Stata: Data access & management

Software Used

STATA 15

Files Used

bhps.dta

StataIntroduction.do

Revision Information

Version	Date	Author	Changes made
1.0	July 2008	Adam Whitworth & Kate Wilkinson	Created
1.5	Sept 2009	Neli Demireva	Major Updates
2.1	March 2013	Neli Demireva	Revised and Updated
2.2	July 2013	Ladislav Kozak	Revised and Updated
3.4	October 2016	Ines Rombach	Revised and Updated
4.0	February 2018	Bethan Copsey	Revised and Updated
4.1	June 2018	Bethan Copsey	Minor edits to exercises
4.2	January 2019	Bethan Copsey	Minor edits to exercises
5.0	October 2019	Pradeep Virdee	Minor edits throughout

Copyright

Useful Information

ITLP Portfolio: http://portfolio.it.ox.ac.uk

How to Use This Course Book

This handbook accompanies the taught session for the course. Each section contains a brief overview of a topic for your reference and some sections are followed by exercises.

The Exercises

Exercises are arranged as follows:

- A title and brief overview of the tasks to be carried out;
- A numbered set of tasks, together with a brief description of each;
- The lecturer and demonstrator will be at hand to help out with any queries during the exercises. At the end of the course, solutions will be provided.

Some exercises, particularly those within the same section, assume that you have completed earlier exercises. Your lecturer will direct you to the location of files that are needed for the exercises. If you have any problems with the text or the exercises, please ask the lecturer or one of the demonstrators for help.

This book includes plenty of exercise activities – more than can usually be completed during the hands-on sessions of the course as well as some tasks that can be performed as a homework. These are clearly outlined throughout the course book.

Writing Conventions

Certain conventions are used to help you to be clear about what you need to do in each step of a task.

- Stata commands are presented with a small font on a new line similarly to the official Stata syntax conventions.
- A button to be clicked will look **like this**.

Objectives

From this course book you should:

- Be able to open Stata .dta and .do files
- Be able to compile and execute .do files
- Be familiar with your data by describing and listing variables
- Be able to perform some basic manipulations of the data such as keeping, deleting, generating and recoding variables
- Understand how to use univariate statistical computations such as summarize and tabulate

Contents

1 Stata: Introduction

Stata is a Data Analysis and Statistical Software Package for Professionals.

However, is Stata for me? Do I need it and would it be useful for me? One way of thinking about these questions and trying to find an answer is by comparing Stata to other similar statistical software packages.

How does Stata differ from similar software packages? (Just some thoughts)		
SPSS	**Stata**	**SAS**
Advantages	***Advantages***	***Advantages***
Widespread use	Syntax is logical and fairly intuitive.	Can handle very large datasets (Stata or SPSS will be fine for most needs).
Cheap licenses from the University.	Good for programming so ideal for tricky or repetitive tasks.	Syntax and programming style similar to previous programming languages.
Click menus – simple to learn and will do most things.	Large range of statistical analyses possible.	Cheap licenses from the University.
Can paste menu-driven commands to syntax files.	Very flexible – can write and download programmes.	
Simple to make and edit graphs	Fast to run and less prone to crashing.	
	Growing in popularity and in supportive community that you can consult: http://www.stata.com/statalist/	
Disadvantages	***Disadvantages***	***Disadvantages***
Syntax not terribly logical – difficult to check it and hard to write.	Still relatively expensive to buy. Stata is sold in UK by Timberlake Consultants and students are able to buy a cheaper GradPlan version.	Programming is not as flexible.
Relatively slow to run and tendency to crash. Not great for programming; not flexible.	Primarily syntax driven but can be used through menus as with SPSS.	For non-programmers, syntax not as intuitive to learn or to do certain tasks.

It is obvious that Stata's advantages and full potential is realized with extensive and continuous use (Stata is very good for programming and for handling large datasets) and for first time users it may seem forbidding and esoteric. In reality, it is much easier than you think and you should just give yourself time to adjust to its ways of dealing with data.

Stata is available for Windows, Unix, and Mac computers. This tutorial in particular uses the Windows version but most of the contents applies to the other platforms as well. The standard Stata version is called Stata/IC (or Intercooled Stata) and can handle up to 2,047 variables. There is a special edition called Stata/SE that can handle a greater amount of variables up to 32,766 (and also allows longer string variables and larger matrices), and a version for multicore/multiprocessor computers called Stata/MP, which has the same limits but is substantially faster. The number of observations is limited by your computer's memory.

There are also various ways in which data in SPSS, SAS or Excel format can be transformed into Stata format. Stata can easily use or export data in these formats.

1.1. A few words about the Data that we are going to use today

We are going to use data from the **British Household Panel Survey.** The *British Household Panel Survey* (BHPS) is conducted by the ESRC UK Longitudinal Studies Centre (ULSC), together with the Institute for Social and Economic Research (ISER) at the University of Essex. The main objective of the BHPS is to further understanding of social and economic change at the individual and household level in Britain. Therefore, it contains a wide range of information. It was designed as an annual survey of each adult member (aged 16 years and over) of a nationally representative sample of more than 5,000 households, making a total of approximately 10,000 individual interviews. The same individuals are re-interviewed in successive waves and, if they leave their original households, all adult members of their new households are also interviewed. Children are interviewed once they reach the age of 16. Major topics in the first three waves of the panel survey are household organisation, the labour market, income and wealth, housing, health and socio-economic values. Further information about the survey could be found at: http://www.data-archive.ac.uk/findingData/bhps.asp

1.2. Structure of the data

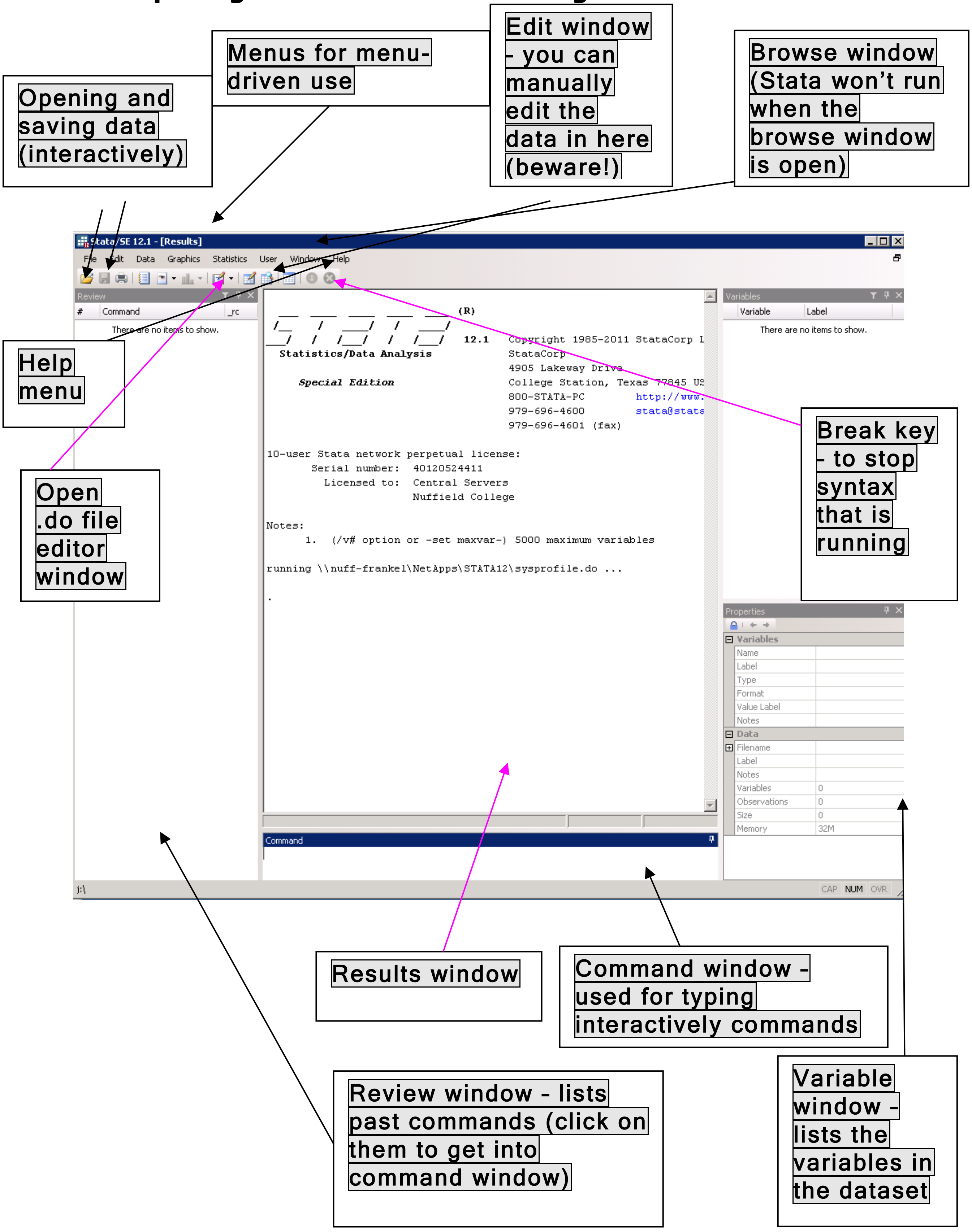
BHPS survey is a panel survey. Information for the respondents is collected at each wave. Some variables such as gender have been recorded only once but most variables will have values recorded for each of the waves. The data that we are going to use today, however , includes only data from a single assessment time point.

A wide format with repeated measurements looks like this:

Variable Name	**Personal identifier**	**Age1 (at wave 1)**	**Age2 (at wave 2)**	**Age3 (at wave 3)**	**Age4 (at wave 4)**
1	**145087984**	**52**	**52**	**52**	**52**

As you go along, you can learn various ways in which you can transform the data from wide into long format and vice versa if you need to do so for your analyses. This course will focus specifically on getting a feel of the data in Stata and introduce you to the cleaning and manipulation of the dataset in Stata. A full list of the variables in our practice dataset can be found at the end of this manual.

