The Australian Economic Review
MELBOURNE INSTITUTE of Applied Economic and Social Research
The Australian Economic Review, vol. 50, no. 2, pp. 236–50

For the Student

Empirical Research: Getting Started with Stata

Daniel Tiong*

Abstract

This article provides a guide to approaching empirical research work for the first time, as well as obtaining a foundation in the use of the statistical software package, Stata. The emphasis revolves around data cleaning and its importance in writing a good research essay. I also cover a basic introduction to Stata, the importance of do-files, common data problems and good coding practice. In closing, I advise that not everything in research needs to go to plan, but that good research practice makes it easier to adapt when plans do not work out.

1. Introduction

When beginning research work for the first time, feeling overwhelmed is not an uncommon occurrence. For example, the question of where to start—or for that matter, where to end—a research essay is not one that is easily addressed in the coursework of a typical undergraduate degree. In contrast to coursework, research challenges one's ability to be critical and think 'outside the box'; that is, coming up with creative solutions to one's problems. Research is something that seems daunting at first glance, but that should not be a deterrent for those who are genuinely interested in answering questions about the world. This is illustrated with the saying: 'If we knew what it was we were doing, it would not be called research, would it?'[1]

Consequently, not all research advice can be described as 'one size fits all'. Advice that works for someone may not work and/or be applicable to someone else.[2] In this vein, the focus of this article is twofold. First, I present a general overview of empirical research in Section 2. I then follow this discussion with an introduction to the basics of the statistical software package, Stata, in Section 3, with an emphasis on data cleaning and management. Section 4 concludes with some advice for good research practice.

2. Empirical Research

2.1 Types of Research

Broadly speaking, a research paper falls into one of three categories: theoretical, empirical and mixed—the latter of which is a mixture of the first two.

* Department of Economics, The University of Melbourne, Victoria 3010 Australia; email <tiongd@unimelb.edu.au>.
I would like to thank Mick Coelli and various anonymous referees for helpful comments and suggestions which have contributed greatly to this article.

A theoretical paper is one that generally does not involve data.[3] Beginning from some reasonable assumptions on how the world works, theoretical research constructs—or expands on—models that then try to predict what would happen under certain scenarios. For example, the concept of a Nash equilibrium is one such theoretical concept that those familiar with game theory will recognise.[4]

On the other end of the spectrum, empirical research is heavily reliant on data. In particular, the end goal of empirical research is often the intersection of three elements: an interesting research question, data suited to answering it and an empirical identification strategy (usually by estimating something causal) with which the question is answered. In the case of the latter, descriptive statistics and correlations alone are often not enough to provide a convincing argument. Problems with any of these elements will have repercussions on the others: in particular, bad data management will reflect poorly on your research overall.

While both theoretical and empirical work take different approaches, there exists plenty of advice that is general enough for both fields. In particular, choose a topic in the field(s) that interests you the most. Doing this benefits you on two fronts. First, you will often be much happier discussing something you enjoy, whether it is with friends and family, colleagues or even in a seminar! Second, this gives you the motivation to keep going with your research, no matter what difficulties you may encounter along the way.

This is supplemented by Creedy's (2001) observation that even if you encounter a dead end, you need to have the willingness and energy to pursue such avenues in the first place. There is no such thing as a 'best' way to do research and research work is fraught with challenges. Yet, it makes it all the more satisfying when you produce your research work at the end of the day.

In practice, the challenges of empirical research are almost always messier than what is covered in undergraduate econometrics. In fact, having a fully fleshed-out research idea, the appropriate data and a watertight empirical strategy from the very beginning would be considered the exception, not the norm. For instance, knowing what your data look like and how to manage them is just as important as having a solid econometric foundation. To put this in perspective, even a perfect identification strategy is not reliable if you do not properly manage your data.[5] Examples range from missing observations to changes in industry classifications or being unaware of potential sample selection bias in your data. To an extent, undergraduate econometrics is like being taught what to do with a million dollars, assuming you had a million dollars to begin with.

2.2 Data and Econometric Models

Even within empirical research, there are different needs. Broadly speaking, there are three types of data that we work with: cross-sectional, time-series and panel data. Each of these types of data have their own set of econometric models, each with their own requirements on what variables need to go in. For example, the simple linear regression:

$$y_i = \beta_0 + \beta_1 x_i + e_i \tag{1}$$

requires data on both y_i and x_i for researchers dealing in *cross-sectional* data; that is, data recorded at a single point in time. Researchers with observations on individual variables, but recorded at many points in time—that is, *time-series* data—can run the same model as well, but replacing y_i with y_t and x_i with x_t.

Another example is that of instrumental variables. If the linear regression above exhibits correlation between x_i and the econometric disturbance term e_i, then attempting to identify β_1 as a causal effect is not valid. However, if a valid instrumental variable z_i is available, then the two-stage least squares approach would allow one to identify the causal effect β_1. However, on top of needing data on y_i and x_i, one would also need data on the instrumental variable z_i Beyond these examples, the choices of econometric models available will depend on the data you have. More examples may be found in Sub-section A1.

Overall, empirical research is a slow process: a lot of groundwork needs to occur

before you proceed to the econometric analysis. The aspects of data management and cleaning are not mechanical tasks that we do once and then never touch again. An empirical researcher should keep in mind that such data work is unpredictable. Perhaps, some new data may arrive. Your data may suddenly change units of measurements midway through the sample. Incorrect spelling or typos might be found in countries or addresses. All of these create problems that we need to deal with as part of the research process.

