Applied Financial Econometrics using Stata
1. Introduction to Stata (& Reproducible Research)

Stan Hurn

Queensland University of Technology
& National Centre for Econometric Research

Housekeeping

- Stan Hurn
  s.hurn@qut.edu.au
- This course of lectures will feature Stata 13. See
  ▸ http://www.stata.com
- Due to licensing issues, you are not able to access Stata during the lectures, but all the materials to reproduce the results reported during the course are available from
  ▸ http://www.ncer.edu.au/events

Stata

Stata is a reasonably fast, powerful statistical package with

- smart data-management facilities,
- a wide array of up-to-date statistical techniques,
- and an excellent system for producing publication-quality graphics
- a large group of users with plenty of lively exchange on the web
- a built in matrix programming language called **Mata**

The bad news is that Stata is fairly expensive and there is a startup cost in learning to use it (or at least there is for me!)

Which Edition

There are a number of different editions of Stata:

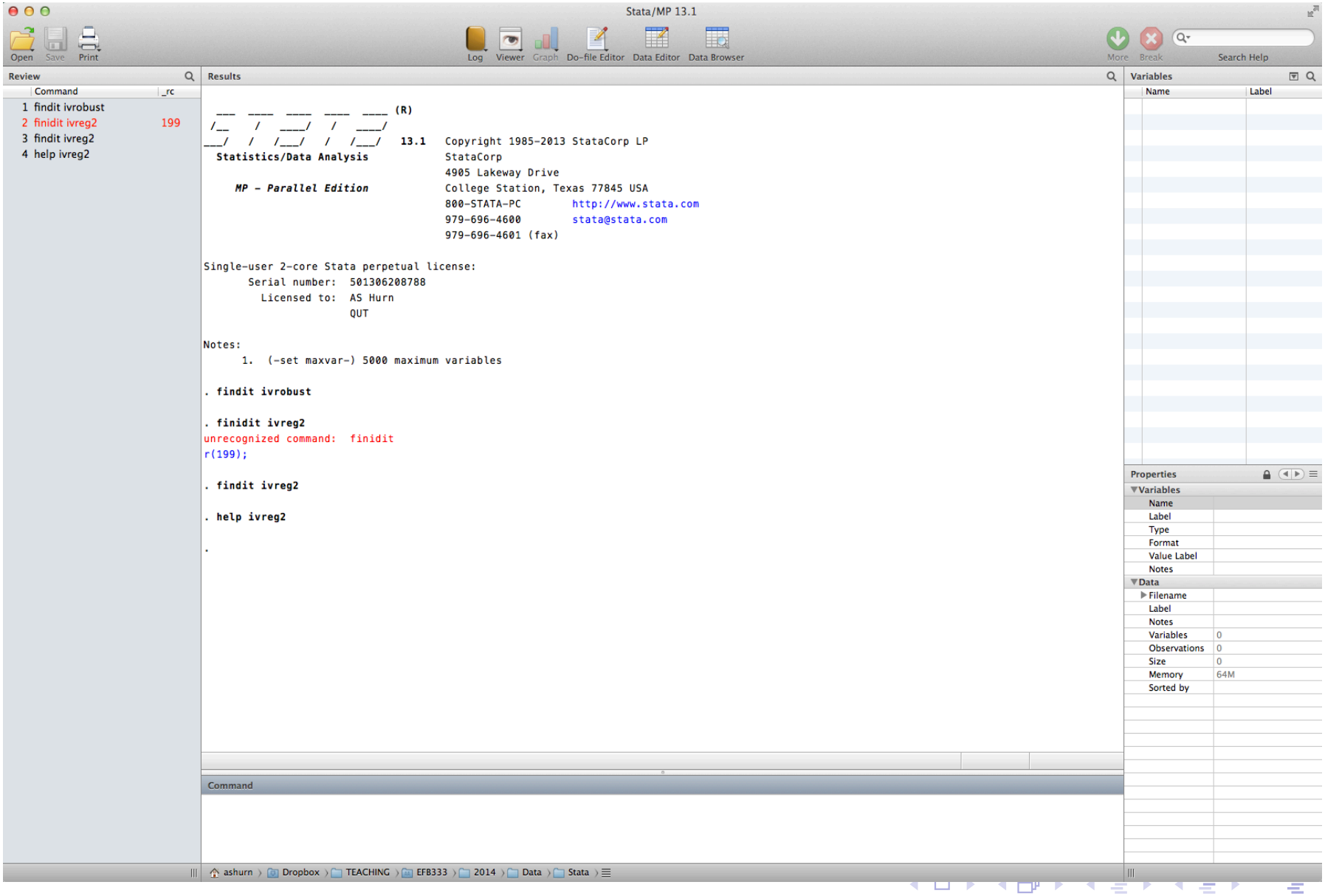
- Stata/MP: The fastest version of Stata (for dual-core and multicore/multiprocessor computers)
- Stata/SE: Stata for large datasets
- Stata/IC: Stata for moderate-sized datasets
- Small Stata: A version of Stata that handles small datasets (for students only)

References

There are a few excellent texts for learning Stata.

