Advanced Stata Programming

Ryan Kessler

Federal Reserve Bank of Boston

Summer 2012

Macros

A Stata **macro** is what most programming languages call a variable; they can be used to store content, numeric or string, which can then be inserted elsewhere in the code by invoking the macro. Macros can be local or global in scope, with the former existing solely within the program or do-file in which they're defined and the latter remaining available until they're dropped (`macro drop macroname`) or Stata is closed.

Local macros are invoked by placing the macroname between a backtick (‘) and a forward tick (’); global macros are invoked by preceding the macroname with the dollar sign ($). In either case, brace ({}) can be used to clarify meaning and to form nested constructions. For example:

```
* Local macros
local l1 "p"
local l2 "i"
gen `l1'`l2' = c(pi)
```

```
* Global macros
global pat "T:/census/ACS/"
do ${pat}acs2009use.do
```

We often use macros instead of **scalars** — a similar construct used to store numbers and strings (of less than 244 characters). But we really shouldn't.

1. Using a scalar will be faster than using a macro, because a macro must be converted from its internal (binary) representation into a printable form.
2. More importantly, these conversions into and out of binary representation can result in a loss of accuracy if the numeric quantity is not an integer.

By storing the result of a computation — for example, a variable's mean or standard deviation — in a scalar Stata need not convert its value and hence the result is held in Stata's full numeric precision.

```
local  x = sqrt(2)
scalar x = sqrt(2)

di (‘x’ == sqrt(2))
> 0
di ( x  == sqrt(2))
> 1
```

```
sysuse auto, clear

sum weight, detail
scalar q1 =r(p25)
scalar q3 =r(p75)
di "IQR = " q3 - q1
```

Two things to consider when assigning values to macros:

1. Beware the evaluating equal sign

```
local equal = 2*2
local noequal 2*2
display "`equal'"
> 4
display "`noequal'"
> 2*2
```

   The equal sign should be excluded in string assignments — with it, the macro is reduced to 244 characters or less; without it, macros can hold up to 1,081,511 characters!

2. Compound quotes (`" "')

```
local ra_leave ""David Coyne" "Kate Fritzsche"
"Tram Nguyen" "Rich Ryan"";
display "`ra_leave'";
> David not found
> r(111);
display `"`ra_leave'"';
"David Coyne" "Kate Fritzsche" "Tram Nguyen" "Rich Ryan"
```

Macro extended functions — typically of the form `local macroname : ...` — simplify data management tasks and bypass the 244 character cap.

```
// Assigning to *months* a list of the 12 months
local months `c(Months)'

// Replicating this list of months 3 times
local thr_months : display _dup(3) "`months' "

// Accessing the length of the macro
local len_months : length local thr_months
display "`len_months'"
> 258

// Replacing all instances of June with Junio
local thr_months : subinstr local thr_months "June" "Junio", all

// Putting list into alphabetical order
local thr_months : list sort thr_months

display "`thr_months'"
> April April April August August August December December...
> Junio Junio March March March... September September September
```

```
#delimit;
local ra_list1 `""Tamas Briglevics" "Angela Cools" "Sean Connolly"
"David Coyne" "Matthew Curtis" "Julia_Dennett" "Vladimir Yankov"
... "Yifan Yu" "Hanbing Zhang" "Chuanqi Zhu""';

local l1 = length(`"`ra_list1'"');
local l2 : length local ra_list1;

di "Truncated length = `l1'";
> Truncated length = 245
di "Actual length    = `l2'";
> Actual length    = 387

local ra_leaving `""David Coyne" "Kate Fritzsche"
"Tram Nguyen" "Rich Ryan""';

local ra_list2 : list ra_list1 - ra_leaving;
local wc1 : word count `ra_list1';
local wc2 : word count `ra_list2';

di "# of RAs in early July  = `wc1'";
> # of RAs in early July  = 24
di "# of RAs in late August = `wc2'";
> # of RAs in late August = 20
```

Structures (Loops)

Stata provides 3 structures for cycling through lists of values — variable names, numbers, text — and repeating commands:

1. `foreach`
2. `forvalues`
3. `for`

The `foreach` loop, introduced with Stata 7 (2001), is probably the most useful of the three; `forvalues` is really a special case of `foreach`, intended for cycling through certain kinds of `numlists`. The `for` command is rarely used — it is, in my opinion, difficult to understand and debug. In addition to these structures, Stata does of course have a `while` loop.

From Nick Cox's "How to face lists with fortitude"

If you know `for` and like it, then stick with it. But remember that when the going gets tough, the tough get going.

```
// Looping over all variables in memory
foreach x of varlist * {
    capture gen log_`x' = log(`x')
}
```

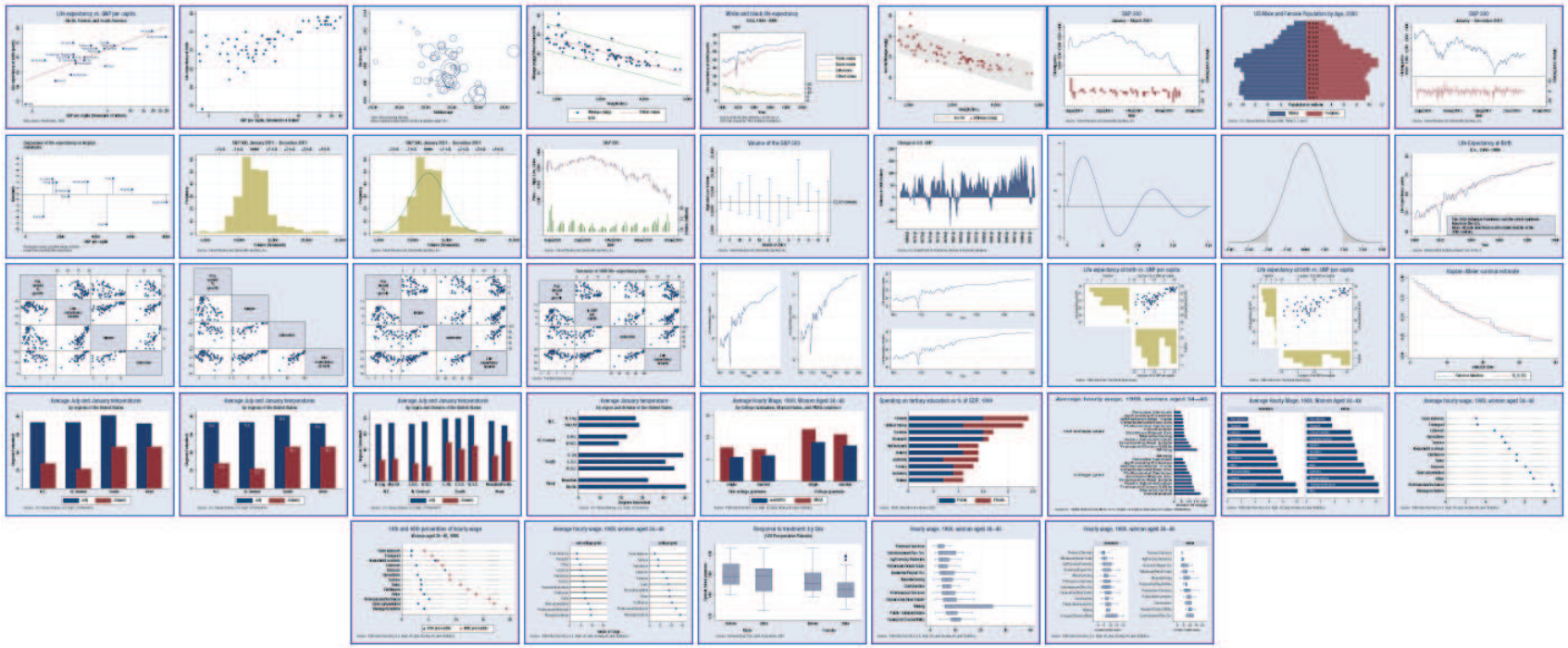
```
// Looping over years of PSID
foreach y of numlist 1968/1997 1999(2)2009 {
    gen inc_`y' = income if year == `y'
}
```

```
// Looping over elements in a macro
local months `c(Months)'
foreach m in `months' {
    di "`m'"
}
```

```
// Counting down from 10, by 2's
local i=10
while `i' >= 0 {
   di "`i'"
   local i = `i'-2
}
```

```
// Counting down from 10, by 2's
forval i=10(-2)0 {
   di "`i'"
}
```

Stata comes with a number of standard graphs: scatter and line plots, bar and pie charts, etc.



