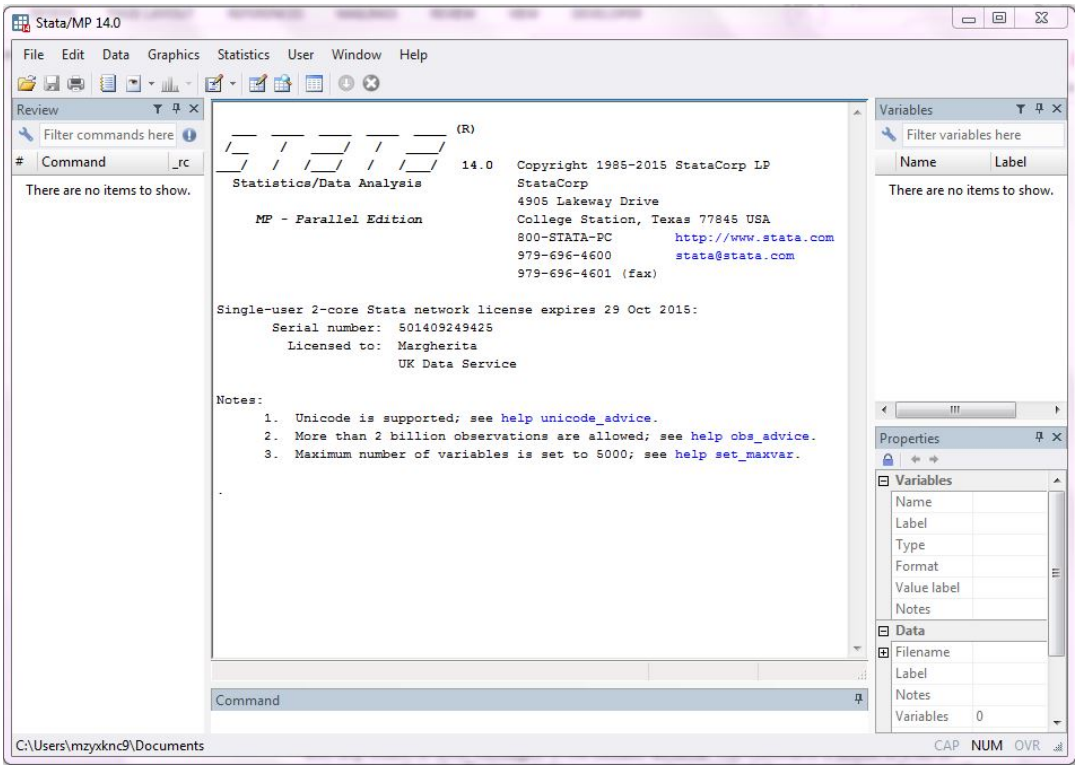
1. What is Stata 12.1?

Stata 12.1 is a statistical software package for data analysis. You can use Stata by pointing and clicking, or by using the command syntax. Stata can be used to manipulate, analyse, and produce graphics of your data. Stata can support complex analysis, and, as it is so programmable, developers and users continue to add new features.

2. How to view data in Stata

The dataset used in the following examples is the Labour Force Survey, October-December 2012: Teaching Dataset. The Labour Force Survey asks people aged 16 and over living in the UK about their working lives. The teaching dataset is a cut-down version of the full dataset. This dataset can be downloaded from the UK Data Service website, after completing a short registration.

When you open Stata 14, you will see the following screen:



The **Command window** is where you type your commands, which are then displayed, along with any results or error messages, in the **Results window**. The command is added to a list of past commands in **the Review window**, so you can keep track of the commands you have used. The **Variables window** contains a list of variables in your data file, and the **Properties window** shows the properties of the individual variables.

2.1. Setting memory size

Stata improves processing speed by holding data in memory while processing, rather than accessing data from a hard drive. Stata 14 has automatic memory adjustment, meaning that the size of the memory allocated to Stata by your computer adjusts automatically to account for the

size of the datasets used. However, in older versions of Stata, you will have to set the size of the memory manually.

In all version of Stata prior to 12.1, the default memory allocated is roughly one megabyte. For most datasets, it will probably be necessary to increase this allocation. A memory size of 16 MB or slightly higher will be enough for most purposes. To set the memory at 20 MB, type the following into the **Command window**, and hit the return key.

Notice that the command appears in **the Review window**, and Stata's response to your command is given in the **Results window**

If you want to set the memory at 20 MB permanently, use the following syntax:

```
. set memory 20m, permanently
```

This means that each time you open Stata, a memory of 20 MB will already be assigned. Notice that a comma separates the option from the command. This is a general syntax feature for specifying options in Stata commands. Note also that Stata allows for shortening of commands, and the above command can also be expressed as:

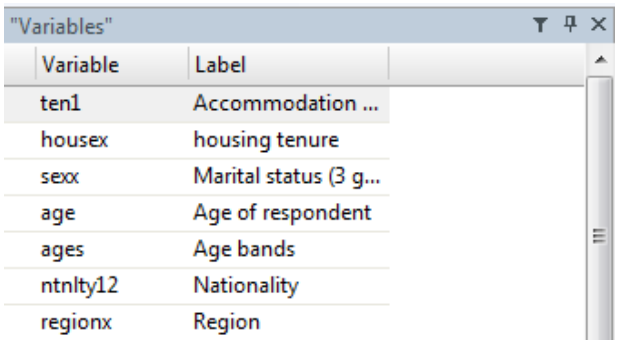
```
. set mem 20m, perm
```

2.2. Opening a file

To open your dataset by pointing and clicking, click on the Open button:

Navigate to the location of your data, click on the filename, and then click Open.

The command will save to the **Review window** as before, and the **Variables window** should now contain a list of your variables:



The syntax to open a file is as follows:

```
. use <filename>, clear
```

The **clear** command at the end of the instruction means that any data that you currently have stored in memory will be erased.

Throughout this guide, words in italics in parentheses such as <filename> indicate where a specific name of a file or variable needs to be added. Underlined portions of command words indicate abbreviations, which can be used instead of typing out the full commands.

2.3. Changing the default directory

You can also alter the directory from which Stata loads and saves data by using the cd 'change directory' command. For example, if you were to type in the following:

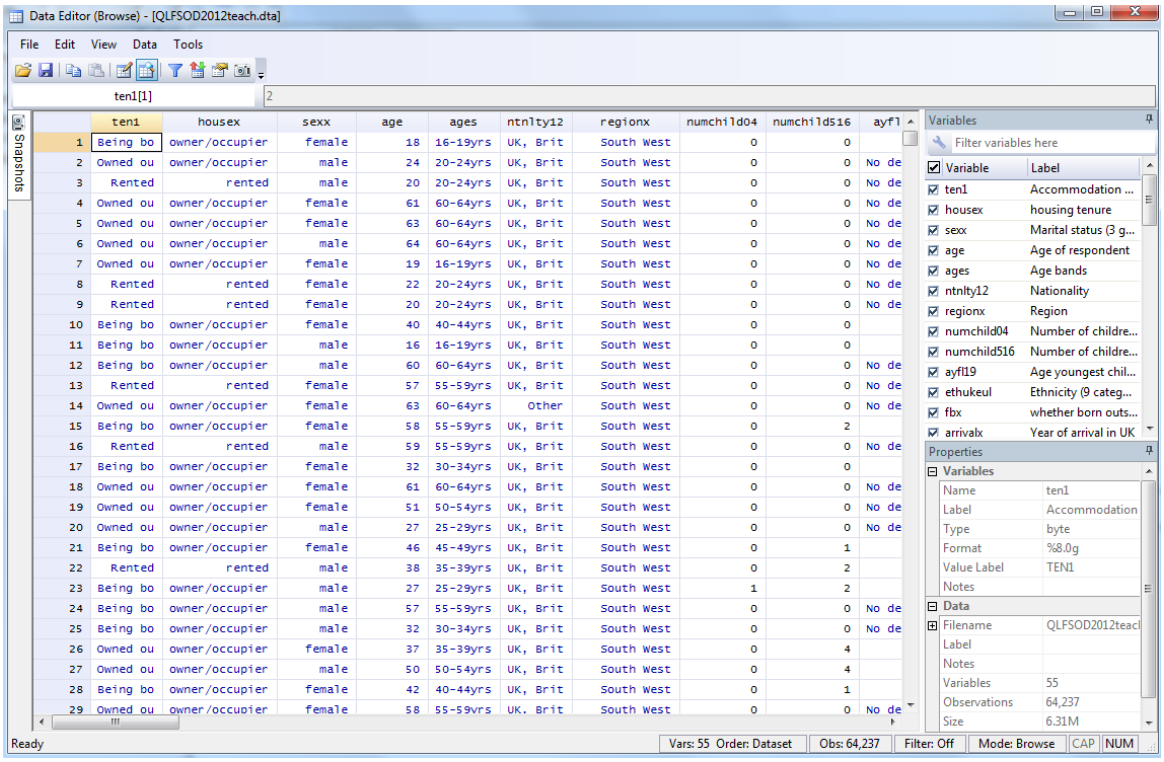
```
. cd C:\Data_Stata
```

all load and save options would now operate to and from this specified location. This means you can simply enter the **use <filename>** command without specifying the file directory each time.

2.4. Opening the data browser

Unlike some other statistical packages, the data in Stata is not immediately visible upon opening a data file. To view the raw data, you will have to open the **Data Editor (Browse)** window. This window allows you to view the data, but not to change its values.

- Each column represents a variable in the survey, and is labelled with the name of that variable. This is often a response to a question or derived from answers to a question or several questions. Double-clicking the name of the variable at the top of the column will bring up the properties of that variable in the **Properties window**, including the label and type.
- Each row represents an individual respondent, this might be a person (as in the Labour Force Survey), or a household, or family unit, or other unit. These rows are often referred to as *cases* (or *observations*). Each row is numbered.



Variables in Stata can be stored as numeric or string. Numeric variables would be anything with a numeric response - scale variables such as people's age, or ranked or unranked categorical variables. String variables contain text.

Use **the Data Editor (Edit)** button to view the data and make changes to values.

Stata will not run commands when the **Data Editor (Browse)** or **(Edit)** are open.