- Christopher F. Baum *An Introduction to Stata Programming*, 2009. College Station, TX: Stata Press.
- Christopher F. Baum *An Introduction to Modern Econometrics Using Stata*, 2006. College Station, TX: Stata Press.
- Colin Cameron and Pravin Trivedi *Microeconometrics Using Stata (Revised Edition)*, 2010. College Station, TX: Stata Press.
- Michael N. Mitchell *A Visual Guide to Stata Graphics (Third Edition)*, 2012. College Station, TX: Stata Press.

Stata Front End for Mac



Extending Stata

A feature of Stata is that it is continually being expanded. A command, to Stata, is an instruction to perform some action. Commands may be either:

- "built in" commands (those elements so frequently used that they have been coded into the Stata kernel); or
- written in Stata's own programming language using an .ado file.

The importance of this program design goes far beyond the limits of official Stata as you are able to acquire new Stata commands from a number of sources.

Update Faciity

Built-in commands change fairly frequently and are distributed as bug fixes, enhancements to existing commands and even entirely new commands during the lifetime of a given major release. One of Stata's great strengths is that it can be updated over the Internet.

- issuing the command **update all** from the command line will check for more recent versions of either Stata's executable (the kernel) or the ado-files.
- alternatively from the **Menu** you click
  Help - Check for updates

All that is required to update Stata therefore is that you have a licensed copy of Stata and access to the Internet.

Beyond Official Stata

There are two particularly important websites

- The Stata Journal (SJ), a quarterly refereed journal, is the primary method for distributing user contributions. The Stata command **findit** will locate commands that have been documented in the SJ and with one click you may install them in your version of Stata.
- The Boston College Boston Statistical Software Components (SSC) Archive contains all new general-purpose commands written by individuals to extend Stata functionality.
  ssc new lists new packages
  ssc hot reports on the most popular packages
  adoupdate, update updates your installed packages.

Issuing the command **search** plus a few keywords will present results from a keyword database and from the Internet: for instance, FAQs from the Stata website, articles in the Stata Journal and Stata Technical Bulletin, and downloadable routines from the SSC Archive and user sites.

Setting up a Project

I like having the following directory structure.

- /do
  - /do/working
  - /do/logfiles
- /dta
- /graphics

You can move between folders like this.

- "../../folder" – moves back two directories
- "../folder" – moves back one directory
- "./folder" – accesses a subfolder of the current directory

Remember to assign the **working directory** when you start a session.

Working Directory

- The foot of the Stata screen contains an important piece of information: the Current Working Directory, or cwd.
- The cwd is the directory to which any files created in your Stata session will be saved. Likewise, if you try to open a file and give its name alone, it is assumed to reside in the cwd.
- If the file is in another location, you must change the cwd either using the menus
  File - Change Working Directory ....
  or issue a command in the do file

  ```
  cd ~/Dropbox/Teaching/Singapore/do
  ```

File Extensions

File extensions usually employed (but not required) include:

```
.ado automatic do-file (defines a Stata command)
.dct data dictionary, optionally used with infile
.do do-file (user program)
.dta Stata binary dataset
.gph graphics output file (binary)
.log text log file
.smcl SMCL (markup) log file, for use with Viewer
.raw ASCII data file
.sthlp Stata help file
```

These extensions need not be given (except for .ado). If you use other extensions, they must be explicitly specified.

The Log File

- For reproducible research it is worthwhile behaving like a bench scientist and keeping a lab notebook — the Stata log file plays this role.
- A log file is a record of the Results window. It records all commands and all textual output as it happens.
- Log file is written to disk and protects against power failures or computer crashes.
- Stata can save the file in one of two different formats.
  - Stata Markup and Control Language (SMCL) format — the default.
  - plain text.
- SMCL files can be translated into a variety of formats readable by applications other than Stata .

Stata Command Syntax

Stata command syntax follows strict rules.