1.3. Opening Stata and understanding the Stata interface

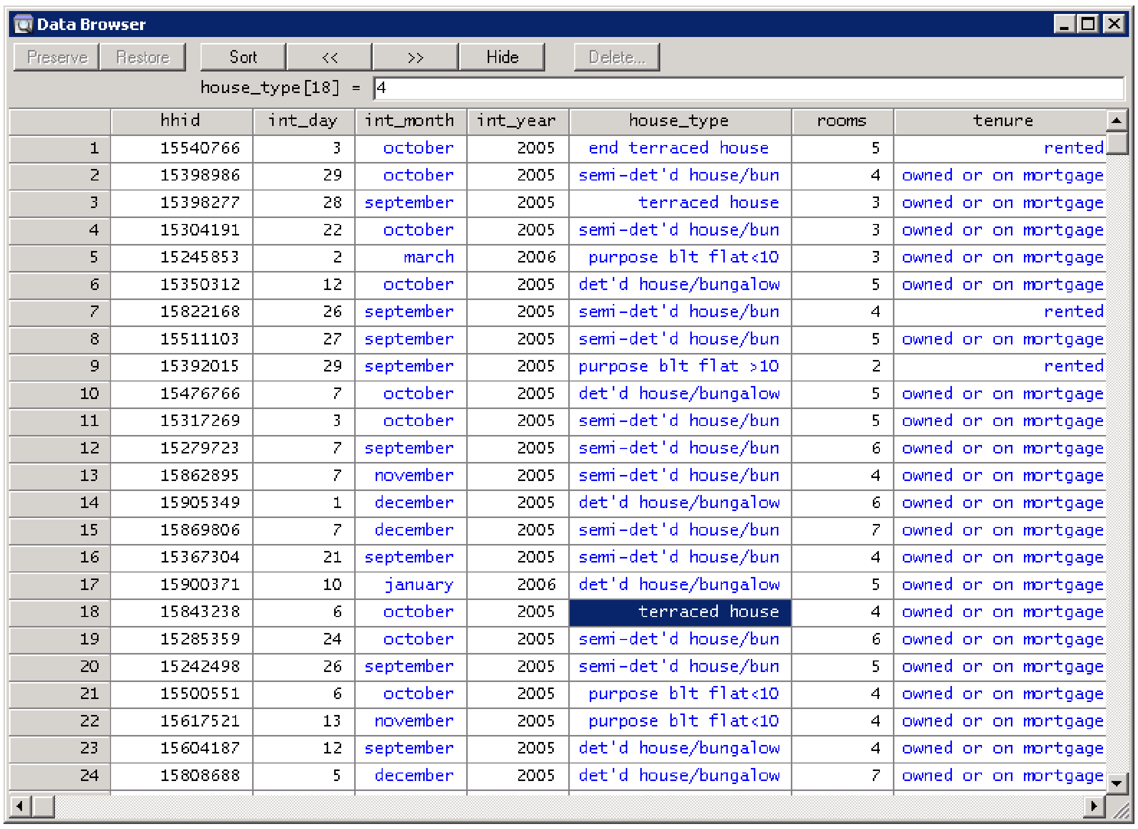

Opening and saving data (interactively)
Menus for menu-driven use
Edit window - you can manually edit the data in here (beware!)
Browse window (Stata won't run when the browse window is open)
Help menu
Open .do file editor window
Break key - to stop syntax that is running
Results window
Command window - used for typing interactively commands
Review window - lists past commands (click on them to get into command window)
Variable window - lists the variables in the dataset
Stata/SE 12.1 - [Results]
File Edit Data Graphics Statistics User Window Help
Review
# Command _rc
There are no items to show.
Variables
Variable Label
There are no items to show.
Properties
Variables
Name
Label
Type
Format
Value Label
Notes
Data
Filename
Label
Notes
Variables 0
Observations 0
Size 0
Memory 32M
Command
CAP NUM OVR

```
  ___  ____  ____  ____  ____ (R)
 /__    /   ____/   /   ____/
___/   /   /___/   /   /___/   12.1   Copyright 1985-2011 StataCorp L
  Statistics/Data Analysis            StataCorp
                                      4905 Lakeway Drive
     Special Edition                  College Station, Texas 77845 US
                                      800-STATA-PC        http://www.
                                      979-696-4600        stata@stata
                                      979-696-4601 (fax)

10-user Stata network perpetual license:
       Serial number:  40120524411
         Licensed to:  Central Servers
                       Nuffield College

Notes:
      1.  (/v# option or -set maxvar-) 5000 maximum variables

running \\nuff-frankel\NetApps\STATA12\sysprofile.do ...

.
```

1.4. What does our data look like in Stata?

The time has come to open Stata. This can be done by going to the Start menu on your computer. Then go to All Programs, find Stata and activate the program.

Let's open a Stata file first (Stata data files' extension is dta): click on the Open icon the first one below the File icon and choose our practice dataset: bhps household file. Files will be located on the H: drive (or another place as directed by your teacher).



house_type[18] = 4

	hhid	int_day	int_month	int_year	house_type	rooms	tenure
1	15540766	3	october	2005	end terraced house	5	rented
2	15398986	29	october	2005	semi-det'd house/bun	4	owned or on mortgage
3	15398277	28	september	2005	terraced house	3	owned or on mortgage
4	15304191	22	october	2005	semi-det'd house/bun	3	owned or on mortgage
5	15245853	2	march	2006	purpose blt flat<10	3	owned or on mortgage
6	15350312	12	october	2005	det'd house/bungalow	5	owned or on mortgage
7	15822168	26	september	2005	semi-det'd house/bun	4	rented
8	15511103	27	september	2005	semi-det'd house/bun	5	owned or on mortgage
9	15392015	29	september	2005	purpose blt flat >10	2	rented
10	15476766	7	october	2005	det'd house/bungalow	5	owned or on mortgage
11	15317269	3	october	2005	semi-det'd house/bun	5	owned or on mortgage
12	15279723	7	september	2005	semi-det'd house/bun	6	owned or on mortgage
13	15862895	7	november	2005	semi-det'd house/bun	4	owned or on mortgage
14	15905349	1	december	2005	det'd house/bungalow	6	owned or on mortgage
15	15869806	7	december	2005	semi-det'd house/bun	7	owned or on mortgage
16	15367304	21	september	2005	semi-det'd house/bun	4	owned or on mortgage
17	15900371	10	january	2006	det'd house/bungalow	5	owned or on mortgage
18	15843238	6	october	2005	terraced house	4	owned or on mortgage
19	15285359	24	october	2005	semi-det'd house/bun	6	owned or on mortgage
20	15242498	26	september	2005	semi-det'd house/bun	5	owned or on mortgage
21	15500551	6	october	2005	purpose blt flat<10	4	owned or on mortgage
22	15617521	13	november	2005	purpose blt flat<10	4	owned or on mortgage
23	15604187	12	september	2005	det'd house/bungalow	4	owned or on mortgage
24	15808688	5	december	2005	det'd house/bungalow	7	owned or on mortgage

You can also choose to work with only a subset of variables (large datasets such as the BHPS can have as many as 1000 of variables inside – remember information for each variable is collected at several waves).

1.5. An important thing to pay attention to: Interactive use vs. syntax do-files

Stata can be used both interactively by typing commands into the command window or by using the menus as we have done so far, or Stata can be used with **syntax do-files**. There is (mostly) no difference between these ways of working with Stata in terms of the commands that can be used or in terms of what those commands do. There are, however, significant differences in terms of the user's ability to i) keep a record of exactly what they have done to the data, and ii) be able to rerun whole projects making small changes to the syntax if required. Also, there are some important commands which can only be used with syntax, such as loops and programming.

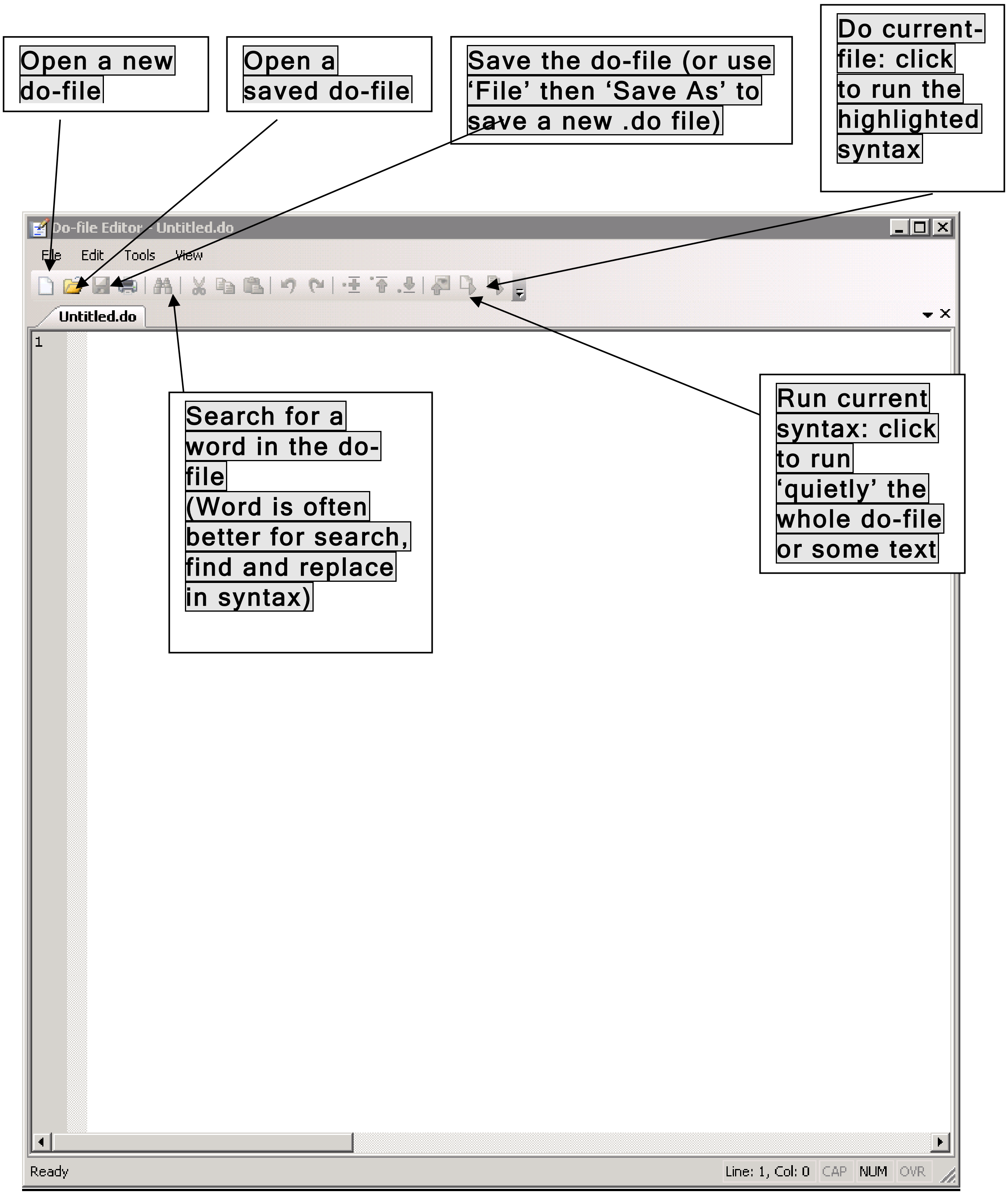
There are two ways to use Stata interactively:

- commands are simply typed into the command window (at the bottom of the picture above) and the user hits enter, or
- the user can use the drop-down menus.

Using Stata interactively runs the command and the data is changed. The command goes into the review window (for the duration of the session it is possible to see what has been done – it is also possible to right-click items in the review window and send them to a .do file). This can be OK from time to time but this approach has a serious disadvantages: Firstly, after the **session closes there is no record of what has been done to the data; secondly, it is not at all easy to undo or redo something during the same session, moreover it is impossible to do so once Stata has been closed.** Hence, you would not be able to show your supervisor, colleagues or funder what you have done to the data, and if you realise you need to change something after you have run the commands for the first time (which is virtually always the case) then this will be impossible to do easily.

As a consequence, most academic and professional users write and save syntax instead. All software programmes of this type can use syntax and in Stata this is done using do-files. A do-file is recognised by Stata as a syntax do-file rather than as a data file but a do-file is basically just a text editor like Word or Notepad. Instead of using Stata interactively, a do-file is the document where all of the syntax for a project is written. This is saved and kept so that there is a permanent record of what was done and so that the syntax can easily be changed or amended as the project develops, and this amended syntax file can then be used to rerun the whole project again if needed. In terms of its importance, if you have your **raw** data files then having a do-file – rather than having the final data – is the most important thing because the do-file can be used to replicate the whole project just by running it again. This goes against what many people intuitively feel which is that the data which is created throughout is the most important thing. Our aim in writing do-files is to create a single syntax file which we can run from start to finish in one go in order to create all the data we need for a particular project without any manual, interactive involvement. This complete do-file can then be passed to a colleague or supervisor to check or use, or can be amended if needed and the whole project rerun easily.

To open a do-file go to 'Window', click on 'Do-file editor' and then 'New do-file' or alternatively click on the 'Open new .do file' icon in the main results window. A new do-file appears:


Open a new do-file

Open a saved do-file

Save the do-file (or use 'File' then 'Save As' to save a new .do file)

Do current-file: click to run the highlighted syntax

Search for a word in the do-file (Word is often better for search, find and replace in syntax)

Run current syntax: click to run 'quietly' the whole do-file or some text

2 Common commands at the start of a do-file

It is common to see all or some of the following at the top of a syntax file:

2.1. Clear

The *clear* command is often used at the start of a do-file – this clears Stata of any data that is open so that the session can begin afresh. If there is data open then you will need to type clear before you can use your own data – remember to save any data which is open first if necessary.

2.2. Setting more off

Another preference might be to type set more off in the command window and this allows the results window to run without seeing the –more- message when the results window becomes full. However, if you run a command that generates a lot of output which you want to look at on the screen then you may not be able to see all the output if you have typed set more off as it only allows you to scroll through a limited section of the output window. The command set more on restores the –more- message.

2.3. The display command

Stata can work as a calculator using the *display* command.

```
dipslay 2+2
```

NB. Stata commands are case-sensitive, display is not the same as Display and the latter will *not* work. Commands can also be abbreviated; the documentation and online help underlines the shortest acceptable abbreviation of each command and we will do the same in this manual.