Typing raw data into Stata

To type raw data into Stata, first clear any current data, by typing:

```
. clear
```

Then go to the **Data Editor (Edit)** and type in the data. By default, the variables are called var1 var2 var3 var4 and so forth. To rename and label each variable individually, type:

```
. rename var1 = id

. label var id "identification no."
```

To rename all the variables using one instruction, use **renvars:**

```
. renvars v*\id sex age income
```

It is good practice to **compress** the data within Stata before saving to reduce the amount of memory required to hold the dataset. Type:

```
. compress
```

2.5. The lookfor and describe commands

A convenient way to search variables for names or descriptions is the **lookfor** command. For example, if you were interested in the number of hours each respondent works, in the **Command window**, you could type:

```
. lookfor hours
```

This will search all variable names or descriptions in the dataset for the expression 'hours'. This command is especially useful with very large datasets.

```
. lookfor hours

              storage   display    value
variable name   type    format     label      variable label
------------------------------------------------------------------------------
ptimehrs        byte    %8.0g      ptimehrs   Whether part-time (work <31 hours per week)
ttushr          byte    %8.0g      ttushr     Total usual hours in main job
```

Another way to look at what variables are in a dataset it to use the **describe** command. In the **Command window**, type

```
. describe
```

Typing **describe** alone without any variable names brings up information on all variables within a dataset.

```
. describe

Contains data from C:\Users\mzyxknc9\OneDrive - JISC\Margherita JISC\USUT\Datasets
  obs:        64,237
 vars:            55
 size:     6,616,411
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
ten1            byte    %8.0g      ten1       Accommodation details
housex          byte    %8.0g      housex     housing tenure
sexx            byte    %8.0g      sexx       Sex
age             byte    %8.0g      age        Age of respondent
ages            byte    %8.0g      ages       Age bands
ntnlty12        int     %8.0g      ntnlty12   Nationality
regionx         byte    %8.0g      regionx    Region
numchild04      byte    %8.0g                 Number of children aged 0-4
numchild516     byte    %8.0g                 Number of children aged 5-16
```

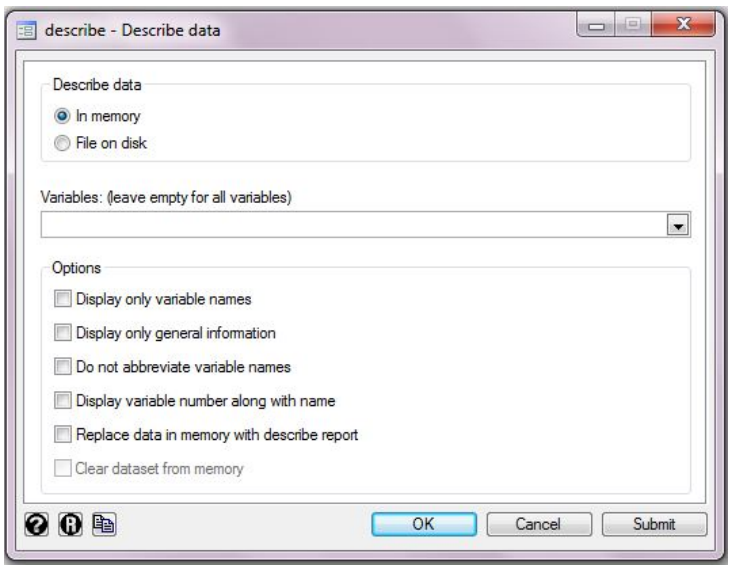
Otherwise, you can type variable names after the **describe** command to obtain details for a subset or single variable.

```
. describe numchild04
```

The underlining of the first two letters of the **describe** command indicates that this command can be shortened to these two letters. So writing the following will also perform the **describe** command:

```
. de numchild04
```

If you prefer, you can navigate to this same information using the menu system. ***Data > Describe data > Describe data in memory or in a file***. From here, you can either select a variable from the dropdown menu, or leave blank to describe all variables. Click *OK*.



Using the 'help' and 'findit' functions

You can find out about the format and function of a command by using the **help** command. For example, you could use this to find information out about the **describe** command:

```
. help describe
```

You also can search for a command if you do not know its full name using the **findit** instruction:

```
. findit describe
```

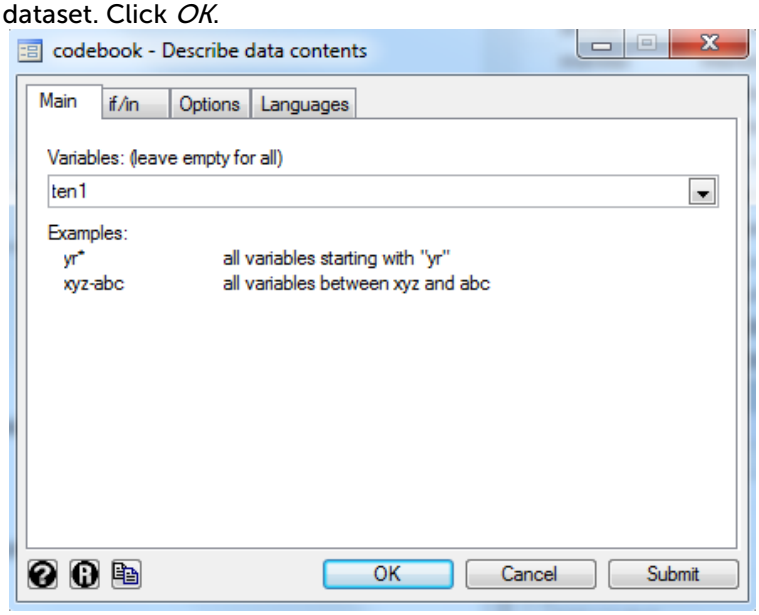
To supplement this guide, you may find it useful to use these commands to find out further information about each command.

2.6. The **codebook** command

The **codebook** command can be used to obtain more detailed information about the values of specific variables.

```
. codebook ten1
```

As before, you can obtain this same information by navigating through the menu system – ***Data > Describe data > Describe data contents (codebook)***. Again, you can select a particular variable using the dropdown menu, or leave blank to return information on all variable in your dataset. Click *OK*.



```
ten1                                                    Accommodation details

            type:  numeric (byte)
           label:  ten1

           range:  [-8,5]                       units:  1
    unique values:  6                       missing .:  0/64,237

      tabulation:  Freq.   Numeric  Label
                      38        -8  No answer
                  14,557         1  Owned outright
                  28,539         2  Being bought with mortgage or
                                    loan
                     308         3  Part rent
                  20,339         4  Rented
                     456         5  Rent free
```

2.7. Saving your data under a new name

If you are planning to make any changes to your dataset, such as recoding variables, it is a good idea to save a copy of your dataset under a different filename so that you have a copy of the data in its original format.

```
. save <filename>
```

As with the **use** command, if you do not specify the directory prior to the filename (and have not changed directory using the **cd** command) this will save into the default directory (c:\data). You can also navigate to save the file, using ***File > Save as...***
If you are happy to overwrite your file with any changes you have made, type

```
. save, replace
```

3. Exploratory analysis in Stata

The dataset used in the following examples is the Labour Force Survey, October-December 2012: Teaching Dataset. The Labour Force Survey asks people aged 16 and over living in the UK about their working lives. The teaching dataset is a cut-down version of the full dataset. This dataset can be downloaded from the UK Data Service website, after completing a short registration.

3.1. Creating a log of your output

It is recommended to make a log of your output. The default setting for Stata is for the output to be temporary, meaning that the results history in the output window is limited to a particular size. Creating a log file allows you to create a permanent record for future reference which can be printed out.

You can start logging your results by clicking on the log button and saving your log under your chosen name. Log files are saved with the extension .smcl

Alternatively, you can open a log file by typing

```
. log using <filename>
```

substituting *<filename>* for the name you wish to give to your log file. When using the **log** command, the options **replace** *<filename>* or **append** *<filename>* are available, giving you the option to replace an existing log of the same file name with a new one, or to start appending a log from the bottom of an already existing file as denoted by *<filename>*.

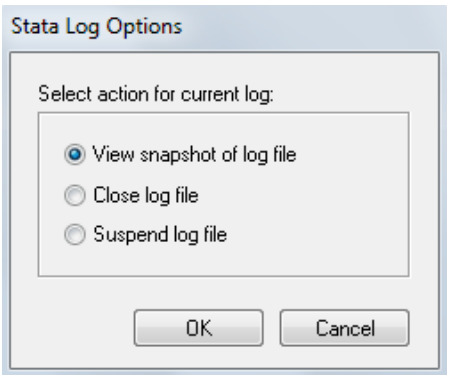
Logs can be suspended, restarted, or closed by typing

```
. log off

. log on

. log close
```

or by clicking on the log button to open the following **Stata Log Options** box:



3.2. Creating a one-way frequency table

The **tabulate** command can be used to create one-way frequency tables.

```
. tabulate sexx
```

As underlining the first two letters of tabulate indicates that it can shortened to those two letters, the above command can also be expressed as:

```
. ta sexx
```

This gives the following output, giving the frequency of males and females in the first column, with the percentage in the second column, and the final column showing the cumulative percentage:

```
. tabulate sexx
```

Sex	Freq.	Percent	Cum.
male	30,413	47.34	47.34
female	33,824	52.66	100.00
Total	64,237	100.00	

You can use the **codebook** command to see which numerical values are connected to the labels 'male' and 'female'. It is important to understand the values underpinning the variable labels when you recode variables. Alternatively, you can add the option **nolabel**:

```
. tabulate sexx, nolabel
```

This runs the one-way frequency table without the labels, so you can see the underlying numerical values:

```
. tabulate sexx, nolabel
```

Sex	Freq.	Percent	Cum.
0	30,413	47.34	47.34
1	33,824	52.66	100.00
Total	64,237	100.00	

Note that in Stata options like *nolabel* are separated from the rest of the command using a comma. If you do not use the comma, Stata will misinterpret *nolabel* as a variable and return an error message.

The tab1 command

You can create one-way frequency tables for multiple variables in a single command by using **tab1**. The following creates three frequency tables:

```
. tab1 sexx marcivx fbx
```

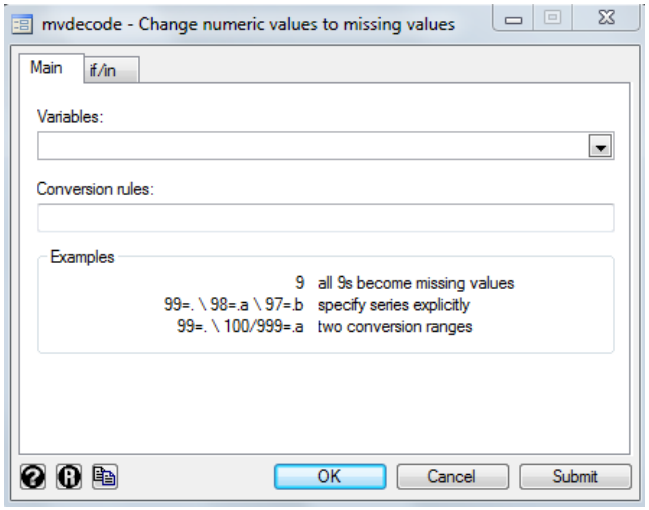
Type **help tab1** for more information.