- The fundamental syntax of all Stata commands follows a template.
- Not all elements of the template are used by all commands, and some elements are only valid for certain commands. But where an element appears, it will appear in the same place, following the same grammar.
- Stata is case sensitive. Commands must be in lower case.

The general syntax of a Stata command is:

```
[prefix_cmd:] cmdname [varlist] [=exp] [if exp]
                      [in range] [weight] [using...]
                      [,options]
```

where elements in square brackets are optional for some commands.

Prefix Commands

A number of Stata commands can be used as prefix commands, preceding a Stata command and modifying its behavior.

- **by:**
  repeats a command over a set of categories
- **statsby:**
  repeats the command and collects statistics from each category.
- **rolling:**
  runs the command on moving subsets of the data (usually time series).
- **simulate:**
  which simulates a statistical model
- **bootstrap:**
  computation of statistics from resampled data
- **jackknife:**
  runs a command over jackknife subsets of the data.

Two Important Commands

- **generate** is used to produce new variables in the dataset.
- **replace** must be used to revise an existing variable. Note that **replace** must always be typed out in full ...

Note

- **generate** and **replace** operate on all observations in the current data set, producing a result or a missing value for each.
- you can restrict **generate** and **replace** to operate on a subset of the observations with the **if [exp]** or **in range** qualifiers. For example, to list observations at the end of the current data set, use

  ```
  if -5/l
  ```

  to see the last five.

Using Logicals

You can take advantage of the fact that the exp specified in generate may be a logical condition rather than a numeric or string value. This allows producing both the 0s and 1s of an indicator (dummy, or Boolean) variable in one command. For instance:

```
generate large = (pop > 5000000) & !mi(pop)
```

This condition makes use of two logical operators: & (AND) and ! (NOT) to add the qualifier that the result variable should be missing if pop is missing, using the **mi()** function. The third logical operator is the Boolean OR, written as |. Note also that a test for equality is specified with the == operator. The single = is used only for assignment.

The egen Command

Stata is not limited to using the set of defined generate functions. The **egen** (extended generate) command makes use of functions written in the Stata ado-file language. A number of **egen** functions provide row-wise operations similar to those available in a spreadsheet: row sum, row average, row standard deviation, and so on.
Users may write their own **egen** functions. In particular, use

```
findit egenmore
```

for a very useful collection.

.do Files

- Stata may be used in an interactive mode, and those learning the package may wish to make use of the menu system.
- When you execute a command from a pull-down menu, it records the command that you could have typed in the Review window, and thus you may learn that with experience you could type that command (or modify it and resubmit it) more quickly than by use of the menus.
- The do-file editor which allows you to easily enter, execute and save sequences of commands, or program fragments. Working in this way promotes reproducibility if all the results are generated by a series of .do files.

Re-using Results

Each of Stata's commands stores its results. Using stored results is particular important in constructing .do files. There are two important classes of commands.

- **r-class** and the command **return list**
  command **summarize** generates a number of scalars, such as r(N), the number of observations; r(mean), the mean; r(Var), the variance; etc. The available items are shown by return list for a r-class command. The contents of these scalars may be used in expressions.

```
quietly summarize price
summarize price if price > r(mean)
```

- **e-class** and the command **ereturn list**
  An e-class command will return e() scalars, macros and matrices: for instance, after regress, the local macro e(N) will contain the number of observations, e(r2) the R^2 value, e(depvar) will contain the name of the dependent variable, and so on.

Macros and Scalars

You should use these constructs whenever possible to avoid creating variables with constant values.

- **global**
  A global variable holds its value for the entire Stata session. The name of global is like a pointer in C and to access its value you need to de-reference it using **$name**.
- **local**
  A local variable holds its value while the current do-file is executed. It behaves like the global and must be de-refenced as follows **‘name’**.
- **scalar**
  a scalar can contain a single number (at maximum precision).