Stata graphic galleries:

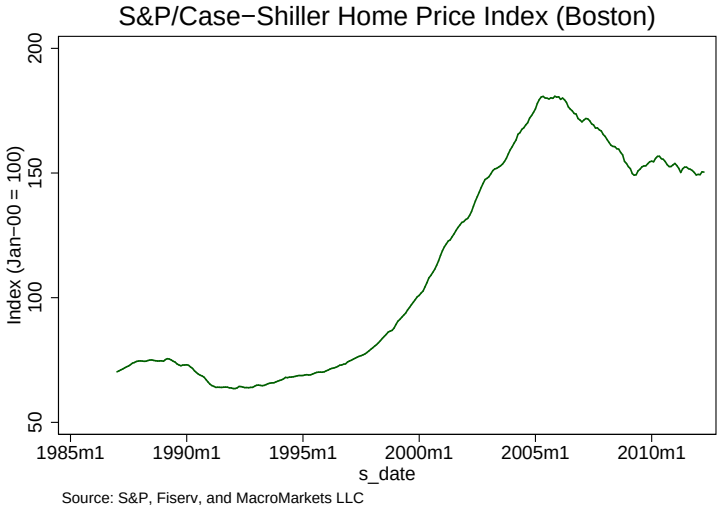
http://www.ats.ucla.edu/stat/Stata/library/GraphExamples/default.htm
http://www.survey-design.com.au/Usergraphs.html
http://www.stata.com/support/faqs/graphics/gph/statagraphs.html

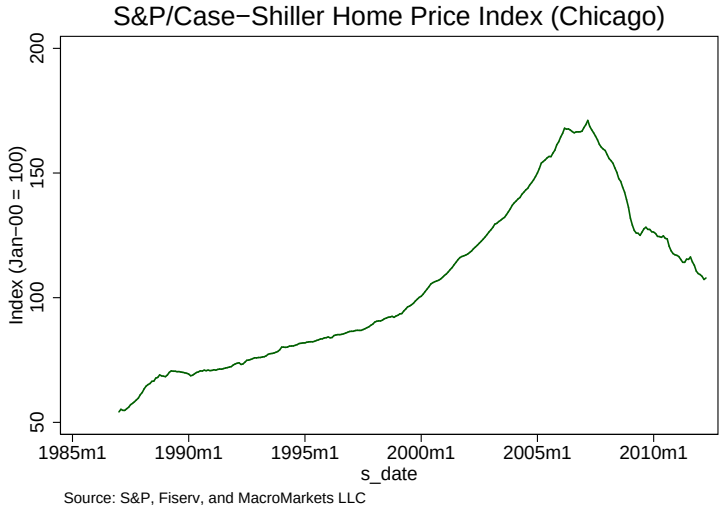
Back to our Case-Shiller house price data. Let's generate simple plots of the Boston and Chicago Indexes.

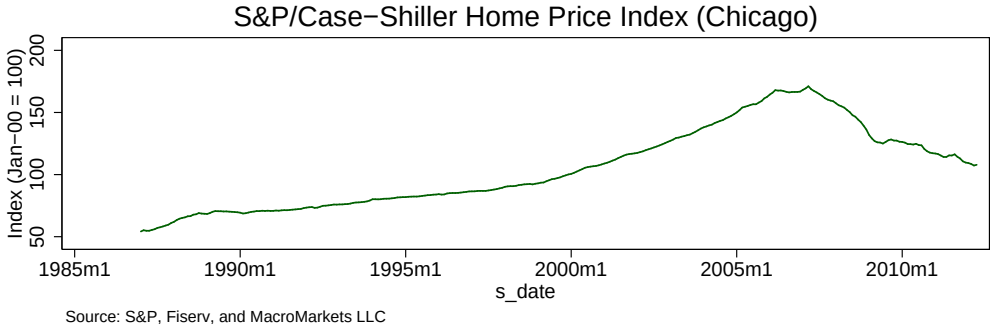
```
// Chicago
line chicago s_date, ytitle("Index (Jan-00 = 100)")          ///
title("S&P/Case-Shiller Home Price Index (Chicago)")          ///
note("Source: S&P, Fiserv, and MacroMarkets LLC")             ///
name(fig_chicago, replace)

// Analogous figure for Boston
line boston s_date, ytitle("Index (Jan-00 = 100)")            ///
title("S&P/Case-Shiller Home Price Index (Boston)")           ///
note("Source: S&P, Fiserv, and MacroMarkets LLC")             ///
name(fig_boston, replace)
```

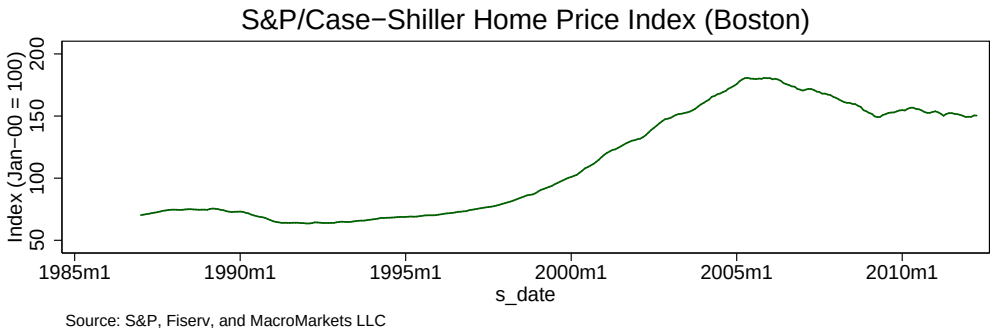
Using the `graph combine` command we can place both graphs on the same canvas and specify that share common axis scales.

```
graph combine fig_chicago fig_boston, cols(1) xcommon ycommon
```


S&P/Case−Shiller Home Price Index (Boston)
Index (Jan−00 = 100)
50
100
150
200
1985m1
1990m1
1995m1
2000m1
2005m1
2010m1
s_date
Source: S&P, Fiserv, and MacroMarkets LLC


S&P/Case−Shiller Home Price Index (Chicago)
Index (Jan−00 = 100)
50
100
150
200
1985m1
1990m1
1995m1
2000m1
2005m1
2010m1
s_date
Source: S&P, Fiserv, and MacroMarkets LLC


S&P/Case−Shiller Home Price Index (Chicago)
Index (Jan−00 = 100)
50
100
150
200
1985m1
1990m1
1995m1
2000m1
2005m1
2010m1
s_date
Source: S&P, Fiserv, and MacroMarkets LLC


S&P/Case−Shiller Home Price Index (Boston)
Index (Jan−00 = 100)
50
100
150
200
1985m1
1990m1
1995m1
2000m1
2005m1
2010m1
s_date
Source: S&P, Fiserv, and MacroMarkets LLC