2.4. Features of do-files: *Saving a do-file*

The do-file is saved by clicking on 'File', 'Save As' in the *do-file editor's menus* – when using Stata interactively data is saved (and opened) from the *main results window*. It is clearly important that you save your syntax so that you do not lose any of your work. Note that you can only save .do files interactively in the .do-file editor window.

2.5. Annotating do-files with comments

It is a good idea to get into the habit of adding notes of what the steps of the syntax is doing. This is helpful when returning to the work after some time or for helping others (colleagues, supervisors, etc.) to understand what the syntax is doing. Note that /* and */ can be used to write notes in the syntax and Stata ignores everything between the first /* and the closing */. It does not matter how many stars are placed within these slashes. Alternatively, some people prefer to use just stars without slashes (* *notes* *) to put annotations in and this works just the same. Some people add annotations for each section of syntax, others prefer to annotate every single step of the syntax.

2.6. Browse window must be closed

In older versions (10 or earlier) Stata will not run (either interactively or a do-file) if the browse window is open.

2.7. Opening do-files from folders

If you go through Windows Explorer or My Computer and open a do-file directly from a folder then the Stata interface and the do-file editor will be opened at the same time. However, it is more common to open Stata, open a new do-file by clicking Window, then Do-File Editor, then New Do-File. You can then use the open icon in the .do file editor window to open the do-file in the same way as you would open a Word document. This enables you to have more than one do-file open at the same time.

2.8. Delimit

Where lines of syntax are long the do-file can extend rightwards to cater for this, but users sometimes prefer to see all of their syntax within the window without having to scroll rightwards. Lines of syntax must always begin with a command and so it is not possible to just hit enter to continue part of a line of syntax to the next line. In order to break the line of syntax, one option is to use three forward slashes /// at the end of the line of syntax and then it is possible to hit enter and continue the line of syntax on to the line below in the do-file: Stata sees the /// and understands this as a line break, and so understands the two (or more) lines as a single line of syntax and does not produce an error message when the following line does not begin with a command.

It is considered good practice to ensure that commands can be read easily within the command window or when printed.

Alternatively, some users prefer to change the delimit (which is the keystroke which identifies the end of a line of syntax). This is by default set to the enter key but it is possible to change this to something else (typically ;). The advantage of this is that Stata then looks for the ; as the end of the line of syntax, irrespective of whether this 'line' of syntax is on one or more lines. Hence, by changing the *delimit to ;* it is possible to hit enter to break up long lines of syntax and then to just include a ; at the end of that syntax line. Programmers often also prefer ; as the delimit as this is common in other software.

To change the delimit to ; (or any other key) the syntax is:

```
#delimit ;
```

While this option is set, commands can run over several lines, and each command needs to end with ';'.

To change the *delimit* back to the default of the enter key type:

```
   #delimit cr
```

Note: this command can only be used in .do (and .ado) files.

Hence, if we had one particularly long line of syntax that we wanted to put on multiple lines we could use /// or we could type:

```
#delimit ;

/* change the delimit from carriage return to ; */

Long line of syntax

   Which spans multiple Lines in the

            do-file and ends with the new delimit  ;

/* the syntax ends with the new delimit to signal the end of
that line of syntax */

#delimit cr

/* change the delimit back to carriage return (or not if you
prefer ;) */
```

The ; delimit can be frustrating when wanting to run only sections of the do-file because Stata returns to the default unless specified in the actual bit of the do-file which is highlighted and run at that time. **NB**: **Notice that the #delimit ; does not produce any feedback messages apart from the error message if you try to use it interactively in the command window. It only works when you put it in a do file.**

NB: Baum, C. (2009: 28) "Use #delimit ; sparingly (Stata is not C). To deal with long lines, use ///and continue on the next line of your do file. You can use the triple-slash continuation on any number of consecutive lines of text."

2.9. Placing items from the Review window into a .do-file

For users who use Stata interactively, it is possible to make a .do file containing the contents of the Review window which is advisable for interactive users. To do this simply right-click on the Review window and 'Save Review Contents' to a .do file.

2.10. Data types

In Stata, the major distinction is between numeric (black/blue in the Browser window) and string variables, i.e. text (red). Different data management tasks often involve conversions between numeric and string variables. For example, if you import data from Excel (xls, xlsx or cvs format), it is likely that Stata has transported it and considers it a string variable despite the fact that its contents are numeric. String variables can hold up to 224 in length. Stata's numeric data types are byte, int, long, float and double. The byte, int, and long types can hold only integer values. Float and double can hold very long numbers.

3 Opening and saving data

3.1. Opening data in Stata format

Stata data files end in .dta (do-files end in .do) and to open them the syntax is:

```
use filename.dta
```

For example, if we had a Stata data file called 'bhps' saved in thefollowing location on the H: drive - H:\Data\Stata\Original then the syntax to open the file from this folder path would be:

```
use "H:\Data\Stata\Original\bhps.dta"
```

This command alone will result in an error message if data are already open in Stata and changes have been made (i.e. any syntax has been run) but these data have not been saved. This is because opening the new data would cause the data which is open to be lost, and Stata will give you an error message to warn you of this. It is therefore common to see the clear option with the use command in syntax files. The syntax would therefore be:

```
use "H:\Data\Stata\Original\bhps.dta" , clear
```

When using Stata interactively the 'open file' icon in the main Stata window is used to open a data file too and this is done in much the same way as you would open a document in Word.

3.2. Using Windows Explorer to get folder and file paths correct

It can be time-consuming to write out complete file paths (as above) by hand and there is a chance that small mistakes will be made when doing so which will lead to difficulties in opening the data as any misspelled folder/file names will not be recognized. A safer way to pick off folder paths and file names is to use Windows Explorer. To do this right-click on the 'Start' icon in the bottom left of the screen and left click on ''My Computer' to open Windows Explorer. Browse through the locations on the computer to find the folder that you wish to open the file from, double-click it to get inside the folder (so you can see the files you want to open), highlight the folder path from the bar across the top of the screen and then copy and paste it into the .do file. Next, single left-click the file you wish to open so that you are able to copy the filename, and then paste this filename onto the end of the folder path – be sure to have a slash in the path before pasting in the file name onto the end. Now you can surround it all by quotation marks, add the use command, and the folder and file names should work correctly.

Also, if all files are opened from and saved to the same location, the "change directory" command can be used. This directs Stata to the main folder, and requires only the subfolders and/ or filenames to be entered when opening or saving files.

```
cd "H:\Data\Stata\Original\"
use bhps.dta
```

3.3. Opening data in comma/ tab delimited formats

The *insheet* command can be used to open data from an Excel worksheet or any tab-delimited text files. Despite its name *insheet* does not read binary spreadsheets (file types .xls). Both tab and comma options are available. In order to do so your data file must be set up in a certain way:

- the first row contains the variable names and the data start on the second row;
- missing numeric data should be coded as an empty cell;

when some spreadsheets create a .csv file they do not add commas to the end of a line if the cells at the end of that line are empty. This will confuse Stata which relies on the commas to tell where the values are. You can avoid this problem by adding columns of 1's (or any other character) to the end of each row of data. This variable can be dropped once the data is in Stata;

the file must be specifically saved as a 'comma separated values' (.csv) file in Excel. This can be done by clicking 'File', 'Save As', then choosing 'comma separated values'.

Once the file has been saved as a .csv file then the syntax to open this file in Stata is:

```
insheet using filename.csv
```

For example:

```
insheet using "H:\Data\Excel\Original\excel_names.csv", clear
```

It is most important to check whether the command has produced the desired file. Check for example, if the number of observations that was in the text file is the same as in the new file, using the summarize command.

- Another possibility is to use the *infile* command. An important distinction exists between *insheet* and *infile*. With *infile*, you could use the if expression and in range qualifier to selectively input data. For example, you could use in 1/1000 to read only the first 1000 observations.

The structure of the command is:

```
infile varname1 varname2 varname3 using filename.csv
```

3.4. Opening data saved in Excel format (e.g. xls/ xlsx files)

Newer versions of Stata allow for data to be read in from Excel (.xls or .xlsx. formats). In the command below, 'firstrow" indicates that the first row in the dataset consists of the variable names, rather than variable values. If no variable names are saved in the first row of the excel file, then the *firstrow* option is not used.

More information can be accessed typing the command "help import excel"

```
import excel using "H:\Data\Excel\Original\data.xls", ///
clear firstrow
```

However, the formatting and coding that can be obtained when importing files in Stata format is lost when importing data in Excel format.

3.5. Saving data in Stata

As in programmes such as Word, there are two ways to save data:

- it is possible in Stata either to open a dataset, make changes to it and then save over the same file. When using Stata interactively, this is done by clicking the save icon on the menu bar, *or*
- it is possible to open a dataset, make changes to it, and save it to a new dataset - this leaves to initial dataset opened unchanged. When using Stata interactively, this is done the same way as it would be done in Word: click 'File', 'Save As', specify a location and *new* filename, and save with the .dta ending (Stata data files end .dta).

However, it is much better practice to use Stata commands to save data sets. These can be saved in .do files to ensure that a record exists of all the changes made to the data, and the datasets saved. Commands for saving data in Stata and excel format are shown below.

3.6. Saving data to a new file in do-files

To save a data file in a do-file the following syntax is used:

```
save filename.dta
```

For example, if we want to save a file to the Data\Stata\Modified folder on the H: drive we would type:

```
save "H:\Data\Stata\Modified\work file.dta"
```

By specifying the file path and new file name as we have done here, a new file is saved with whatever changes have been made to the dataset since we opened it. Importantly, by saving the dataset to a new filename, the initial dataset which we opened remains unchanged. This is what is often done with raw data files for example, which are usually never changed or overwritten to enable us to go back to the original data if needed. Saving with a new file path or filename in a do-file is equivalent to 'File', 'Save As', specifying location and new filename if using Stata interactively (i.e. bullet two above).

This syntax works if a dataset of that name does not already exist in the specified folder. However, if a dataset with this name does exist in this folder then Stata will not let us write over it using this syntax. Instead, it will give us an error message telling us that a file of this name already exists in this location and asking us if we are sure that we wish to copy over it with this new data. To make Stata write over the file you simply add the *replace* option. For example, if we wanted to run the syntax for a second time (i.e. the file already exists) and replace the existing file (we might for instance have improved the syntax since the last time we ran it) then we would write:

```
save "H:\Data\Stata\Modified\saved data.dta", replace
```

The *replace* option overwrites any previous version of the file which exists in this folder and so is commonly used by users who write syntax files and often run their syntax multiple times as they gradually make changes to it. Therefore, only use the replace option when you are sure that it is OK to write over the existing file because once replaced there is no way to retrieve the original file. It is usually fine in a syntax do-file to use the replace option because it will replace files with the most recent (and presumably most correct) versions as the syntax is improved over time.

However, it is good practice to **! never overwrite any original raw data files** at all in case they are needed for some reason in the future.

3.7. Saving over the same file in a do-file (use with caution)

Alternatively, if a data file were open at the time, we had made changes to it and did actually just want to write over the version of the file we opened with the changed version we would not need to specify the file path and could just type:

```
save, replace
```

This overwrites the data file that is used at the time this command is run. The previous version of the file (i.e. the one that we opened) is changed permanently. It is advisable to use *save, replace* very cautiously: it is not good practice to make any changes to any raw datasets, and using *save, replace* can often cause problems if running the syntax several times (e.g. if generating new variables and afterwards using *save, replace* can cause problems. This is because, if the .do file is run again, there will be an error message when you come to generate a variable again because it already exists in the dataset). It is usually a good idea to save to a new work file rather than use '`save, replace`'.

3.8. Saving for previous versions of Stata

If you are working on a later version of Stata and use the save command then you will not be able to open that data file in earlier versions of Stata: this can cause a few problems if for example one person in your research team has a copy of Stata 13 but everyone else uses Stata 12. The same also applies with earlier versions.

A useful command to avoid this problem is '`saveold`'. This command is used in exactly the same way as the save command outlined above except that it will allow files saved in Stata 13 to be opened with versions 12, 11 etc.

