Automating Your Work:
An Introduction to Programming in Stata

Shawna N. Smith

31 March 2012

…but why?

- Kinsey International Survey of Relationships (Heiman et al. 2011)
  - First question: Are there country-level differences in variables of interest?
    - 5 countries: US, Spain, Brazil, Germany, Japan
    - 2 genders: Males & females
    - 30 variables of interest= 300 potential differences
- Dissertation work
  - Research question: Are there differences in 'political minority' labor market cleavages across countries?
    - 3 countries
    - 6 'political minorities'
    - 2 genders
    - 3 outcomes variables = 108 sets of logit coefficients
- Comparative measurement modeling (Medina, Smith & Long 2009)
  - Research question: Does measurement modeling approach matter affect discovery of significant country-level differences?
    - (17 countries x 6 variables) * (17 countries * 6 variables)
    - 2 factors x 3 estimations ≈ 8400 F-tests

Roadmap

- Writing effective do-files {brief intro}
- Automation
  - Macros
  - Saved results
  - Loops
  - Matrices
  - Ado-files {preview}

The Workflow of Data Analysis Using Stata

By J. Scott Long

- This talk is premised on *Chapter 4: Automating your work*
- For example files: type `findit workflow` and follow the instructions

{intro} Writing effective do-files

- **Robust**: To be robust, a do-file must produce *exactly the same result* when run at a later time or on another computer
- **Legible**: To be legible, a do-file must be documented and formatted so that it is *easy to understand* what is being done

Robust

- Self-contained
- Include version control
- Exclude directory information
  - Never hardcode your directory! Rather, set your working directory before you start your work
  - E.g., use:

```
cd _working
```
{in command line, pre work}

not:

```
saveas "/Users/shawnasmith/Proj1"
```

Legible

- Use comments {early/often}
- Use alignment and indentation
- Use short lines {<80 characters}
- Limit the use of abbreviations

LITMUS TEST

Imagine you go back to this do-file after three years.
Can you follow **exactly** what you've done before?


```
capture log close
log using <name>, replace text

// program:<name>.do
// task: <what the do-file does>
// project:<which project>
// author: <who wrote it> \<when>

// #0
// program setup
version 12
clear all
set linesize 80

// #1
// <description of 1st task>

// #2
// <description of 2nd task>

log close
exit
```

Automating Your Work

- **Macros**
- Saved results
- Loops
- Matrices
- Ado-files {preview}

Macros

Macros are the simplest tool for automating your work.

- A macro assigns a *string of text* or a *number* to an *abbreviation*
- Two types of macros: **LOCALS** & **GLOBALS**

GLOBAL	LOCAL
Persists until you delete or exit Stata	Can only be used within the (a)do-file in which they are defined
Do-files can use macros created by other do-files	When program ends, local disappears
NOT ROBUST; not self-containing	**ROBUST**

Syntax

- `local` *localname "string"*

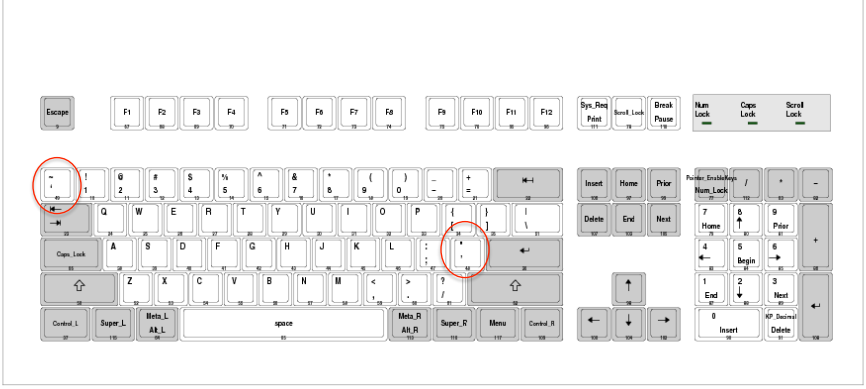
```
local rhs "var1 var2 var3"
display "The local rhs contains: `rhs'"
```

- `local` *localname = expression*

```
local ncases = 198
display "The local ncases equals: `ncases'"
```

Expression* is limited to 80 characters; *"string"* is limited to 67,784 characters. Usually better to use *"string"


```
display "The local rhs contains: `rhs'"
```



{EXAMPLE}

I want to estimate the model:

```
. regress y var1 var2 var3
```

I can create the local `rhs` with the names of the independent or right-hand-side variables:

```
. local rhs "var1 var2 var3"
```

Then, I can write the `regress` command as:

```
. regress y `rhs'
```

i.e., the command:

```
regress y `rhs'
```

works exactly the same as

```
regress y var1 var2 var3
```


Macros can be combined to specify a sequence of nested models. First, I create locals for four groups of independent variables:

```
. local set1_age   "age agesquared"
. local set2_educ  "wc hc"
. local set3_kids  "k5 k618"
. local set4_money "lwg inc"
```

Next, I specify four nested models. The first model includes only the first set of variables and is specified as:

```
. local model_1 "`set1_age'"
```

The macro model_2 combines the content of the local model_1 with the variables in local set2_educ:

```
. local model_2 "`model_1' `set2_educ'"
```

The next two models are specified the same way:

```
. local model_3 "`model_2' `set3_kids'"
. local model_4 "`model_3' `set4_money'"
```

Next, I check the variables in each model:

```
. display "model_1: `model_1'"
model_1: age agesquared

. display "model_2: `model_2'"
model_2: age agesquared wc hc

. display "model_3: `model_3'"
model_3: age agesquared wc hc k5 k618

. display "model_4: `model_4'"
model_4: age agesquared wc hc k5 k618 lwg inc
```

Using these locals, I estimate a series of logits:

```
. logit lfp `model_1'
. logit lfp `model_2'
. logit lfp `model_3'
. logit lfp `model_4'
```