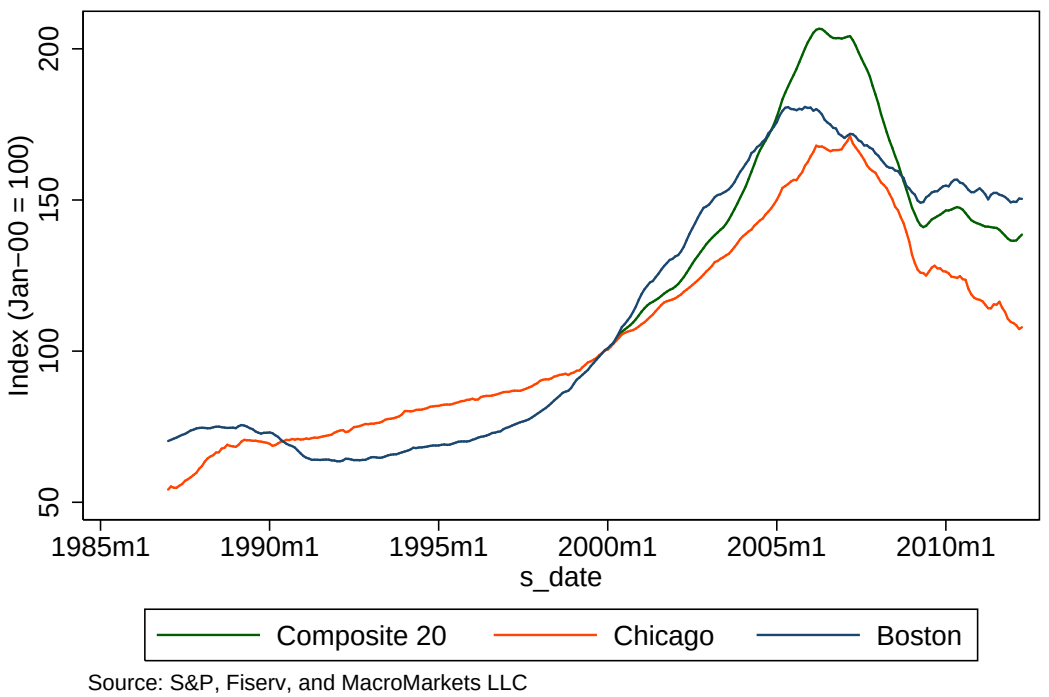
Still, it would be nicer to have a more direct comparison. Stata's `twoway` command can be used to combine multiple (compatible) graph types into the same figure; `twoway` takes multiple other graph commands as arguments, each in parentheses.

```
// Composite 20, Chicago, and Boston
twoway (line comp20 s_date) (line chicago s_date)              ///
(line boston s_date), ytitle("Index (Jan-00 = 100)")           ///
title("S&P/Case-Shiller Home Price Indices")                   ///
note("Source: S&P, Fiserv, and MacroMarkets LLC")              ///
legend(label(1 "Composite 20") label(2 "Chicago")              ///
label(3 "Boston") cols(3)) name(comp_chi_bos, replace)
graph export "comp_chi_bos.eps", as(eps) replace
```

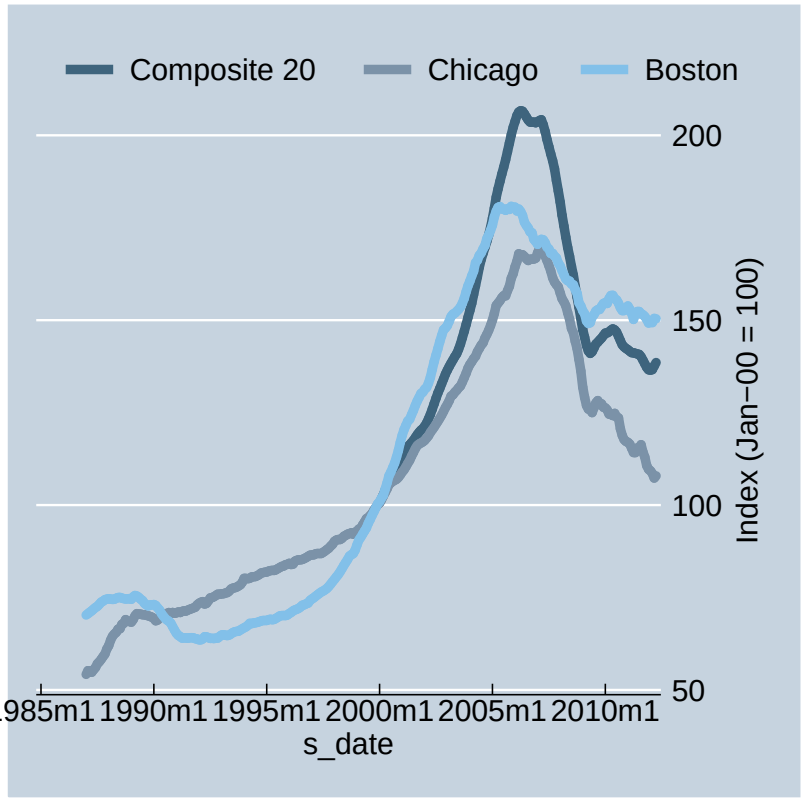
We might consider changing this graph's overall appearance. Using the `set scheme` command, we can do just that (type `graph query, schemes` to obtain a list of available themes). For example:

```
set scheme economist
```

`set scheme s1color`



`set scheme economist`



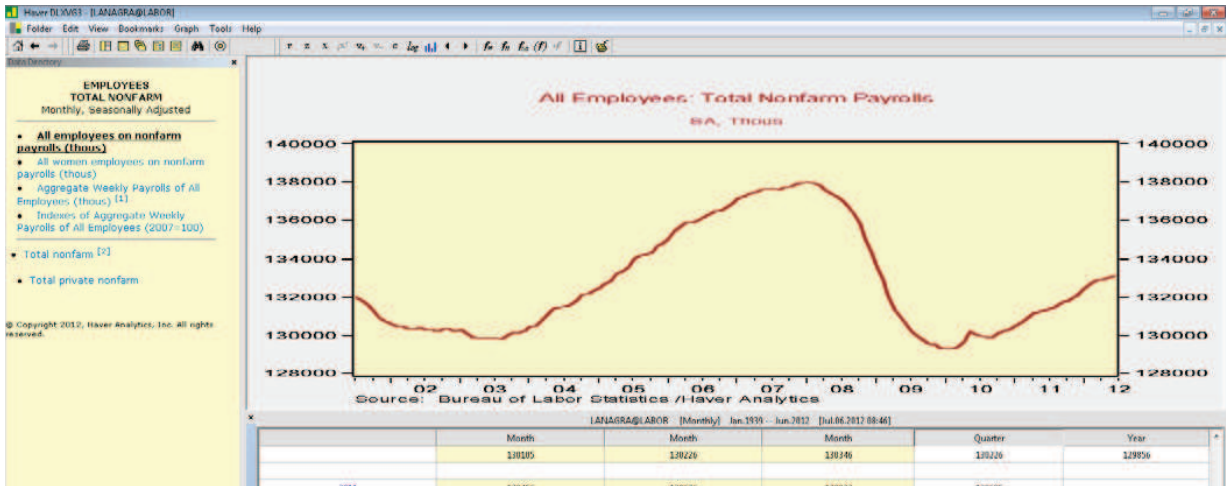
Time now for some different data.

Haver

Stata provides a command, `haver use`, that allows you to load data from the Haver Analytics database, provided you know the appropriate mnemonics.

There is one problem, though: Haver data lives on your `C:` drive and hence cannot be accessed via the Linux Cluster. To use these Haver commands, you must use PC Stata.

Once we identify a data series, all we need to know is the mnemonic and the file in which it's stored, both of which are listed right below the default chart in Haver.