3.9. Saving to Excel format files

If you wish to save your data in excel format (.xls. or .xlsx), then the following command can be used:

```
export excel using "H:\Data\Excel\Modified\bhps1.xls", ///
 replace firstrow(varlabels)
```

In this command, the first row in the excel spreadsheet will consist of the variable labels (rather than variable names) that were used in the Stata dataset.

However, you will use any value labels you may have applied to the data.

If you wish to save your data in csv format (which Excel can open) then you should use the *outsheet* command. The *comma* option can be specified to save a comma delimited file (and replace can also optionally be used):

```
outsheet using "H:\Data\Excel\Modified\saved data.csv", ///
comma replace
```

For those who also have StatTransfer installed then the *stcmd* command may be useful. This essentially allows you to carry out the step of transferring data between formats from within Stata. To download this .ado programme and its accompanying help file type *findit stcmd* in the command window.

Exercise 1 Opening and saving data (15 minutes)

- *Practice opening and saving data in different formats using a .do file*

Task 1

- Open a new do file.
  Note: In these exercises, you need to write your own code. Write all of your commands for the following tasks in this do file.
- Open the Stata dataset with the name bhps.dta, which is saved in H:/Data/Stata/Original using the *use* command in Stata.

  Hint: the Stata code to open data in Stata format is described in section 3.1
- Take a minute to look at the data in the Data Editor window, and have a look around the results window. How many observations and variables are in the dataset?
- Save the dataset in H:/Data/Stata/Modified using the *save* command in Stata. Name the newly created dataset bhps_save.dta.

  Hint: Ensure that the file path and name for the new dataset are correctly specified to avoid overwriting the original dataset.
  The Stata code to save data in Stata format is described in section 3.6

Task 2

- Open the Excel dataset with the name excel_names.xls, which is saved in H:/Data/Excel/Original using the *import excel* command in Stata.
  Be aware that in this dataset, the variable names are given in the first row of the excel spreadsheet.

  Hint: the Stata code to open data in Excel format is described in section 3.4
- Take a minute to look at the data in the Data Editor window, and have a look around the results window. Does the data appear to be different from that read in during task 1?
- Save the dataset in H:/Data/Excel/Modified using the *export excel* command in Stata. Name the newly created dataset excel_names_save.xlsx .

  Hint: the Stata code to save data in Excel format is described in section 3.9

Task 3

- Also open the dataset named excel_nonames.xls, saved in H:/Data/Excel/Original.

  Hint: Be aware that this dataset contains no variable names in the first row. How do you have to change the code?

Task 4

- Save the Stata .do file in H:/StataProgs. Name the .do file StataCourse1. *Hint: this task has to be performed manually; there is no command for this.*

4 Getting your head around Stata syntax

4.1. General structure

The general structure of syntax in Stata is:

```
command variable(s) [if] [in], options
```

Commands always start on a new line and lines of syntax have to start with the command. One command should not run to the next line (although see 'delimit' in section 2.8). Stata does not mind whether there is more than one space between words where this does not affect the meaning of the syntax (there must be at least one space between words though). Stata is case-sensitive and lower case should always be used for commands.

For example, say we wanted to make a binary variable *'int2005'* to identify if the respondent in our dataset was interviewed in 2005 where *'int2005'* takes the value 1 if the interview was in 2005 and the value 0 if it was not in 2005 (it was in 2006 for example). A Variable 'int_year' which identifies the year in which the interview took place already exists in our dataset. So what we want to do is to create a variable int2005 with values equal 0 and then to tell Stata to change the value to 1 if the value for int_year is 2005. To do this we could write:

```
generate int2005 = 0
replace int2005=1 if int_year==2005
```

In this syntax it can be seen that the commands (generate and replace) start the line, next comes the variable followed by the values to apply to it, and finally come any if conditions. Note that the if condition takes a **double equal sign** in Stata (see p.24 for other mathematical operators to use with if conditions). In this example what we firstly generate a variable called 'int2005' and set it to 0 for all observations. Next we replace 'int2005' to 1 in cases where the interview took place in 2005 (i.e. int_year==2005). Note that **once the variable is generated we cannot generate it again (unless we first drop it) and we instead replace values of this existing variable**.

Let's assume that we want to **drop** the following two variables from the dataset now:

```
drop int_day rooms
```

Again the structure of the syntax line is that the command comes first (i.e. drop) followed by the variables we wish to drop from the dataset.

rename is the command to rename variables. How would we rename the house_type variable so that it was instead called type_of_housing? Stata syntax is fairly logical so it might not be a surprise to learn that to do this you would type:

```
rename house_type type_of_housing
```

If we wanted summary statistics of an interval level variable (for instance car_value) then we could use the summarize command to give the mean, minimum, maximum, and some other summary statistics:

```
summarize car_value
```

Finally, many commands in Stata have options which can be specified by typing a comma and then the desired options. summarize is one such command where the detail option can be added to the command to gain additional summary statistics (e.g. percentiles, median, etc). The syntax would be:

```
summarize car_value, detail
```

4.2. If conditions

Many commands can be carried out with 'if' conditions and this means that the command will only be carried out for observations (rows) which satisfy the 'if' condition.

The mathematical operators used in Stata are similar to many other programmes, with the exception of 'equal to' which is a **double** equal sign when used as part of an if condition:

==	!= *or* ~=	>	<	>=	<=	&	|
Equal to (2 = signs)	**Not equal to**	**Greater than**	**Less than**	**Greater than or equal to**	**Less than or equal to**	**And**	**Or**

Remember: a missing numeric value (.) is always considered higher than all other numeric values. So, if you are recoding a numeric age variable (age) into a categorical age variable (age_old) then be very careful about writing:

```
gen age_old=0
replace age_old=1 if age>70
```

The latter command will basically replace all missing values with the value of 1.

Instead you should write

```
replace age_old=1 if age>70 & age!=.
```

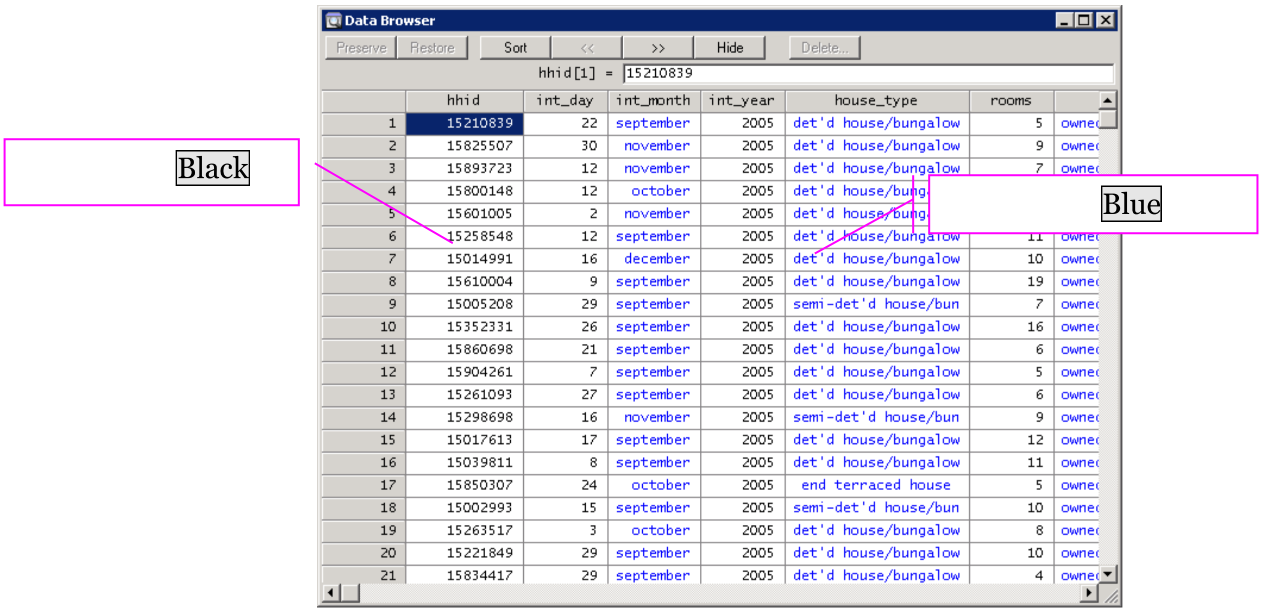
This last sentence can be written as:

```
replace age_old=1 if age>70 & !missing(age)
```

4.3. Getting a feel for the data – taking a first look at the variables: describe

Once the data is open it is likely you will want to have a look at the variables, their coding and their data values to become more familiar with the data. One way to do this is through the browse window, within which you can sort variables and use the scroll bar on the right to see the largest and smallest values for the sorted variable. Missing numeric values are coded as **.** in Stata and are treated as the highest numeric value – higher than all non-missing numbers. If you have missing data you would be advised to code it as . too because Stata expects this and some of its commands are set up to recognise . as missing and to treat it accordingly. This means that you probably have to do some recoding of your variables as in survey data, values such as 'No answer' may be coded as '-8' and 'Does not apply' as '-9' and you would normally want to recode them into **.** when you are analysing the data.

Data formats in the browse window



hhid[1] = 15210839

	hhid	int_day	int_month	int_year	house_type	rooms	
1	15210839	22	september	2005	det'd house/bungalow	5	owne
2	15825507	30	november	2005	det'd house/bungalow	9	owne
3	15893723	12	november	2005	det'd house/bungalow	7	owne
4	15800148	12	october	2005	det'd house/bung	[illegible]	[illegible]
5	15601005	2	november	2005	det'd house/bung	[illegible]	[illegible]
6	15258548	12	september	2005	det'd house/bungalow	11	owne
7	15014991	16	december	2005	det'd house/bungalow	10	owne
8	15610004	9	september	2005	det'd house/bungalow	19	owne
9	15005208	29	september	2005	semi-det'd house/bun	7	owne
10	15352331	26	september	2005	det'd house/bungalow	16	owne
11	15860698	21	september	2005	det'd house/bungalow	6	owne
12	15904261	7	september	2005	det'd house/bungalow	5	owne
13	15261093	27	september	2005	det'd house/bungalow	6	owne
14	15298698	16	november	2005	semi-det'd house/bun	9	owne
15	15017613	17	september	2005	det'd house/bungalow	12	owne
16	15039811	8	september	2005	det'd house/bungalow	11	owne
17	15850307	24	october	2005	end terraced house	5	owne
18	15002993	15	september	2005	semi-det'd house/bun	10	owne
19	15263517	3	october	2005	det'd house/bungalow	8	owne
20	15221849	29	september	2005	det'd house/bungalow	10	owne
21	15834417	29	september	2005	det'd house/bungalow	4	owne

In the browse window:

cells shown in black are the numeric values of numeric variables;

cells shown in blue are the value labels of numeric variables – whilst these may show words in the cell, the bar at the top of the browse window shows that these are numeric variables;

cells shown in red are string variables. The country variable is a string variable for example, despite looking numeric.

To change how variables are display in the browse window right-click the variable and then click 'Show/Hide All Value Labels'. Alternatively it is possible to go through the 'Prefs' tab in the main results window and to change the General Preferences.

To check a variable double-click it in the browse window to open the 'Variable Properties' dialog box. In the format box a 'g' denotes a numeric variable and an 's' denotes a string variable.

As well as the browse window some other commands are also useful to get a feel for the data. 'Describe' is useful when you are less familiar with variables as it describes the variable(s) and also tells you if the variable has a value label attached to it (and if so what it is called). To describe all of the variables in a dataset the syntax would be:

```
describe  _all
```

and to describe the tenure variable the syntax would be:

```
describe  tenure
```

4.4. List

You can also develop a better understanding of the missing data by listing the missing values (for one variable or combination of variables):

```
list tenure if tenure==.
```

Another useful way that you can run commands on multiple variables is to use the * key. This is best explained with examples. If we typed

```
describe  car*
```

then we would be asking Stata to describe all variables which start with 'car' irrespective of what they have after this. * can be used at the end of the variable name (as here), at the start, or indeed in the middle of the variable name. This works with many commands so, similarly, if we wanted to drop all variables ending in 'temp' we could simply type:

```
drop  *temp
```

As well as describing the variables, the 'describe' command also sets out whether each variable has a set of value labels attached to it and, if so, what those value labels are called. These value labels refer to the values (i.e. categories) which that variable can take – if you generate your own variables you may want to create labels both for the variable and for the values of the variable as well. Looking at the variable for housing type (house_type) the value label is listed as hh_type. In order to see the values of this label type:

```
label list hh_type
```

You might want to look at what values the first ten (or some other value) observations in the data have for some variable. For example, if you wanted to see the first ten values of household type (house_type) in the data the syntax would be as below.

```
/*list the first ten values of house_type in the dataset*/

list house_type  in 1/10
```

Note that this tells you the values for the first ten observations in the data, and therefore re-sorting the data in a different way will result in different observations coming to the top of the dataset.

