Lesson 1: Getting Started

This lesson is a gentle introduction to Stata, including why it's used, its interface, do files (the script files Stata uses), how to load in and save data, how to make use of the help system, setting the working directory, using the data browser and editor, and importing, exporting, appending and merging data.

I use Stata version 14.2 in the videos, but the overall look and the basic programs in Stata haven't changed markedly since Stata 12.

1.1 Why Use Stata?

Stata is a general-purpose statistical package, with the following advantages:

- It is quick
- It handles data very well
- The graphics facilities are (reasonably) good
- Users can write new commands, so Stata can be and is constantly updated
- Writing code for analyses is intuitive and (relatively) easy to understand
    - Compared to R and Python, I prefer writing code in Stata, and the Stata help files are usually much better than the equivalent R and Python help/man pages

Stata is best used by typing commands in the command window or, even better, in Stata's text editor, which is called a do file (see **Section 1.3**). In do files, you can write and execute several commands individually or together. Data manipulation and analysis is carried out through these commands, so it's simple to repeat an analysis if the commands are saved in a do file. Stata can also be used in a point-and-click way through using the drop-down menus at the top of the screen, but I won't be using these in the lessons and don't recommend using them in practice: writing out script and recording it in a do file is replicable, and I think probably the fastest way of learning how to use Stata, even if it's more difficult to start with.

Throughout this workbook, commands are written in **`bold courier font`**, the same font used in Stata. For example, most Stata commands follow the same format:

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

where:

- **`command`** is the name of the command to be used, e.g. **`summarize`**
- **`{variable(s)}`** is a list of one or more variables, e.g. age
- **`if`** … restricts the command to a selection of observations that fulfil a criterion, e.g. participants that are over 40 years of age
- **`in`** … restricts the command to a selection of observations based on their order, e.g. the first 50 observations
- **`, [options]`** - any options a command can use, always typed after a comma

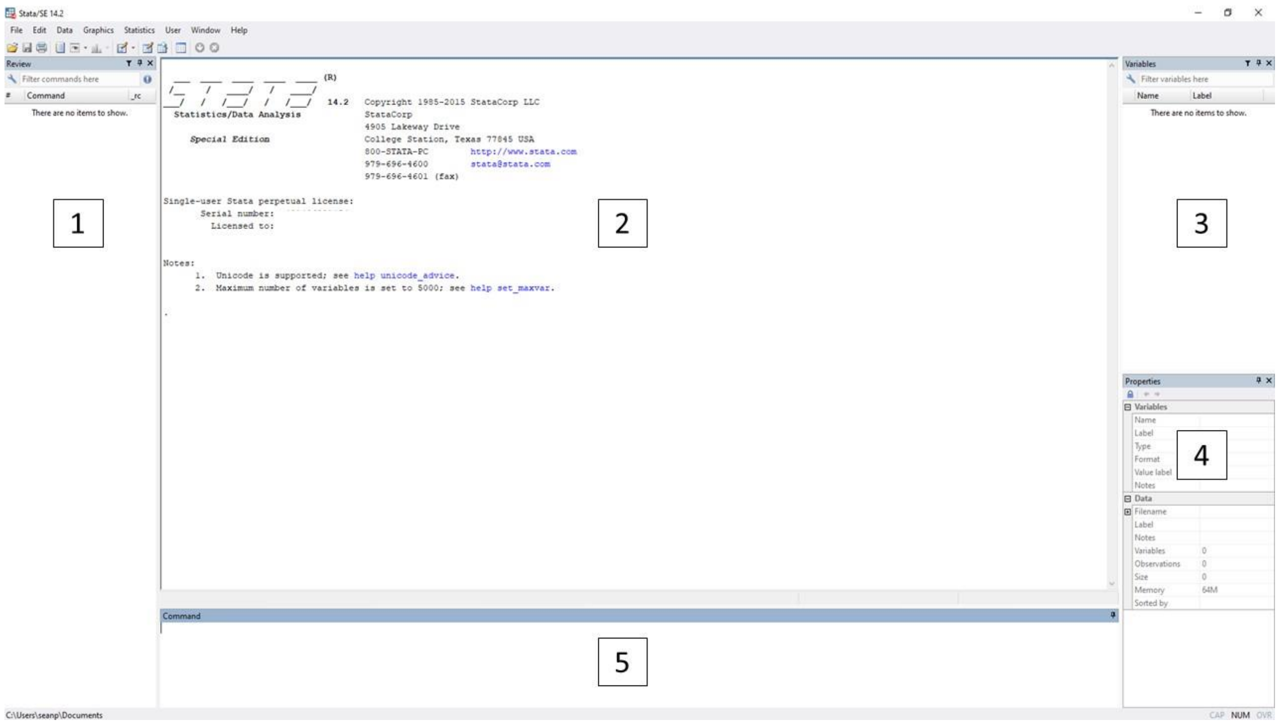
Most commands require just the command name and a list of one or more variables, though some commands either require or allow you to specify some options too. Commands can usually be shortened to save time, for example the **`tabulate`** command (wee use this later) can be shortened to **`tab`**. After each command is introduced, I'll put the short form (if any) I typically use as well, and after the first example, I will use the shortened form.

During this course, we'll go through plenty of examples of useful commands.