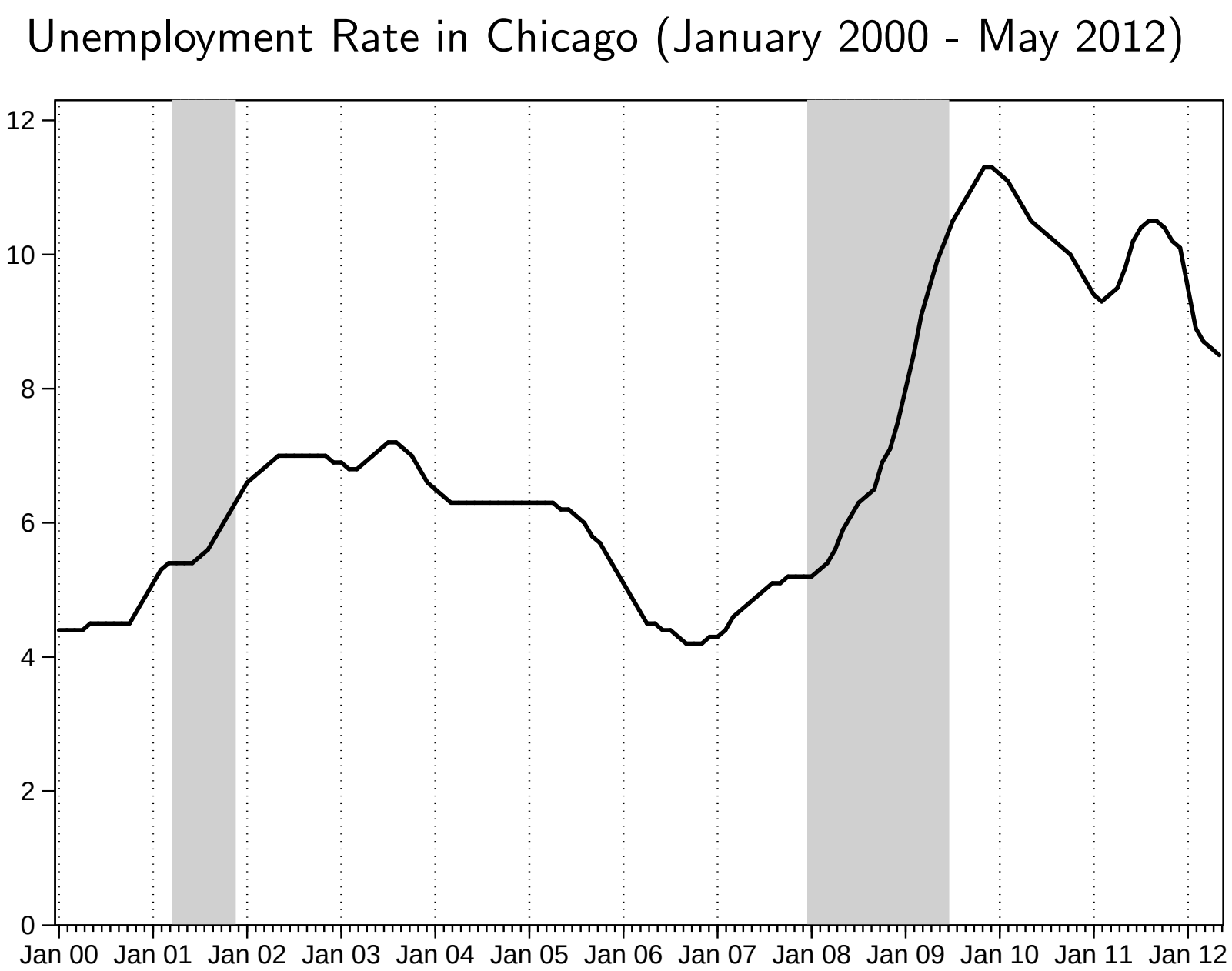


Say we want to plot Chicago's unemployment rate from January 2000-present, with recession shading.

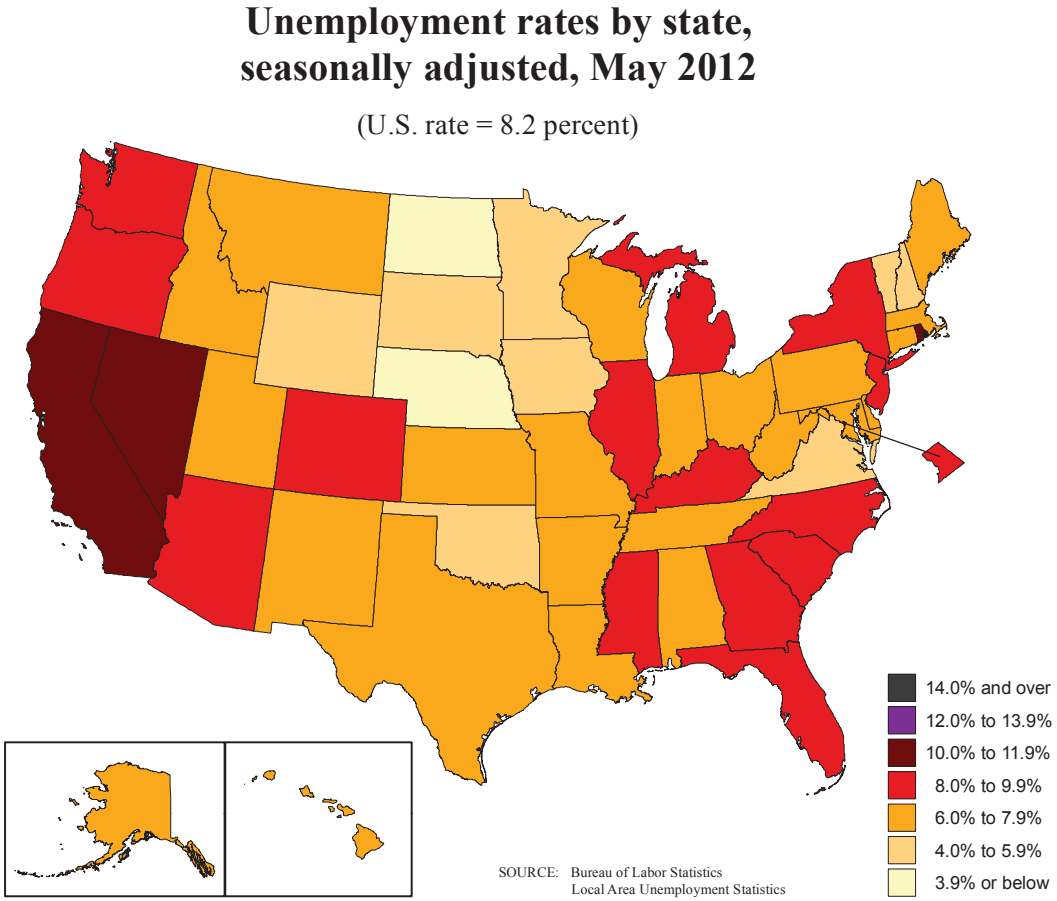
Chicago's unemployment rate is readily available in Haver under the mnemonic `CNJRA`, which lives in the regional employment file `EMPLR.DAT`. The NBER recession indicator is available as well (`RECESSM2`) but must be pulled separately since it lives in `USECON.DAT`.

```
tempfile unemployment

haver use CNJRA using "C:/Haver/EMPLR.DAT", clear
save `unemployment'

haver use RECESSM2 using "C:/Haver/USECON.DAT", clear

* Merging unemployment data with recession indicator
merge 1:1 time using `unemployment', keep(match) nogenerate
```

```
// Generating/formatting date variable
gen mydate  = dofm(time)
gen date  = ym(year(mydate),month(mydate))
format date %tmMon_YY
tsset date

// We want our plot to start on January 2000
keep if year(mydate) >= 2000

// Making appropriate adjustment to recession indicator
sum CNJRA
gen recess = (r(max)+1)*RECESSM2

#delimit;
twoway (bar recess date, fintensity(100) lcolor(gs13) fcolor(gs13))
(tsline CNJRA, tlabel(#10, grid gextend gstyle(dot) labsize(small))
tmtick(##11)clcolor(black) clwidth(medthick) ytitle("")
ylabel(0(2)`=r(max)', nogrid angle(horizontal) format(%9.0f)
labsize(small)) ttitle("") graphregion(color(white))
plotregion(margin(zero)) legend(off));
graph export "chicago_unemp.eps", as(eps) replace;
#delimit cr
```