{The whole thing}

```
. local set1_age   "age agesquared"
. local set2_educ  "wc hc"
. local set3_kids  "k5 k618"
. local set4_money "lwg inc"

. local model_1 "`set1_age'"
. local model_2 "`model_1' `set2_educ'"
. local model_3 "`model_2' `set3_kids'"
. local model_4 "`model_3' `set4_money'"

. display "model_1: `model_1'"
model_1: age agesquared

. display "model_2: `model_2'"
model_2: age agesquared wc hc

. display "model_3: `model_3'"
model_3: age agesquared wc hc k5 k618

. display "model_4: `model_4'"
model_4: age agesquared wc hc k5 k618 lwg inc

. logit lfp `model_1'
. logit lfp `model_2'
. logit lfp `model_3'
. logit lfp `model_4'
```

Automating Your Work

- Macros
- **Saved results**
- Loops
- Matrices
- Ado-files {preview}

Saved results

- Stata commands send results to your log file but **also** save those results to memory.

***Drukker's Dictum**:*

Never type anything that you can obtain from a saved result

- This information can be moved into macros and matrices, and used in many ways.

{Example I}

Let's say I wanted to compare descriptive statistics for males & females.
I use the sum command to compute the mean for females.

```
. sum job if fem

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
         job |       159    2.102453    1.006129          1        4.8

. return list

scalars:
                  r(N) =  159
              r(sum_w) =  159
             r(mean) =  2.102452833697481 // scalar for mean of job
              r(Var) =  1.012295852242676
               r(sd) =  1.006129142924841 // scalar for sd of job
              r(min) =  1
              r(max) =  4.800000190734863
              r(sum) =  334.2900005578995

. local jobmnF = r(mean)
. local jobsdF = r(sd)

. di "The mean for females is `jobmnF' (SD=`jobsdF')."
The mean for females is 2.102452833697481 (SD=1.006129142924841).
```


{Example I}

Now I do the same thing for males.

```
. sum job if !fem

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
         job |       249    2.317068    .9448164          1       4.64

. local jobmnM = r(mean)
. local jobsdM = r(sd)

. di "The mean for males is `jobmnM' (SD=`jobsdM')."
The mean for males is 2.317068263708827 (SD=.9448164073633196).
```

Now let's create one final local that contains the difference in the two means.

```
. local jobdfmnMF = `jobmnM'-`jobmnF'

. di "Difference in mean job prestige between males & females is `jobdfmnMF'."
Difference in mean job prestige between males & females is .2146154300113463.
```

{Example 2}
Use `prvalue, save & dif` to calculate discrete change for ΔSD centered on the mean for a continuous variable in our model (age).

The old way

```
. sum age

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
         age |       753    42.53785    8.072574         30         60

. di 42.53785 - (8.072574/2)
38.501563

. di 42.53785 + (8.072574/2)
46.574137

. qui prvalue, x(age=38.501563) rest(mean) save label(.5SD-)
. prvalue, x(age=46.574137) rest(mean) dif label(.5SD+)

:::
```


The new way

```
. local c "age"
. sum `c'

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
         age |       753    42.53785    8.072574         30         60

. return list

scalars:
                  r(N) =  753
              r(sum_w) =  753
             r(mean) =  42.53784860557769 // scalar for mean of age
              r(Var) =  65.16645121641095
               r(sd) =  8.072574014303674 // scalar for sd of age
              r(min) =  30
              r(max) =  60
              r(sum) =  32031

. local sddn = r(mean) - (r(sd)/2)
. local sdup = r(mean) + (r(sd)/2)

. qui prvalue, x(`c'=`sddn') rest(mean) save label(.5SD-)
. prvalue, x(`c'=`sdup') rest(mean) dif label(.5SD+)

:::
```

Question:

I discover that my age variable is actually coded as categories [thus not continuous] & decide to change my C variable to income.

Which parts of the above code do I need to change if:
[1] I 'hardcoded' my numbers; &
[2] I used the locals & scalars?

Automating Your Work

- Macros
- Saved results
- **Loops**
- Matrices
- Ado-files {preview}

{foreach} & {forvalues} loops

- **Loops** let you execute a group of commands *multiple times*
  - By combining macros with loops, you can speed up tasks, ranging from creating variables to estimating models.

Anytime you write a command more than once, consider using a loop!

Syntax: {foreach}

- `foreach` *localname* `in` | `of` *list-type list* {
  commands referring to `*local-name*'
  }

  - `foreach` name **`in`** n1 n2 n3 {
  - `foreach` var **`of varlist`** var1-var10 {
  - `foreach` var **`of varlist`** var* {

Syntax: {forvalues}

`forvalues` *localname* = *range* {

commands referring to `*localname*'

}

```
forvalues nage = 40(5)80 {
forvalues n = 1/100 {
```

Syntax	Meaning	Example	Generates
#1(#d)#2	From #1 to #2 in steps of #d.	1(2)10	1, 3, 5, 7, 9
#1/#2	From #1 to #2 in steps of 1.	1/10	1, 2, 3, ..., 10
#1 #t to #2	From #1 to #2 in steps of (#t-#1)	1 4 to 15	1, 4, 7, 10, 13

{Example 1: Using loops to create new variables}

I have a four-category ordinal variable y with values from 1 to 4. I want to create the four three binary variables that indicate if y<2, y<3 and y<4 (1 if yes, 0 if no).

Old way

Use three generate commands:

```
. generate y_lt2 = y<2 if y<.
. generate y_lt3 = y<3 if y<.
. generate y_lt4 = y<4 if y<.
```

New way

```
1> foreach cutpt in 2 3 4 {
2>     generate y_lt‘cutpt’ = y<‘cutpt’ if y<.
3> }
```

The first time through, line 2 is evaluated as

```
. generate y_lt2 = y<2 if y<.
```

The second time through, line 2 is evaluated as

```
. generate y_lt3 = y<3 if y<.
```

etc.