3. Getting Started in Stata

As alluded to in Section 2, you need data if you want to run an econometric model. However, this assumes that your data are already in the correct format that your model needs. As a general rule, you should never assume that this is the case (and consider yourself lucky if it is!). An example of this is illustrated in Sub-section 3.1. For empirical researchers, dealing with data is a fact of everyday life, especially data cleaning. Heuristically, data cleaning is the process of correcting any mistakes in your raw data and transforming them into something that your model can use.

However, the data cleaning process does not end there. Good data management also involves carefully documenting *how* you clean your data and providing these details, perhaps as part of an appendix in a final research essay. Transparent research practices will make your work more credible in the eyes of those who read your work. If you work in a statistical package like Stata (or Eviews, R and SAS, to name a few), you can also document your steps via the use of *comments*, which I discuss later.

For now, one of the first questions to ask here is: 'Why Stata?' What can it do and why would we want to use it? The description on the Stata website (<http://www.stata.com/why-use-stata/>) reads:

> *Stata is a complete, integrated statistical software package that provides everything you need for data analysis, data management, and graphics.*

Anyone who uses Stata will generally use it for all aspects of data analysis, data management and graphics. Our interest here lies in the data management aspect, which must be done before either the data analysis or graphics. More specifically, I introduce basic commands in Stata, do-files, common data problems (such as missing observations, variable generation and strings and floats), plus custom packages for Stata.

To begin with, Stata is an example of what we call *command-line software*. The 'command-line' aspect comes from the fact that if you want Stata to do something, you often have to explicitly type in the *command* that you want Stata to do. This feature gives Stata a significant advantage over more commonly used programs, such as Excel, in terms of transparency. Formulae entered into cells are not particularly easy to read, for example. In this sense, the command-line features of Stata make any data modification far more clear. However, simpler tasks can also be done using Stata's drop-down menus. For example, if you need help with a given command, the **help** (or **findit**) command will bring up documentation on any command you specify. Alternatively, you can use the 'help' menu item to bring up a search box which you can use to look up commands.

When you open up Stata, you will be greeted with Stata's interface. The interface is highlighted in Figure 1 and contains the following:

(i) variable list

(ii) properties

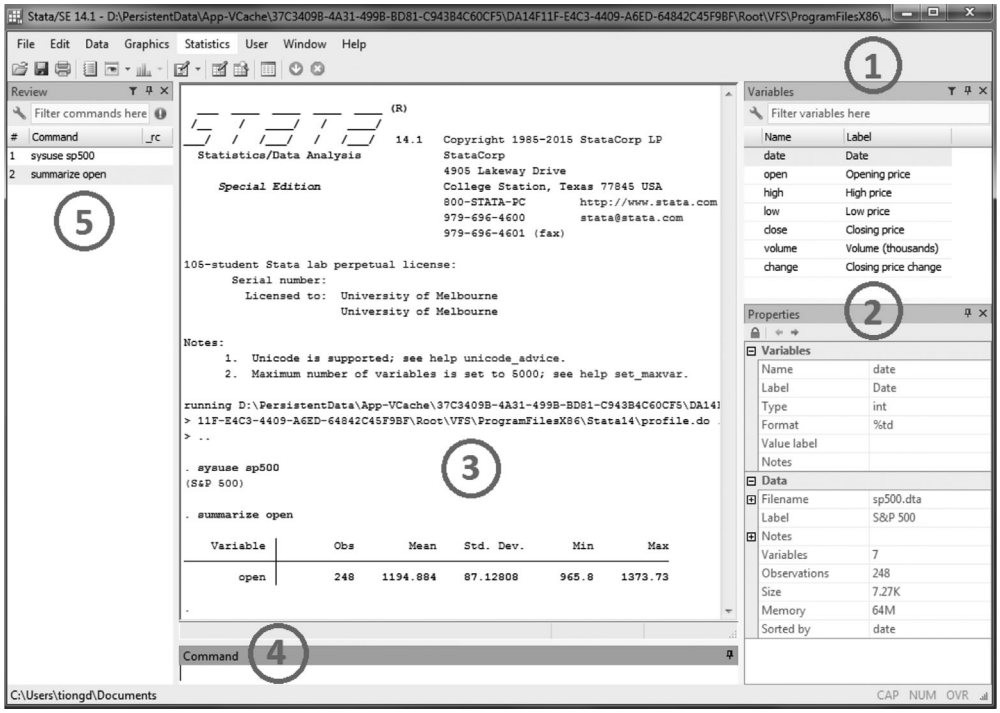
(iii) results (output)

(iv) command (input)

(v) command history.

Whenever you type anything into (iv), Stata will read it as a command and will give you the results of that command in (iii). With that said, for larger projects, using (iv) can become quite troublesome, especially if your project spans a long period of time. For such projects, we store

Figure 1 The Stata Interface



our commands in what we call *do-files*, which will be discussed later in this section.

3.1 Syntax and Data Formats

Commands in Stata follow a type of structure that we call 'syntax'. Essentially, this is like putting words together in English to make a full sentence. You can already imagine that if you took the words in a sentence and rearranged them at random, you would not expect to get back something that makes sense. Stata commands work on this principle. Every command you run in Stata has its own syntax, in the sense that you need to enter them in certain ways so that Stata knows what you are asking it to do.