1.2 Stata's Interface

Stata opens with five main windows, and each window gives us different information:

1. Review: shows all commands that have been run in this Stata session
2. Results: displays the output of commands
3. Variables: displays a list of all variables in a dataset, as well as their labels
4. Properties: shows summary information about the current dataset
5. Command: you enter commands here



```
  ___  ____  ____  ____  ____ (R)
 /__    /   ____/   /   ____/
___/   /   /___/   /   /___/   14.2   Copyright 1985-2015 StataCorp LLC
  Statistics/Data Analysis            StataCorp
                                      4905 Lakeway Drive
     Special Edition                  College Station, Texas 77845 USA
                                      800-STATA-PC        http://www.stata.com
                                      979-696-4600        stata@stata.com
                                      979-696-4601 (fax)

Single-user Stata perpetual license:
       Serial number:
         Licensed to:

Notes:
      1.  Unicode is supported; see help unicode_advice.
      2.  Maximum number of variables is set to 5000; see help set_maxvar.

.
```

1.3 Do Files

Stata's text editor, a do file, is opened by clicking this button (below the **Graphics** and **Statistics** menus on my screen):

Do files allow you to write out and save all the commands you want to run. This is extremely useful for keeping track of commands, and for editing and re-running analyses.

To execute the commands in do files, highlight the line or block or lines that you want to run and press **ctrl+d**, or click this button in the do file editor:

If you are following along with the videos, feel free to either type commands into the command window (number 5 above – this is what I'll be doing to save switching between windows), or type the commands into the do file and run them individually (this is how I work in practice). I recommend using a do file so you can keep a copy of all the commands we'll use, though I've provided do files with all the commands we'll use in lessons.

In a do file, every command needs to be on a separate line. If you want to extend the command across multiple lines, you need to type `///` at the end of each line where you want to continue to the next line.

Additionally, if you want to insert a comment into a do file (a line of text that isn't run as code, which appears green in newer versions of Stata), then start a new line with an asterisk (`*`). If you want a whole section to be comments (or just not run), then start a new line with `/*` above where you want to comment to start, and `*/` on a new line just below where you want the comment to end. Finally, if you want to add a comment at the end of a command, type in `//` and any following text will be a comment.

1.4 The Dataset

I've created a simulated dataset that we'll be looking at in this lesson, **Lesson_01.dta**, which contains information for the following variables:

1. ID number
2. Age (years)
3. Sex (male or female)
4. Height (cm)
5. Weight (pounds)
6. Hair colour (black, blonde, brown, grey, red)
7. Systolic blood pressure (mmHg)
8. Diastolic blood pressure (mmHg)
9. Accommodation (own, mortgage, rent)
10. Exercise per week (numeric categories)
11. Calories consumed per day
12. Income (£s)
13. Run a marathon in the past year (yes or no)
14. Current smoker (yes or no)
15. Favourite colours
16. Date of blood pressure measurement
17. Date of recruitment

I've made it so that the dataset needs some editing before it can be analysed, which we'll cover in the next couple of lessons.

1.5 Loading and Saving Data

There are many ways to load a Stata data file (a **.dta** file); one of the simplest ways is to click the "open" icon as for any other program. This loads the dataset into Stata memory, and displays the command Stata used to do this:

```
use "{folder}\Lesson_01.dta", clear
```

You could have typed in the command directly, either into the command window or a do file. You can also load in a dataset by dragging and dropping a Stata data file directly into the results window. You probably won't see the option I've included above, `clear`, as this only appears automatically when you are loading in data on top of existing data that isn't saved. We'll go through more about options and Stata trying to stop you losing data in a bit. For now, bear in mind that *Stata has no undo button*: changes are permanent, which is presumably why Stata is quite keen for you to not accidentally overwrite or delete data. Also bear in mind that when typing folders or file names, you'll likely have to enclose the locations in quotation marks.

Once loaded into memory, you will see all the variables in the dataset in the upper right-hand window. The lower right-hand window displays some information about the dataset, for example that it has **16 variables** and **1000 observations**, i.e. 16 columns of data and 1000 rows, excluding the names of the variables.

We are now going to save a copy of the dataset to work with: in general, it is safest to always keep a master, unchanged copy of a dataset, and work on a copy. Because you can save the commands you use to clean and analyse a dataset in a do file, you don't need to change the master copy at all. This means that if you make a mistake, then you can go back and just fix the do file, then re-run all the code.

As with opening a dataset, there are a couple of options for saving a dataset. You can either use the "save" icon and choose a folder to save in, or use the **`save`** command:

```
save "{folder}\Lesson_01_{initials}.dta", replace
```

Anything in curly brackets in this workbook needs to be replaced by something else, in this case your initials. The option `replace` allows Stata to overwrite any saved Stata files with the same name in the same folder. It is optional, but Stata will give an error message if any file exists with the same name. If you don't specify the folder location, Stata will save to the folder acting as the *working directory* (see **Section 1.7**).