Looping Commands

Stata has three looping constructs: **foreach**, **forvalues** and **while**. The three loops will all be used to demonstrate summing 4 variables together

```
// generate 100 observations of 4 uniform random variables
clear
set obs 100
set seed 10101

generate x1var = runiform()
generate x1var = runiform()
generate x1var = runiform()
generate x1var = runiform()

generate sum = x1var + x2var + x3var + x4var
```

This can be achieved as follows using loops.

foreach loop

The **foreach** loop is used to loop over items in a list of variable names

```
// remember sum already exists so reset it
quietly replace sum = 0

foreach var of varlist  x1var x2var x3var x4var {
quietly replace sum = sum + ‘var’
}
```

Each variable in the loop is referred to by the local macro **var** so that it must be de-referenced in subsequent uses. The choice of **var** as the local macro name is arbitrary BUT the word **varlist** is necessary.

forvalues loop

The **forvalues** loop iterates over consecutive values. In the following code the local macro **i** is used as an index and so must be dereferenced when referring to it subsequently.

```
// sum still exists
quietly replace sum = 0

forvalues i = 1/4  {
quietly replace sum = sum + x‘i’var
}
```

Another useful notation for the loop is

```
forvalues i = 1(2)11
```

so that the index does from 1 to 11 in increments of 2

while loop

A **while** loop continues until a condition is no longer true.

```
// sum still exists
quietly replace sum = 0
local i 1

while `i' <= 4  {
quietly replace sum = sum + x`i'var
local i = `i' + 1
}
```

The **continue** command provides a way of ceasing execution of a loop prematurely.

Foreach loop again

In many cases, the **forvalues** command will allow you to substitute explicit statements with a single loop construct.
The **foreach** command, however, is even more useful. It defines an iteration over any one of a number of lists:

- the contents of a varlist (list of existing variables)
- the contents of a newlist (list of new variables)
- the contents of a numlist (list of integers)
- the separate words of a macro
- the elements of an arbitrary list

Building up a local macro

```
levelsof region, local(regid)
local alleps
foreach c of local regid {
regress lexp gnppc if region ==‘c’
predict double eps‘c’ if e(sample), residual
local alleps " ‘alleps’ eps‘c’ "
}
```