3.3. What are missing values?

It may be preferable to perform your analyses to produce valid percentages; a percentage that excludes certain responses that are not of interest, such as 'unknown' responses. You can do this by setting these responses to missing. When you receive data from the UK Data Service or other sources, missing values may not be coded into Stata recognised missing values. They may take on a negative number, or some other format. This means that it is up to you to recode missing values, and to decide how missing data will be handled within you analysis. More detailed information about handling missing data can be found at the Missing Data website.

In the Labour Force Survey, October-December 2012: Teaching Dataset, 'No answer' and 'Does not apply' responses have not been set to missing in Stata.

You can set 'No answer' and 'Does not apply' responses to missing by selecting ***Data > Create or change data > Other variable-transformation commands > Change numeric values to missing***. This opens a window box titled **mvdecode – Change numeric values to missing**. Select your variable/s from the dropdown menu. The conversion rules can be specified according to your needs. For example, entering '9' in the conversion rules will set all values of 9 to missing. If you wish to specify different missing values so that you can differentiate between them, you can do this here by following the examples given by Stata in the **mvdecode** window.



Setting missing values can also be achieved through syntax, using the command **mvdecode**.
. mvdecode <variable>, mv(<value>)

So, for example, the missing values of the variable *jobtyp* "Permanent or temporary job" are coded '-8' and '-9' for the responses 'No answer' and 'Does not apply' accordingly. To set them to missing, you can type
. mvdecode jobtyp, mv(-8, -9)

Note that this syntax codes all the values to '.' to denote that they are missing. If you want to set separate missing values for each response, you can type
. mvdecode jobtyp, mv(-8=.a \ -9=.b)

Some datasets have common values, such as minus values, to denote missing values. If there are common values across the dataset that you wish to set to missing, you could type
. mvdecode _all, mv(-8=.a \ -9=.b)

To unset any missing values, you can use **mvencode**
. mvencode _all, mv(.a=-8 \ .b=-9)

Values set to missing only show up in analysis in Stata if you specifically request them. We can see from the **codebook** command that there are no missing values for the variable 'Sex' (sexx).

If we look at another variable, we can see this isn't always the case.

```
. codebook jobtyp
```

```
jobtyp                                                  Permanent or temporary job

                  type:  numeric (byte)
                 label:  jobtyp

                 range:  [1,2]                        units:  1
         unique values:  2                    missing .:  25,664/64,237

            tabulation:  Freq.   Numeric  Label
                        36,168         1  Permanent
                         2,405         2  Not permanent in some way
                        25,664         .

.
```

If we run a one-way frequency of this variable, we can see that the values for .a and .b are missing from the analysis. This is because Stata treats any value preceded by '.' as missing. To see these values included in the frequency, you can add the option **missing**:

```
. tabulate jobtyp, missing
```

Permanent or temporary job	Freq.	Percent	Cum.
Permanent	36,168	56.30	56.30
Not permanent in some way	2,405	3.74	60.05
.a	25,643	39.92	99.97
.b	21	0.03	100.00
Total	64,237	100.00	

3.4. Comparing two variables

Cross-tabulations show the relationship between two (or more) categorical variables. For example, you may be interested in marital status by sex. When creating cross-tabs, it is possible to specify a number of statistics as options. These include row percentages (**row**), column percentages (**col**), and whether missing value frequencies are contained in the table (**missing**).

```
. tab marcivx sexx, r m
```

Whether married/Civil Partner (living with or separated)	Sex male	female	Total
Not married or in civ	11,653	11,543	23,196
	50.24	49.76	100.00
Married/Civil partner	18,760	22,281	41,041
	45.71	54.29	100.00
Total	30,413	33,824	64,237
	47.34	52.66	100.00

The **tab2** command creates all possible crosstabs from a number of variables specified in the command. If you were interested in marital status by gender and whether the respondent was born outside the UK, typing the following syntax would produce crosstabs of marital status by whether born outside UK, marital status by sex, and whether born outside the UK by sex, with row percentages and missing values included:

```
. tab2 marcivx fbx sexx, r m
```

```
-> tabulation of marcivx by fbx
```

Whether married/Civil Partner (living with or separated)	whether born outside the UK No answer	no	yes	Total
Not married or in civ	20,544	2,627	25	23,196
	88.57	11.33	0.11	100.00
Married/Civil partner	34,609	6,387	45	41,041
	84.33	15.56	0.11	100.00
Total	55,153	9,014	70	64,237
	85.86	14.03	0.11	100.00

```
-> tabulation of marcivx by sexx
```

Whether married/Civil Partner (living with or separated)	Sex male	female	Total
Not married or in civ	11,653	11,543	23,196
	50.24	49.76	100.00
Married/Civil partner	18,760	22,281	41,041
	45.71	54.29	100.00
Total	30,413	33,824	64,237
	47.34	52.66	100.00

```
-> tabulation of fbx by sexx
```

whether born outside the UK	Sex male	female	Total
No answer	26,304	28,849	55,153
	47.69	52.31	100.00
no	4,075	4,939	9,014
	45.21	54.79	100.00
yes	34	36	70
	48.57	51.43	100.00
Total	30,413	33,824	64,237
	47.34	52.66	100.00

To examine the level of association between sex and marital status using a chi-square test, type:

```
. tab2 marcivx sexx, chi2
```

```
-> tabulation of marcivx by sexx
```

Whether married/Civil Partner (living with or separated)	Sex male	female	Total
Not married or in civ	11,653	11,543	23,196
Married/Civil partner	18,760	22,281	41,041
Total	30,413	33,824	64,237

```
          Pearson chi2(1) = 121.8147   Pr = 0.000
```

To create a three-way cross-tabulation of sex, marital status and whether born in the UK, use the **table** command:

```
. table marcivx fbx sexx
```

Type **help table** for more details.

3.5. Creating summary statistics

The **summarize** command can be used to create summary statistics for continuous variables such as *hourpay* "gross hourly pay":

```
. sum hourpay
```

Variable	Obs	Mean	Std. Dev.	Min	Max
hourpay	64,237	-5.33279	9.958271	-9	932.5

You can add the option **detail** for a wider range of summary statistics.

```
. sum hourpay, detail
```

Gross hourly pay

	Percentiles	Smallest		
1%	-9	-9		
5%	-9	-9		
10%	-9	-9	Obs	64,237
25%	-9	-9	Sum of Wgt.	64,237
50%	-9		Mean	-5.33279
		Largest	Std. Dev.	9.958271
75%	-9	259.28		
90%	9.2	265.4	Variance	99.16716
95%	14.79	320.51	Skewness	16.38635
99%	27.53	932.5	Kurtosis	1270.904

Note that the **summarize** command ignores any missing data in your variable.

It is also possible to combine the functions for **tabulate** and **summarize** to create tables of summary statistics for sub-groups. For example, to summarise women's hourly pay:

```
. sum hourpay if sexx==1, detail
```

Note that commands testing for equality require a double equals sign. If you want to say that a variable is NOT equal to a particular value, use "~=".

You can see by the number of observations that some cases (in this instance all men) are no longer included in the summary statistics.

Gross hourly pay

	Percentiles	Smallest		
1%	-9	-9		
5%	-9	-9		
10%	-9	-9	Obs	33,824
25%	-9	-9	Sum of Wgt.	33,824
50%	-9		Mean	-5.588356
		Largest	Std. Dev.	8.272273
75%	-9	80.13		
90%	8.41	90.8	Variance	68.43049
95%	13.31	92.3	Skewness	2.636868
99%	23.34	105.8	Kurtosis	11.73068

The "~=" operator is one of many which can be used in Stata to make conditional statements in commands. The table below gives some other commonly used operators:

Arithmetic		Logical		Relational (numeric and string)	
+	addition	~	not	>	greater than
-	subtraction	!	not	<	less than
*	multiplication	|	or	>=	> or equal

/	division	&	and	==	equal
^	power			~=	not equal
				!=	not equal
+	string concatenation				
Note that a double equal sign (==) is used for equality testing					

Does your variable make sense?

Note that there are some very small values in the gross hourly pay variable (0.47 is 47 pence per hour!). You might want to investigate how this variable was created to understand why some of the values are so low. A few unfeasibly low values may be due to coding errors in data entry. Otherwise, you should consult the documentation that comes with all datasets (such as the User Guide) to find out how the variable was created. In a fuller analysis, you may have to make decisions on how to 'clean' errors, outliers, and improbable values on continuous variables such as income.

You can also combine the functions **tabulate** and **summarize** to create tables of summary statistics for sub-groups. To obtain the mean hourly pay for men and women by whether they are living as a couple or not, type

```
. ta marcivx sexx if (statusx==1& hourpay>0), su(hourpay) means
```

The option **means** specifies that we only require the mean values for each category.

Means of Gross hourly pay

Whether married/Civil Partner (living with or separated)	Sex male	female	Total
Not marri	11.567124	10.33204	10.909788
Married/C	17.19179	12.580982	14.759209
Total	15.401034	11.855858	13.525392

Married people could get higher pay than non-married people for both men and women because non-married people may be younger, have less work experience and seniority, and so thus be paid less. To consider whether the relationship between marital status and income differs between different age groups (given by the variable ages), use the **bysort**: command:

```
. bysort ages: tabulate marcivx sexx if (statusx==1&hourpay>0), su(hourpay) means
```

Here are the first three tabulations of results:

-> ages = 16-19yrs

Means of Gross hourly pay

Whether married/Civil Partner (living with or separated)	Sex male	female	Total
Not marri	5.4423226	5.5854348	5.52
Total	5.4423226	5.5854348	5.52

-> ages = 20-24yrs

Means of Gross hourly pay

Whether married/Civil Partner (living with or separated)	Sex male	female	Total
Not marri	8.4877273	7.9824866	8.2106598
Married/C	12.145833	8.6618519	9.7338462
Total	8.6249063	8.0282294	8.2930513

-> ages = 25-29yrs

Means of Gross hourly pay

Whether married/Civil Partner (living with or separated)	Sex male	female	Total
Not marri	11.310758	10.370728	10.822271
Married/C	12.192985	10.676536	11.325751
Total	11.565539	10.472854	10.97986

The table command

Note that if you wish to present your results with fewer decimal places you must use the **table** command and **format ()** option instead of **tabulate**:

```
. table marcivx sexx if (statusx==1&hourpay>0), c(mean hourpay) format(%9.2f)
```

Whether married/Civil Partner (living with or separated)	Sex	
	male	female
Not married or in civil partnership	11.57	10.33
Married/Civil partner(living with and se	17.19	12.58

Type **help table** for more details.