One of the first examples of syntax that we will see is that of loading in data. Before doing any data work, Stata first needs to know where your data are being stored. The folder where data are stored is called your *working directory* and follows the syntax:

```
cd [“]directory_name[“]
```

where **cd** is the command for setting a working directory and **directory_name** is the 'path' to where your data are stored. Note that actually typing in the expression **directory_name** will result in Stata giving you an error: you must replace it with the appropriate path, which differs from person to person. Anything in closed brackets is optional, subject to exceptions. For example, if your directory name has spaces in it, then you must use quotation marks, otherwise Stata will not be able to interpret your command correctly.[6]

So, for example, if you stored your data in a folder called **C:\Data** (for Windows users), then you could type in:

```
cd C:\Data
```

But if the path was instead **C:\Data files**, then you would need to type in:

```
cd “C:\Data files”
```

in order for the command to work.

Table 1 Commands for Loading Data in Stata

Command	*Description*
use	Loads Stata datasets
insheet [**using**]	Loads .csv files
import excel	Loads Excel spreadsheets

The next task will be to get Stata to open, or 'read in', your data file. Stata can handle a variety of different data formats and has commands for each of those formats. A list of common commands is provided in Table 1.

For more specialised data formats, typing in **help import** into Stata—or searching it via the help drop-down menu—will reveal a larger variety of commands for reading in data.

In particular, Stata will treat *columns* in a spreadsheet as individual variables. Each row is often then considered to be a single observation. Generally, you will need to make sure your data are in the appropriate format before loading them into Stata. One such example is the interest rate data provided by the Reserve Bank of Australia (RBA). Figure 2 illustrates what the raw data look like in an Excel spreadsheet and Figure 3 illustrates the same data when loaded into Stata.

Figures 2 and 3 illustrate a number of immediate data problems that need to be addressed before one can continue:

(i) The first six lines contain information that, while useful for describing the variables, is not suitable for data analysis itself.

(ii) Variables should generally consist of numbers. The second column (cash rate target) is one such example, but there are words in the columns that will interfere with regressions and descriptive statistics.

(iii) The data are time-series, but Stata has not yet recognised them as such.

(iv) The variable names at the top are generic (that is, the variables are called v1, v2, v3 and so on).

Figure 2 Raw Data on Reserve Bank of Australia (RBA) Interest Rates, Excel Spreadsheet

	A	B	C	D
1	F1 INTEREST RATES AND YIELDS – MONEY MARKET			
2	Title	Cash Rate Target	Change in the Cash Rate Target	Interbank Overnight Cash Rate
3	Description	Cash Rate Target on date	Change in the Cash Rate Target (in percentage points)	Interbank Overnight Cash Rate on date
4	Frequency	Daily	as announced	Daily
5	Type	Original	Original	Original
6	Units	Per cent	Per cent	Per cent
7				
8				
9	Source	RBA	RBA	RBA
10	Publication	13-Jan-17	13-Jan-17	13-Jan-17
11	Series ID	FIRMMCRTD	FIRMMCCRT	FIRMMCRID
12	4-Jan-11	4.75		4.75
13	5-Jan-11	4.75		4.75
14	6-Jan-11	4.75		4.75
15	7-Jan-11	4.75		4.75

Figure 3 Raw Data on Reserve Bank of Australia (RBA) Interest Rates, Stata Editor

	v1	v2	v3	v4
1	F1 INTEREST RATES AND YIELDS – MONEY MARKET			
2	Title	Cash Rate Target	Change in the Cash Rate Target	Interbank Overnight Cash Rate
3	Description	Cash Rate Target on date	Change in the Cash Rate Target (in percentage points)	Interbank Overnight Cash Rate on date
4	Frequency	Daily	as announced	Daily
5	Type	Original	Original	Original
6	Units	Per cent	Per cent	Per cent
7				
8				
9	Source	RBA	RBA	RBA
10	Publication date	13-Jan-17	13-Jan-17	13-Jan-17
11	Series ID	FIRMMCRTD	FIRMMCCRT	FIRMMCRID
12	4-Jan-11	4.75		4.75
13	5-Jan-11	4.75		4.75
14	6-Jan-11	4.75		4.75
15	7-Jan-11	4.75		4.75

Figure 4 Reserve Bank of Australia (RBA) Data, Suitable for Analysis in Stata

	date	Target	TargetDiff	OvernightC~e
1	04jan2011	4.75	.	4.75
2	05jan2011	4.75	.	4.75
3	06jan2011	4.75	.	4.75
4	07jan2011	4.75	.	4.75
5	10jan2011	4.75	.	4.75

All of the above can be addressed in Stata, although it is sometimes convenient to do certain tasks in, say, Excel.[7] An example of the data after some cleaning work is shown in Figure 4.

Details on how Figure 4 is obtained are provided in Sub-section A2. A general rule of thumb in this case is to *keep a back-up of your raw data* before modifying them in any way. If you delete something from your raw data and accidentally save it, then that creates problems, especially if such data are not publically available (for example, from a survey or experimental data).