Unemployment Rate in Chicago (January 2000 - May 2012)
12
10
8
6
4
2
0
Jan 00
Jan 01
Jan 02
Jan 03
Jan 04
Jan 05
Jan 06
Jan 07
Jan 08
Jan 09
Jan 10
Jan 11
Jan 12

Say we want to recreate a map posted by the BLS

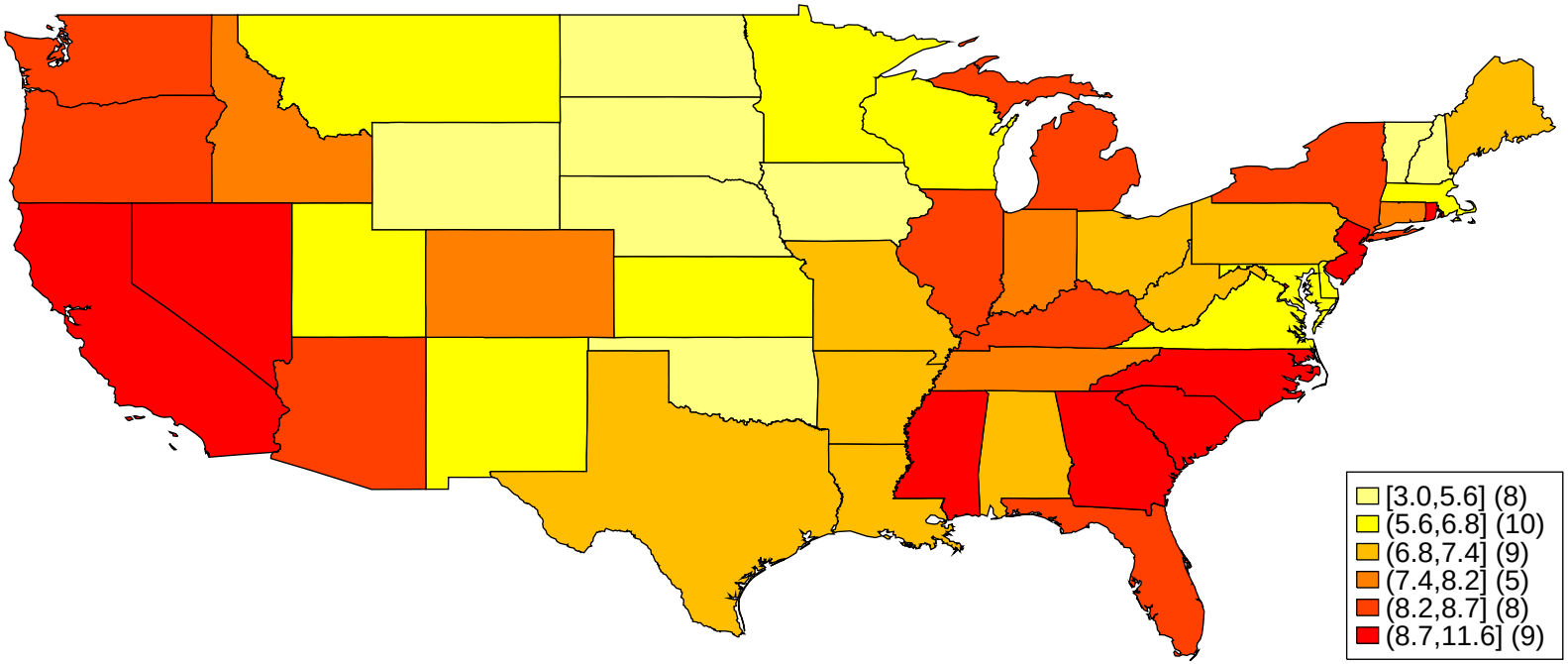


We can do so using the `shape2dta` and `spmap` commands.

```
// Pulling data from Haver
haver use ALRA AKRA AZRA ARRA CARA CORA CTRA DCRA DERA FLRA      ///
GARA HIRA IDRA ILRA INRA IARA KSRA KYRA LARA MERA MDRA MARA      ///
MIRA MNRA MSRA MORA MTRA NERA NVRA NHRA NJRA NMRA NYRA NCRA      ///
NDRA OHRA OKRA ORRA PARA RIRA SCRA SDRA TNRA TXRA UTRA VTRA      ///
VARA WARA WVRA WIRA WYRA using "C:/Haver/EMPLR.DAT", clear

tempfile unemployment
gen id=_n
keep if id==_N

// Renaming to facilitate reshape, wide --> long
foreach x of varlist *RA {
   local stub = subinstr("`x'","RA","",1)
   rename `x' RA`stub'
}
reshape long RA, i(id) j(state) string
save `unemployment'
```

```
* Bringing our shapefile into Stata
shp2dta using "states.shp", replace                                    ///
database("state_data") genid(id)                                       ///
coordinates("state_coordinates")

use "state_data", clear
rename STATE_ABBR state

// Merging with unemployment data
merge 1:1 state using `unemployment', keep(match)

// Restricting to the continental US
keep if state != "AK" & state != "HI"

// Formatting unemployment rate
format RA %7.1f

// Generating map
spmap RA using "state_coordinates", id(id)                             ///
legenda(on) legtitle() fcolor(Heat) legorder(lohi)                     ///
legcount clnumber(6) legend(region(lcolor(black)                       ///
fcolor(white)) size(*2) position(4))

graph export "state_unemp.eps", as(eps) replace
```

Unemployment Rate (SA), May 2012



Outputting Results

Anyone who performs empirical research is familiar with the tedious task of turning estimation output into tables, with appropriate handling of standard errors or t-statistics, p-values, significance stars and presentation of summary statistics.

Stata's `estimates` commands make that a bit easier by allowing you to `estimates store` and produce a crude but readable table from several sets of output, with some control over the format and contents of the table, with `estimates table`. But they leave us a long way from producing publishable-quality results.

Fortunately, a number of folks — namely, Ben Jann (`estout`), Roy Wada (`outreg2`), and Ian Watson (`tabout`) — have created easy-to-use routines that turn sets of estimates into publication-quality tables.

There's a battle for first place on the `ssc hot` list between Ben Jann's `estout` and Roy Wada's `outreg2`.

Ben Jann

Roy Wada

We'll discuss Ben Jann's `estout` and its related wrappers, but I suggest you become familiar with both.

First, note that `estout` has a website — `http://repec.org/bocode/e/estout/` — on which you'll find explanations of all of the available options and numerous worked examples.

To use `estout`, you merely apply the `eststo:` prefix:

```
eststo clear
eststo: reg price mpg weight
eststo: reg price mpg weight foreign
eststo: logit foreign mpg price
eststo: logit foreign mpg price weight
```

Then, to produce a table, we use `esttab` — an easy-to-use wrapper for `estout`, which has many options to control the exact format and content of the table.

```
esttab using "esttab_auto.tex", replace label eqlabels(none)
```

This will create a LaTeX table in that file. A file destined for Excel would use the `.csv` extension; for word, `.rtf`.