{Example 2: Build a loop}

Let’s return to our earlier `prvalue` example.

Except this time, let’s assume we want to examine changes in predicted probabilities for a standard deviation change in each of our 3 continuous variables.

Here’s what we did:

```
. local c “age”
. sum `c’
. local sdup = r(mean) + (r(sd)/2)
. local sddn = r(mean) - (r(sd)/2)

. qui prvalue, x(`c’=`sddn’) rest(mean) save label(SD-)
. prvalue, x(`c’=`sdup’) rest(mean) dif label(SD+)
```

We could repeat this process for each of our c variables.
Or…

We can build a loop!

First, we'll define a local containing all of our continuous variables

```
. local cont      age ed prst
```

Then we build our `foreach` loop

```
. foreach var in `cont' {
  2.          sum `var'
  3.          local sdUP = r(mean) + (.5*r(sd))
  4.          local sdDN = r(mean) - (.5*r(sd))
  5.          prvalue, x(`var'=`sdDN') rest(mean) save label(DN)
  6.          prvalue, x(`var'=`sdUP') rest(mean) dif label(UP)
  7.          }
```

What happens the first time we go through the loop?
The second?

{aside} The `q' local

In Stata, we can use the command `quietly` to do something without showing output. This can be useful when we need to use certain commands but don't need to see te output.

E.g., in this case we presumably have the descriptive statistics elsewhere & don't need to see the `sum` output again—but we have to perform the command to grab the numbers.

```
. local q "quietly"

. foreach var in `cont' {
  2.          `q' sum `var'
  3.          local sdUP = r(mean) + (.5*r(sd))
  4.          local sdDN = r(mean) - (.5*r(sd))
  5.          `q' prvalue, x(`var'=`sdDN') rest(mean) save label(DN)
  6.          prvalue, x(`var'=`sdUP') rest(mean) dif label(UP)
  7.          }
```

HINT: Define your `q` local at the top of your do-file. While you are writing your code, asterisk (*) it out. Once your programs run, run them once '*noisily*' to check for errors. **THEN**, finally, remove the asterisk & run again *quietly*.

{aside} Using `display` commands in loops

In addition to functioning as a calculator, `display` can also help you to annotate your work in Stata & are especially helpful with loops.

Let's revise our loop to include some of these commands…

```
. foreach var in `cont' {
  2.          `q' sum `var'
  3.          local sdUP = r(mean) + (.5*r(sd))
  4.          local sdDN = r(mean) - (.5*r(sd))
  5.          di "============================"
  6.          di "**`var': change from `sdDN' [S] to `sdUP' [C]"
  7.          `q' prvalue, x(`var'=`sdDN') rest(mean) save label(DN)
  8.          prvalue, x(`var'=`sdUP') rest(mean) dif label(UP)
  9.          di "============================"
 10.          }
```

To see what this does, let's look at the output:


```
. foreach var in `cont' {
  2.          `q' sum `var'
  3.          local sdUP = r(mean) + (.5*r(sd))
  4.          local sdDN = r(mean) - (.5*r(sd))
  5.          di "============================"
  6.          di "**`var': change from `sdDN' [S] to `sdUP' [C]"
  7.          `q' prvalue, x(`var'=`sdDN') rest(mean) save label(DN)
  8.          prvalue, x(`var'=`sdUP') rest(mean) dif label(UP)
  9.          di "============================"
 10.          }
============================
**age: change from 36.54593888588001 [S] to 53.32497258381034 [C]

ologit: Change in Predictions for warm

Confidence intervals by delta method

                    Current     Saved    Change   95% CI for Change
     Pr(y=1SD|x):    0.1305    0.0945    0.0360  [ 0.0276,   0.0444]
     Pr(y=2D|x):     0.3540    0.3007    0.0533  [ 0.0407,   0.0659]
     Pr(y=3A|x):     0.3773    0.4174   -0.0401  [-0.0502,  -0.0300]
     Pr(y=4SA|x):    0.1381    0.1873   -0.0492  [-0.0604,  -0.0380]

              yr89       male      white        age         ed       prst
Current=  .39860445  .46489315   .8765809  53.324973  12.218055  39.585259
  Saved=  .39860445  .46489315   .8765809  36.545939  12.218055  39.585259
   Diff=          0          0          0  16.779034          0          0
============================
============================
**ed: change from 10.63764157757564 [S] to 13.79846832211908 [C]

:::
```

Question:

How would you modify the loop to look at a change from min to max in each c variable?

{Example 3: Creating interactions using loops}

I need interactions between the binary variable `fem` & a set of independent variables

First, let's set up a local for the variables we want to interact:

```
. local int "mar kid5 phd"
```

Now let's reference this local in a loop to create interactions

```
. foreach var in `int' {
  2.          gen femX`var' = fem*`var'
  3.          label var femX`var' "Female*`var'"
  4.          }
```

To examine the new variables and their labels, I use `codebook, compact`:

```
Variable   Obs Unique      Mean  Min   Max  Label
-----------------------------------------------------------------------------
fem        915      2  .4601093    0     1  Gender: 1=female 0=male
femXmar    915      2  .2459016    0     1  Female*mar
femXkid5   915      4  .1147541    0     3  Female*kid5
femXphd    915     76  1.415104    0  4.62  Female*phd
-----------------------------------------------------------------------------
```

{Example 3 (extended): Nested loops}

What if I wanted to create two sets of interactions: one for men & one for women [e.g., for a fully-interactive model]?