3.2 Some Basic Commands

For the purposes of experimenting, Stata also ships with a number of 'canned' datasets, which we can freely read in and modify at any time. One such example is the **sp500** dataset, which can be brought up by simply typing the command:

```
sysuse sp500
```

into (iv) and pressing 'Enter'. Even if you do not have any specific model in mind, you can still obtain descriptive statistics. Some examples can be found in Table 2.

Such commands need what we call a *variable list*, or *varlist*, to work. One of the variables in the **sp500** dataset is **open**, so if you type in something like **summarize open**, you will get back summary statistics for the variable **open**:

```
. summarize open

    Variable |    Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
        open |    248    1194.884    87.12808      965.8    1373.73
```

The same results are also displayed in Figure 1. If we are interested in the summary statistics for **open** over a certain range, then we could also use an **if** statement, which in this case will ask Stata to provide summary statistics, but only over the range specified. For example, if we ran the command:

```
summarize open if open > 1200
```

Then, we would get the following summary statistics table:

```
. summarize open if open > 1200

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
        open |       119    1269.352    49.06138    1200.47    1373.73
```

A further discussion of **if** statements is provided in Sub-section A7. From here, you can take it a step further and run a simple regression in Stata. You can do this with the command **regress**. Here is an example (albeit useless): if you wanted to regress the variable **open** on another variable, such as **close**, you would type in:

```
regress open close
```

It would give you the results of the simple linear regression:

$$open_t = \beta_0 + \beta_1 close_t + e_t \quad (2)$$

where e_t is the usual econometric disturbance term. Even with a simple command

Table 2 Examples of Descriptive Statistics in Stata

Command	*Description*
summarize	Provides summary statistics
tabulate	Gives a breakdown of every value a variable takes
histogram	Creates the histogram of a variable
edit and **browse**	Allows you to look at the dataset (the former lets you edit it directly)

like **regress**, you can modify it in many ways:

(i) **regress open close, noconstant** (suppresses the constant and runs the regression $open_t = \beta_1 close_t + e_t$ instead).

(ii) **regress open close, vce(robust)** (gives heteroscedasticity-consistent standard errors instead of the usual ordinary least squares (OLS) standard errors).

Finally, Stata also accepts abbreviated versions of most commands. For example, **regress** can be shortened to **reg**. The **tabulate** and **summarize** commands can be referred to by **tab** and **sum**, respectively.

3.3 Do-Files and Code Annotation

Generally, the majority of research work in Stata does not occur in the command window. Any project that takes more than a few hours will probably have many commands that need to be input every time you start work on the project again. Ideally, there should be a way for you to write down those commands so that writing them again is not necessary.

In Stata, the way to do this is through *do-files*, which are text files that run all of your code line-by-line. That is, whenever Stata sees a line, it will interpret it as a command by default unless you specify otherwise.[8] This means that a do-file keeps a record of how you use your data for cleaning, econometric modelling and figure generation, which others can also use.

It also means that when you finish up your work at the end of the day, you can come back the next day and start right where you left off. Other benefits of do-files include: (i) being able to spot and correct mistakes more easily; (ii) more convenient ways of reproducing results for yourself or for your colleagues; and (iii) having a more organised workplace. It is also simpler to write 'statements' and 'loops' in a do-file, but we only discuss these briefly in Sub-section 3.4. Further discussion of these may be found in Sub-section A7.

I personally begin a do-file with the following commands:

```
clear all
set more off
```

The command **clear all** clears everything from memory to avoid interference with previous files that might have been run. The command **set more off** disables a feature of Stata that can often be more trouble than it is worth (an example of this is illustrated in Sub-section A3).

Once the commands above have been run, you can then type in your commands for whatever kind of data cleaning, regression or figure generation you happen to be interested in. Once that is all done, you may then want to save your data. In order to do so, you can use the **save** command, followed by the name that you want the dataset to have.[9]

If you want to overwrite the file that you are saving, the **save** command by itself will not work and you will need to include the **replace** option in the command. So for example, if you had a file named '**sp500.dta**' and you wanted to overwrite it, you might type in:

```
save sp500.dta, replace
```

in which case, the file would be overwritten.

A form of good practice when writing a do-file is that you can (and should) leave 'comments' in your do-file describing what your code does. The general consensus I get when talking to others who use Stata is: anything that seems obvious to you when you write your code today is much less obvious when you come back to it in 1 month's time. Comments make it easier to read not just your code, but also others' code.

Stata will 'ignore' comments that it sees in your do-file. Specifically, this means that Stata will not try and interpret lines in do-files that are 'commented out' as commands to run. There are three distinct ways in which you can comment out lines of code:

(i) * is for single-line annotation.

(ii) // is for in-line annotation.

(iii) /* and */ are used for larger comment blocks. In fact, /* begins the comment and will continue until you use */ to end it.

So, if we wanted to bring together all of the concepts we used from Sub-sections 3.1 and 3.2, an example do-file with annotation would look something like the script below. I also include an example of an **if** statement, which in this case asks Stata to provide summary statistics, but only if the condition holds.

```
* Preamble
clear all                      // Clear data from memory
set more off

cd "C:\Data"                   // Working directory (change if needed)

sysuse sp500                   // Load canned Stata dataset

* Summary statistics
summarize open                 // Summary statistics for open
summarize open if open > 1200  // Summary statistics, restricted range

* Regression
regress open close             // Simple linear regression

* Save the dataset
save sp500.dta, replace
```

A log of the output from this do-file is presented in Sub-section A4.