Say we were interested in a particular housing type - for instance terraced housing – and we wanted to know how many people in the dataset lived in terraced housing. First we could use the 'label list' command shown above to find out that terraced housing is coded 4 in the house_type variable. Then the syntax to count the number of instances of terraced house in the data using the count command would simply be:

```
/*how many observations have house_type of value 4 (ie terraced
house)*/

count if house_type==4
```

This could also be done using other commands that we will come to later.

4.5. Codebook

Codebook is extremely useful command to obtain information on a variables, i.e. the format type, information on the label, the number of category labels etc. You can use it with a particular variable such as house_type.

```
codebook house_type
```

You can also use codebook to have a look at the different categories of a variable and the numerical values associated with them. This is particularly useful when a variable has a lot of categories and this information can be very important for subsequent coding.

```
codebook house_type
```

4.6. Missing Values

Stata has a variety of missing value codes: from system missing . to .a and .z. They are treated as large positive values, and when your dataset is sorted, they would come at the bottom of the dataset. Therefore if you use an expression if variable <. you would definitely exclude the missing values of that variable from the analysis.

Stata's standard approach of handling missing data is to exclude any missing observations from the computation. For the `generate` and `replace` commands, any missingness is preserved in the newly created variable. In univariate statistical commands, such as `summarize`, only non-missing cases are included. For multivariate statistical commands, Stata practices listwise deletion, in which each observation for which there is missingness on any of the variables included is deleted. The same applies to `correlate` command, although the `pwcorr` command computes pairwise correlations using all the available data for each pair of variables.

Several Stata commands, however, deal with missing data in a nonstandard way. The egen rowwise functions (rowmx, rowmean, rowtotal) all ignore missing values.

The empty or null string "" is also treated as a missing value.

You can use the 'mvdecode' command to recode various numeric values, such as -9, -8, -999 to missing. The 'mvencode' performs the opposite function. For more information, see the Help file.

Exercise 2 Getting a feel for the data (10 min)

- *Read the bhps file in again to ensure you are using the original .dta data*
- *Describe and get a feel for the data*

Task 1

- Read in the bhps file from H:/Data/Stata/Original, as in exercise 1.
- To get an initial idea of the main variables considered in this exercise, list the first 20 observations of the following variables:
  hhid, tenure, inc_lab, country and int_month
  Hint: information on the list command is provided in section 4.4
- From this output, how many of these households rent their accommodation?
- From this output, which household has declared a labour income of zero?

Task 2

- Run the *describe* command to look at all variable descriptions and the names of the value labels for all variables in the dataset.

  Hint: the Stata code for this command described in section 4.3.
- Specifically look at the following variables: tenure, inc_lab and country.
  Of these three variables, which one is in string format: tenure, inc_lab or country?
  What is the value label for tenure?
  What is the variable label for inc_lab?

  Hint: run the describe command again, but this time only for the above named variables.

Task 3

- What values can the interview month variable (int_month) take and what do they mean? Find the answer using two different methods:

  - 1. Use the *codebook* command
  - 2. Use the describe command to find the value label for the int_month variable and then the *label list* command to list the value labels.

  Hint: the Stata code for the codebook command described in section 4.5.
  The label list command is described in section 4.4.

- Compare the answers obtained when using methods 1 and 2. Are there any differences in the information displayed?

5 Managing and manipulating the data – basic data management commands

Drop and keep are fairly self-explanatory commands which can be used for variables or for observations – drop removes the specified variables or observations whilst keep retains only the specified variables or observations.

5.1. Drop variables

Drop can be used to drop particular named variables such as kids and emp in the first example below

```
drop kids emp
```

or like many other commands drop can also be made to act on a number of variables at the same time. In the example below we drop all variables from exp_food to car_value in the dataset as well as all variables beginning with toi (i.e. the two variables relating to access to toilet).

```
drop exp_food-car_value toi*
```

5.2. Drop observations

Drop can act on observations as well as variables. Say we were only interested in cases with particular values on the housing type variable (house_type) and we wanted to drop all observations with values other than these. First we could describe the variable to find out about its value label (its value label is called hh_type), and then list the value labels attached to the house_type variable:

```
describe house_type /*shows the value label is hh_type*/
label list hh_type /*show the coding of the variable */
```

Seeing the codes, we decide that we want to drop all cases that relate to business premises, sheltered accommodation, institutional housing, those labelled as 'other', and those missing information for this variable (i.e. those coded 9, 13, 14 and 15; remember that missing data is coded . – which Stata takes to be a very large number).

This is done with the syntax below - note the use of double equal signs after the if condition and the three forward slashes to break onto a new line in the .do file (this is not necessary as the .do file can keep going across but some users prefer to be able to see all of their syntax in the do-file window and so may wish to break lines – ass discussed in section 2.8):

```
drop if house_type > 999 | house_type==9 | ///
       (house_type>=13 & house_type <=15)
```

5.3. Keep variables

`Keep` does the opposite of the `drop` command. It will keep certain variables and drop all others:

```
keep hhid-garden inc* hh_wt-int_dur
```

5.4. Keep observations

`Keep` can also be used to keep only specified observations. Here we keep only those cases where the interview year is 2005 or 2006 – brackets may be needed for more complex if statements:

```
keep if int_year == 2005 | int_year == 2006
```

5.5. Preserve and Restore

There may be occasions when you wish to work on a subset of the data temporarily; maybe you have to consider a subgroup for only a single aspect of your analysis.

You can preserve your data before doing any manipulations to it through keep and drop.

Simply type:

```
preserve
```

After you finish working with a subset of the data and want to come back to the original data, type:

```
restore
```

Be aware that you have to run the preserve and restore commands from the .do file at the same time. If run separately, Stata will be unable to restore your data.

5.6. Browse

The 'browse' command is used when you want to open the browse window but want to only see selected variables. This can be helpful if you have lots of variables in the data and want to compare a few of them visually. Note that all of the variables in the underlying data remain intact even though you only see those selected in the browse window. To only see house value and house type for example we would type:

```
browse hhvalue house_type
```

5.7. Sort and gsort

If you want to look at the data and sort it then often this can be done interactively by clicking into the browse window, selecting the variable of interest and clicking sort.

However, if you should need to do a sort in your syntax (e.g. for a merge) then the sort command will do this:

```
sort hhvalue
```

Here for instance we sort by house value (hhvalue). Sort always does an **ascending sort so this means that missing values – which are usually . – end up at the bottom of the data file** as they take the highest numeric values in Stata.

You can add a second variable (e.g. sort int_year hhvalue), which will show the value of the houses in ascending order first for the earlier interviews, then for those conducted later.

```
gsort –hhvalue
```

Here for instance we do a descending sort on house value; the negative sign dictates the direction of the sort. The command `gsort +hhvalue` is also perfectly valid and means just the same as `sort hhvalue`.

It is also possible to sort by more than one variable at a time (nested sorts). Using the `sort` command with multiple variables sorts all the variables in ascending order. e.g. `sort int_year hhvalue`, will show the value of the houses in ascending order first for the earlier interviews, then for those conducted later.

When using gsort the variables can be sorted in either direction and need not all be sorted in the same direction. For example, to sort the data by month (ascending) and house value (descending) we type:

```
gsort +int_month -hhvalue
```

5.8. Generate – creating a new variable

The `generate` command is used to create a new variable. For example, to create a variable called count and to set it equal to 1 for every observation the syntax would be:

```
generate count=1
```

Generate can be abbreviated to gen (or even to g). So, to generate a new variable called int_year2 and set it equal to the value of an existing variable int_year we would type:

```
gen int_year2=int_year
```

Variables can either be **string** or **numeric**. The examples above create numeric variables, and it is usually easier to work with numeric variables where possible. To make a string variable use quotation marks:

```
gen job = "statistician"
```

Mathematical equations can also be used to generate new variables. For example, you may have variables describing the height and weight of a participants, which you can use to generate a new variable BMI.

5.9. Replace

Replace overwrites the values of variables which already exist. In the following example, therefore, we make a new variable called int_season (interview season) and set it equal to missing (.) for every observation. Next we use the replace command to replace values of int_season with new codes based upon the month in which the interview took place (int_month) – (i.e. Jan, Feb and March take the code 1 in int_season etc).

```
gen int_season=.
replace int_season=1 if int_month==1 | int_month==2 | int_month==3
replace int_season=2 if int_month==4 | int_month==5 | int_month==6
replace int_season=3 if int_month==7 | int_month==8 | int_month==9
replace int_season=4 if int_month==10 | int_month==11 | int_month==12
```

Note that `generate` (like other commands) requires only a single equal sign whilst 'if' conditions (unlike commands) take a double equal sign. Given that replace overwrites values you should be careful when using this command on existing variables – it is usually safer to use `generate` and then `replace` (as above), or `recode`, rather than overwriting existing variables (we will come to `recode` in a moment). Once a variable exists it is not possible to `generate` another variable with the same name and to make changes to this variable the `replace` command should be used, or you can drop the variable and `generate` it again.

Alternatively, to generate a new string variable called deprivation, set it equal to blank in all cases, then reset it with the word "lowincome" if the individual has low income (<10000) and "above10000" if the person does not have low income (>=10000 and not missing) you would type:

```
gen deprived = ""
replace deprived = "low income" if inc_lab < 10000
replace deprived = "above 10000" if inc_lab >= 10000 & inc_lab !=.
```

Always be careful with creating string variables – you will most certainly have to recode them at a later stage; that is why it is more useful to create numeric variables.

5.10. Label

The label command is used, firstly, to label variables and, secondly, to attach value labels to variables: labelling variables appropriately is helpful for future reference to remind yourself of what the variable actually is, whilst labelling values is pretty much essential. For example, we can label the int_season variable that we have created above as follows (this is called attaching a variable label):

```
label variable int_season "season of interview"
```

Above, we created four possible values for this variable using the replace command (1=jan, feb, mar; 2=apr, may, jun; 3=jul, aug, sep; 4=oct, nov, dec). We can now label these four codes as spring, summer, autumn and winter. Attaching labels to variable values has two steps.

Step one is to define the labels and the syntax for this is `label define` *`labelname value1 label1 value2 label2`*...... In this example therefore the syntax to define the label called seasons is:

```
Step 1     label define seasons 1 spring 2 summer 3 autumn 4 winter
```

If you want the label values to be more than one word then put quotation marks around them when defining the label (e.g. `label define seasons 1 "season of spring"`...)

Having defined the seasons label, step two is to attach this label set to the variable int_season. The syntax to do this is: '`label values` *`variable_name label_name`*'. In this example therefore the syntax to attach the seasons label to the int_season variable would be:

```
Step 2     label values int_season seasons
```

NB. The label exist independently of the variable and it is therefore possible to attach this label set to several variables if needed.