First, let's set up a local for the group variable {gender, in this case} & one for the variables we want to interact.

```
. local mf        "male fem"
. local rhs       "mar kid5 phd"
```

Then, we can modify the loop from the prior slide by nesting it within another loop.

```
. foreach g in `mf' {                          //loop through men & then women
  2.   foreach var in `rhs' {                  //loop through vars
  3.         gen `g'X`var' = `g'*`var'
  4.         label var `g'X`var' "`g'*`var'"
  5.         }                                 //closes inner loop [vars]
  6.    }                                      //closes outer loop [gender]
```

What is the first variable created? The last?

{Aside: 'Tagging' new variables}

In our last command, we created six new variables. Anytime I create a new variable, I like to add a note to it that tells me the date of creation & name of the do-file that created it—I call it a ***tag***.

The easiest way to do this is to set up locals at the beginning of the do-file containing the tag information. Then, at the end of my do-file, I loop through & tag all new vars.

```
//  #0
//  program setup
. version 11
. clear all
. local dte       "2012mar30"
. local job       "iids12-automate"
. local tag       `job'.do sns `dte'

:::

//  #6
//  tag new variables
. foreach v of varlist maleXmar-femXphd {
  2.         note `v': `tag'
  3.         }
```

```
. note maleXmar-femXphd // displays notes for vars

maleXmar:
  1.  icpsrcda11-automate.do sns 2011aug09

maleXkid5:
  1.  icpsrcda11-automate.do sns 2011aug09

maleXphd:
  1.  icpsrcda11-automate.do sns 2011aug09

femXmar:
  1.  icpsrcda11-automate.do sns 2011aug09

femXkid5:
  1.  icpsrcda11-automate.do sns 2011aug09

femXphd:
  1.  icpsrcda11-automate.do sns 2011aug09
```


Automating Your Work

- Macros
- Saved results
- Loops
- **Matrices {putting it all together…}**
- Ado-files {preview}

{Example 1: Descriptives}

Matrices can be especially good for collecting descriptive statistics, e.g., by group. Much easier to assess differences in a single matrix than across several matrices.

Before we can start writing our matrix program though, we need to think up what we want out matrix to look like& specify its dimensions [rows & columns].

In this case, I want to use the science data to compare means for four different variables across the prestige of the PhD granting institution [(1) Adequate to (4)Distinguished].

What are the dimensions of matrix?

{Let's get started!}

```
. matrix desc = J(4,4,.) // create empty matrix with 4 rows & 4 columns
. matrix colnames desc = female enroll pubtot jobclass // label columns
. matrix rownames desc = 1_Adeq 2_Good 3_Strong 4_Dist
. local irow1 = 0 // initialize a counter that will indicate row for info
```


{Example 1: Descriptives}

***Set up our matrix**

```
. matrix desc = J(4,4,.) // create empty matrix with 4 rows & 4 columns
. matrix colnames desc = female enroll pubtot jobclass // label columns
. matrix rownames desc = 1_Adeq 2_Good 3_Strong 4_Dist
. local irow1 = 0 // initialize a counter that will indicate row for info
```

***Set up our loop to fill in matrix**

```
forvalues n = 1(1)4 { // loop through levels of phd prestige
        local icol1 = 0 // initializes counter for cols
        local ++irow1 // this adds 1 to the counter for rows
        di _newline "**Means: phd=`n' // for which value of phdclass?
        di "============================"
                foreach var in female enroll pubtot jobclass { // loops thru vars
                        local ++icol1 // adds 1 to counter for cols
                        `q' sum `var' if phdclass==`n'
                        matrix desc[`irow1', `icol1'] = r(mean)
                        } // close loop for vars
        mat list desc // This allows us to watch the empty matrix fill in
                } // close loop for phdclass level
```

***Print matrix**

```
mat list desc
```

{Example 1: Descriptives}

***Print matrix**

```
mat list desc
```

```
desc[4,4]
             female     enroll     pubtot   jobclass
  1_Adeq  .26666667  5.5128205  12.155556  1.7777778
  2_Good  .31067961    5.78125  9.1650485  2.0869565
3_Strong  .19402985  5.3174603  13.641791  2.7391304
  4_Dist  .53763441      5.525  13.430108  2.6415094
```


{Example 2: Gender differences over range of C}

Let's set up a model. I want to see if there is a significant gender difference in productivity of young scholars. To measure this, I create a new binary variable `star` that indicates whether or not a scholar has more than 3 articles in the last three years of their PhD.

```
. gen star = 0
. replace star = 1 if art>=3 & art !=.
(216 real changes made)
```

Then I run my model, using the `rhs` local defined earlier + a control for gender (`male`):

```
. logit star male `rhs', nolog

Logistic regression                               Number of obs   =        915
                                                  LR chi2(4)      =      20.35
                                                  Prob > chi2     =     0.0004
Log likelihood = -489.87655                       Pseudo R2       =     0.0203

------------------------------------------------------------------------------
        star |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
        male |   .5371478   .1683767     3.19   0.001     .2071355    .8671602
         mar |   .3064048   .1898271     1.61   0.107    -.0656495    .6784592
        kid5 |  -.2998166   .1220281    -2.46   0.014    -.5389873    -.060646
         phd |   .1737595   .0808904     2.15   0.032     .0152173    .3323017
       _cons |  -2.091095   .3112341    -6.72   0.000    -2.701102   -1.481087
------------------------------------------------------------------------------
```

{Example 2: Gender differences over range of C}

We want to look at gender differences in the probability of being a 'star' at different levels of PhD prestige. Commands in Stata store information in memory, which we can grab. This includes both *scalars* [as seen from `sum`, prior], but also *matrices*.