3.4 Variable Generation and Some Common Data Problems

A very common application of Stata's data management is that of variable generation, often through the use of the **generate** command. For instance, suppose you had the **sp500** dataset and were interested in what the spread was between the opening price and closing price on any given day. To do that, you can run the command:

```
generate spread=high - low
```

How about the difference in the closing price from day to day? The **generate** command can also handle this as well. That said, the syntax is a bit more involved. The specific command would look something like this:

```
generate close_diff=close[_n] - close[_n-1]
```

What the command above says is this: create a variable called **close_diff**, such that at any 'observation number' denoted by '**_n**', its value will be equal to the value of **close** at that observation, less the value of **close** at the observation before (denoted by '**_n-1**').

There are some variable generation tasks that **generate** cannot carry out. For those tasks, Stata also comes with the **egen** (extensions to generate) command. As the name suggests, it can generate variables beyond what the **generate** command can do. This ranges from variables that take the value of summary statistics (mean, median, standard deviation, minimum, maximum) to variables that join together the values of other variables (also known as 'concatenation').

Beyond variable generation, there are also other commands which you may find useful for data manipulation (see Table 3).

I now will also highlight some other examples of common data problems that you may encounter:

Missing observations. More common than not, some observations simply will not have any data recorded at all. This could be due to error or other factors, such as unemployed persons not having a wage recorded, survey

Table 3 Commands for Data Manipulation in Stata

Command	*Description*
sort [**varlist**]	Sorts variables from lowest to highest
order [**varlist**]	Orders specified variables according to the list provided
keep [**varlist**]	Keeps specified variables (and drops everything else)
drop [**varlist**]	Drops specified variables (and keeps everything else)
rename	Renames a variable
replace	Replaces specific values of a variable

participants failing to respond to a question or some other sort of sample selection bias. Generally, Stata will mark missing observations with a 'dot'. However, raw data sometimes mark missing observations with terms such as 'N/A' (not applicable) or codes such as '999' for not applicable or 'no response'. This means that you will need to carefully read any documentation that comes with the raw data since both of these cases create issues for data work.[10]

I discuss the case of N/A immediately below. For the latter, you will need to replace such codes with Stata's dots for missing variables; otherwise, they will bias estimates considerably. It is also worth noting here that any zero observations are not the same thing as missing observations. Stata will include the former in regressions, while it will ignore missing observation dots.

Strings and floats. Stata can accept words as data, not just numbers. Any variable containing letters is considered to be a 'string' variable, regardless of whatever else the variable happens to contain. On the other hand, any variable that contains only numbers and/or missing observations is considered 'numeric', or more commonly, a 'float' variable.[11]

String variables usually arise in situations, such as addresses and names, as well as missing observations. Stata generally does not accept string variables in regressions, so in the case of missing observations that have the form N/A, for example, you would not be able to run a regression with that variable unless you correct this. If you encounter such problems, the **replace** and **destring** commands can be useful, along with **if** statements.

Date–time translation. Time can be measured in minutes, hours, days, weeks, months and years, to name just a few. Moving between all of these frequencies—or even getting Stata to recognise these in the first place—is a headache in its own right. If you have any panel or time-series data, this is an area to watch out for. You may type in **help datetime translation** for documentation on this, but be warned: it is quite extensive.

Dataset merging. If you have multiple datasets on hand but would like them all in one place, then you will need to merge them. The **merge** and **append** commands are used specifically for this purpose.

Reshaping. Panel data—that is, data recorded both over time and over individuals—come in two distinct formats: 'long' and 'wide'. A long format dataset has a variable that holds the 'context' of the data and another variable that lists the data values. On the other hand, a wide format dataset gives every variable its own column. Table 4 is a hypothetical example that illustrates this.

The three columns on the left illustrate a dataset in 'long' format and the three columns on the right are the same dataset, but in 'wide' format. In Stata, the **reshape** command allows you to move between these two formats, as needed. An example of how to do this may be found in Sub-section A6.

Statements and loops. A substantial portion of data work involves the use of logical statements. For example, you might ask yourself whether it is possible to easily obtain summary statistics and a histogram on wages only for those who are employed. The answer to this is 'Yes': if we assume that such a dataset records a wage of zero for anyone unemployed, then an example command could look like:

Table 4 Example of Long and Wide Format Dataset

Country	*Year*	*GDP*	*Country*	*GDP2016*	*GDP2017*
Australia	2016	10	Australia	10	20
Australia	2017	20	New Zealand	30	40
New Zealand	2016	30			
New Zealand	2017	40			

```
histogram wage if wage > 0
```

Such statements can become complicated very quickly. A more detailed discussion of this is left for Sub-section A7.

3.5 User-Written Programs and Additional Resources

As versatile as Stata is, some researchers have specific requirements that Stata may not necessarily cover. For these cases, other Stata users create and contribute their own packages of commands to the Boston College Statistical Software Components (SSC) archive, which can be found and installed using the **ssc install** command in Stata. Such commands also come with documentation in a similar fashion to Stata's help documentation. Some potentially useful packages are provided in Table 5.

Outside of additional functionality, there are many places that you may refer to if you ever need help on certain parts of your Stata code. For any problem you encounter, there is a good chance that you will not be the first one to have such a problem! To this end, I provide below a list of resources that you may find useful for your own work:

(i) Stack Overflow (<stackoverflow.com>): a community of programmers, where a lot of questions are asked and a lot of them are answered. This includes Stata, too!