	(1) Price	(2) Price	(3) Car type	(4) Car type
Mileage (mpg)	-49.51	21.85	0.160**	-0.169
	(-0.57)	(0.29)	(3.04)	(-1.83)
Weight (lbs.)	1.747**	3.465***		-0.00391***
	(2.72)	(5.49)		(-3.86)
Car type		3673.1***		
		(5.37)		
Constant	1946.1	-5853.7	-4.379***	13.71**
	(0.54)	(-1.73)	(-3.62)	(3.03)
Observations	74	74	74	74

t statistics in parentheses

* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

Adding to the previous table each model's BIC.

```
esttab using "esttab_auto_stat.tex", replace                ///
label eqlabels(none) stats(bic N, fmt(%7.0f %7.0f)          ///
labels(BIC Observations))
```

	(1) Price	(2) Price	(3) Car type	(4) Car type
Mileage (mpg)	-49.51	21.85	0.160**	-0.169
	(-0.57)	(0.29)	(3.04)	(-1.83)
Weight (lbs.)	1.747**	3.465***		-0.00391***
	(2.72)	(5.49)		(-3.86)
Car type		3673.1***		
		(5.37)		
Constant	1946.1	-5853.7	-4.379***	13.71**
	(0.54)	(-1.73)	(-3.62)	(3.03)
BIC	1379	1357	87	67
Observations	74	74	74	74

t statistics in parentheses

* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

Ian Watson's `tabout` allows us to produce oneway or twoway tables of frequencies and/or percentages, as well as summary statistics.

```
tabout foreign rep78 using "tabout_auto.tex", replace   ///
 c(freq) style(tex)
```

	Repair Record 1978					
Car type	1 No.	2 No.	3 No.	4 No.	5 No.	Total No.
Domestic	2.0	8.0	27.0	9.0	2.0	48.0
Foreign	0.0	0.0	3.0	9.0	9.0	21.0
Total	2.0	8.0	30.0	18.0	11.0	69.0

```
tabout foreign using "tabout_auto2.tex", replace sum    ///
c(mean mpg median mpg mean weight median weight) style(tex)
```

Car type	Mean mpg	Median mpg	Mean weight	Median weight
Domestic	19.8	19.0	3,317.1	3,360.0
Foreign	24.8	24.5	2,315.9	2,180.0
Total	21.3	20.0	3,019.5	3,190.0

If all else fails, you can turn to the `file` command, which reads and writes ASCII text and binary files.

```
ttest mpg, by(foreign)

file open  ttest using "ttest.csv", write replace
file write ttest "mean diff., std. err., p-value"           _n
file write ttest (r(mu_1)-r(mu_2)) "," (r(se)) "," (r(p))
file close ttest
```

mean diff.	std. err.	p-value
-4.9458042	1.3621621	0.00052542

Bottom line: you should never be copying and pasting Stata output from the results window or a log file!

Programs & Ado-files

First, some Stata nomenclature:

1. Stata formally defines a **program** as a set of Stata commands that includes a `program` statement.
2. An **ado-file** is simply a file that stores a program (see, for example, `viewsource estout.ado`).

You can easily write a Stata program, stored in an ado-file, that handles all the features of official Stata commands such as the `if` and `in` options. You can even write a help file that documents its operation, for your benefit and for those with whom you share the code.

But you probably shouldn't! In most cases, existing Stata commands — official or user-written — will perform the tasks you need. Use Stata's search features such as `findit` and the Stata user community (Statalist) to ensure that the program you envision has not already been written.

A Stata program adds a command to Stata's language. The name of the program is the command name, and the program must be stored in a file of that same name with extension `.ado`, and placed on the adopath — i.e., the list of directories that Stata will search to locate programs.

A program begins with the `program define program_name` statement, which usually includes the option `, rclass`. The program name should not be the same as any Stata command, nor the same as any accessible user-written command!

The `syntax` command will almost always be used to define the command's format. For instance, a command that accesses one or more variables in the current data set will have a `syntax varlist` statement. The `syntax` statement does allow you to specify `[if]` and `[in]` arguments, which allow commands to limit the observations used. The `syntax` statement may aslo include a `using` qualifier, allowing your command to read or write files.

Really, any feature you find in an official Stata command can be implemented with an appropriate `syntax` statement.

A sample program from `help return`:

```
program mysum, rclass
    syntax varname
    tempvar new
    quietly {
        count if !missing(`varlist')
        return scalar N = r(N)
        gen double `new' = sum(`varlist')
        return scalar sum = `new'[_N]
        return scalar mean = return(sum)/return(N)
    }
end
```

This program can be executed as `mysum varname`. It prints nothing but places three scalars in the return list. The values `r(N)`, `r(sum)`, and `r(mean)` can now be referred to directly.

```
mysum mpg
display r(N) "    " r(sum) "    " r(mean)
>74    1576    21.297297
```

Monte Carlo simulation

Let's consider a population model $Y = \beta_0 + \beta_1 X + \epsilon$, where the predictor X takes on one of three values, $X_i \in \{1, 2, 3\}$, and the response $Y \sim N(X_i, 1)$. Furthermore, let's suppose there are 100 units at each value of X: $n = 300$ and $n_1 = n_2 = n_3 = 100$.

We will use a simulation to demonstrate that

1. $\hat{\beta}_1$ is unbiased for β_1
2. we must use a robust variance-covariance matrix in a context of heteroskedasticity.

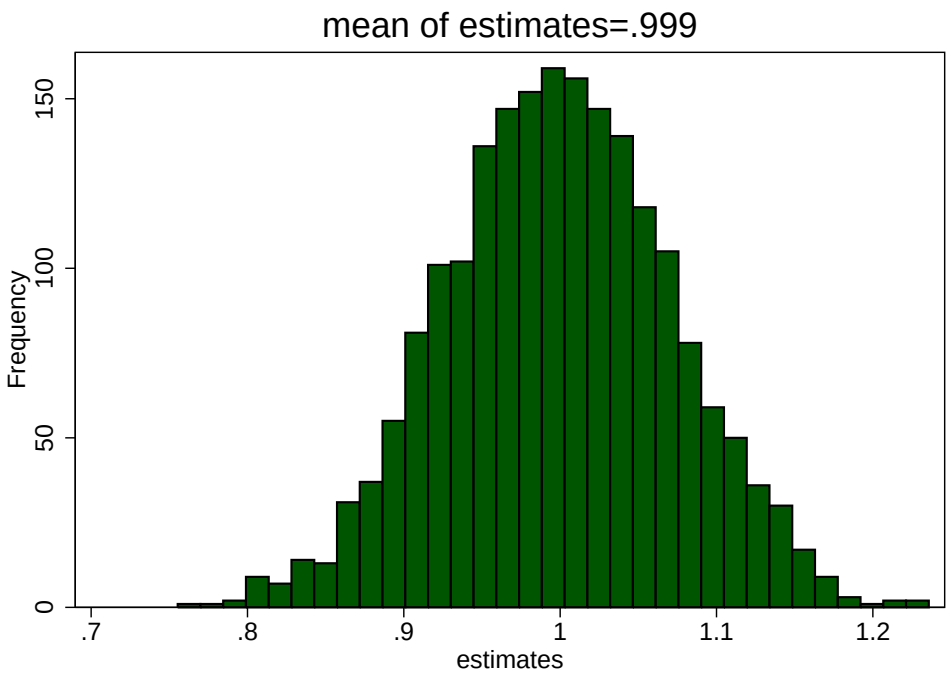
Simulations in Stata are often done using a loop and the `postfile` command or a program and the `simulate` command. Given that we're discussing programs, we'll use the latter — i.e., we'll write a program that sets up the experiment and specifies what is to be done in one replication and then `simulate` the program a specified number of times.

```
capture program drop ols_sim
program define ols_sim, rclass
   syntax [, NONCONstant robust]
   set obs 300
   tempvar y x
   gen `x'=1 in 1/100
   replace `x'=2 in 101/200
   replace `x'=3 in 201/300
   if "`nonconstant'"!="" gen `y'=rnormal(`x',`x'^2) in 1/300
   else gen `y'=rnormal(`x',1) in 1/300

   reg `y' `x', `robust'
   return scalar b1=_b[`x']
   test `x'=1
   return scalar pv=r(p)
end
```