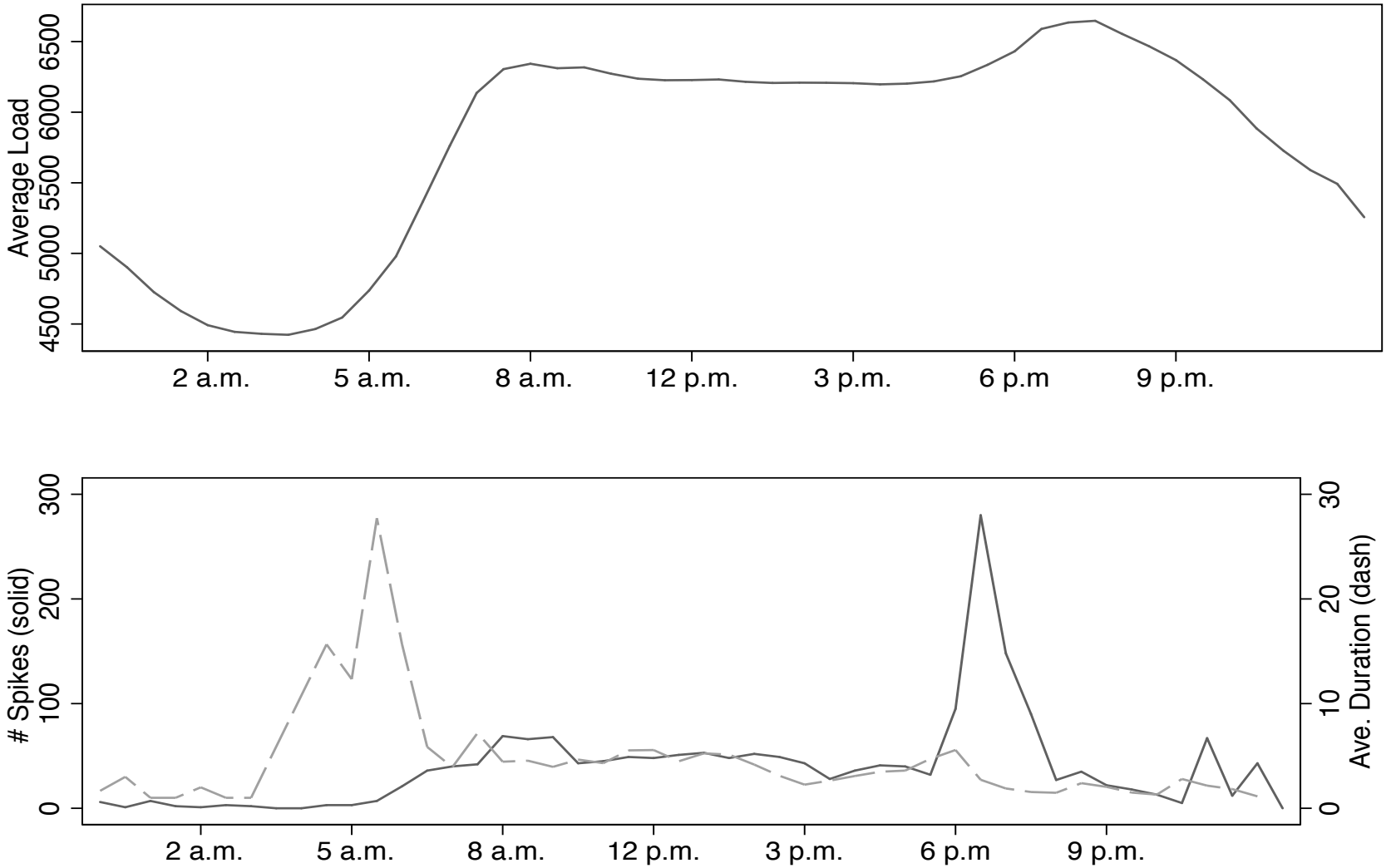
Automating Reporting of Results

The holy grail of reporting results is to be able to transfer results directly form the statistical package into the desired word processing format without the need for manual data entry. Sata provides a number of hand tools in this regard.

1. **tabstat** provides a compact table of summary statistics.
2. **estimates store [name]** will save the output from the current estimation procedure and the related **estimates table [names]** will provide a nice table of the output from the stored results.

Generating Publication Quality Tables

- Ben Jann's package **estout** is a comprehensive way of producing publication quality tables for use in LaTeX. This is described as a wrapper for the Stata command **estimates table**.
  - http://repec.org/bocode/e/estout
- A companion program **estadd** allows for additional user specified statistics to be added to the output.
- A simplified version of **estout** is is available as **esttab** and the utility command **eststo**.
- Transforming output directly into tables enhances reproducibility by removing the scope for human error in transcribing the results.


Average Load
4500
5000
5500
6000
6500
2 a.m.
5 a.m.
8 a.m.
12 p.m.
3 p.m.
6 p.m
9 p.m.
Spikes (solid)
0
100
200
300
Ave. Duration (dash)
0
10
20
30
2 a.m.
5 a.m.
8 a.m.
12 p.m.
3 p.m.
6 p.m
9 p.m.

Tabstats

```
. qui gen dummy07 = (datetime > clock("1jul2007 00:00", "DMYhm"))
.
. estpost tabstat durations unexpected_load qniflow , ///
>                         statistics(mean sd max min) by(dummy07)
Summary statistics: mean sd max min
     for variables: durations unexpected_load qniflow
  by categories of: dummy07
```

dummy07	e(durat~)	e(unexp~)	e(qnifl~)
0			
mean	3.823491	29.20877	286.8378
sd	5.424092	90.57061	402.9354
max	74	377	1547.7
min	1	-229	-350
1			
mean	3.195155	52.1726	445.2416
sd	3.571507	83.84323	476.7142
max	27	311	1535.15
min	1	-247	-5234.82
Total			
mean	3.619089	36.67906	338.3676
sd	4.906958	89.07235	434.613
max	74	377	1547.7
min	1	-247	-5234.82

```
.
. esttab using "./working/tabstats.tex", nomtitle nonumber noobs append ///
>                 cells("durations(fmt(3)) unexpected_load(fmt(3)) qniflow(fmt(3))")
(output written to ./working/tabstats.tex)
```

Tex Version

	durations	unexpected_load	qniflow
0			
mean	3.823	29.209	286.838
sd	5.424	90.571	402.935
max	74.000	377.000	1547.700
min	1.000	-229.000	-350.000
1			
mean	3.195	52.173	445.242
sd	3.572	83.843	476.714
max	27.000	311.000	1535.150
min	1.000	-247.000	-5234.820
Total			
mean	3.619	36.679	338.368
sd	4.907	89.072	434.613
max	74.000	377.000	1547.700
min	1.000	-247.000	-5234.820

Beauty is ...