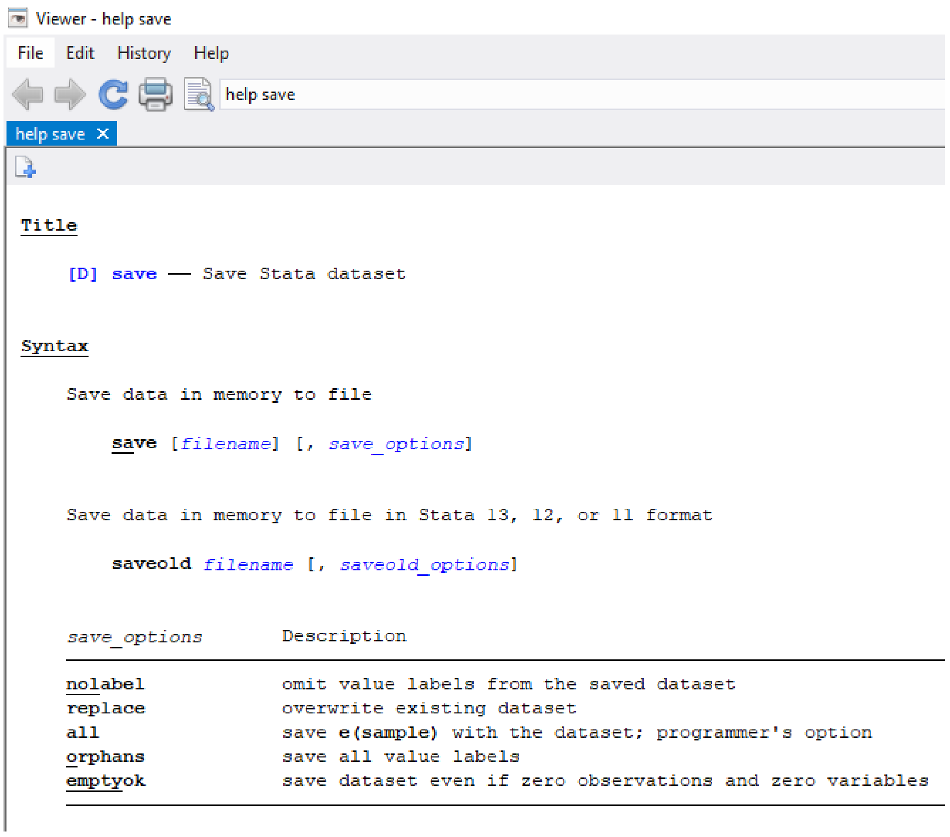
1.6 Help

Stata has an extensive help system, where every command has its own help page. General help pages for different topics are also available. We've now seen the commands **`use`** and **`save`**, so could look up the help files for each. For any command, just type:

```
help {command}
```

where **`{command}`** is the name of the command you want to look up. Let's try:

```
help save
```



```
Title

    [D] save — Save Stata dataset


Syntax

    Save data in memory to file

        save [filename] [, save_options]


    Save data in memory to file in Stata 13, 12, or 11 format

        saveold filename [, saveold_options]


    save_options          Description
    ---------------------------------------------------------------------------
    nolabel               omit value labels from the saved dataset
    replace               overwrite existing dataset
    all                   save e(sample) with the dataset; programmer's option
    orphans               save all value labels
    emptyok               save dataset even if zero observations and zero variables
    ---------------------------------------------------------------------------
```

This brings up the help file for the **`save`** command. The **`Syntax`** section details how the command needs to be written to be used. Stata syntax (the grammar of Stata) differentiates between bits of the command that are necessary and those that are optional: parts of the command in square brackets are optional. Here, you can see the **`[filename]`** and **`[, save_options]`** are optional because they are in square brackets, so you could just type **`save`** as a valid command.

The help files will always list the syntax of the command, detail any options, and usually give more information and some examples. However, not all commands are written by Stata, and for user-written commands, help files can be of variable quality. In my experience, the help files for almost all programs are pretty good though.

1.7 The Working Directory

The *working directory* of Stata is the folder in which Stata operates – you can see in the bottom left of the screen where Stata is currently operating. This is the folder Stata looks to load and save datasets if you don't provide a full file path, meaning you can save some time and effort setting your working directory at the top of a do file or start of a Stata session, and only specifying file names when loading and saving data.

The necessary command for setting the working directory is:

```
cd "{folder}"
```

The folder location can either be typed out, or by selecting the *copy address as text* option when right clicking on the address bar of the folder you want Stata to work in, or by copying the folder location from the **`use`** command we used earlier.

The command **`cd`** stands for *change directory*.