These commands:

```
. prvalue, x(male=1 phd=3) save label(M)
. prvalue, x(male=0 phd=3) dif label(F)
```

Generate the following stored matrices:

```
. matrix dir
      _PEtemp[3,7]
    pedifsep[2,1]
     pelower[7,2] // Matrix for lower CI bound
     peupper[7,2] // Matrix for upper CI bound
      pepred[7,2] // Matrix that includes discrete change
      peinfo[3,12]
      pebase[3,7]
       PE_in[1,7]
     PE_base[1,7]
     PRVinfo[1,12]
    PRVlower[2,2]
    PRVupper[2,2]
     PRVmisc[1,2]
     PRVprob[1,2]
     PRVbase[1,7]
      _PRVsav[1,6]
   pegrad_pr[2,8]
```


{Example 2: Gender differences over range of C}

To display the matrix `pepred`, type:

```
. matrix list pepred

pepred[7,2]
                 c1          c2
1values           0           1
  2prob   .81984824   .18015175 // Predicted prob for current [2,2]
  3misc  -1.5153197           .
 saved=   .72674537   .27325466 // Predicted prob for saved [4,2]
 saved=  -.97817189           .
 saved=   .09310287  -.09310292 // Discrete change [6,2]
 saved=  -.53714782           .
```

NB: These matrices are not always well-labeled. If you want to make use of them, you will need to spend a bit of time on the front-end ensuring you know where your #s of interest are.

{Example 2: Gender differences over range of C}

We can make use of these stored matrices to generate our own matrix of discrete change coefficients & confidence intervals.

```
matrix dc = J(11,4,.) // create empty matrix with 11 rows & 4 columns
matrix colnames dc = x dc dcLB dcUB // label columns
local irow1 = 0 // initialize a counter that will indicate row where to put info

forvalues n = 0(.5)5 {
         local ++irow1 //this adds 1 to the counter
         di _newline "**M-W: phd=`n'
         di "============================"
         `q' prvalue , x(male=0 phd=`n') save rest(mean) lab(F)
         prvalue , x(male=1 phd=`n') diff rest(mean) lab(M)
         matrix dc[`irow1',1] = `n'
         matrix dc[`irow1',2] = pepred[6,2]
         matrix dc[`irow1',3] = pelower[6,2]
         matrix dc[`irow1',4] = peupper[6,2]
         mat list dc
         di "============================"
         }
```


{Example 2: Gender differences over range of C}

Final Output

```
. mat list dc

dc[11,4]
              x         dc        dcLB        dcUB
 r1           0  .06709394  .01943566  .11475222
 r2          .5  .07124487  .02296249  .11952726
 r3           1  .07550052  .02633549  .12466554
 r4         1.5  .07984015  .02948088  .13019943
 r5           2  .08423951  .03234022   .1361388
 r6         2.5  .08867085  .03487714  .14246456
 r7           3  .09310292   .0370809  .14912494
 r8         3.5  .09750138  .03896644  .15603632
 r9           4  .10182886  .04057072    .163087
r10         4.5  .10604556  .04194696  .17014416
r11           5  .11010976  .04315759  .17706193
```

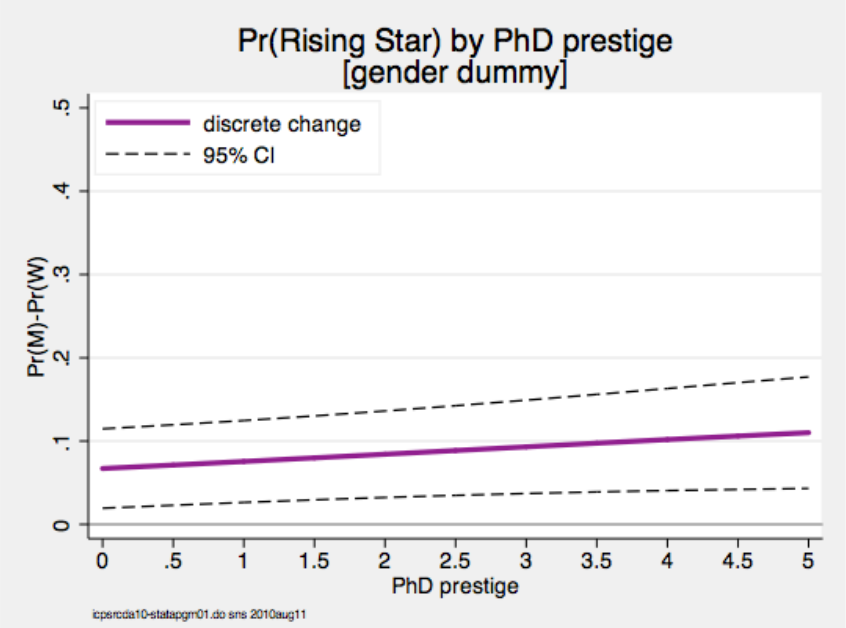
{Aside} `svmat` command

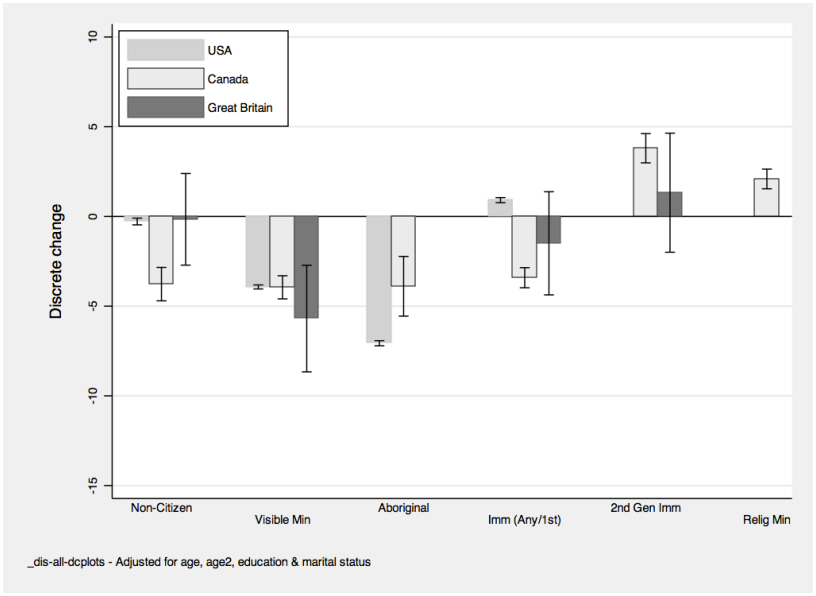
We can change matrix columns to variables with `svmat` command:

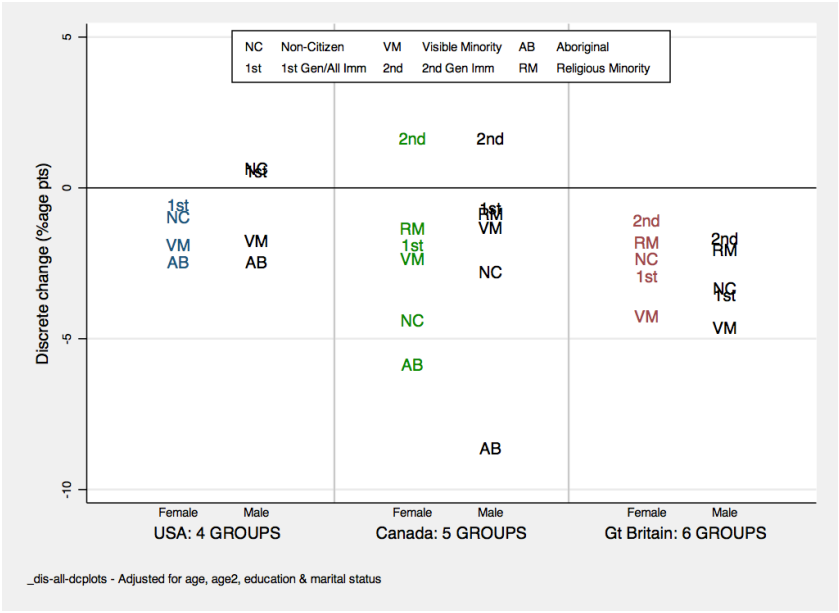
```
svmat dc , names(col)
label var x "value of x"
label var dc "discrete change"
label var dcLB "95% CI"
label var dcUB "95% CI"
```

& plot!

```
twoway ///
        (connected dcLB x, msymbol(i) clpat(dash) clwidth(medium) clcolor(gs0)) ///
        (connected dc x, msymbol(i) clpat(solid) clwidth(thick) clcolor(purple)) ///
        (connected dcUB x, msymbol(i) clpat(dash) clwidth(medium) clcolor(gs0)) ///
        , ytitle("Pr(M)-Pr(W)") ylabel(0(.1).5) ///
        xtitle("PhD prestige") xlabel(0(.5)5) ///
        legend(pos(11) order(2 1) ring(0) cols(1) region(ls(none))) ///
        title("Pr(Rising Star) by PhD prestige" "[gender dummy]") ///
        yline(0, lcolor(gray) lwidth(medium)) ///
        caption("`tag'", size(vsmall))
```


USA
Canada
Great Britain
Discrete change
10
5
0
-5
-10
-15
Non-Citizen
Visible Min
Aboriginal
Imm (Any/1st)
2nd Gen Imm
Relig Min
_dis-all-dcplots - Adjusted for age, age2, education & marital status


NC Non-Citizen
1st 1st Gen/All Imm
VM Visible Minority
2nd 2nd Gen Imm
AB Aboriginal
RM Religious Minority
Discrete change (%age pts)
5
0
-5
-10
Female
Male
USA: 4 GROUPS
Canada: 5 GROUPS
Gt Britain: 6 GROUPS
_dis-all-dcplots - Adjusted for age, age2, education & marital status

Automating Your Work

- Macros
- Saved results
- Loops
- Matrices
- **Ado-files {preview}**

Ado-files

- Ado-files are like do-files, except that they are automatically run
  - "Ado" = **A**utomatically-loaded **do**-file
- Stata 12 has more than 2,000 ado-files
- When you run a command, you can't tell whether it's part of the executable or an ado-file…

Stata users like you can write new commands & use them just like official Stata commands!!

Ado-files: An Example

Rescaling a variable

Equation for rescaling a variable:

newvalue =
(oldvalue - oldmin) x
((newmax-newmin)/(oldmax)-oldmin)) +
oldmin + (newmin-oldmin)

rescale.ado


```
*! version 3.1.0 \ shawna smith 2010-08-10
//  pgm:        rescale.ado
//  task:       pgm to rescale variables
//  author:     sns \ 2010-08-10
//  note:       used in byr files; rhss paper

capture program drop rescale
program define rescale
        version 11.1
        args oldvar newvar newmin newmax
        gen `newvar'=.
        label var `newvar' "`oldvar' rescaled `newmin' to `newmax'"

        qui sum `oldvar'
        local oldmin=r(min)
        local rangequota=(`newmax'-`newmin')/(r(max)-r(min))
        replace `newvar'=(`oldvar'-`oldmin')* ///
                `rangequota'+`oldmin'+(`newmin'-`oldmin')

end
exit

*!Update 2010-08-10: v3.1.0 changed version to 11.1
**Update 2008-05-19: v3.0.0 added variable labels
**Update 2008-05-10: v2.0.0 added gen of new variable in lieu of replacement
**Update 2008-05-06: v1.1.0 changed equation
**2008-05-05: v1.0.0 written
```

{In practice}

```
. rescale phd phd100 1 100
(915 missing values generated)
(915 real changes made)

. codebook phd phd100, compact

Variable   Obs Unique      Mean    Min    Max  Label
-----------------------------------------------------------------------
phd        915     83  3.103109   .755   4.62  PhD prestige
phd100     915     83  61.14562      1    100  phd rescaled 1 to 100
-----------------------------------------------------------------------

. corr phd phd100
(obs=915)

             |      phd   phd100
-------------+------------------
         phd |   1.0000
      phd100 |   1.0000   1.0000
```


{Homework: wd.ado}

A program that I use everyday is a program for setting & updating my working directory.