3.6. Using weights in Stata

Most Stata commands can deal with weighted data. To do a cross tabulation of sex by marital status using the weight *pwt11*, type:

```
. ta sexx marcivx [fweight=pwt11]
```

Sex	Whether married/Civil Partner (living with or separated)		Total
	Not marri	Married/C	
male	8,772,886	11,665,551	20,438,437
female	7,704,447	12,916,397	20,620,844
Total	16,477,333	24,581,948	41,059,281

You can abbreviate the weight type in the command. So the following gives the same result as above:

```
. ta sexx marcivx [fw=pwt11]
```

Grossing weights

Note that the numbers in the weighted cross tabulation are much larger than in the unweighted cross tabulation. This is because the weight *pwt11* is a weight that grosses up to the population size. Bear in mind that although the figures in the weighted cross tabulation look large, they are based on a much smaller sample of the population. Grossed-up figures like this are often quoted to the nearest thousand.

Stata allows four kinds of weights:

- fweights (frequency weights)
- pweights (sampling weights)
- aweights (analytic weights)

- iweights (importance weights)

Type **help weight** for more information about these different types of weight.

Commands in Stata each have a "natural" weight type that Stata associates with that analysis. If you type 'w' in place of the weight type in the command, Stata will display the type of weight it assumes and the results. Thus typing:

```
. ta sexx marcivx [w=pwt11]
```

gives the following output:

```
. ta sexx marcivx [w=pwt11]
(frequency weights assumed)
```

Sex	Whether married/Civil Partner (living with or separated) Not marri	Married/C	Total
male	8,772,886	11,665,551	20,438,437
female	7,704,447	12,916,397	20,620,844
Total	16,477,333	24,581,948	41,059,281

For more information about weights in Stata, type:

```
. help weight
```

4. Data manipulation in Stata

The dataset used in the following examples is the Labour Force Survey, October-December 2012: Teaching Dataset. The Labour Force Survey asks people aged 16 and over living in the UK about their working lives. The teaching dataset is a cut-down version of the full dataset. This dataset can be downloaded from the UK Data Service website, after completing a short registration.

4.1. Creating variables using the **generate** command

The **generate** command allows you to create new variables. To generate a new variable called sex1 based on an existing variable sexx, you would type:

```
. generate sex1=sexx
```

The **label variable** command provides a label for your variable:

```
. label variable sex1 "sex of respondent"
```

Label define and **label values** define the coding for your new variable. **Label define** establishes a set of labels for a set of values, and **label values** attaches labels defined using **label define** to the values of a specified variable:

```
. label define sexlabel 0 male  1 female

. label values sex1 sexlabel
```

Note that labels created through the **label define** option are stored independently to variables. This means that one label can be assigned to a number of different variables. For example, you can create a new variable *sex2* and reuse *sexlabel*:

```
. gen sex2=sexx

. label values sex2 sexlabel
```

Renaming variables

If you wish to change the name of a variable, you can use the **rename** command:

```
. rename sex2 sex3
```

Using **rename** does not alter the variable and value labels associated with a variable.

Reserved words and system names in Stata

There are a number of *reserved words* and *system names* that have specific meanings in Stata. They should not be used as variable names when you create your variables.
System names in Stata are reserved for specific operations. For example:

_n	running number of the current observation
_N	total number of observations
_all	all variables

Reserved words have specific meanings in Stata commands and include:

double	long	float	byte	if	using

For full lists and more information about reserved words and system names, type:

```
. help reswords

. help _variables
```

Generate also allows you to create a variable with values based on the mathematical transformation of another variable. To create a variable for age squared, type:

```
. gen agesquared=age^2
```

The superscript is indicated by a "^" symbol.
To create a variable *weekpay* for the weekly pay (based on a 35 hour week and hourly pay), type:

```
. gen weekpay=hourpay*35
```

Another common use for the **generate** command is to create indicator or 'dummy variables'. These are typically binary variables (holding valid values of 1 or 0) which can be used to enter categorical variables into statistical models.

To create a variable *age40* that takes the value 1 if the respondent is 40 years or older, and the value 0 if the respondent is under 40, use the **generate** command alone or in combination with its accompanying **conditional** or **if** options. Here are these two alternative ways of doing this:

```
. gen age40_a=age >=40 if age ~=.
```

or alternatively:

```
. gen age40_b=cond(age >=40, 1,0) if age ~=.
```

To check that these 2 commands have given identical results, type:

```
. tab age40_a age40_b
```

4.2. Using the replace command to change the values of existing variables

The **replace** command can also be used as a way of creating dummy variables:

```
. gen age40=1 if age >=40

. replace age40=0 if age40 ~=1 & age ~=.
```

The commands **label define** and **label values** can also be used in conjunction with **replace** to assign labels to your new values.

```
. label define AGEG40 1 “Aged 40 or older” 0 “Less than 40”

. label values age40 AGEG40
```

Avoid accidentally changing missing values

System missing values in Stata (“.”, “.a” etc.) are coded as a number higher than the numerical value of all other numerical values. Care must consequently be taken when using the >= “greater or equal to” or > “greater than” operators. For example, if missing data was present in your analysis and you typed the following:

```
. replace varx=1 if varz>4000
```

Stata would assess missing values to be greater than 4000 and code them as 1.
To avoid this, you would type:

```
. replace varx =1 if varz > 4000 & varz ~= .
```

Note that this applies to all uses of ‘greater than’ or ‘greater or equal to’ operators and not just when using the **replace** command.

4.3. Recoding variables

Recode is used to collapse the number of categories in a variable.

Look at the **codebook** and survey documentation before recoding to understand how the variable is currently coded or the range of values it currently takes, and whether there are any missing or inapplicable cases which need to be accounted for.

Do not alter original variables – create ‘replica’ variables

Instead of altering original variables, create ‘replica’ variables identical in their values to original variables and perform any recoding on these variables. The advantage of this is that you will keep the integrity of the original values of the variables in your dataset. The original variables can then provide a check to your recoding. They can also be indispensable to correcting otherwise potentially irreversible changes that result from your data manipulation.

To create a dummy variable white to indicate white versus non-white ethnicity using **recode**, type:

```
. codebook ethukeul

. gen white = ethukeul

. recode white 1=1 2/9=0 *=.
```

To label the variable and its coding and to test whether the recode worked as expected, type:

```
. label variable white "White versus non-white"
```

```
. label define WHITE 1 "White" 0 "Non-white"

. label values white WHITE

. tab eth white
```

The label name may be in capitals or lower case letters.

The symbols / and * in recode

Stata understands / to mean 'through' (6/10 in the present context thus means 6 through 10). The symbol * in the context of recoding instructions means 'remaining' or 'all others'.

To create an *agegroup* variable based on the variable age, first create a replica variable and recode the replica variable into 6 categories:

```
. codebook age

. gen agegroup=age

. recode agegroup min/19=1 20/29=2 30/39=3 40/49=4 50/59=5 60/65=6 *=.
```

To label the variable and its coding and to test whether the recode worked as expected, type:

```
. label variable agegroup "age in 10-year groups"

. label define AGEGGROUP 1 "16-19yrs" 2 "20-29yrs" 3 "30-39yrs" 4 “40-49yrs” 5 “50-59yrs” 6 “60-69yrs”

. label values agegroup AGEGGROUP

. tab age agegroup
```

Instead of using **generate** and **recode** as separate commands, you can also use **generate** as an option of **recode**. The name of the new variable is defined in the brackets that follow the **generate** option:-

```
. recode ethukeul 1=1 2/9=0 *=., generate(white)
```

5. Graphics in Stata

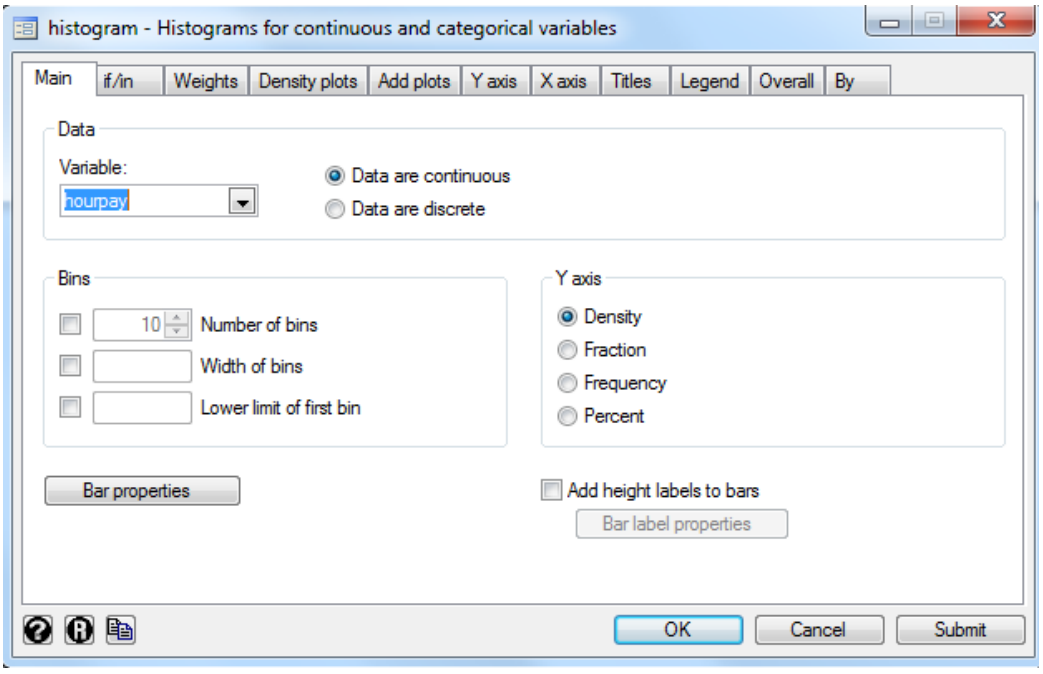
The dataset used in the following examples is the Labour Force Survey, October-December 2012: Teaching Dataset. The Labour Force Survey asks people aged 16 and over living in the UK about their working lives. The teaching dataset is a cut-down version of the full dataset. This dataset can be downloaded from the UK Data Service website, after completing a short registration.