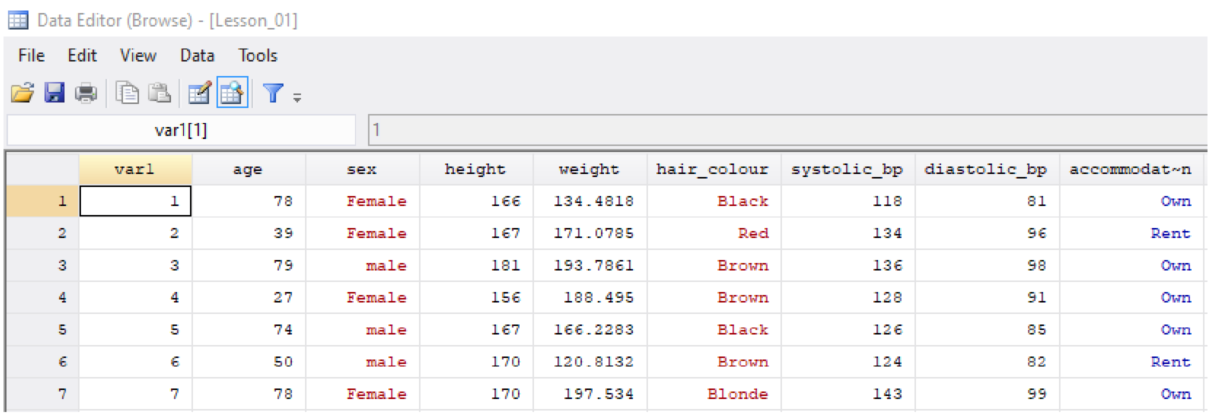
1.8 Looking at Your Data

A Stata dataset can be thought of as a spreadsheet, containing some number of columns (variables) and rows (observations). Each variable has a unique name which can be used to specify the variable in commands. There are rules about variable names:

- Variables have to start with a letter or an underscore
- Variables can't have more than 32 characters
- Variable names can't be used more than once
- Variable names can't have spaces or symbols in them (except an underscore)

Stata is also *case-sensitive*. This means that Stata will regard, for example, **Age** and **age** as two unique and completely separate variables. For this reason, it is usually recommended to keep all variable names in lower case (it's also easier to write in code if it doesn't need capitals).

The data can be directly viewed using the **Data Editor** buttons (below the **Statistics** and **User** menus on my screen): The left-hand button is the **`edit`** button, and the right-hand button is the **`browse`** button. Both buttons bring up a spreadsheet view of the dataset, but you can directly edit data using the **`edit`** button. This isn't usually recommended, as it's easy to make mistakes directly editing data and it isn't replicable, but there are situations where it's useful. Although the commands don't come up when you click these buttons, you can type **`edit`** and **`browse`** into the command window to bring up the same spreadsheet views.



	var1	age	sex	height	weight	hair_colour	systolic_bp	diastolic_bp	accommodat~n
1	1	78	Female	166	134.4818	Black	118	81	Own
2	2	39	Female	167	171.0785	Red	134	96	Rent
3	3	79	male	181	193.7861	Brown	136	98	Own
4	4	27	Female	156	188.495	Brown	128	91	Own
5	5	74	male	167	166.2283	Black	126	85	Own
6	6	50	male	170	120.8132	Brown	124	82	Rent
7	7	78	Female	170	197.534	Blonde	143	99	Own

You'll notice that the values in most variables are black, while some values are red and some blue. These correspond to different data types: black values are numeric (i.e. composed entirely of numbers and missing values), red values are strings (i.e. text or characters, e.g. "male", "brown", "Blue; Yellow; Red"), and blue values are labelled numbers (e.g. 1 = "Own", 2 = "Mortgage", 3 = "Rent" etc.). This distinction is made for each variable and is set once the data is created or loaded into Stata. You can't add strings to numeric variables, or numbers to string variables (unless they are put in quotation marks to become strings). This means it's really important to know what kind of variable you're working with at any time, since numbers and strings are dealt with entirely differently: the general rule is that strings need to be enclosed with quotation marks, otherwise Stata gets confused. Labelled numeric variables act as numbers, but are labelled, so whenever you look at them (or use a command to summarise information about the variable), you see something more informative than simple numbers. This is especially useful for binary and categorical variables, i.e. variables with 2 or more possible options.

You’ll notice full stops (periods) in the numeric variables – these are missing values. Missing values are coded in Stata as incredibly large numbers, but this doesn’t come up much apart from in the context of *if statements*, which we deal with in **Lesson 3**. All you need to remember for now is that if you see a full stop in a numeric variable, it’s a missing numeric value. Missing string values, on the other hand, are literally blank.