(ii) Statalist (<statalist.org>): the official Stata forum. Similar to Stack Overflow above, there is plenty of active discussion on all things Stata.

(iii) Institute for Digital Research and Education at University of California, Los Angeles (UCLA) (<http://www.ats.ucla.edu/stat/stata/>): a good resource on common applications of Stata. Detailed examples are given for all of their examples, of which there are many.

Table 5 Useful Statistical Software Components (SSC) Packages

SSC package	*Description*
outreg2	Options for exporting regression output
eststo	Options for exporting regression output
tabout	Options for exporting summary statistics
unique	Returns number of missing observations
ivreg2	Additional IV regression options

Outside of these resources, there is always Google—although chances are, your searches will take you to one of the resources above. Your colleagues, as well as your advisor(s), are also able to help you. In fact, good research practice will help you when it comes to feedback, both in the speed and the quality with which you receive feedback.

4. General Research Advice

Beyond the use of Stata, there are a number of factors that you may want to keep in mind when writing an empirical research essay for the first time:

Treat your work like someone else will read it. This includes not only the essay itself, but also the Stata code that you write. Readable code is paramount if you want to get any (useful) advice from your advisor or other colleagues about the methods that you are using.

Name your variables appropriately. This applies both to your code and to your essay. Shortened variable names are okay in some cases, but the readability of your essay falls if you use those variables in places where you do not necessarily need to; for example, a regression results table.

Make your code flexible. Do not assume that your data work is ever done. More often than not, you will have to deal with data work not only at the start of your research, but also during the research process itself. Common examples include having to adapt to new data or changing research direction, both of which can be incredibly time-consuming if you do not write your code with these possibilities in mind.

Research is a function of obituaries. This is something my advisor told me when I first began with my own work. Even the best of us are not perfect. In fact, we fail all the time at the various things that we try. Perhaps, a certain idea does not work out or the model is not appropriate for the data. In any research paper you pick up, there is always an untold story of obituaries behind how the paper was made. Regardless, what everyone sees at the end of the day is the finished result—and for us, that can be the most satisfying part, by far.

The last thing to keep in mind is that research progress is essentially a random process. You can never be guaranteed that something you do will work the way you expect, to the point where you could spend a whole week not making any progress, then have a day where you make breakthrough after breakthrough, before going back to encountering difficulties again. The key to getting around this is to work at it consistently. At the end of the day, research is not a 100 m sprint—it is more like a marathon. You are in it for the long haul with a research paper, but as long as you keep working at it, you will eventually end up with a research essay that you will be proud of!

March 2017

Appendix 1: Further Discussion

Additional detail on key sections in the article may be found here. In particular, I provide a further discussion of econometric models, along with the code used to create the various figures found in the article. I also go into further detail on loops and logical statements for any who may be interested.

A1. Econometric Models: Examples

In this section, I briefly present some examples of econometric models, all of which can be implemented in Stata.

To begin with, consider cross-sectional data. An example that uses these data is the *binary probit* model which models the probability of an outcome variable y_i that could be either 0 or 1. To illustrate this, consider the simple linear regression model:

$$y_i = \beta_0 + \beta_1 x_i + e_i \tag{A1}$$

If the response variable y_i was a binary outcome variable, then one way to model this would be to use:

$$\begin{aligned}\Pr(y_i = 1|x_i) &= \int_{-\infty}^{\beta_0+\beta_1 x_i} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z^2}{2}\right) dz \\ &= \Phi(\beta_0 + \beta_1 x_i)\end{aligned} \tag{A2}$$

where $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution. In this vein, you can also write:

$$\Pr(y_i = 0|x_i) = 1 - \Pr(y_i = 1|x_i) \tag{A3}$$

How about a model for time-series data? One of the most ubiquitous models is that of the autoregressive of order one (AR(1)) model:

$$y_t = \beta_0 + \beta_1 y_{t-1} + e_t \tag{A4}$$

which could be extended by adding a lag of an independent variable to the right hand side, which would result in the autoregressive distributed lags(1,1) model:

$$y_t = \beta_0 + \beta_1 y_{t-1} + \delta_1 x_{t-1} + e_t \tag{A5}$$

Lastly, we consider a model for panel data. One of the simplest examples to begin with is the fixed-effects model, which models the response y_{it} for an individual at time t through the specification:

$$y_{it} = x_{it}\beta + \alpha_i + u_{it} \tag{A6}$$

where x_{it} is a $1 \times k$ vector of time-varying regressors, α_i is the individual time-invariant fixed effect and u_{it} is the econometric disturbance term.

Stata is able to run all of these models. In particular, the time-series models above can be run as per a standard OLS regression. The probit model can be run using the **probit** command (which extends to multiple possible responses with the **oprobit** (ordered probit) command. The fixed-effects model can be run

with the **xtreg** command alongside the **fe** option for fixed effects.