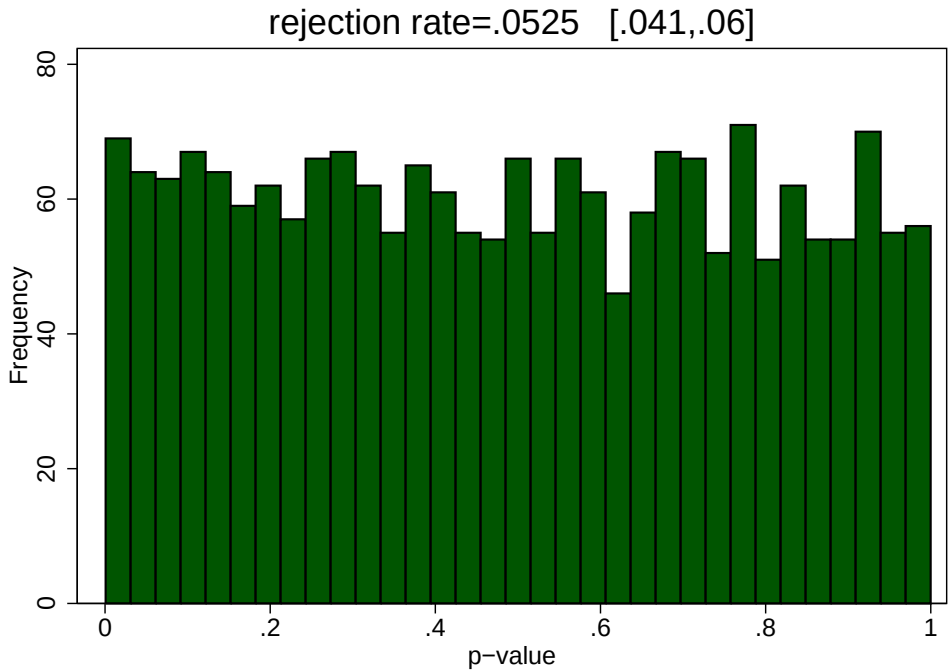
We use the `simulate` command to run `ols_sim` 2000 times:

```
simulate b1=r(b1) pv=r(pv), reps(2000): ols_sim
```

$\hat{\beta}_1$ is unbiased for β_1



test size $\approx$ nominal size

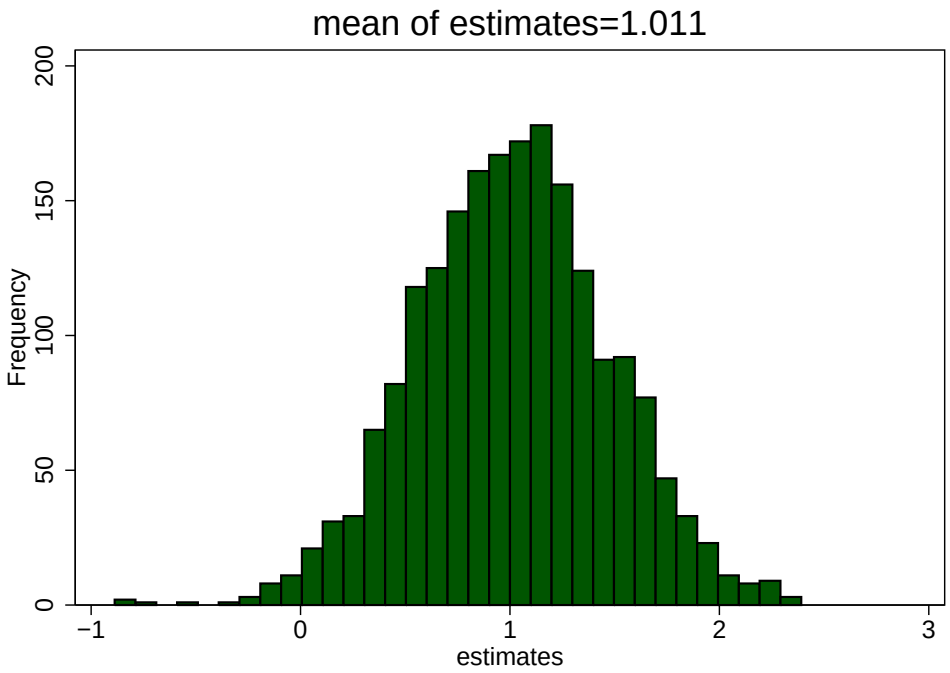


Note: `cii 2000 2000*0.05` yields an exact binomial confidence interval for the rejection rate (in square brackets).

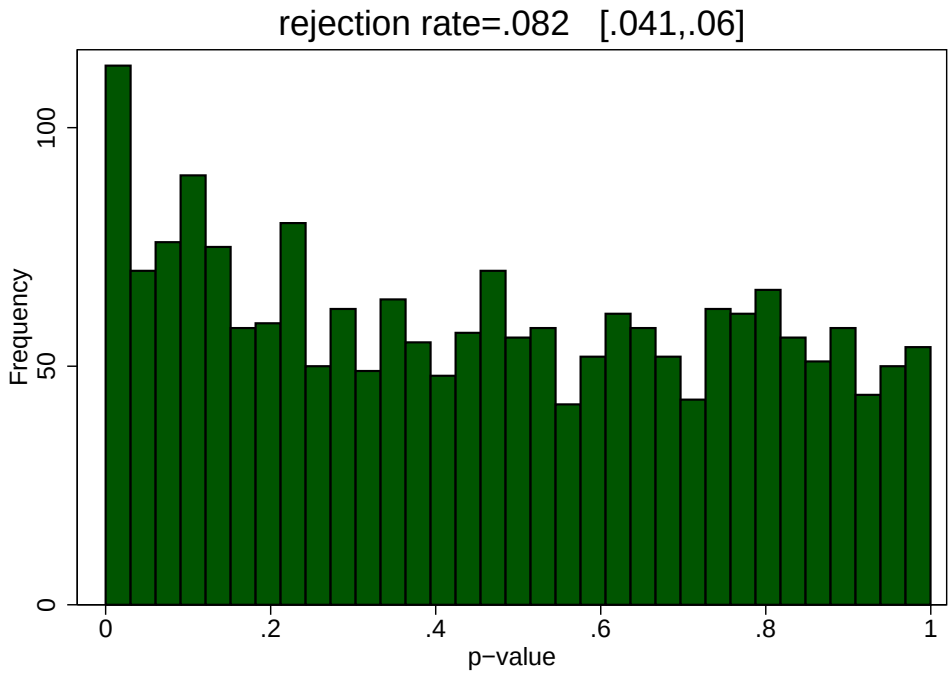
Now suppose we introduce some heteroskedasticity by invoking the `nonconstant` option. Will $\hat{\beta}_1$ still be unbiased for β_1? Will the rejection rate still be ≈ 0.05?

```
simulate b1=r(b1) pv=r(pv), reps(1000): ols_sim, noncon
```