5.1. Producing a histogram

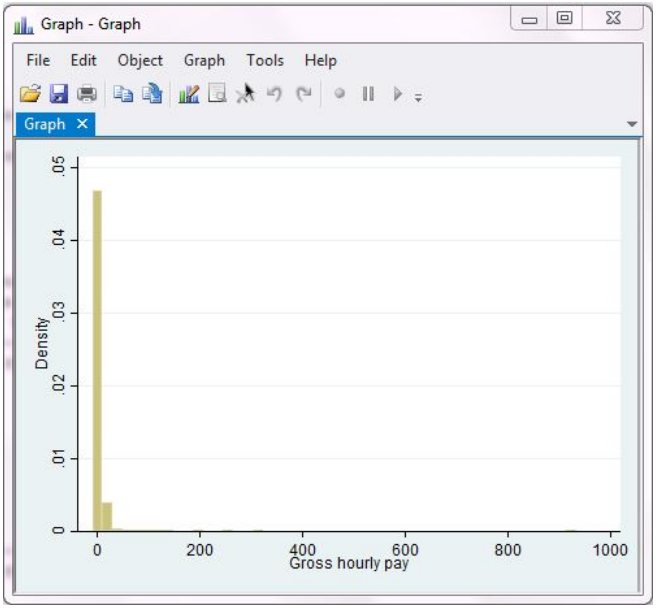
The menu system is a good way to produce graphics because the graphic command structure is different to the usual structure of Stata commands. However, once you are familiar with the structure of graphics commands, you may find it easier to use the command **histogram**.

To create a histogram using the menu system:

- Open the histogram dialogue by choosing **Graphics> Histogram**
- Select *hourpay* using the drop down menu and click OK



The following graph will appear in a new window:



You can use the options in the tabs to alter the graph. If/in allows you to select the range of data to display in your graph (e.g. *hourpay* greater than 0 and less than 200). The *Weights* tab allows you to apply frequency weights to your graph. You can add other plots, alter the appearance of the axes, add titles, subtitles, notes and captions and make other alterations to the appearance of your graph using the tabs.

The following histogram of *hourpay* only includes cases where *hourpay* is greater than 0 and less than 200, is weighted using *pwt11*, displays percentages in the y-axis, has the number of bins selected (at 30) and has a title "Hourly pay (pounds)" and a note about the source of the data and weighting:

To save the graph, right click on it.

Graph formats

To import to a word document, save the graph as a windows metafile format (.wmf). Stata can also export graphs to .png or .tiff formats, which may be preferable when producing results for publication.

To create these graphs using the command **histogram**, type:

```
. histogram hourpay
```

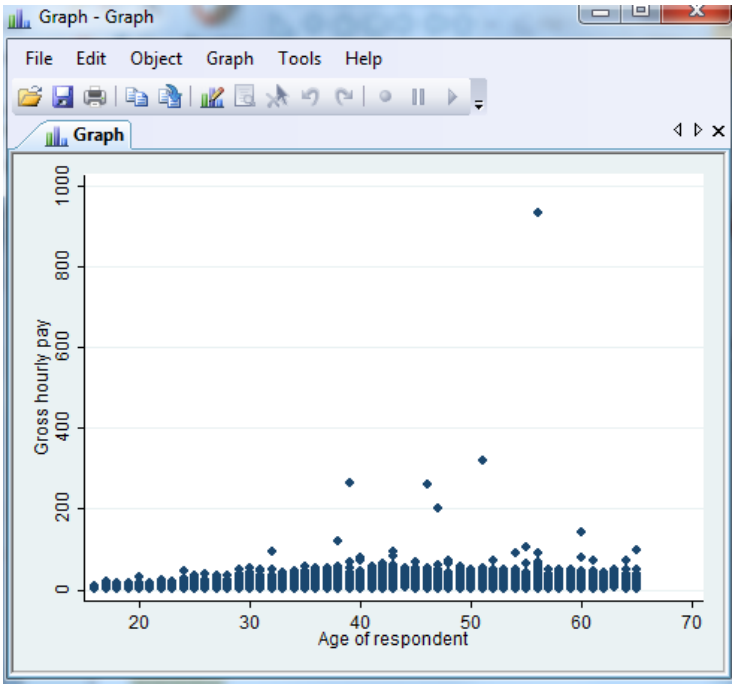
```
. histogram hourpay if hourpay >0 & hourpay <200 [fw=pwt11], bin(30) percent
scheme(s2color) title(Hourly pay (pounds))
```

For more information about the **histogram** command and its options, type:

```
. help histogram
```

5.2. Producing a two-way scatter plot

To produce a two-way scatter plot of men's hourly pay by age using the menu system, select: **Graphics >Twoway graph**. Click on the **Create...** button to select the type of graph. Select scatter from the list of graphs and then select *age* as your x variable and *hourpay* as your y variable in the main tab. The graph appears in a new window:



As with the histogram, you can use options to apply a weight, to limit the procedure to those cases where hourpay >0 and hourpay<=200 and sexx==1 and to give the graph an appropriate title, subtitle and a note about the source of the data and weighting:

To create these graphs using the **twoway** command, type:

```
. twoway scatter hourpay
```

```
. twoway (scatter hourpay age if hourpay>0 & hourpay<=200), title(Women's hourly pay (pounds) by age) subtitle (hourly pay>0 and <= 200 pounds) note("Source: Labour Force Survey, October-December 2012.")
```

Note that the text in the title, subtitle and note need not be enclosed by quotation marks ("") unless it includes punctuation that Stata uses in its command syntax (such as a comma).

For further graphics options, type:

```
. help graph
```

6. Do-files in Stata

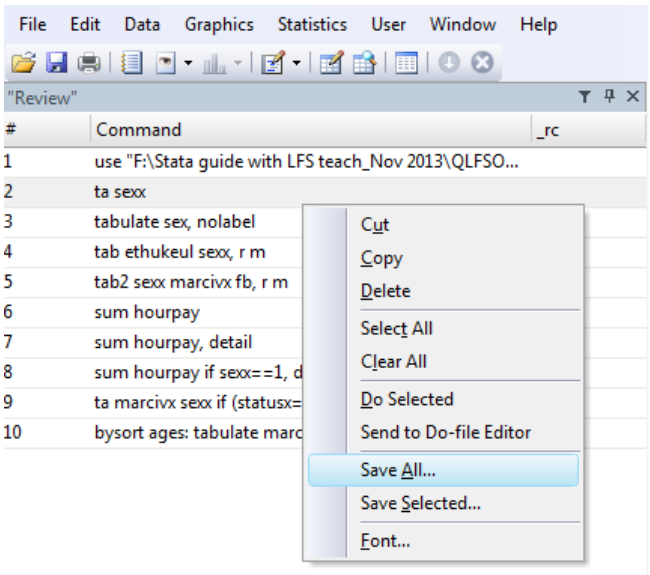
The dataset used in the following examples is the Labour Force Survey, October-December 2012: Teaching Dataset. The Labour Force Survey asks people aged 16 and over living in the UK about their working lives. The teaching dataset is a cut-down version of the full dataset. This dataset can be downloaded from the UK Data Service website, after completing a short registration.

6.1. Saving your commands in a .do file

A do-file is a set of commands that can be saved, edited and used again in the future. Whilst conducting your analysis, it is helpful to produce a do-file that takes you from your original data, through your data manipulation to your complete analysis. These files are saved with a **.do** extension at the end.

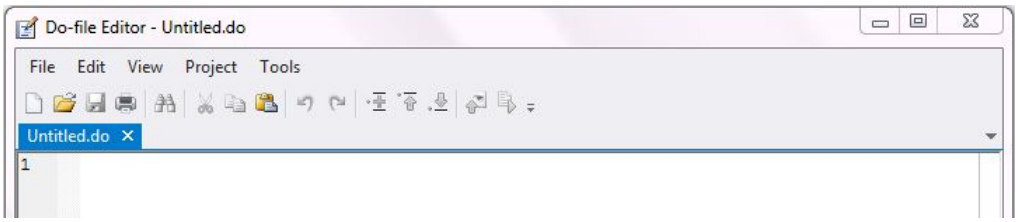
One way you can save the commands you have run is by saving the contents of the **Review window.**

- Click on the **Review** tab to make the **Review window** visible (if it is not already)
- Right click in the window and select **Save All...** in the pop up
- You will then be prompted for a name and location to save the do-file to



However, it is more advisable to establish a do-file prior to running any commands and to run commands directly from this file.
To open the do-file editor, go to the menu at the top of the screen and click on the do-file icon.

The editor should have this menu bar at the top of an otherwise blank screen.



The do-file editor has some familiar commands such as save, cut & paste, print etc. The find and replace commands may be particularly useful for larger variable recoding tasks.

6.2. Running your commands in a .do file

Type the following commands into the do-file, substituting the name and location of your teaching dataset file for "<filename>", and click on  to run them:

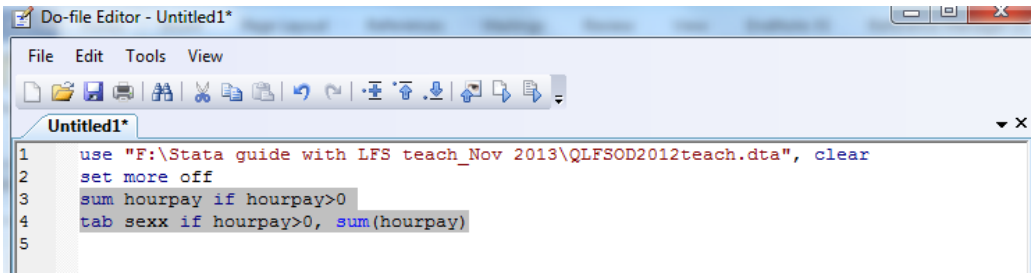
```
use <filename>, clear
set more off
sum hourpay if hourpay>0
tab sexx if hourpay>0, sum(hourpay)
```

The command **set more off** stops Stata from pausing for a prompt.