`wd.ado`

Allows me to use the command:
`wd` *project*

To set my current directory to the necessary working directory for each project, e.g.:

```
. wd iids12
/Users/shawnasmith/Documents/_teaching/iids/2012/

. wd dis-chap1
/Users/shawnasmith/Documents/_dissertation/_analysis/_chap1/_working

. wd icpsr12
/Users/shawnasmith/Documents/_teaching/icpsr/2012/

. wd working
/Users/shawnasmith/Desktop/_working/
```

```
*! version 1.5.0      \ shawna smith 2010-07-15
//  pgm:        wd.ado
//  task:       general program to change the working directory
//  project:    working directory
//  author:     sns \ 2010-07-15
//  note:       this is based on mcd.ado by david drukker/ from wf by jsl

capture program drop wd
program define wd
          version 11.2
          args dir
          if "`dir'"==”iids" {
                    cd /Users/shawnasmith/Documents/_teaching/iids/2012
          }
          else if "`dir'"=="desk" {
                    cd /Users/shawnasmith/Desktop/
          }
          else if "`dir'"=="icpsr12" {
                    cd /Users/shawnasmith/Documents/_teaching/icpsr/2012
                    }
          else if "`dir'"=="working" {
                    cd /Users/shawnasmith/Desktop/_working
          }
          else if "`dir'"=="" { // list current working directory
                    cd
          }
          else {
                    display as error "Working directory `dir' is unknown."
          }
end
exit
```


A few final notes on ado files

- Ado files can be written in your do-file editor [or any text editor] but should be saved with the extension *.ado*
- In order to run, they need to be saved in your **PERSONAL** Stata directory
  - To find out where this is, use the command `sysdir`

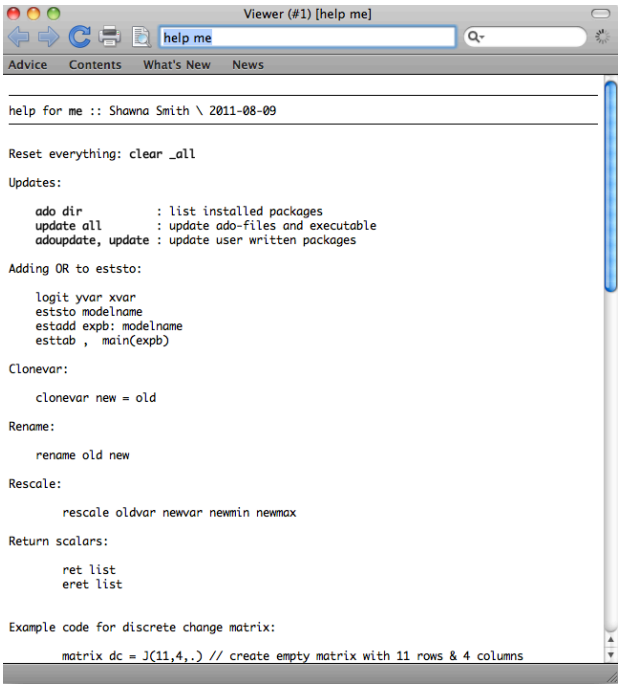
```
. sysdir
   STATA:  /Applications/Stata/
 UPDATES:  /Applications/Stata/ado/updates/
    BASE:  /Applications/Stata/ado/base/
    SITE:  /Applications/Stata/ado/site/
    PLUS:  ~/Library/Application Support/Stata/ado/stbplus/
PERSONAL:  ~/Library/Application Support/Stata/ado/personal/
OLDPLACE:  ~/ado/
```

{Aside} Me.hlp

- Help files provide information about commands we use in Stata—e.g., we type in `help prgen` to discover options, specific syntax, etc.
  - If you start writing programs & making them available for Stata users, you will want to create this files for each new program you author.
  - However, anyone can create a file with the extension `.hlp` & save it.

{Aside} Me.hlp

- I use a help file named `me.hlp` to give me quick access to information that I frequently use—e.g., summaries of options, fragments of code that I can copy and paste into a do-file.
- I put `me.hlp` in my PERSONAL directory. Then when I use the command **`help me`**, a Viewer opens and I can quickly access this information.



Advanced examples

{Example A1: Fully-interactive model for gender differences}

Let's say we decided to look into effects of gender on being a rising academic star in more detail, by allowing the effects of our IVs to vary by gender. To produce a graph similar to the one in the prior slide, what would we need to do differently?

CHANGE #1:

Estimate a fully-interactive model using our interaction variables [constant suppressed]

```
. local rhsM      "male maleXmar maleXkid5 maleXphd"
. local rhsF      "fem femXmar femXkid5 femXphd"

. logit star `rhsM' `rhsF', nocon nolog
```


CHANGE #2:

Again we want to do a loop through `prvalue, save` & `dif`. But our variables are more difficult to deal with this time. So, let's create some more locals!