5.11. Recode

'`Recode`' is another commonly used command and is used to change the values (and, if you wish, value labels) of variables. '`Recode`' is designed for recoding numeric variables and if you wish to recode a string variable you should use '`generate`' and '`replace`' as in the example above. It is possible to recode a variable into itself (i.e. to replace a variable with a new version of itself) or to recode a variable into a new variable which is made in the process – it is often safer to recode into a new variable and to keep the original version of the variable. **To recode into a new variable the generate option is included; if this is not included then the original variable is overwritten.**

The 'recode' command in Stata also enables you to add value labels to the groups created at the same time. Therefore, it would be simpler to use recode to do what we have just done above (i.e. generate, replace, define value label, attach value label):

```
   drop int_season


recode int_month (1 2 3=1 spring) (4/6=2 summer) ///
(7/9=3 autumn) (10/12=4 winter) (.=.),  generate(int_season)
```

Note here that the 'gen' option at the end is where we specify that we want to recode into a new variable (int_season) rather than recoding int_month with a condensed version of itself – in this way the original int_month variable is not altered. The contents of the first bracket instructs Stata to take all codes of 1 (jan), 2 (feb) and 3 (march) in the original variable (int_month) and set them all equal to value 1 in the new variable, and label this value of 1 as spring. Note that 4/6 means 4 to 6 inclusive (i.e. 4, 5 & 6). Missing values remain coded as missing values in the new variable in the syntax above – note that it is not possible to label missing values using recode. Note that the categories in the recode statement evaluate sequentially. Other possible options instead of . are '-9' (relating to non-missing values, usually the missing values are transferred to missing automatically). In previous versions of Stata, 'miss' and 'nonmiss', 'else' (meaning all other values which have not yet been coded for) – it is not possible to use more than one of these three and if one is used then it is the last category specified before the options (as above). Again, it is also possible to recode multiple variables with the same coding structure at the same time. See the Help menu for more detail.

5.12. Rename

Now that we have created this new variable we might decide that we would rather name it differently. Unsurprisingly, the rename command is used to rename variables. To change the name of the int_season variable to season_of_interview therefore we would type:

```
   rename int_season season_of_interview
```

After being renamed, a variable retains all of its properties, such as format, variable label, value labels etc.

Exercise 3 Basic data management commands (20 minutes)

- *Read the bhps file in again to ensure you are using the original data*
- *Practice dropping/ keeping variables and observations*
- *Generate new variables, apply value and variable labels*

Task 1

- Delete (“drop”) all variables beginning with ‘toi’.

  Hint: the Stata code for this command described in section 5.1.

Task 2

- Sort the dataset so that the household with the lowest labour income (inc_lab) is at the top of the dataset.
  Check the data editor to verify the command worked.
  Hint: the relevant Stata code is described in section 5.7
- Now sort the data by year of the interview and the household ID

Task 3

- Assume that in your analysis, you are only interested in cases where the house_type is terraced, end terraced, semi-detached or detached. Only keep data for these types of houses.

  Hint: use the label list command to find out about value labels in the house_type variable. ***Remember that house_type is the variable name; hh_type is its label.***

  Hint: the Stata code to drop/ keep observations is described in sections 5.2 and 5.4. You will have to use an if-statement.

Task 4

- Create a binary variable (i.e. a variable that takes the values 0 or 1) called ‘data2005’ and make it equal to 1 for cases where the interview took place in 2005 and equal to 0 otherwise.
  Hint: the variable with information on the year of the interview is called int_year. The code to create and replace new variables is described in sections 5.8 and 5.9.
- Create a value label called *year_2005*, with the categories 1= “Yes (2005)” and 0 = “No (2006)”
  Hint: the Stata code to create a new value label is described in section 5.10.
- Then apply the new value label to the variable data2005, which you have just created.
  Hint: the relevant Stata code is described in section 5.10
- Finally, apply the following variable label to the newly created variable (data2005) : “Has the interview been performed in 2005?”
  Hint: the relevant Stata code is also described in section 5.10.

Task 5

- Sometimes, the *recode* command may be easier to use if new variables are created to include categories based on another variable.
- Therefore, use the *recode* command to create a new variable called ’data2006’. Base the new variable on the variable int_year and make it equal to 1 for cases where the interview took place in 2006 and equal to 0 if the interview took place in 2005. Make sure that values 1 are labelled “Year 2006” and values 0 are labelled “Year 2005”.
  Hint: details of the recode command are provided in section 5.11.
- Your supervisor is not happy with the variable name. Change it from ‘data2006’ to ‘Interview06’
  Hint: information on the rename command is provided in section 5.12

Task 6

- For each household, calculate the percentage of the total income that is derived from the labour income and name this variable lab_percent. You perform this calculation as follows:
  gen lab_percent = inc_lab / inc_total * 100
  Find out the mean value of this figure (type: su lab_percent)

6 Exploratory analyses – calculating sample statistics

The analyses in this section are unweighted but in reality it may be necessary to use weights if you are using survey data. Weighting is not considered within the remit of this course. (Weighting is a procedure that corrects for the over-sampling of a particular group and the deviation from proportional representation.)

6.1. Summarize (also summ or su)

For a range of basic statistics the `summarize` command is often the most useful. This gives basic statistics and adding the detail option provides additional useful information - percentiles, interquartile range, mean, median, standard deviation, etc. The syntax is simple:

```
summarize hhvalue

summ hhvalue, detail
```

Many (if not most) commands in Stata can be abbreviated and these can be found in the manuals and Stata help pages as the underlined part of the command: in the case of `summarize` this can be shortened to `summ` as above (or in fact it can be abbreviated further to sum or even `su`).

If we wanted summary statistics of house value for terraced houses only (i.e. house_type==4) then we would simply need to add an 'if' condition to the syntax:

```
summ hhvalue if house_type==4, detail
```

Alternatively, bysort can be used to calculate summaries for every category of the variable (as opposed to the example above where we used an 'if' condition to select just one category of the house_type variable). Bysort is very handy – it carries out a sort and at the same time conducts the command for the specified by-group(s). The syntax to calculate summary statistics of house value for each different category of house type separately would be as follows:

```
/*combine summarize with bysort*/

bysort house_type: summ hhvalue
```

Bysort can be abbreviated to bys to give:

```
bys house_type: summ hhvalue
```

6.2. Centile

The ‘`centile`’ command can be used to give more information about the percentile values within any variable if you want this information. For those unfamiliar with percentiles, they can be thought of as giving the value of the observation at a particular point in the distribution. For example, one common usage of percentiles in to break the data into ten equally sized blocks – deciles – and then to look at the differing compositions of the bottom or top deciles. Another use of centiles is to obtain the values of the median and interquartile range (although this can also be obtained with `summarize, detail`). The syntax to do this is:

```
centile hhvalue, c(25 50 75)
```

If we wanted to calculate values for each centile then the syntax would be:

```
centile hhvalue, c(10 20 30 40 50 60 70 80 90)
```

This could be shortened to:

```
centile hhvalue, c(10(10)90)
```

The first number in the brackets tells Stata which centile to begin with (10), the number in the central brackets tells Stata the size of the increments to move in (10) and the final number tells Stata which centile to go up to (90). This syntax therefore gives us the value for the deciles within this variable.

6.3. Xtile

‘`xtile`’ is used to create deciles, quintiles or other groupings. Whilst ‘`centile`’ tells us about the values of these groupings, `xtile` places our observations into the groupings. For example, if we wanted to divide the houses within the data into ten groups based upon their house value (i.e. deciles) and where group one was the 10% of houses with the lowest value and group 10 was the 10% of houses with the highest value then we could use `xtile`. The syntax would be:

```
xtile hhvalue_decile = hhvalue, nq(10)
```

If we wanted to create 5 groups of data (i.e. quintiles) then we would change the nq(10) to nq(5), and so on.

6.4. Tabulate – tables of frequencies and percentages

`'Tabulate'` is a commonly used command that gives tables of frequencies and percentages, either one-way or two-way. To get a simple one-way table of the number of rooms in the accommodation (rooms) for example the syntax is:

```
/**one-way tables - tabulate**/

tabulate rooms
```

If we thought that the number of rooms was likely to differ according to the type of house then we could do a two-way table of these two variables:

```
   /**two-way tables**/

   tab rooms house_type
```

`'Tabulate'` gives us frequencies by default but we can add in column or row percentages too, if we wish, by adding these as options. By default, tabulate does not report missing values and it is often sensible to report these so that analyses are not skewed by failing to consider any missing values. To include column percentages and to report missing cases within the table too, we would type:

```
/*include column percentages*/

tab rooms house_type, col miss
```

and to also add row and column percentages and take out the frequencies the syntax would be:

```
/*include row and column percentages*/

tabulate rooms house_type, col miss row nofreq


tab rooms house_type, row col chi2

/*gives you frequencies, row and column percentages and strength of
association*/
```

6.5. Tabstat

'Tabstat' is another command which creates tables of summary statistics. 'Tabstat' has the advantage of being particularly flexible: it allows you to run calculations for more than one variable at a time and to generate a number of different statistics which the user specifies.

For example, if we wanted to know the mean, range, min, max, standard deviation, median and interquartile range of the house value and house cost variables then this is easily done with 'tabstat':

```
tabstat hhvalue hhcost, ///
 stats(mean range min max sd p25 median p75)
```

Instead of using the bysort option at the start of the command line many commands also accept a by option to generate summaries by category:

```
tabstat hhvalue, by (house_type) ///
stats(mean min max range sd median p25 p75)
```

The order of the options does not matter and so we could equally have written:

```
tabstat hhvalue, ///
stats(mean min max range sd median p25 p75) by(house_type)
```

6.6. Table

The `'table'` command can also be used to provide statistical data for variables, many of which can also be produced using `'tabstat'` with `'bysort'` or the `'by'` option. The syntax structure for the `'table'` command is as follows, where c refers to the content which you want in the columns and the variables for which you want output for. Therefore, if we wanted to know the mean house value and standard deviation of the house value for each type of housing then the syntax would be:

```
table house_type, c(mean hhvalue sd hhvalue)
```

Note that the required summary statistic needs to be specified for each variable.

If we suspected that these values differed according to whether the property had a garden or not then we could run syntax with the garden variable as the by option. For each housing type, the syntax would tell us the mean number of rooms as well as the mean and standard deviation of house value for properties of each type with and without a garden:

```
/*are the values different for homes with and without a garden?*/
table house_type, c(mean hhvalue sd hhvalue) by(garden)
```

As mentioned above, the bysort option can also be used at the start of the command line as well as at its end – in many instances the exact syntax used will depend upon what works best in the situation needed. The syntax below is therefore equivalent to the previous syntax (although the output generated will be presented slightly differently):

```
bys garden: table house_type, c(mean hhvalue sd hhvalue)
```

Note that `'tabulate'` and `'table'` are different commands – `'tabulate'` produces frequencies and row/column percentages whilst `'table'` (and `'tabstat'`) produces summary statistics (mean, median, range, standard deviation, percentiles, etc).

6.7. Statsby

Finally, another command which is useful for creating a file of various summary statistics by groups is `'statsby'`, although this is more difficult to use. Every analysis command in Stata (e.g. summarize, tablehis, regress, etc) saves results, meaning that the results you see on the screen are temporarily held in memory and it is possible for the user to access them and use them. Just to note that `'statsby'` is a command which makes use of these saved results when analysing by sub-groups.

The basic structure of the command is:

```
statsby [exp_list] [, options ] :  command
```

For example, to analyse labour income for each house type using the 'summarize, detail' command and to save all the statistics generated this command type:

```
statsby , by(house_type) ///
 saving("H:\Output\income by house type.dta"): ///
 summ inc_lab, detail
```

Note that the [exp_list] part is blank in the above example so by default the command saves all the statistics generated by summarize, detail.