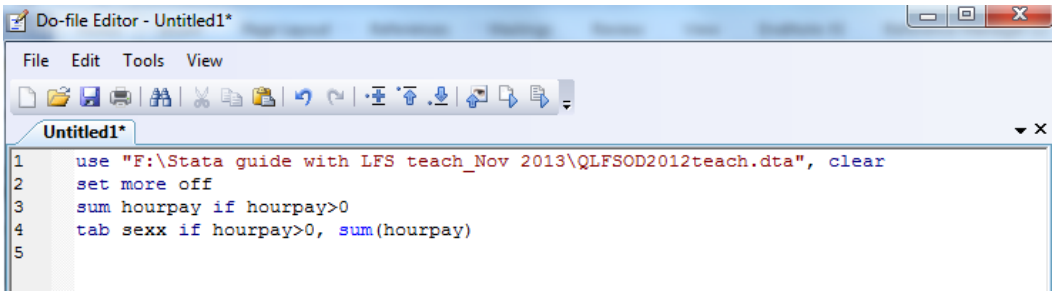
Minimise the do-file editor by clicking on  near the top right corner. You will see in the main results window that all the commands in your do-file have been executed. The end of the do-file is denoted by:

```
end of do-file
```

You can run only some of the commands in your do-file by selecting the commands you want to run and pressing the run button.

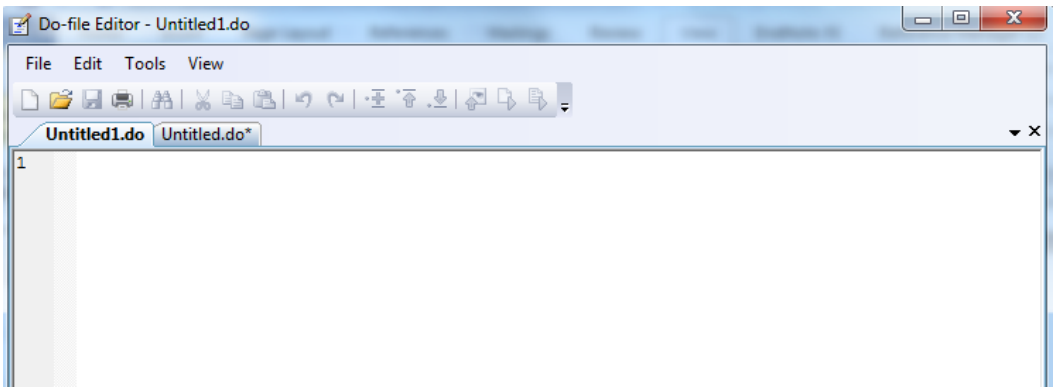


The colours in the do-file editor help you to observe at glance the kinds of comments and commands that you have in your do-file.



Use **Edit> Preferences...** to change the do-file editor colour scheme.

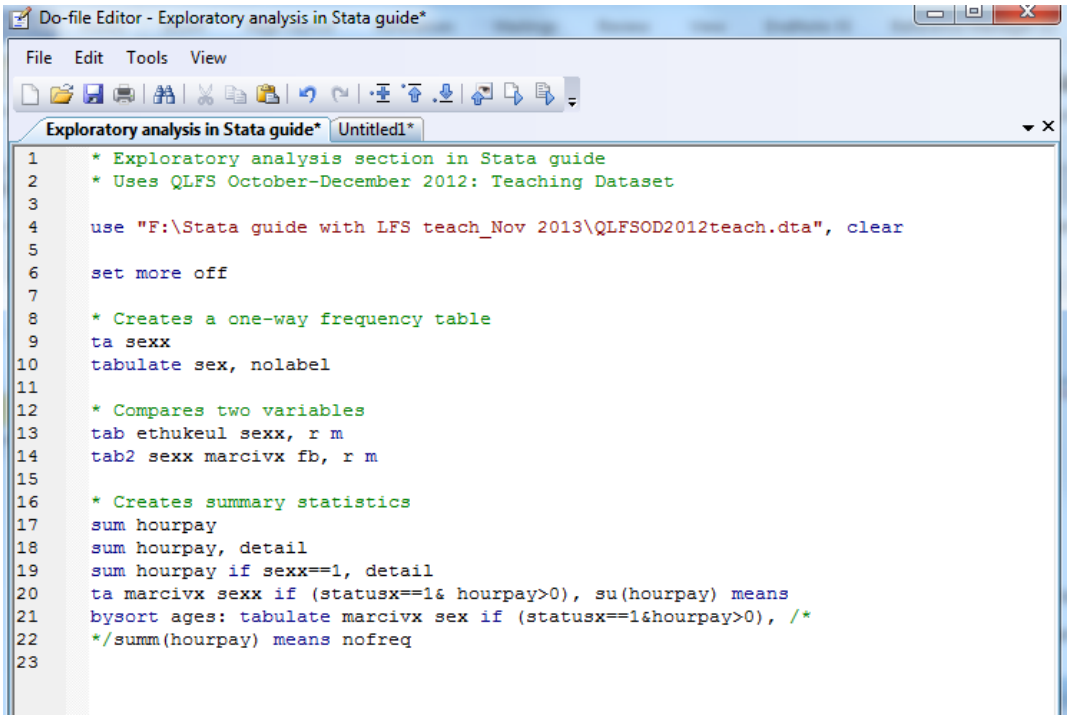
You can open up multiple do-files in one do-file editor. These are represented as different tabs along the top of the editor:



When you open existing or new do files, these will appear as extra tabs along the tab bar.

6.3. Adding notes and writing long commands in a .do file

You can add notes or comments to your commands to explain what you have done and why. Comments are preceded by two forward slashes // or by an asterisk *. For long commands, you can use /* at the end of one line, and */ at the beginning of the next to tell Stata that the text on different lines is part of the same command. Comments and spaces will make your do-file easier to follow both for yourself and others:



Note that the final command is split over two lines using /* and */ so that the whole command appears on the screen. This do-file has been saved as 'Exploratory analysis in Stata guide' (the title can be seen in the tab at the top) so that it can be rerun or changed at a later date.

7. Using hierarchical data in Stata

The dataset used in the following examples is the English Housing Survey 2011-2012: Household Data (EHS). This survey asked over 13,000 households about their housing and local environment. This dataset can be downloaded from the UK Data Service website, after completing a short registration.

Hierarchical datasets consist of data at more than one level of measurement where lower level data are nested in one or more higher levels. For example, in a survey in which individuals within households are interviewed, both individual and household variables may available. Examples of individual variables might include sex, age, level of education and ethnicity, and examples of household level variables might include region, tenure and number of people in household. A household identifier is used to link individuals to households.

Hierarchical data are stored either as a single file containing variables at multiple levels (e.g. an individual level file containing individual and household variables) or in a multiple file format with one (or more) file at each level of measurement (e.g. an individual level file and a separate household level file). The information contained in the two formats is identical and Sections 7.1 and 7.4 of this chapter in effect show how to move between the two formats.

The examples in this guide use two level hierarchical data from the EHS as an example but the methods shown also apply to more complex hierarchical data. For more information about hierarchical data, see the guide *'What are hierarchical files?'*

7.1. Selecting one individual per household using *if* or **keep**

The file ***people.dta*** from the ***interview*** folder in the EHS contains individual and household variables contained in a single dataset (e.g. *sex* at the individual level and *dvhhsize,* household size, at the household level) and contains a household identifier *aacode*. Analyses conducted using this data file are at the individual level by default.

If you want to use these data to do household level analyses, you could select one person in each household[1] by selecting only the Household Representative Person (HRP) in each household.

Household Representative Person (HRP)
The *Household Reference Person* (HRP) is a term commonly used in survey data to indicate a member of the household who is equivalent to a head of household. Where there are joint householders, the HRP is defined by the Office for National Statistics (who produce many of the UK large-scale surveys) as the individual with the highest income. If the joint householders have the same income, the eldest is selected. The HRP can be used in analysis to 'represent' the household in terms of income, say, or to select a single member of each household.

In ***people.dta***, the variable *hrp* is coded as following:

1 = person is the HRP
0 = person is not the HRP

[1] Note that the EHS contains household level datasets that are more appropriate for household analysis.

To select the HRP only, add an 'if' qualifier to your command. The following shows a command to create a crosstab of *sex* by legal marital status *xmarsta* and column percentages for the HRP only:

```
. tab sex xmarsta if hrp==1, col
```

Because we have selected only one case per household, the resulting table can be interpreted as a household level analysis. The table shows not only the column percentages, but also counts for cells and margins.

You should check that the counts in the table match the number of households expected in the file. This information can be obtained from previous exploratory analysis, or preferably, exploratory analysis confirming information in the dataset's documentation.

Instead of using *if* in your command, you can drop cases by selecting only the HRP in each household. Save the dataset under a new name so as not to overwrite the original dataset. Check that this has worked as intended by doing a frequency of the variable *hrp*.
Type:

```
. keep if hrp==1
. save <newfilename>
. tab hrp
```

HRP variables
The way in which the HRP's identity is stored varies from dataset to dataset. For example:

- You may have a binary variable that indicates whether the respondent is the HRP or not.

- You may have a variable which indicates the person number of the HRP. You will then need to test whether the respondent's own person number is the same as that of the HRP.

- You may have a variable that indicates the respondent's relationship to the HRP.

See the documentation that comes with your dataset to find a variable that identifies the HRP or some other variable that will allow you to select one individual per household.

7.2. Using **collapse** to create household level summary variables from individual level data – in a new household level file

Suppose you have individual and household variables in a single individual level file and you wish to create a new summary household-level variable (e.g. the age of the oldest person in the household) using individual level data. The example below shows a simple example of computing the age of the oldest person. It assumes that you have a household ID and age variable.

Individual data	Summary variable age_max in household
Person 1 Household 1: age = 47	age_max = 47
Person 2 Household 1: age = 43	
Person 3 Household 1: age = 18	
Person 1 Household 2: age = 74	age_max = 74

Use the individual level dataset **people.dta**. First sort the data by the household ID variable (aacode), then use the **collapse** command to create a new variable *oldest* (the age of the oldest person in the household).

```
. sort aacode
. collapse (max) oldest=age, by(aacode)
. save <newfilename2>
```

These commands will close the file in memory and put the new file in memory in its place. Save your file first if you want to keep any changes.