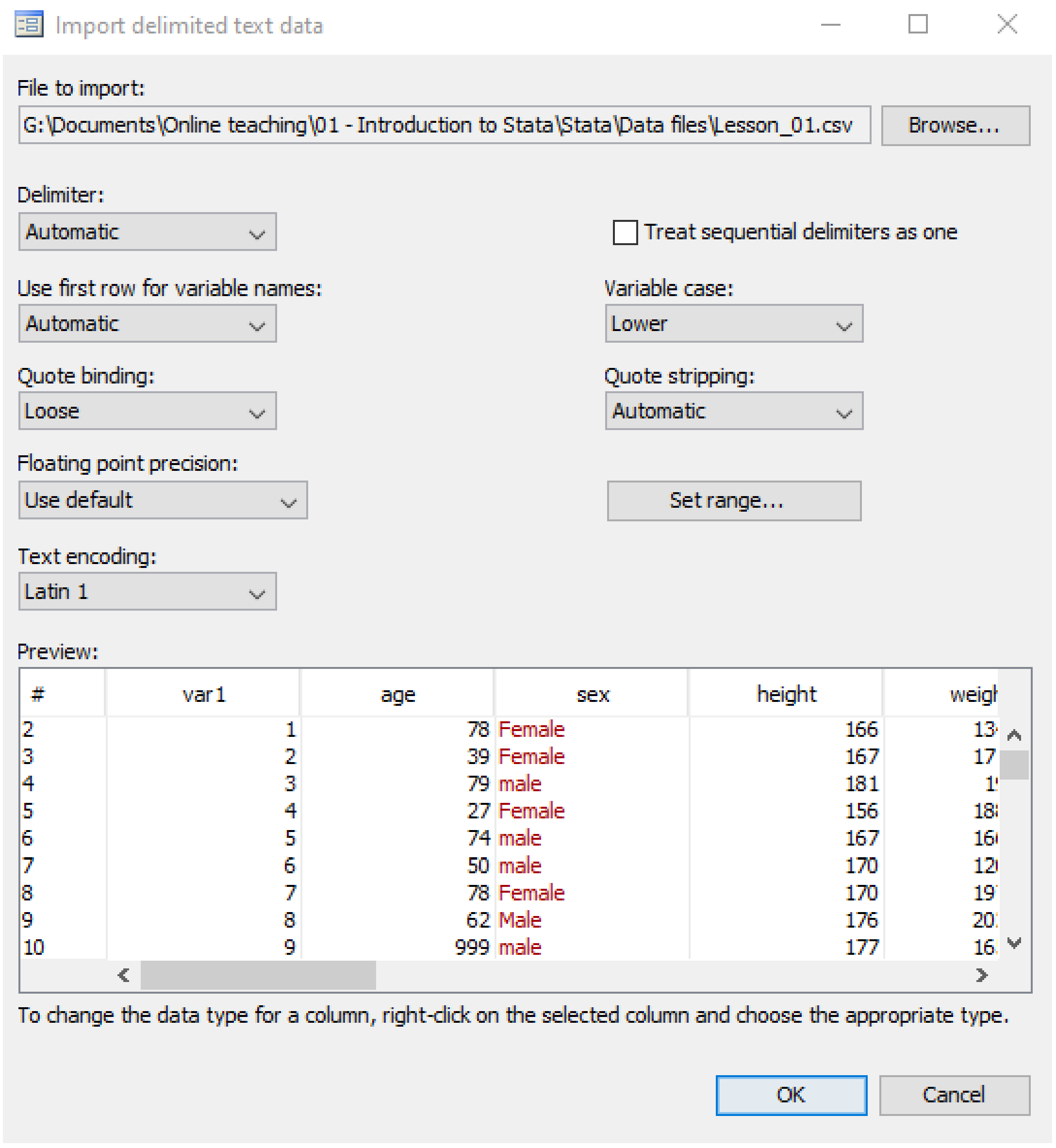
1.9 Importing, Exporting, Appending and Merging Data

Importing data

In addition to opening .dta files, Stata can import other types of data files too, for example Excel worksheets and .csv (comma-separated values) files.

The **`import`** command can be used to import files, though the first time you import a new file, it is often simpler to use the ***Import*** option under **File**. From there, you'll need to select the correct import option, usually either an ***Excel spreadsheet*** or a ***Text data*** file This brings up an import window that has a preview of what the data will look like once imported. Stata will attempt to import the data correctly automatically, but you may need to change some of the options to get the data to look right. Once you've correctly imported the file you want, you can copy the command into a do file for later use, or save the data as a .dta file and use that instead.

For example, if we wanted to import the **Lesson_01.csv** file, we could use the ***Import*** option under **File:**



File to import: G:\Documents\Online teaching\01 - Introduction to Stata\Stata\Data files\Lesson_01.csv

Delimiter: Automatic

☐ Treat sequential delimiters as one

Use first row for variable names: Automatic

Variable case: Lower

Quote binding: Loose

Quote stripping: Automatic

Floating point precision: Use default

Text encoding: Latin 1

Preview:

#	var1	age	sex	height	weigl
2	1	78	Female	166	13
3	2	39	Female	167	17
4	3	79	male	181	1
5	4	27	Female	156	18
6	5	74	male	167	16
7	6	50	male	170	12
8	7	78	Female	170	19
9	8	62	Male	176	20
10	9	999	male	177	16

To change the data type for a column, right-click on the selected column and choose the appropriate type.