A2. Reserve Bank of Australia Data: The Cleaning Process

Below is an example of how one might potentially clean such data, as illustrated in Figures 2 and 3. In practice, doing all of this via Stata is not always the most efficient way: certain tasks are simple enough to do before importing your data into Stata. The code itself is not necessarily the most efficient, but it does lead us to the final results that we see in Figure 4.

```
/* ------------------------------------------------------
RBA data, cash rate data cleaning example

The data comes from
http://www.rba.gov.au/statistics/tables/#interest-rates

using the data "Interest Rates and Yields --
Money Market -- Daily -- F1" in .csv format.
------------------------------------------------------ */

* Preamble
clear all
set more off

cd "C:\Data" // Set the working directory
insheet using f1-data.csv

keep v1-v4 // for illustration

* All variables in string format, rename them with this in mind
rename v1 date_str
rename v2 Target_str
rename v3 TargetDiff_str
rename v4 OvernightCashRate_str

* Get rid of the descriptive parts of the spreadsheet
generate id = _n
drop if id <= 11 // Brute force, from observation of initial data

* Destring all of the variables
destring Target_str,            generate(Target)
destring TargetDiff_str,        generate(TargetDiff)
destring OvernightCashRate_str, generate(OvernightCashRate)

* Special case of date-time translation for the dates
generate date = date(date_str,"DM20Y")
format date %td // Daily frequency, time formatting
order date

* Keep only the variables we need
keep date Target TargetDiff OvernightCashRate
```

A3. The Set More Off Command

If the 'more' option is not turned off, then a simple example illustrates the consequence. Consider the following snippet of code:

```
clear all
set more on
sysuse sp500
tabulate open
```

A truncated version of output looks like this:

```
. tabulate open

    Opening |
      price |      Freq.     Percent        Cum.
------------+-----------------------------------
    1102.84 |          1        0.40       14.92
    1103.25 |          1        0.40       15.32
    1104.61 |          1        0.40       15.73
     1106.4 |          1        0.40       16.13
    1106.46 |          1        0.40       16.53
     1115.8 |          1        0.40       16.94
    1117.58 |          1        0.40       17.34
    1118.33 |          1        0.40       17.74
    1118.54 |          1        0.40       18.15
--more--
```

Every time you press a key, more output will display until it runs out of additional output to display. As one might guess, the most irritating part is that you will not be able to enter in any additional commands until all the output is displayed.

A4. Do-File Output

Below is the output from the do-file example discussed in Sub-section 3.3.

```
. * Preamble
. clear all                       // Clear data from memory

. set more off

. cd "C:\Data"
C:\Data

. sysuse sp500                    // Load canned Stata dataset
(S&P 500)

.
. * Summary statistics
. summarize open                  // Summary statistics for open

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        open |       248    1194.884    87.12808      965.8    1373.73

. summarize open if open > 1200  // Summary statistics, restricted range

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        open |       119    1269.352    49.06138    1200.47    1373.73

.
. * Regression
. regress open close              // Simple linear regression

      Source |       SS       df       MS              Number of obs =     248
-------------+------------------------------           F(  1,   246) = 7095.32
       Model |  1812220.67     1  1812220.67           Prob > F      =  0.0000
    Residual |  62831.0496   246  255.410771           R-squared     =  0.9665
-------------+------------------------------           Adj R-squared =  0.9664
       Total |  1875051.72   247   7591.3025           Root MSE      =  15.982

------------------------------------------------------------------------------
        open |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       close |   .9868791    .011716    84.23   0.000     .9638027    1.009956
       _cons |    16.3735   14.02771     1.17   0.244    -11.25625    44.00324
------------------------------------------------------------------------------

.
. * Save the dataset
. save sp500.dta, replace
(note: file sp500.dta not found)
file sp500.dta saved

.
end of do-file
```

A5. Destringing Variables

We first establish two cases of missing observations, as highlighted in the main article: one where there exists a 'N/A' observation (thus making the whole variable a string) and one where a missing observation is denoted with a number, say '999'. We keep this to three observations for simplicity. The preamble generates these data, which you are welcome to ignore. The generated dataset looks like this:

id	var1	var2
1	N/A	999
2	10	999
3	N/A	5

In the case of the N/A observations, the step is to first replace these with Stata's dots. When you do so, you first need to make sure that you respect the fact that **var1** is a string variable. In other words, the dots must be enclosed in double quotation marks. Once that is done, then the **destring** command will do the rest of the work. In this section, an **if** statement comes in very handy by telling Stata that we only want to replace the observations that say N/A in the first place.

The do-file is shown below.

```
* Preamble
clear all
set more off

* Data generation: ignore this part unless you want to replicate this
set obs 3

generate id   = _n

generate var1 = "N/A"
replace var1  = "10" in 2

generate var2 = 999
replace var2  = 5 in 3

* Case 1: Destringing a variable
replace var1 = "." if var1 == "N/A" // Replace N/A with a dot in string format
destring var1, generate(var1_num)   //  Create  a  destrung  version  of  the  original
variable

* Case 2: replacing missing observations
replace var2 = . if var2 == 999     // Replace all instances of 999 with dots
```

A6. Reshaping Data

The following do-file outlines the reshaping process for the example dataset illustrated in Sub-section 3.4.

```
* Preamble
clear all
set more off

* Data generation in long format (feel free to ignore this)
set obs 4

generate country = "Australia"
replace country  = "New Zealand" in 3/4

generate year = 2016
replace year  = 2017 in 2
replace year  = 2017 in 4

generate GDP  = 10 * _n

* Reshaping the data: wide format
reshape wide GDP, i(country) j(year)

* Send the data back to long format
reshape long
```

A7. Local Macros, Statements and Loops

The more familiar you become with Stata, the more you will find yourself asking from it. A very important aspect of data management is the ability to examine data that meet certain conditions. For example, you might like to run a regression only on observations that are not equal to zero. Or, you might need to generate up to 10 lags of a time-series variable. Conditional logic, through the form of **if**, **foreach** and **while** commands will help in doing this. For simple tasks, this is quite intuitive, but can get complicated very quickly.