7.3. Using **egen** to create household level summary variables from individual level data – in the original individual level file

The **egen** command can be used to produce statistical summaries, *by* subgroups as defined by a variable. In order for this to work you first need to **sort** the data by that variable, so cases with common values are grouped together.

```
. sort aacode
. egen oldest = max(age), by(highed1)
```

The **sort** and **egen** commands may be combined. Type:

```
. bysort highed1: egen oldest = max(age)
```

7.4. Using **merge** to attach household data to an individual level file

Suppose you have hierarchical data held in two files, one with individual level variables and one with household level variables and in which each contains the household ID *aacode*. To attach the household and individual level datasets to create an individual level dataset, first sort all files to be combined by the household ID variable *aacode*, save and close them. Using a household level file from the EHS ***generalfs11.dta*** (in the derived folder when the data are downloaded).

The **merge** command can be used in a range of ways including a one-to-one and match and one-to-many merges. Distributing household level records to individuals within the households does not require additional specification and you do not need to indicate which file is the look up table and which is the lower level file.

To undertake a 'one to many' match you can use a command of the format:

```
. sort aacode
. merge 1:m aacode using <other file>
```

where the <other file> is the file on disk that you wish to merge with the file in memory. It does not matter whether the file in memory is the household or individual file, however you will need to change the 1:m to m:1.

Return to the individual file by reopening *person.dta*. The following assumes that you have done so.

```
. sort aacode
. merge aacode using <householdfile.dta>
```

The '_merge' var indicates whether this has worked and should have values of 3 if data in the file has come from both the files.

```
. ta _merge
```

You should now find that cases with the same value of *aacode* have values of *oldest* in common. You can eyeball this in your data to make sure this has worked.

Check that the merge has worked as intended
If you merged all the household variables into the individual level dataset, the same information should be in the new file as there was in your two original files. Look at the data:
- does it look right? Is the merged dataset at the correct level of measurement? Are there lots of missing data that weren't in the original datasets? If so, should there be?
- do frequency tables of the same variables in both the old and merged individual level files. The results should be the same.
- select only the HRP in the merged file and do frequency tables of some of the household level variables. Then do the same using the original household file. The results should be the same.

8. Linking and merging in Stata

The dataset used in the following examples is the English Housing Survey 2011-2012: Household Data (EHS). This survey asked over 13,000 households about their housing and local environment. This dataset can be downloaded from the UK Data Service website, after completing a short registration.

Many datasets come as a single file but others are supplied in multiple files. These files may contain:

1. Information about the same cases at the same level of measurement (e.g. two files containing information about the same households but with different variables in each)
2. Hierarchical data: data that contain more than one level of measurement nested within another (e.g. individuals within households in individual level file(s) and household file(s)).
3. Data with the same variables but different cases (e.g. the same survey conducted in England and Wales with the data for each country in its own file)

Combining together (*merging*) multiple files that contain the same cases involves using one or more *matching variables* that uniquely identify each case (e.g.an individual ID variable and/or household ID etc.) to link the data from the same case in different files.

8.1. Using **joinby** to link multiple files at the same level of measurement

In the EHS, the ***generalfs11*.dta** file contains the weight and other administrative variables and ***interviewfs11.dta*** contains the answers to the household questionnaire. Both files are at the household level and contain all the cases. To analyse the interview data using the household weight, these two files must be merged.

To combine ***interviewfs11.dta*** and ***generalfs11.dta***, open ***interviewfs.dta***, sort it by the household ID variable aacode, save and close the data:

```
. use <interviewfs11.dta>, clear
. sort aacode
. save <interviewfs11.dta>, replace
. clear
```

Do the same with the ***generalfs11.dta*** data but leave this dataset open:

```
. use <generalfs11.dta>, clear
. sort aacode
. save <generalfs11.dta>, replace
```

Now use the **joinby** command with the household ID variabl aacode to link the interview data to the open generalfs11.ddta file. Save under a new name, *years1011.dta*:

```
. joinby aacode using <interviewfs11.dta>
. save <household11.dta>, replace
```

Check that the new dataset has the expected number of variables in it. Type:

```
. describe
```

Merging files when one file contains information about some of the cases only
You can use the same method to merge two files with the same level of measurement but for which one file contains a subset of the cases contained in the other. Because this is not a 1:1 correspondence between the cases as in the example above the choice of data to open first does affect the resulting merged dataset.

- If you open the file with all the cases first and add the subset file to it, you will obtain a file with all the cases in it. The dataset will contain values where they exist and missing values (system missing blanks) for the cases not contained in the subset.

- If you open the subset file first and add the other file to it, you will obtain a file with all the variables in both datasets but only for the subset of cases.

8.2. Using **merge** to attach household data to an individual level file

Datasets sometimes come with multiple files because the files contain data at different levels (e.g. individual, and household).
When the data are of the following format, you may add the higher level data to the lower level file (e.g. add household variables to an individual level dataset):

- a higher level file (e.g. household level), **and**
- a lower level file (e.g. individual level) where the lower level data are nested within the higher level data (e.g. individuals within households), **and**
- both files contain a matching variable to allow the higher level to be identified in each file e.g. both contain a household ID variable.

The data described are an example of hierarchical data. See Section 7.4 in Chapter 7 of this guide about using hierarchical data in SPSS for how to attach higher level data to lower level data in a hierarchical dataset.

8.3. Using **append** to merge files with different cases but the same variables

If two datasets have the same variables but different cases you may want to combine the files. There is no matching variable because there is no linking of cases as they are different cases in each file. Examples include combining survey data with the same questions held in a file for England and a file for Wales to create a dataset for England and Wales together, or combining several years of data from the same series of surveys to increase sample size (as long as there is a new sample in each survey).

Check documentation for coding differences between the two files
Before combining datasets, consider recoding to standardise the coding across time if there are variables where coding has changed between years. ALWAYS check the user documentation to see whether there has been *any changes.

To combine two years of the household level files, ***generalfs10.dta*** and ***generalfs11.dta*** from 2010-2011 and 2011-2012 respectively, open the 2010-2011 data, create a *year* variable, save under a new name and close the data:

```
. use <generalfs10.dta>, clear
. ge year= 2010
. save <year10>, replace
. clear
```

Do the same with the 2011-2012 data but leave this dataset open:

```
. use <generalfs11.dta>, clear
. ge year= 2011
. save year11, replace
```

Now use the **append** command to join the 2010-2011 file to the bottom of the 2011-2012 file in memory (now called ***year11***) and save under a new name, ***years1011.dta***:

```
. append using year10
. save years1011, replace
```

To order the data by year, use the sort command:

```
. sort year
```

For each year, the variable *aacode* taken together uniquely identifies households. To check whether this variable still uniquely identifies households, use the **duplicates** command:

```
. duplicates report aacode
```

If *aacode* no longer uniquely identifies households in the combined file, add the *year* variable to the duplicates report. Type:

```
. duplicates report aacode year
```

Since aacode and person together uniquely identify households in each year, aacode and year together should uniquely identify households in the new combined dataset. Type **help duplicates** for further information about the **duplicates** command.

9. Introduction to estimation commands in Stata

The dataset used in the following examples is the Labour Force Survey, October-December 2012: Teaching Dataset. The Labour Force Survey asks people aged 16 and over living in the UK about their working lives. The teaching dataset is a cut-down version of the full dataset. This dataset can be downloaded from the UK Data Service website, after completing a short registration.

9.1. The structure of estimation commands in Stata

In Stata, commands that produce statistical models are referred to as *estimation commands*. Most estimation commands in Stata follow a similar structure so that once you have learned the basic estimation procedure for one command, you will be able to produce results using a wide range of procedures.

For more information about estimation commands, type:

```
. help estimation commands
```

```
. help estcom
```

An example of an estimation command is linear regression command **regress**:

```
. regress depvar [varlist] [weight] [if exp] [in range] [, level(#) beta robust
noconstant noheader]
```

The regress command is followed by the dependent variable. Subsequent listed variables define independent/explanatory variables. Sampling weights can also be defined in the statement, as can conditional statements **[if exp]** or a range of cases **[in range].** These latter two options allow for the selection of a subset of data.

A number of other options can be specified following the comma in estimation commands. A full list of options for the **regress** command can be viewed by typing:

```
. help regress
```

9.2. Example: Multiple Linear Regression using the **regress** command

Multiple linear regression can be used to consider the extent to which a set of explanatory (independent) covariates predict the values of a continuous outcome (dependent) variable.

> **Know your data before conducting analyses**
>
> Prior to analysis, you should check the distribution of continuous or scalar variables, check and clean your variables, perform descriptive analyses, handle missing values, and consider potential transformations for your variables.

The variables used in this section are:

- *grsswk* (gross weekly pay)
- *ethukeul* (ethnicity - see codebook below)
- *sexx* (female = 1, male = 0);
- *marcivx* (marital status: Married/Civil partner(living with and sep.) = 1, not married or in civil partnership = 0)

Here is the coding for the ethnicity variable *ethukeul:*

```
. codebook eth,tab(10)
```

```
ethukeul                                          Ethnicity (9 categories) UK level

                  type:  numeric (byte)
                 label:  ETHUKEUL

                 range:  [1,9]                        units:  1
           unique values:  9                        missing .:  0/64237
       unique mv codes:  1                          missing .*:  42/64237

            tabulation:  Freq.   Numeric  Label
                        57160         1  White
                          574         2  Mixed/Multiple ethnic groups
                         1454         3  Indian
                         1035         4  Pakistani
                          365         5  Bangladeshi
                          339         6  Chinese
                          686         7  Any other Asian background
                         1574         8  Black/African/Caribbean/Black
                                         British
                         1008         9  Other ethnic group
                           42        .b
```

9.3. Choosing a comparison category for categorical variables

When entering a categorical variable into a model, one category acts as the reference group to which all other category values of a variable are compared. By default Stata will treat the category assigned the lowest numerical value as the comparison group. You can choose the reference category using the command **char**. To declare the first category (white) as the reference group for the ethnicity variable, type:

```
. char ethukeul [omit]1
```

Stata remembers the category selected and omits it from every subsequent model unless you specify another category.

Selecting comparison groups

When selecting the omitted category, choose a theoretically and empirically meaningful category. For instance, in relation to the ethnicity variable, to use the 'Other ethnic group' category may not be meaningful if you do not have a clear idea about who the 'Other ethnic group' are. The reference group should also be of a fairly large size, otherwise this could affect the stability of the models. Stata will give a message encouraging the use of another reference category if it considers the one you have chosen is too small.

9.4. Identifying indicator variables using the xi: command and i.

To tell Stata that a model contains categorical independent variables, you use the prefix **xi:** "interaction expansion" before your regression statement. You also need to prefix each of the categorical variables in the model with **i**.