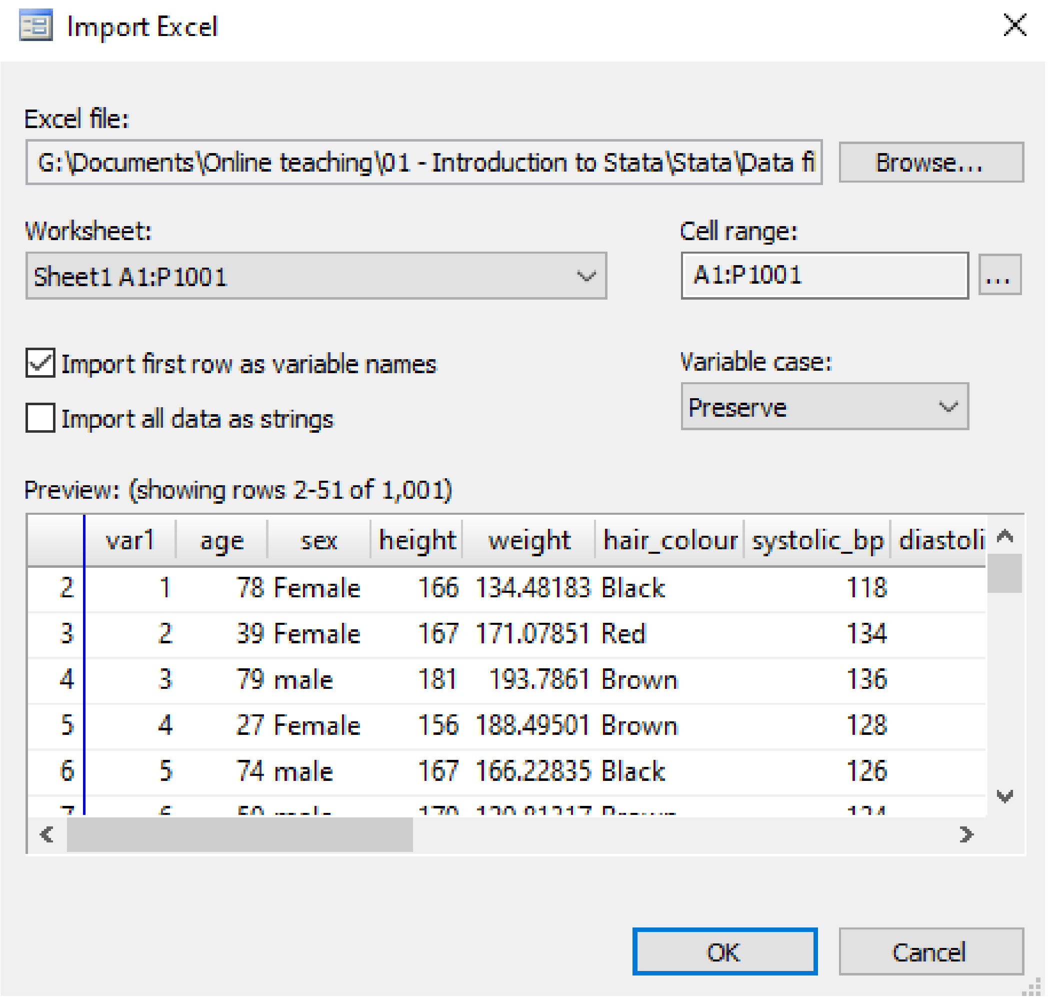
This gives us the following command:

```
import delimited "G:\Documents\Online teaching\01 -
Introduction to Stata\Stata\Lesson_01.csv", clear
(16 vars, 1,000 obs)
```

If we've set the working directory, we could remove the folder location and just write:

```
import delimited "Lesson_01.csv", clear
(16 vars, 1,000 obs)
```

Importing from Excel is similar, though we also need to specify that we want a particular Excel sheet (note the quotation marks in the command) and that we want the first row of observations to be treated as the variable names:



Import Excel

Excel file: G:\Documents\Online teaching\01 - Introduction to Stata\Stata\Data fi

Worksheet: Sheet1 A1:P1001

Cell range: A1:P1001

[✓] Import first row as variable names

[ ] Import all data as strings

Variable case: Preserve

Preview: (showing rows 2-51 of 1,001)

	var1	age	sex	height	weight	hair_colour	systolic_bp	diastoli
2	1	78	Female	166	134.48183	Black	118	
3	2	39	Female	167	171.07851	Red	134	
4	3	79	male	181	193.7861	Brown	136	
5	4	27	Female	156	188.49501	Brown	128	
6	5	74	male	167	166.22835	Black	126	

This gives us the following command:

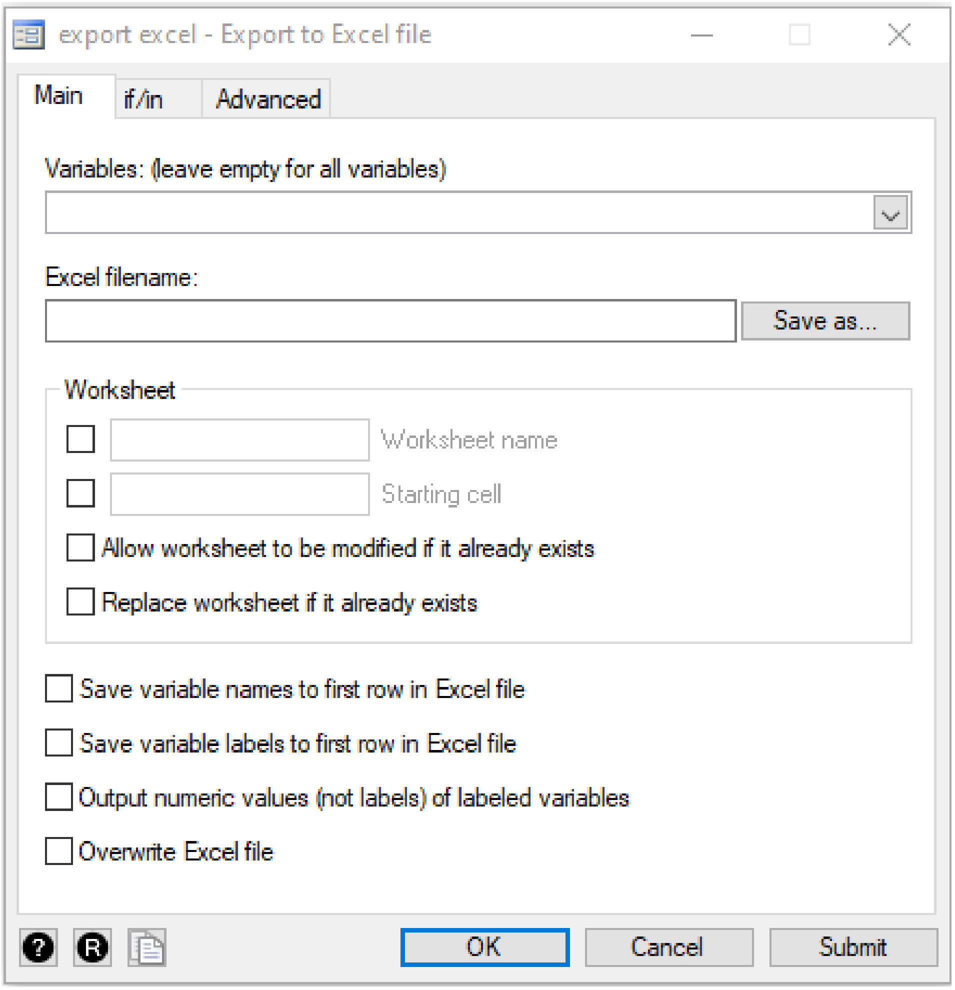
```
import excel "Lesson_01.xlsx", sheet("Sheet1") firstrow clear
```