In order to use these commands, the concept of a *macro* is needed. Simply speaking, it is something that stores values. These values could be words, numbers or even both. Macros are either global (for the entire Stata session) or local (for the do-file only). I will focus on local macros here. The syntax to define a local macro is:

```
local <name> [=] <value>
```

where **<name>** is the name you give the macro and **<value>** will be the value that the macro takes. In order to use a macro, you will need to use single quotation marks of the form **‘’**, with emphasis on the first quotation mark (this is the one that comes under the tilde key on most keyboards). The commands below illustrate some uses for local macros:

```
local x 1
display 'x'

local y sp500
sysuse 'y'
```

In the case of the last line, this is effectively the same as typing in the command **sysuse sp500**.

As alluded to earlier, macros are commonly used in running conditional statements. One of the most frequently used is the **if** statement, which works in the form:

```
if exp{
stata commands
}
```

In short, if a certain condition is met, then Stata will run it. Otherwise, it will not do anything. The syntax above also begs the question: Does one necessarily have to write an **if** statement in the manner above? The answer is 'No', but it is considered good practice to begin a new line upon writing a conditional statement and indent any commands that appear inside.

An example of how the **sp500** dataset might use this is below:

```
sysuse sp500
local model 1
if('model' == 1){
regress open close
}
```

Notice that in this case, I defined a local macro called **model** that took a value of 1. The next part then tells Stata: if the macro called 'model' is equal to 1, then run whatever commands are inside the curly brackets.

You might also notice that there was a use of a double equals sign, or ==, rather than the usual = sign. This is because logical operations in Stata are different from regular commands. The most commonly used are found in Table A1.

Table A1 Logical Operations

Logical symbol	*Meaning*
==	Is equal to
!=	Is not equal to
&	And
|	Or
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Table A1 presents the logical symbols in Stata and their meaning. Closely related to the **if** statement are the **foreach** and **while** loops. A **foreach** loop repeats a command over a specified *varlist*. One application of a **foreach** loop is to create lagged values, among others. The (simplified) syntax for such a command is as follows:

```
foreach lname in varlist{
commands referring to lname
}
```

A **while** loop, on the other hand, will continue repeating the commands inside the loop until the conditions specified have been met. The syntax for a **while** loop is:

```
while exp{
stata commands
}
```

Such loops rely heavily on 'counters'. Specifically, the statement 'exp' is often a local macro of some sort that is predefined, right before the loop starts. Inside the while loop itself, we also need a way to 'advance' this counter so that we do not get stuck in an infinite loop (if the counter is not advanced, then the while loop would never end in such a case). Here is an example:

```
local i=1
while('i' < 10){
stata commands
local i='i'+1
}
```

Furthermore, you can also 'nest' loops and statements. In other words, you can also place loops inside other loops. The amount of logic required for this can be quite extensive and can get out of hand, but the practical applications of nested loops are quite extensive.

Endnotes

1. This quotation is most frequently attributed to Einstein.

2. For example, stationarity testing is highly recommended for anything involving time-series, but is not particularly applicable to cross-sectional data by virtue of the latter having only observations for a single time period.

3. An exception to this includes data that have been generated through simulation of the theoretical model.

4. In the example of two players, a 'stable', or 'equilibrium', outcome occurs when neither player has an incentive to unilaterally deviate from the action they are playing, provided they are rational.

5. Hirschberg, Lu and Lye (2005) provide an accessible background to such descriptive statistics for understanding your data better.

6. The reason why this is the case here is because inputs, or 'arguments', of a command in Stata are often separated by spaces. Since your directory name is a single input, Stata will get confused if you have spaces in the directory name itself. The quotation marks in this sense tell Stata that everything inside the quotations should be treated as a single input.

7. For example, removing the first few rows can be done manually and variable names can be assigned before reading the data into Stata. As mentioned later in the article, be sure to make a back-up of your raw data before doing anything of this sort.

8. One example is to use '///' at the end of a line, which means that Stata will interpret the next line as part of the same command.

9. There are different saving commands for other types of data. For example, if you want to save as a .csv file, you can use the outsheet command instead.

10. Sample selection bias is an important issue in its own right, as are the implications for data management. Angrist and Pischke (2014) contains a very accessible introduction to this content. For the more advanced reader, Angrist and Pischke (2009) is an excellent reference.

11. When using Stata, you will find that Stata will highlight strings as red and floats as black in the data editor.

References

Angrist, J. D. and Pischke, J. 2009, *Mostly Harmless Econometrics*, Princeton University Press, Princeton, New Jersey.

Angrist, J. D. and Pischke, J. 2014, *Mastering 'Metrics*, Princeton University Press, Princeton, New Jersey.

Creedy, J. 2001, 'Starting research', *Australian Economic Review*, vol. 34, pp. 116–24.

Hirschberg, J., Lu, L. and Lye, J. 2005, 'Descriptive methods for cross-section data', *Australian Economic Review*, vol. 38, pp. 333–50.