Model 1: The relationship between gross weekly pay and ethnicity

To explore the relationship between gross weekly pay (dependent variable) and ethnic group (categorical independent variable), type:

```
. xi: regress grsswk i.ethukeul if statusx==1 & grsswk~=.
```

Note that this command selects only those who are in paid employment (*statusx*==1) and who do not have missing values for the weekly gross pay variable (*grsswk* ~=.).

This should produce the following results:

```
. xi: regress grsswk i.ethukeul if statusx==1 & grsswk~=.
i.ethukeul        _Iethukeul_1-10     (_Iethukeul_1 for ethukeul==-8 omitted)

      Source |       SS           df       MS      Number of obs   =    38,840
-------------+----------------------------------   F(9, 38830)     =      3.21
       Model |  2989395.02         9  332155.002   Prob > F        =    0.0007
    Residual |  4.0222e+09    38,830  103585.553   R-squared       =    0.0007
-------------+----------------------------------   Adj R-squared   =    0.0005
       Total |  4.0252e+09    38,839  103638.519   Root MSE        =    321.85

------------------------------------------------------------------------------
     grsswk |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
 _Iethukeul_2 |  -17.22364   75.8795    -0.23   0.820    -165.9494    131.5021
 _Iethukeul_3 |  -37.57224   77.94518   -0.48   0.630    -190.3467    115.2023
 _Iethukeul_4 |  -24.58499   76.60996   -0.32   0.748    -174.7424    125.5725
 _Iethukeul_5 |  -63.37222   77.73346   -0.82   0.415    -215.7318    88.98731
 _Iethukeul_6 |  -65.45238   80.58939   -0.81   0.417    -223.4096    92.50484
 _Iethukeul_7 |  -29.60063   80.03896   -0.37   0.712     -186.479    127.2777
 _Iethukeul_8 |  -51.11559   77.67373   -0.66   0.510    -203.3581    101.1269
 _Iethukeul_9 |  -43.25147   76.60832   -0.56   0.572    -193.4057    106.9028
_Iethukeul_10 |  -59.82227   77.26358   -0.77   0.439    -211.2608    91.61629
         _cons |   143.1667   75.86009    1.89   0.059    -5.521009    291.8543
------------------------------------------------------------------------------
```

9.5. Storing estimates

Stata automatically holds results for the last model run. When you run subsequent model, the results for earlier models are not retained by Stata. To assign the name ***model1*** to the model using **estimate store** so that you can recall results and perform further estimation procedures on it at a later time, type:

```
. est store model1
```

Model 2: Adding gender and marital status

The second model adds gender and marital status and **est store** saves the results as model2:

```
. xi: regress grsswk i.eth i.sexx i.marcivx if status==1& grsswk~=.

. est store model2
```

Note that because *sexx* and *marcivx* are coded as binary variables already, using the prefix i. before them in the command does not alter the results. Therefore, model 2 could have been written as:

```
. xi: regress grsswk i.eth sexx marc if status==1& grsswk~=.
```

```
. xi: regress grsswk i.eth sexx marc if status==1& grsswk~=.
i.ethukeul        _Iethukeul_1-10     (_Iethukeul_1 for ethukeul==-8 omitted)

      Source |       SS           df       MS      Number of obs   =    38,840
-------------+----------------------------------   F(11, 38828)    =     52.12
       Model |  58573146.1         11  5324831.46   Prob > F        =    0.0000
    Residual |  3.9666e+09     38,828  102159.351   R-squared       =    0.0146
-------------+----------------------------------   Adj R-squared   =    0.0143
       Total |  4.0252e+09     38,839  103638.519   Root MSE        =    319.62

------------------------------------------------------------------------------
     grsswk |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
_Iethukeul_2 |  -17.75598   75.3564    -0.24   0.814    -165.4564    129.9444
_Iethukeul_3 |  -26.77638   77.40822   -0.35   0.729    -178.4984    124.9457
_Iethukeul_4 |  -34.01025   76.08524   -0.45   0.655    -183.1392    115.1187
_Iethukeul_5 |  -75.06945   77.19894   -0.97   0.331    -226.3813    76.24241
_Iethukeul_6 |  -77.21878   80.03457   -0.96   0.335    -234.0886    79.65099
_Iethukeul_7 |  -28.76202   79.48744   -0.36   0.717    -184.5594    127.0354
_Iethukeul_8 |  -59.67742   77.14135   -0.77   0.439    -210.8764    91.52156
_Iethukeul_9 |  -38.27229   76.08027   -0.50   0.615    -187.3915    110.8469
_Iethukeul_10 |  -70.73305   76.73234   -0.92   0.357    -221.1304    79.66426
        sexx |  -58.11882   3.248701   -17.89   0.000    -64.48635   -51.75128
     marcivx |   52.08097   3.418595    15.23   0.000     45.38044    58.7815
       _cons |   140.0634   75.37326     1.86   0.063    -7.670109    287.7969
------------------------------------------------------------------------------
```

The **estimates** command offers a number functions for handling multiple models. If we forget which of our stored models we are currently using, we can use **estimates query**. Type:

```
. est query
```

```
. est query
(active results produced by regress; also stored as model2)
```

We can also look at the directory of stored estimates using the **est dir** command:

```
. est dir
```

```
. est dir
```

name	command	depvar	npar	title
model1	regress	grsswk	10	
model2	regress	grsswk	12	

To see the results for model 1 again, type:

```
. est replay model1
```

You can change which model is currently active. To make model 1 the active model, type:

```
. est restore model1
```

```
. est restore model1
(results model1 are active now)
```

The **est table** command can be used to visually compare output from your different models. This allows you to compare the values of coefficients following the addition of further variables. The **star** and **eform** options tell Stata to show stars next to parameter estimates to indicate level of statistical significance, and to display the results in 'exponentiated form' respectively (in this case as odds ratios). Type:

```
. est table model1 model2, star eform
```

Variable	model1	model2
_Iethukeul_2	3.310e-08	1.944e-08
_Iethukeul_3	4.815e-17	2.351e-12
_Iethukeul_4	2.103e-11	1.696e-15
_Iethukeul_5	3.005e-28	2.499e-33
_Iethukeul_6	3.753e-29	2.913e-34
_Iethukeul_7	1.395e-13	3.227e-13
_Iethukeul_8	6.321e-23	1.209e-26
_Iethukeul_9	1.645e-19	2.391e-17
_Iethukeu~10	1.046e-26	1.910e-31
_Isexx_1		5.745e-26***
_Imarcivx_1		4.154e+22***
_cons	1.501e+62	6.741e+60

legend: * p<0.05; ** p<0.01; *** p<0.001

To erase specific models from memory, use the **estimates drop <model name>** command to make room for another model. Type **estimates clear** to drop all stored models from memory. Type **help estimate** for further details of commands.

9.6. Post-estimation commands

To assess whether the inclusion of additional variables significantly improves overall model fit, use **testparm** to produce Wald tests. This is one of many post–estimation commands you can implement after fitting your model. Type:

```
. testparm   _Is* _Ima*
```

Note that _Isexx_1 stands for sex and _Imarcivx_1 for marital status. As there are no other terms

beginning with s or m, you can use _Is* and _Im* to stand for the two terms respectively. These are the results:

```
( 1)  _Isexx_1 = 0
( 2)  _Imarcivx_1 = 0

      F(  2, 10520) =  471.58
           Prob > F =    0.0000
```

Interpreting the output of testparm

The output above shows the parameter constraints being tested (the null hypothesis): that the coefficients for *sexx* and *marcvix* are simultaneously equal to zero. The F-test is shown next and is highly significant (the p-value is less than the generally used criterion of 0.05) so the null hypothesis can be rejected. In other words, the test indicates that the coefficients for *sexx* and *marcvix* are not simultaneously equal to zero. Including statistically significant predictors should lead to better model fit, so the test indicates that adding sex and marital status to the model significantly improves overall model fit.

Instead of just comparing all other ethnic groups to the White category, we can also consider whether there are statistically significant differences between any particular two ethnic groups. This is achieved using the **test** command to produce Wald tests.

The following example tests whether there are significant differences between 'Pakistani' and 'Indian' groups, and between 'Pakistani' and 'Bangladeshi' ethnic groups:

```
. test _Iethukeul_3=_Iethukeul_4

. test _Iethukeul_3=_Iethukeul_5
```

```
. test _Iethukeul_3=_Iethukeul_4

 ( 1)  _Iethukeul_3 - _Iethukeul_4 = 0

       F(  1, 10520) =   13.41
            Prob > F =    0.0003

. test _Iethukeul_3=_Iethukeul_5

 ( 1)  _Iethukeul_3 - _Iethukeul_5 = 0

       F(  1, 10520) =    6.83
            Prob > F =    0.0090
```

For more information about **testparm** or **test**, type

```
. help testparm

. help test
```

The help pages will give you more information about post-estimation commands following a particular estimation command. For more about post-estimation commands following a logistic

regression, type:

```
. help logit postestimation
```

For more information about post-estimation commands and some examples of their use, type:

```
. help postestimation commands
```

```
. help postest
```

Further models

Model 1 and Model 2 give some initial indications regarding ethnic differences in income, but they are incomplete in that many other important explanatory variables for income (such as educational attainment or job tenure) are missing from the model, and as a result the current estimates might be biased. Further analysis should be conducted to investigate what other variables should be included and to check that the assumptions of linear regression are not violated, to consider the influence of outliers, to consider the model fit etc.

More information and help with Stata

The help pages available in Stata provide information about Stata commands and how to use them. Type **help** to open Stata help or type **help <command>** for more information about a particular command. The Stata help pages are searchable. The help page for each command gives its syntax, a description of the command, details about options, examples of use and related commands. The help pages also provide overview pages about types of commands and useful commands. For example, for a list of useful commands, type:

```
. help y_41
```

Entering into an internet search engine the name of a command with the word 'stata', or a description of what you want to do in Stata often produces useful results.

Statalist is an independently operated listserver hosted at the Harvard School of Public Health. Anyone with an email address may join. Subscribers may post to the list and join discussion forums. Subscribers may ask questions about Stata ranging from the basic to the very advanced and receive replies from other Statalist users. There is a searchable archive of Statalist discussions.

The website address is: www.stata.com/statalist/

2 November 2015

T +44 (0) 1206 872143
E help@ukdataservice.ac.uk
W ukdataservice.ac.uk

The UK Data Service delivers quality social and economic data resources for researchers, teachers and policymakers.

UK Data Service

E·S·R·C
ECONOMIC
& SOCIAL
RESEARCH
COUNCIL