We could also specify that we wanted a particular cell range, or that we wanted all variables to be imported as strings – this would convert all the numbers to strings.

Exporting data

The **`export`** command can be used to export data to Excel, tab delimited, comma delimited, and other file types. It works similarly to **`import`**, but in reverse. As with **`import`**, the first time you export a file, it makes sense to use the ***Export*** option under **File**, tinkering until you output the data in a format that you want.

In particular, you can select which variables to export (leave blank to export all variables), whether you want variable names or variable labels in the first row and whether you want the data *labels* or the underlying *numbers* (for numeric labelled variables). There are also more options to play around with if you're looking for something specific.


export excel - Export to Excel file

Main | if/in | Advanced

Variables: (leave empty for all variables)

Excel filename: [Save as...]

Worksheet
- [ ] Worksheet name
- [ ] Starting cell
- [ ] Allow worksheet to be modified if it already exists
- [ ] Replace worksheet if it already exists

- [ ] Save variable names to first row in Excel file
- [ ] Save variable labels to first row in Excel file
- [ ] Output numeric values (not labels) of labeled variables
- [ ] Overwrite Excel file

OK | Cancel | Submit

Appending data

If you have multiple datasets that have the same variable names and variable types (string or numeric), you can attach them one on top of the other by *appending* them. The syntax is:

```
append using "{new dataset}.dta"
```

`append` doesn't require the number of variables in the two datasets to be the same: any variables missing in the original or appended datasets will have missing values created for them. For example, if the *age* variable is in the original but not the appended dataset, every observation in the appended dataset will have missing values of *age*.

Merging data

While **`append`** adds observations to a dataset in memory, **`merge`** adds new variables to the same observations to a dataset in memory (although sometimes it adds rows too). This is useful if you have multiple datasets that contain *different* variables for the *same* observations, for example, a dataset containing demographic information about people in a study (age, sex, weight etc.), and another containing health outcomes (blood pressure, disease status etc.).

The **`merge`** command usually requires that there be at least one variable that is common to both datasets, for example an ID variable. You can have more than one matching variable, and you can also match on row number instead of a variable if you know the rows are sorted the same way in both datasets (though this is inherently riskier). The observations in the second dataset are matched to the first by the identification variable(s) or row number.

Merges can be *one-to-one*, where unique IDs in each dataset are tied together, *one-to-many*, where unique IDs in the dataset in memory are tied to potentially multiple instances of each ID in the second dataset, *many-to-one*, where potentially multiple instances of each ID in the dataset in memory are tied to unique IDs in the second dataset, and *many-to-many*, where potentially many instances of each ID in both datasets are tied together. I have often used one-to-one matching, sometimes used one-to-many and many-to-one matching, and don't think I've ever used many-to-many matching.

The syntax is as follows for merging on specified variables (**`m`** means many):

```
merge {1:1, 1:m, m:1 or m:m} {variable(s) common to both datasets
to match on} using "{new dataset}.dta"
```

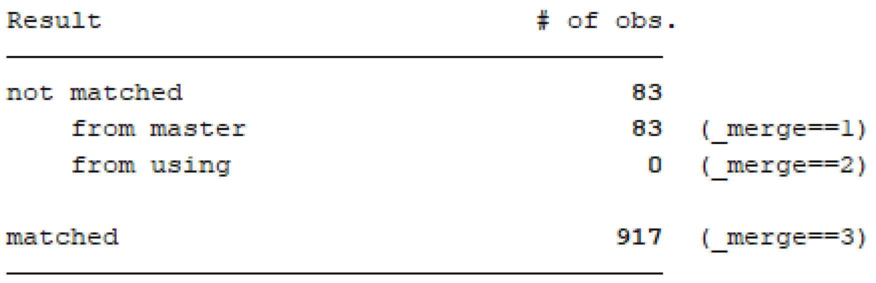
or as follows for merging on observation number:

```
merge 1:1 _n using "{new dataset}.dta"
```

For example, we could add a new variable to the **Lesson_01.dta** file using the **merge_01.dta** file. The **merge_01.dta** file has the same ID variable, called *var1* (temporarily), and an additional variable called *date_of_recruitment*. We're merging using 1:1 matching as there is exactly 1 observation for each ID number in both datasets.