$\hat{\beta}_1$ is still unbiased for β_1



test size $>$ nominal size

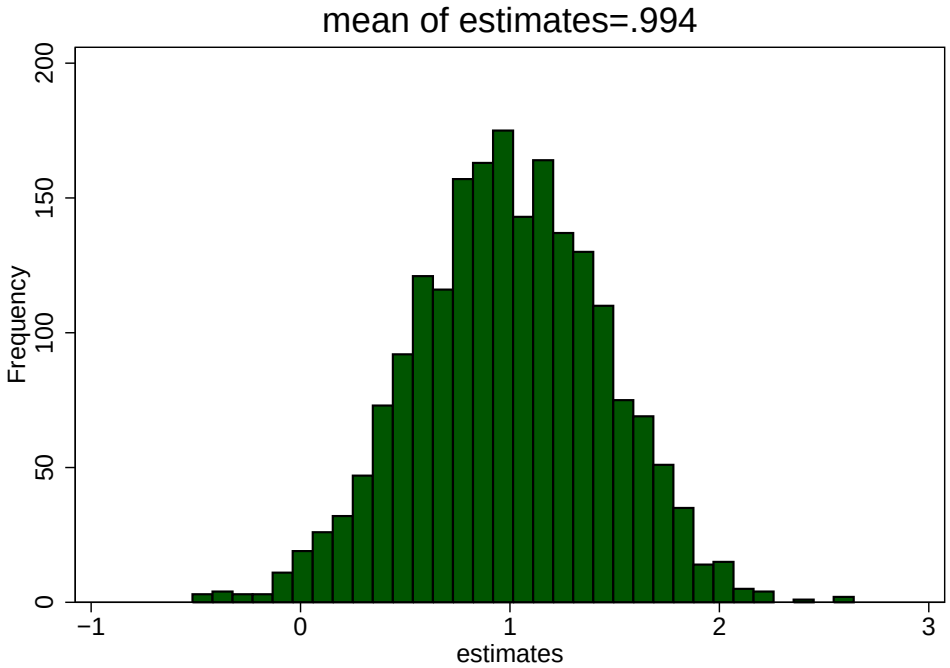


The standard errors inaccurately describe the uncertainty in the estimates, leading to misleading tests and confidence intervals.

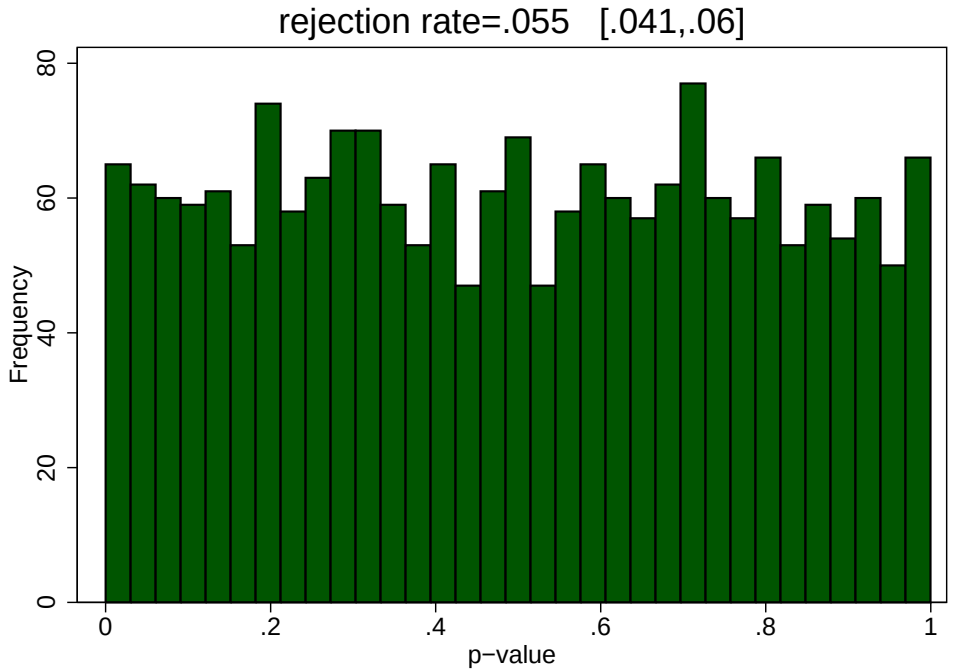
Fortunately, we can easily compute Huber-White standard errors that account for nonconstant variance by specifying the `robust` option

```
simulate b1=r(b1) pv=r(pv), reps(1000): ols_sim, noncon robust
```

$\hat{\beta}_1$ is unbiased for β_1



test size ≈ nominal size



Mata

As of verion 9, Stata contains a full-blown matrix programming language, *Mata*, with many of capabilities of MATLAB and R. Why should you care?

1. Mata is automatically compiled into bytecode and hence runs many times faster than the standard ado-file language.
2. Mata circumvents the limitations of Stata's traditional matrix commands.
3. Mata contains a large library of mathematical and matrix functions, including optimization routines, equation solvers, decompositions, and probability density functions.

I'm not going to dwell on Mata syntax — it's largely similar to C and its handling of matrices is broadly similar to that of other matrix programming languages (see `help mata`).

```
sysuse auto, clear

// Entering Mata
mata
// Defining matrices
x = st_data(., ("mpg"))
y = st_data(., ("price"))
cons = J(rows(x), 1, 1)
X = (x, cons)

// Some linear algebra
beta_hat = (invsym(X'*X))*(X'*y)
e_hat = y - X * beta_hat
s2 = (1 / (rows(X) - cols(X))) * (e_hat' * e_hat)
V_ols = s2 * invsym(X'*X)
se_ols = sqrt(diagonal(V_ols))

// Returning results
st_matrix("beta_hat",beta_hat)
st_matrix("se_ola",se_ols)
// Leaving Mata
end
```

```
. reg price mpg

      Source |       SS       df       MS              Number of obs =      74
-------------+------------------------------           F(  1,    72) =   20.26
       Model |   139449474     1   139449474           Prob > F      =  0.0000
    Residual |   495615923    72  6883554.48           R-squared     =  0.2196
-------------+------------------------------           Adj R-squared =  0.2087
       Total |   635065396    73  8699525.97           Root MSE      =  2623.7

------------------------------------------------------------------------------
       price |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         mpg |  -238.8943   53.07669    -4.50   0.000    -344.7008   -133.0879
       _cons |   11253.06   1170.813     9.61   0.000     8919.088    13587.03
------------------------------------------------------------------------------
```

```
matrix list beta_hat

beta_hat[2,1]
             c1
r1  -238.89435
r2   11253.061
```

```
matrix list se_ols

se_ols[2,1]
             c1
r1  53.076687
r2  1170.8128
```

Shell (!)

Stata can easily invoke commands in the underlying operating system with the `shell` command (the shortcut `!` can also be used and in this context is distinct from its use as a negation operator). In Linux, `shell` invokes the terminal — also known as the shell or console.

Using StatTransfer within Stata to avoid the nightmare that is SAS

```
local psid_sas "/home/a1rek01/psid_sas/"

* Navigating to StatTransfer command
cd /opt/apps/stattransfer/stattransfer11_64/

* Converting files
! ./st copy "`psid_sas'*.sas7bdat" "`psid_sas'*.dta"

* Creating list of resulting .dta files
cd /home/a1rek01/psid_sas/
local stata_list : dir . files "*.dta"
di `""`stata_list'""'
```

We can define a program that returns the date on which a file or folder was last modified and the number days that have elapsed since that date.

```
capture program drop last_mod
program define last_mod, rclass
    syntax, FILEname(string)

    tempfile date
    qui !date -r `filename' `+%d %b %Y'  > `date'

    preserve
    qui insheet using `date', clear nonames
    local today    = date("`c(current_date)'","DMY")
    local mod_date = date(v1,"DMY")
    local diff     = `today' - `mod_date'

    return local lm  `=v1[1]'
    return scalar diff = `today' - `mod_date'
    restore
end

last_mod, filename(/shared/census/ACS/acs2009use.ado)
di r(lm)
> 11 Aug 2011
```

Stata resources

Have questions? Use the following resources:

1. Stata help files
2. Google
3. Stata manuals (located on T-8)
4. Other RA's
5. Statalist (be sure to read the Statalist FAQ's before posting!). You'll probably receive a response within the hour from

Nick Cox

Kit Baum

Much of the information found in this presentation was pulled from the following files:

- Why become a Stata programmer by Kit Baum
- Monte Carlo Simulation in Stata by Kit Baum
- How to face lists with `fortitude` by Nick Cox
- Mata in Stata by Kit Baum
- `estout` website maintained by Ben Jann
- Advanced Graphics Programming in Stata by Sergiy Radyakin