If only the mean and median were required then you could type:

```
statsby mean = r(mean) median = r(p50), ///
by(house_type) ///
saving("H:\Output\income by house type trim.dta", replace): ///
summ inc_lab, detail
```

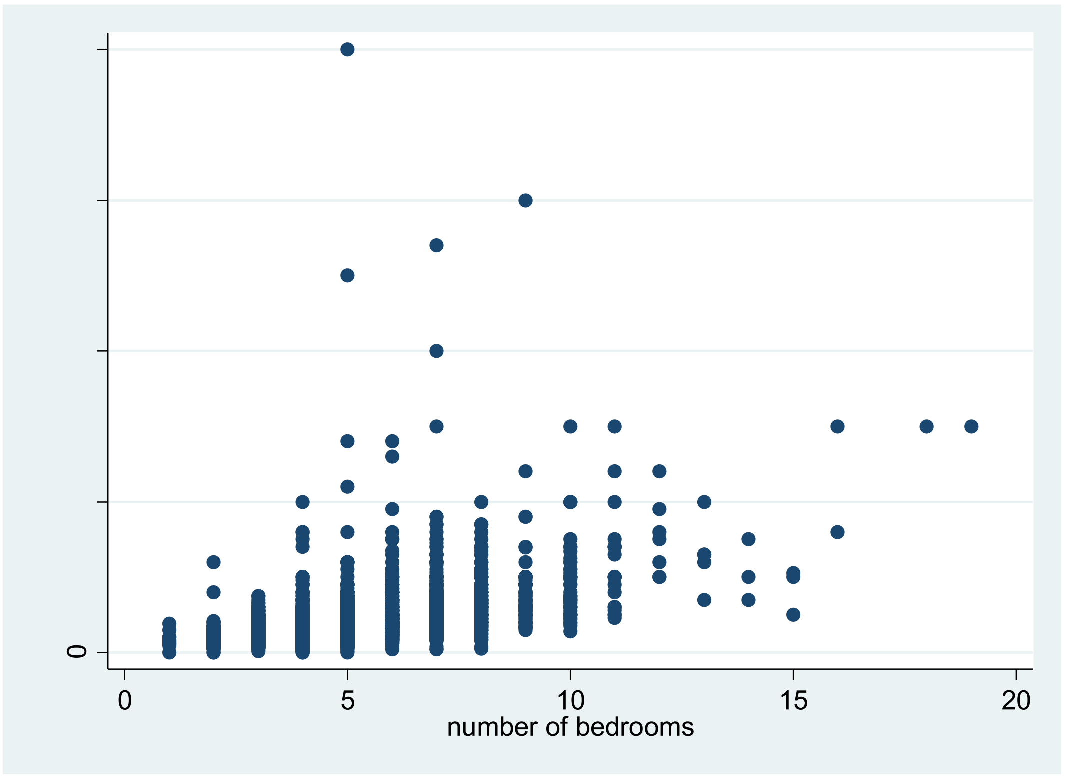
The same principle applies to any analysis command, not just to summarize as used in this example, as each analysis command saves results. The 'statsby' command is flexible but can be complex so it is worth checking Stata's help menu or the manuals for more details if you need to use this command.

6.8. Drawing a scatterplot

You may want to examine the relationship between the variables through a scatterplot. For example, we can hypothesize that as the number of rooms in a house increase, so does it price. To check this type:

```
graph  twoway scatter hhvalue rooms
```

This will produce the following graph.



The graph shows that the relationship is far from linear and there are some notable outliers.

More advanced plot options are covered in the additional Stata courses.

Exercise 4 Summarize the data (20 minutes) • *Read the bhps file in again to ensure you are using the original data* • *Perform univariate statistical analyses and produce summary tables*
Task 1 • Use the *summarize* command to find the minimum, maximum and median values of the hhcost variable. *Hint: use the detail option in the summarize command. The summarize command is described in section 6.1.*
Task 2 • Combine *summarize* with *bysort* to see how the minimum, maximum and median house value varies according to whether house has a garden or not.
Task 3 • What is the median value of a terraced house? *Hint: the median is the 50th percentile. Use the 'detail' option in the summarize command.*
Task 4 • Using *tabulate*, create a two-way table of details on car ownership within the household (car_own) and whether the property has a garden (garden). *Hint: Details on the relevant Stata command are provided in section 6.4.*
Task 5 • Use *tabstat* with the *by option* to calculate the mean, interquartile range and standard deviation of the house value of each property type. *Hint: 'by' should come in the end. The relevant Stata command is described in section 6.5*
Task 6 • Run question 5 again but use the bysort option at the start of the command line rather than the by(house_type) option. Note how the presentation of the output is different but the values calculated are identical.
Task 7 • Use *xtile* to create a new variable *inc_total_decile* which has 10 categories representing the deciles of the total household income (*inc_total*). How many cases are in each decile? *Hint: the xtile command is described in section 6.3. Use the tabulate command to find the number of cases.*

Task 8

Use the *'table'* command to produce output showing the mean food expenditure (exp_food), the maximum number of children in a household(kids) and the median total household income (inc_total) for different categories of car ownership (car_own).

Hint: details on the table command are provided in section 6.6. Use the help function (type help table) to find out how to generate the different summary statistics.

Task 9

- Create a *scatter plot* with the number of rooms (room) on the y-axis and the number of children (kids) per household on the x-axis
  Hint: a brief summary of the scatter command is provided in section 6.8.

7 Appendices

7.1. Help in Stata

This course is part of a series of Stata courses run through the IT Learning Centre which are designed to provide sufficient knowledge for people to become comfortable users of Stata for a range of intermediate and advanced tasks. Details are on the IT Learning Centre website; courses will be repeated each term.

There is also plenty of Stata help available in other places if you need it:

- The 'help' menu in the main Stata window
  - contains options for 'search commands' if you want to find out details and exact syntax for a particular command (this is similar to the material in the manuals but there is often more detail and examples here than in the manual)
  - allows you to search the net or FAQs
- You can just type help and then the name of the command into the command window and this will take you to the help pages for that command
- The manuals are arranged alphabetically and provide much of the information that is in the help menu
- There is plenty of material on Google – often helpful if you're sure there's a command to do it but just can't remember its name
- UCLA web resources are very good: http://statcomp.ats.ucla.edu/stata/default.htm
- The Princeton website is good: http://dss.princeton.edu/online_help/stats_packages/stata/

- Stata's own website is good www.stata.com. In particular, they run NetCourses and list short courses. Stata is unusual in that there is an active web community for very keen users. One useful thing is that Stata users write programmes (ado-files) to do particular tasks which are available online. These can be freely downloaded to your PC so that you can use them in Stata – this is one benefit of the programming flexibility of Stata and these .ado files can be handy for particular, often difficult tasks. The fourth course in this series, *Introduction to programming in Stata*, discusses .ado files.

7.2. Additional Literature

Baum, C. (2009). "An Introduction to Stata Programming", Stata Press Publication

Long, S. and Freese, J. (2006). "Regression models for categorical dependent variables using Stata", Stata Press Publication

7.3. Variables in the Dataset

```
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
hhid            long   %12.0g                 household identification number
int_day         byte   %8.0g                  day of interview
int_month       byte   %9.0g       month      month of interview
int_year        int    %8.0g                  year of interview
house_type      byte   %20.0g      hh_type    type of accommodation
rooms           byte   %8.0g                  number of bedrooms
tenure          byte   %20.0g      tenure     house owned or rented
hhvalue         long   %12.0g                 value of property: home owners
hhcost          long   %12.0g                 original purchase price of
                                                property
monthly_mortg~e int    %8.0g                  last total monthly mortgage
                                                payment
toilet_indoor   byte   %8.0g       loo_in     accom: has indoor toilet
toilet_shared   byte   %8.0g       loo_sh     accom: is indoor toilet shared
garden          byte   %8.0g       gdn        accom: has terrace/garden
total_mortgage  long   %12.0g                 total mortgage on all property
exp_food        int    %8.0g                  total monthly food and grocery
                                                bill
vehicle_access  byte   %8.0g       veh_acc    car or van available for private
                                                use
car_own         byte   %22.0g      car_own    household member owns vehicle
car_value       long   %12.0g                 value vehicle(s) less amount
                                                outstanding
hhsize          byte   %8.0g                  number of persons in household
kids            byte   %8.0g                  number of children in household
pens            byte   %8.0g                  number over pensionable age in
                                                household
emp             byte   %8.0g                  number in employment in household
wage            byte   %8.0g                  number in household of working
                                                age
inc_total       double %10.0g                 annual hh income
                                                (1.9.2004-1.9.2005)
inc_lab         double %10.0g                 annual household labour income
inc_nonlab      double %10.0g                 annual household non-labour
                                                income
inc_pens        double %10.0g                 annual household pension income
inc_bens        double %10.0g                 annual household benefit income
inc_inv         double %10.0g                 annual household investment
                                                income
hh_wt           double %10.0g                 household weight -within uk
                                                estimates
int_place       str6   %9s                    interview location
int_dur         float  %9.0g                  interview duration
country         str3   %9s                    Country of interview
```

7.4. Answers to the exercises

A sheet with the answers to the exercises is provided at the end of the course.

7.5. Do-file for this session

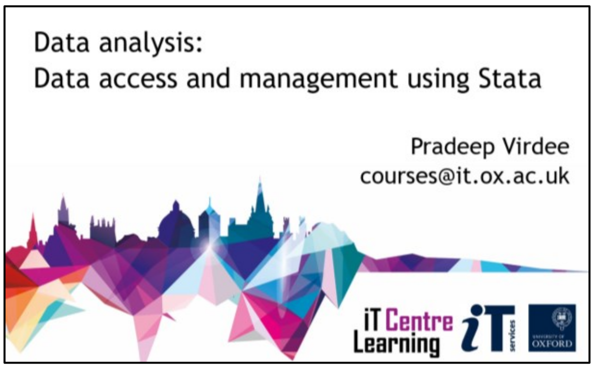
```
/* Stata Course 1: Introduction to Stata - Getting started and basic commands */
* Syntax file to accompany course booklet updated in January 2015 *

clear
set more off

set more on

/* open data set in STATA format from specified file location*/
use "H:\Data\Stata\Original\bhps.dta", clear

*If no file format is specified, Stata, by default, assumes that .dta
* format is
use "H:\Data\Stata\Original\bhps", clear

*opening data in excel format
import excel using "H:\Data\Excel\Original\excel_names.xls", ///
 clear firstrow

*the firstrow option is important to read in variable names saved in the
* first row of the data set. Compare results to:
import excel using "H:\Data\Excel\Original\excel_names.xls", ///
 clear firstrow

*save data in Stata format
save "H:\Data\Stata\Modified\bhps1.dta"

*if we wanted to make changes, and save the dataset again under the
* same name, we have to use the replace option:
save "H:\Data\Stata\Modified\bhps1.dta", replace

*to ensure the datasets can be used in previous versions of Stata
* (version 12 and earlier)
saveold "H:\Data\Stata\Modified\bhps_old.dta", replace


*save the data in Excel format:
export excel using "H:\Data\Excel\Modified\bhps1.xls", ///
 replace firstrow(variables)

*we could also save the variable lables in the first row of the excel sheet:
export excel using "H:\Data\Excel\Modified\bhps1_labs.xls", ///
 replace firstrow(varlabels)

*********************************************************************************

*EXERCISE 1 – Go to page 16 of your course book

*********************************************************************************

/**** Getting a feel for the data - taking a first look at the variables ****/
*make sure data is read in:
use "H:\Data\Stata\Original\bhps.dta", clear

*or - use the "change directory" option if data if opened from/ saved to a
* similar location
cd "H:\Data\Stata"
use "Original\bhps", clear

/*describe variables - this describes the variables and sets out what their
value labels are called*/
describe _all

describe hhid-garden

describe inc*
```

```
/*list the value labels of house_type*/
label list hh_type

/*list the first ten values of house_type in the dataset*/

list house_type in 1/10

*or look at missing values specifically:
list hhid car_own if car_own == .

/*how many observations have house_type of value 4 (ie terraced house)*/

count if house_type==4

*codebook command
codebook car_own

*********************************************************************************

**EXERCISE 2 – Go to page 23 of your course book

*********************************************************************************


/***** Managing and manipulating the data - basic data management *****/

/***drop & keep***/

/*drop variables*/

drop monthly_mortgage total_mortgage car*


/*drop observations*/

describe house_type

label list hh_type

drop if house_type>999 | house_type==9 | (house_type>=13 & house_type <=15)

tab house_type, miss
*to see that categories 9, 13, 14, 15 are no longer in the dataset.
* missing data have also been deleted

/*NB normally missing is . and this is the highest numeric
value: beware! For a string variable missing is "" */

/*keep */

/*1. keep variables*/

keep hhid-garden inc* hh_wt-int_dur

/*2. keep observations if they are 2005 or 2006. NB: if condition requires
double == Alternatives are > < >= <= !=/~= */

keep if int_year == 2005 | int_year == 2006

/*brackets may be needed for more complex if statements*/

/*preserve and restore* - two important ways to come back to the original data.
However, do not always count on that. Always save a copy of your original file
to which you can come if need be. Remember to always preserve exactly before the
action you want remedied. Stata does not keep any 'preserve' memory.*/
use "H:\Data\Stata\Original\bhps.dta", clear
```

```
preserve
drop pens
drop if car_own == 3

tab car_own
restore

tab car_own
/** browse **/

browse hhvalue house_type
*this makes it easier to look at certain variables only, and will be
* particularly useful when calculations/ derivations are to be validated


/***sort***/
use "H:\Data\Stata\Original\bhps.dta", clear

/* sort by house value (hhvalue) (sort does ascending sort. gsort with negative
sign used for descending sort (see below)*/


/*descending sort*/

sort garden
gsort +garden

gsort -hhvalue

/* sorting with multiple variables */

sort tenure hhvalue

gsort +tenure -hhvalue



/**generate**/

/*like many Stata commands generate can be abbreviated - gen is the abbreviated
command*/

/*make a new variable called int_year2 and set it equal to the value of int_year*/

gen int_year2=.
gen int_year3 =100
gen int_year4 = "two thousand and six" if int_year == 2006
*be careful to use quotation marks when creating string variables



/*generate/replace*/

/*NB replace writes over variables which already exist, usually . or 0.
Otherwise, be careful when writing over variables and it's usually safer to recode
into new variables*/

gen int_season=.
replace int_season=1 if int_month==1 | int_month==2 | int_month==3
replace int_season=2 if int_month==4 | int_month==5 | int_month==6
replace int_season=3 if int_month==7 | int_month==8 | int_month==9
replace int_season=4 if int_month==10 | int_month==11 | int_month==12

/* generating string variables*/

gen deprived = ""
replace deprived = "low income" if inc_lab < 10000
replace deprived = "above 10000" if inc_lab >= 10000 & inc_lab !=.


/**label**/

/*label variable*/

label variable int_season "season of interview"


/*label values*/

/*adding value labels to variable codes - there are two steps to
attaching labels to a variable*/

/*Step one: define a label*/

label define seasons 1 spring 2 summer 3 autumn 4 winter

/*Step two: attach the label to the variable (note that it is therefore possible to
attach
this set of labels to more than one variable*/

label values int_season seasons



/**Alternatively, we could have used the recode command to do this - here we recode
into
a new variable which we make (int_season)**/

drop int_season

recode int_month (1 2 3=1 spring) (4/6=2 summer) (7/8 9=3 autumn) ///
 (10/12=4 winter)(.=.), gen(int_season2)

/*NB it is also possible to recode multiple variables with the same coding
structure
at the same time*/



/**rename**/

rename int_season season_of_interview


**********************************************************************************

** EXERCISE 3 – Go to page 30 of your course book

**********************************************************************************


/**** Exploratory analyses - simple statistics ****/

/**summarize (can be written out in full or abbreviated to summ)**/

summarize hhvalue

/*the detail option with summarize gives lots of useful information - percentiles,
interquartile range, mean, median, standard deviation, etc*/

summ hhvalue, detail
su hhvalue, detail

summ hhvalue if house_type==4, detail
label list hh_type
```