Men at 0:

```
. local rhsM0 ""
. foreach v in `rhsM' {
  2.     local rhsM0 "`rhsM0' `v'=0"
  3. }
```

Women at 0:

```
. local rhsF0 ""
. foreach v in `rhsF' {
  2.     local rhsF0 "`rhsF0' `v'=0"
  3. }
```

Men at mean:

```
. local rhsMmn ""
. foreach v in `rhsM' {
  2.     sum `v' if male & e(sample)
  3.     local mn = r(mean)
  4.     local rhsMmn "`rhsMmn' `v'=`mn'"
  5. }
```

Women at mean:

```
. local rhsFmn ""
. foreach v in `rhsF' {
  2.     sum `v' if fem & e(sample)
  3.     local mn = r(mean)
  4.     local rhsFmn "`rhsFmn' `v'=`mn'"
  5. }
```

Question:

Any guesses as to what our matrix loop will look like [& specifically the `prvalue` commands]?


```
matrix rhpred = J(11,8,.)
matrix colnames rhpred = p1phd p1fp p1mp p1dif p1diflb p1difub p1difsig p1difz

:::

forvalues phd = 0(.5)5 {
        local ++irow
        mat rhpred[`irow',`colx'] = `phd'
        prvalue, x(`rhsFmn' `rhsM0' femXphd=`phd') save label(F) // fem means
        mat rhpred[`irow',`colmp'] = r(p1)
        prvalue, x(`rhsMmn' `rhsF0' maleXphd=`phd') dif label(M) // male means
        mat rhpred[`irow',`colfp'] = r(p1)
        mat temp = r(pred)
        local p1dif = temp[2,1]
        mat rhpred[`irow',`coldif'] = `p1dif'
        local p1diflb = temp[2,2]
        mat rhpred[`irow',`collb'] = `p1diflb'
        local p1difub = temp[2,3]
        mat rhpred[`irow',`colub'] = `p1difub'
        local p1difz = temp[2,6]
        mat rhpred[`irow',`colz'] = `p1difz'
        * is ci crossing 0?
        local issig = 0
                if `p1diflb'<0 & `p1difub'<0 { // both neg, sig
                local issig = 1
                }
                if `p1diflb'>0 & `p1difub'>0 { // both pos, sig
                local issig = 1
                }
        local p1difsig = `issig'
        mat rhpred[`irow',`col'7] = `p1difsig'
        }
```

```
Matrix rhpred = J(11,8,.)
matrix colnames rhpred = p1phd p1fp p1mp p1dif p1diflb p1difub p1difsig p1difz

:::

forvalues phd = 0(.5)5 {
        local ++irow
        mat rhpred[`irow',`colx'] = `phd'
        prvalue, x(`rhsFmn' `rhsM0' femXphd=`phd') save label(F) // fem means
        mat rhpred[`irow',`colmp'] = r(p1)
        prvalue, x(`rhsMmn' `rhsF0' maleXphd=`phd') dif label(M) // male means
        mat rhpred[`irow',`colfp'] = r(p1)
        mat temp = r(pred)
        local p1dif = temp[2,1]
        mat rhpred[`irow',`coldif'] = `p1dif'
        local p1diflb = temp[2,2]
        mat rhpred[`irow',`collb'] = `p1diflb'
        local p1difub = temp[2,3]
        mat rhpred[`irow',`colub'] = `p1difub'
        local p1difz = temp[2,6]
        mat rhpred[`irow',`colz'] = `p1difz'
        * is ci crossing 0?
        local issig = 0
                if `p1diflb'<0 & `p1difub'<0 { // both neg, sig
                local issig = 1
                }
                if `p1diflb'>0 & `p1difub'>0 { // both pos, sig
                local issig = 1
                }
        local p1difsig = `issig'
        mat rhpred[`irow',`col'7] = `p1difsig'
        }
```

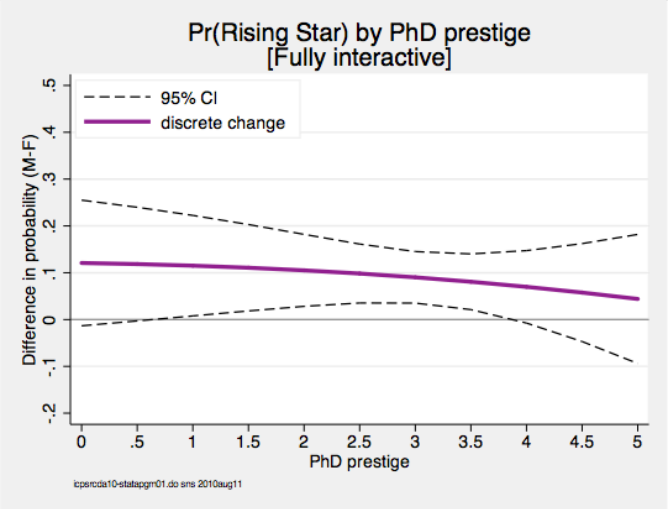

Final Output

```
. mat list rhpred

rhpred[11,8]
          p1phd        p1fp        p1mp       p1dif     p1diflb     p1difub
 r1           0   .09135602   .21217319   .12081718  -.01336658   .25500094
 r2          .5   .10279272   .22123058   .11843786  -.00296005   .23983576
 r3           1   .11547922   .23056147   .11508225   .00772092   .22244358
 r4         1.5   .12950547   .24016455   .11065908   .01837661   .20294154
 r5           2   .14495616   .25003758   .10508142   .02823389   .18192895
 r6         2.5   .16190735   .26017755   .09827021   .03532607   .16121435
 r7           3   .18042251   .27058038   .09015787   .03511293   .14520281
 r8         3.5   .20054837   .28124103   .08069266   .02122691   .14015842
 r9           4   .22231026   .29215345   .06984319  -.00751892   .14720529
r10         4.5   .24570782    .3033106   .05760278  -.04699499   .16220055
r11           5   .27071071   .31470442   .04399371   -.0938314   .18181883

       p1difsig      p1difz
 r1           0   1.7647241
 r2           0   1.9121741
 r3           1   2.1009154
 r4           1   2.3502602
 r5           1   2.6800576
 r6           1   3.0599522
 r7           1   3.2102165
 r8           1   2.6595932
 r9           0   1.7694727
r10           0   1.0793669
r11           0   .62561958
```

We can plot this matrix in several ways. For example, as before, we may want to plot the difference between genders & the CI for the difference.



Alternatively, we may want to plot the predictions for each gender & use other methods for indicating whether the difference is significant (e.g., line style).