First, make sure your **Lesson_01_{initials}.dta** file is loaded into memory and the working directory is set to the folder containing your downloaded datasets (it should still be if you did this earlier), then type:

```
merge 1:1 var1 using "merge.dta"
```



```
Result                           # of obs.
-----------------------------------------
not matched                             83
    from master                         83  (_merge==1)
    from using                           0  (_merge==2)

matched                                917  (_merge==3)
-----------------------------------------
```

Stata displays an output for the results of the merge, showing the number of observations in the original dataset *not matched* in the new dataset (*not matched from master*, coded as a 1), the number of observations in the new dataset not matched with the original dataset (*not matched from using*, coded as a 2), and the number of observations that matched between the datasets (*matched*, coded as a 3). Stata also, by default, creates a variable called *_merge*, which lets you easily tell whether an observation was matched or not using the codes you can see above.

From this output, we can see that 917 observations matched between the **Lesson_01_SH.dta** and **merge.dta** datasets, but there were 83 observations in the **Lesson_01_SH.dta** dataset that weren't in the **merge.dta** dataset: these observations will therefore have missing values of *date_of_recruitment*.

The final command before moving onto the exercise is to save the dataset one more time, as we'll load this in at the start of the second lesson:

```
save "Lesson_01_SH.dta", replace
```

1.10 Exercise

For this exercise, see how you do with the following:

1. Open a fresh copy of Stata

2. Change the directory to the folder containing your downloaded copy of the **Exercise_01.dta** dataset
    a. This dataset is similar to the **Lesson_01.dta** dataset, but with some tweaks

3. Load in the **Exercise_01.dta** dataset

4. Save a copy of the dataset with your initials at the end

5. Have a look at the variables listed in the top right window, and the dataset information in the bottom right window

6. Have a look at the data itself using the spreadsheet view
    a. Take a note of any numeric, string and labelled numeric variables, and remind yourself of their colours and what they mean

7. Append the **Exercise_01_append.dta** dataset
    a. Check that the command added observations to the dataset

8. Merge the **Exercise_01_merge.dta** dataset
    a. You can load up another instance of Stata to take a quick look at the dataset (hint: you'll need to merge 1:1 on the *id* variable)
    b. Check which variable(s) were added to the dataset
    c. The variable will look strange, but that's ok, we'll go through how to **format** variables in a later lesson

9. Open up the help menu for **`merge`** and take a quick look at the available options
    a. See if you can figure out how to merge a dataset without creating the *_merge* variable

10. Save the appended and merged dataset as **Exercise_01_{initials}.dta**

11. Now export this dataset to an Excel spreadsheet
    a. Make sure the variable names are in the header row

1.11 Exercise – Answers

1. *Note: you can have multiple instances of Stata open at once – each new copy has its own settings, and each can have its own dataset loaded, although each copy of Stata can't interact with any others.*

2. To change the directory, use the **`cd`** command and specify the folder where you downloaded the datasets, making sure the folder name is in quotation marks:
   a. **`cd "G:\Documents\Online teaching\01 - Introduction to Stata\Stata"`**

3. To load a dataset, use the **`use`** command, followed by the name of the dataset you want to load. If you've changed the directory, you don't need the full file path:
   a. **`use "Exercise_01.dta", clear`**
   b. You don't need the **`clear`** option unless you are loading data over an unsaved dataset

4. To save a dataset, use the **`save`** command, followed by the name of the file you want to save the dataset as. If you've changed the directory, you don't need the full file path:
   a. **`save "Exercise_01_SH.dta", replace`**
   b. You don't need the **`replace`** option unless you are saving data over an existing dataset

5. You can see the variables and their labels in the top right window, and some summary dataset information in the bottom right window
   a. There should be 14 variables, with 1,000 observations, taking up something like 57 KB

6. Use either of the *Data Editor* buttons to load up the spreadsheet view, or use the **`browse`** or **`edit`** commands
   a. Most variables are black, so are numeric
   b. One variable is red, *var3*, which means it's a string variable
   c. Two variables are blue, *hair_colour* and *accommodation*, which means they are labelled numeric variables (numbers that have a label applied to them to be more informative)

7. To append a dataset, use the **`append`** command:
   a. **`append using "Exercise_01_append.dta"`**

8. To merge a dataset, use the **merge** command:
    a. `merge 1:1 id using "Exercise_01_merge.dta"`
    b. The merged variable, *date_of_recruitment*, is just a list of numbers in the 21,000 range – this is how Stata encodes dates (number of days since 01/01/1960), and we'll need to format it to show an interpretable date

9. To open up the help menu, use the help command:
    a. `help merge`
    b. One of the options, **nogenerate**, looks like it will not create the *_merge* variable
    c. You'll have to remove the *_merge* variable before trying to **merge** again – we'll cover this in the next lesson

10. Save again, making sure to include the replace option:
    a. `save "Exercise_01_SH.dta", replace`

11. You can export using the *Export* interface under *File*
    a. The code that Stata generates will hopefully look something like this:
    b. `export excel using "Exercise_01_SH", firstrow(variables) replace`