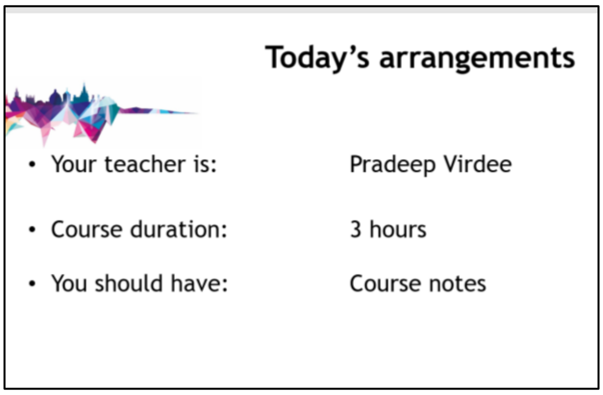
```
*look only at terranced houses

bysort house_type: summ hhvalue
bys house_type: summ hhvalue



/**centile**/

centile hhvalue, c(10)

/*calculate median and interquartile range values*/

centile hhvalue,c(25 50 75)

/*calculate values for each centile between 10 and 90 i.e. start at the
first value (10) and move in increments of the value in brackets (10)
until reaching the third value (90)*/

centile hhvalue, c(10 20 30 40 50 60 70 80 90)

centile hhvalue, c(10(10)90)


/** xtile - creates new variables based on centiles**/

xtile hhvalue_decile=hhvalue, nq(10)
xtile hhvalue_quincile=hhvalue, nq(5)



/**one way tables - tabulate**/
tabulate rooms


/**two way tables**/

/*frequencies*/
tabulate rooms house_type

/*include column percentages*/

tabulate tenure garden,col

/*include row and column percentages*/
tabulate tenure garden,col row

*could also add some statistical tests:
tab tenure garden, row col chi2


/** There are often many ways of doing things in Stata **/
/*Summarize hhvalue by house type. bysort can be used with many
commands to run commands on bygroups (i.e. categories/sub-groups of a variable)*/

bysort house_type:summ hhvalue,detail
bys house_type:summ hhvalue,detail


/**tabstat - Another useful command which can be used to do similar things in
tabstat - this has a large range of statistics that can be selected as options**/

tabstat hhvalue hhcost, stats(mean range min max sd median p25 p75)

tabstat hhvalue, stats(mean min max sd p25 median p75) by(house_type)
*maybe this option produces the nicest output
```

```
/**table command: NB that table and tabulate are different commands. Tabulate
 gives frequencies whilst table is used for more statistical analyses**/

table house_type,c(mean rooms mean hhvalue sd hhvalue)

/*are there values different for homes with and without a garden?*/

table house_type,c (mean rooms mean hhvalue sd hhvalue) by(garden)

/*or rather than using the by option at the end you could use the by sort option
at the start of the syntax line */

bys garden:table house_type,c (mean rooms mean hhvalue sd hhvalue)


/** statsby - to create datasets with the desired summary statistics **/

statsby , by(house_type) ///
 saving("H:\Output\income by house type.dta"): ///
 summ inc_lab, detail

*if only certain output is required, the following information can be provided:
statsby mean = r(mean) median = r(p50), by(house_type) ///
 saving("H:\Output\income by house type trim.dta", replace): ///
 summ inc_lab, detail


*Scatter plots:
*to look at the relationship between the value of a house and the number of
bedrooms
graph  twoway scatter hhvalue rooms


*********************************************************************************

**EXERCISE 4 – Go to page 38 of your course book

*********************************************************************************
```



Data analysis:
Data access and management using Stata

Pradeep Virdee
courses@it.ox.ac.uk



Today's arrangements

- Your teacher is: Pradeep Virdee
- Course duration: 3 hours
- You should have: Course notes

Your safety is important

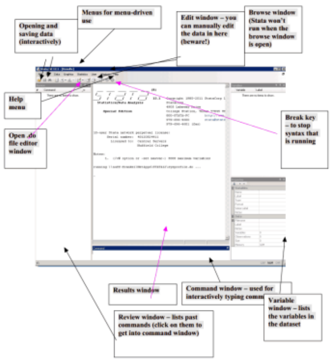
- Where is the fire exit?
- Beware of hazards:
  Tripping over bags and coats
- Please report any equipment faults to us
- Let us know if you have any other concerns

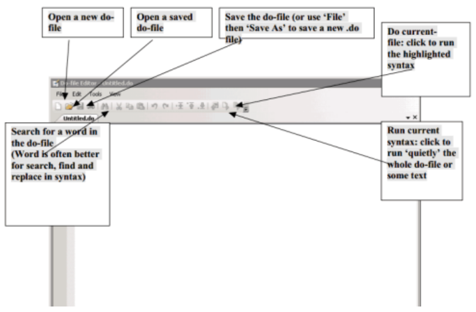
Your comfort is important

- The toilets are along the corridor outside the lecture rooms.
- The rest area has vending machines and a water cooler.
- The seats at the computers are adjustable.
- You can adjust the monitors for height, tilt and brightness.

Today's objectives

- To compile and execute .do files
- Read data into Stata
- To describe and list variables
- To perform some basic data manipulations
- To understand how to use basic univariate statistical computations
- To save Stata .do files


Opening and saving data (interactively)
Menus for menu-driven use
Edit window – you can manually edit the data in here (beware!)
Browse window (Stata won't run when the browse window is open)
Help menu
Open .do file editor window
Break key – to stop syntax that is running
Results window
Command window – used for interactively typing comm
Variable window – lists the variables in the dataset
Review window – lists past commands (click on them to get into command window)


Open a new do-file
Open a saved do-file
Save the do-file (or use 'File' then 'Save As' to save a new .do file)
Do current-file: click to run the highlighted syntax
Search for a word in the do-file (Word is often better for search, find and replace in syntax)
Run current syntax: click to run 'quietly' the whole do-file or some text
Do-file Editor - untitled.do
File Edit Tools View
Untitled.do

General command structure in Stata

command *variable(s)* [if] [in], options

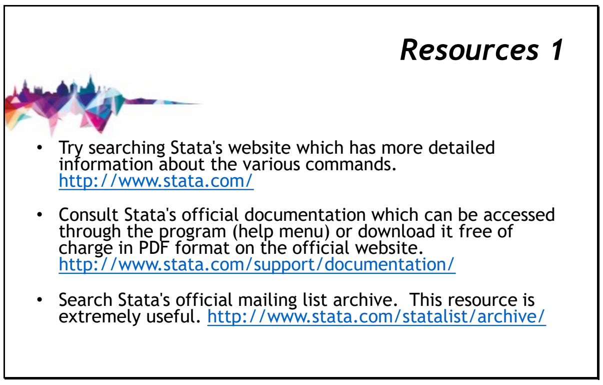
Troubleshooting 1

Today:

During each practical today, we will be able to help you with any problems you are experiencing.

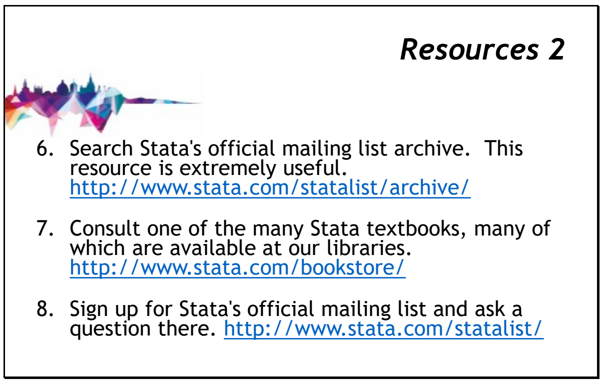
After class

1. Make sure you correctly typed the command and variable name(s) – command names should appear blue in the .do file
2. If you know the exact name of command that you are having difficulties with try using Stata's help function.



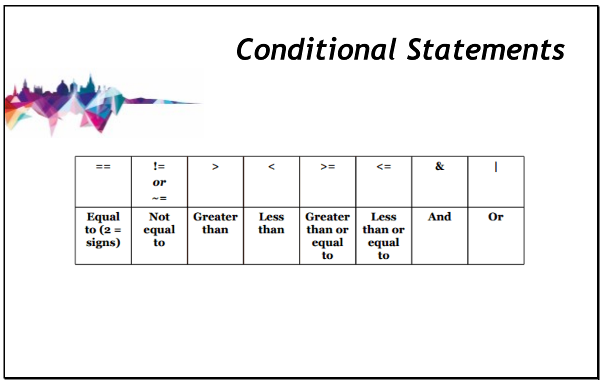
Resources 1

- Try searching Stata's website which has more detailed information about the various commands. http://www.stata.com/
- Consult Stata's official documentation which can be accessed through the program (help menu) or download it free of charge in PDF format on the official website. http://www.stata.com/support/documentation/
- Search Stata's official mailing list archive. This resource is extremely useful. http://www.stata.com/statalist/archive/



Resources 2

6. Search Stata's official mailing list archive. This resource is extremely useful. http://www.stata.com/statalist/archive/
7. Consult one of the many Stata textbooks, many of which are available at our libraries. http://www.stata.com/bookstore/
8. Sign up for Stata's official mailing list and ask a question there. http://www.stata.com/statalist/



Conditional Statements

==	!= *or* ~=	>	<	>=	<=	&	|
Equal to (2 = signs)	Not equal to	Greater than	Less than	Greater than or equal to	Less than or equal to	And	Or