Once you have used **esttab** to write the output, the TeX file may be called and input into your document without the need for manual transcription.

```
\scalebox{0.95}{\begin{minipage}{\hsize}
\begin{stlog}
\input{./do/working/lec1_stats.log.tex}
\end{stlog}
\end{minipage}}
```

Estimation Example

```
. // Probit
. eststo clear
. eststo: qui logit durations01 unexpected_load qniflow sum3hdd sum3cdd maxtemp mir
(est1 stored)
. eststo: qui logit durations01 unexpected_load qniflow sum3hdd sum3cdd maxtemp mir
(est2 stored)
. eststo: qui logit durations01 unexpected_load qniflow sum3hdd sum3cdd maxtemp mir
(est3 stored)
. eststo: qui logit durations01 unexpected_load qniflow sum3hdd sum3cdd maxtemp mir
(est4 stored)
.
. esttab using "./working/table.tex", b(4) se(4) scalars(r2_p ll bic) ///
>             replace booktabs nostar
(note: file ./working/table.tex not found)
(output written to ./working/table.tex)
.
```

Tex Version

	durations01	durations01	durations01	durations01
unexpected_load	0.0010	0.0012	0.0012	0.0012
	(0.0005)	(0.0005)	(0.0005)	(0.0005)
qniflow	0.0008	0.0009	0.0009	0.0009
	(0.0001)	(0.0001)	(0.0001)	(0.0001)
sum3cdd	0.0189	0.0182	0.0232	0.0213
	(0.0109)	(0.0110)	(0.0110)	(0.0110)
dummy07		-0.4186		
		(0.0960)		
dummy08			-0.6742	
			(0.1127)	
dummy09				-0.7730
				(0.1256)
N	2284	2284	2284	2284
r2_p	0.0297	0.0359	0.0413	0.0420
ll	-1498.8437	-1489.2908	-1480.8378	-1479.7399

Estimation Commands

- standard estimation commands
  - `regress`
  - `ivregress`
  - `gmm`
  - `probit`
  - `logit`
  - `var`
  - `sureg`
  - `xtreg, fe re be`
  - `xtabond`
  - `ml`
- two useful additions in Version 13
  - `forecast`
  - `mlexp`

Essential Packages

- the **ivreg2** package written by Baum, Schaffer and Stillman and related package **xtivreg2** by Schaffer.
  - http://ideas.repec.org/c/boc/bocode/s425401.html
- the **xtabond2** package written by Roodman
  - http://www.stata-journal.com/software/sj12-4

Matrix Programming Language

- Since the release of version 9, Stata has contained a full-fledged matrix programming language, Mata, with most of the capabilities of MATLAB, R, Ox or Gauss. Hypothetically it should be reasonably easy to translate the logic of other matrix languages into Mata (this is still on my to do list).
- A large library of mathematical and matrix functions is provided in Mata, including optimization routines, equation solvers, decompositions, eigensystem routines and probability density functions.
- You can use Mata interactively, or you can develop Mata functions to be called from Stata. (I will illustrate the latter use of Mata in the session on testing factor models.)
- Mata functions can access Statas variables and can work with virtual matrices (views) of a subset of the data in memory

An Efficient Division of Labour

- Mata interfaced with Stata provides for an efficient division of labor. In a pure matrix programming language, you must handle all of the housekeeping details involved with data organization, transformation and selection.
- But if you write an do-file that calls one or more Mata functions, the do-file will handle those housekeeping details.
- When the housekeeping chores are completed, the resulting variables can be passed on to Mata for processing.
- Results produced by Mata may then be accessed by Stata and formatted for display.
- Mata can access Stata variables, local and global macros, scalars and matrices, and modify the contents of those objects as needed. If Matas view matrices are used, alterations to the matrix within Mata modifies the Stata variables that comprise the view.

Data Access

- If youre using Mata functions in conjunction with Stata, one of the most important set of tools are Mata's interface functions: the **st_** functions.
- The first category of these functions provide access to data. Stata and Mata have separate workspaces, and these functions allow you to access and update Stata's workspace from inside Mata. For instance, **st_nobs()**, **st_nvar()** provide the same information as **describe** in Stata, which returns **r(N)**, **r(k)** in its return list.
- Mata functions **st_data()**, **st_view()** allow you to access any rectangular subset of Stata's numeric variables, and **st_sdata()**, **st_sview()** do the same for string variables.

st_view()

- One of the most useful Mata concepts is the view matrix, which as its name implies is a view of some of Stata's variables for specified observations, created by a call to **st_view()**. In other words, **st_view()** does not return a matrix that is a copy of the underlying values, but creates a matrix that is a view onto the Stata dataset itself.
- It requires three arguments: the name of the view matrix to be created, the observations (rows) that it is to contain, and the variables (columns). The statement

```
st_view(x, ., varname)
```

  states that the previously-declared Mata vector x should be created from all the observations (specified by the missing second argument) of **varname**.
- Note that **st_data()** does copy the Stata matrix and therefore is memory hungry.