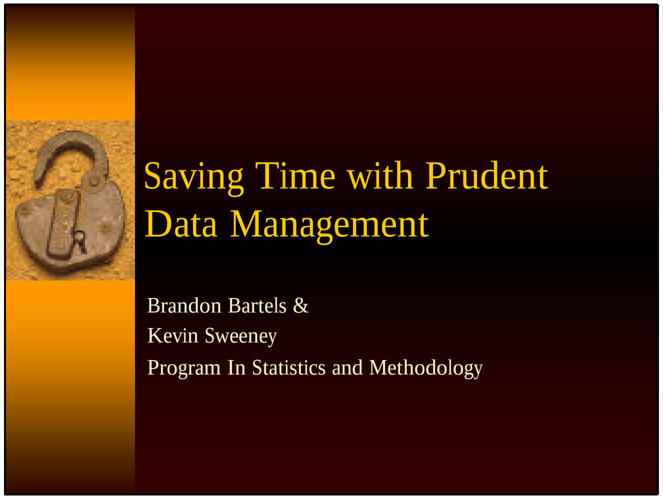
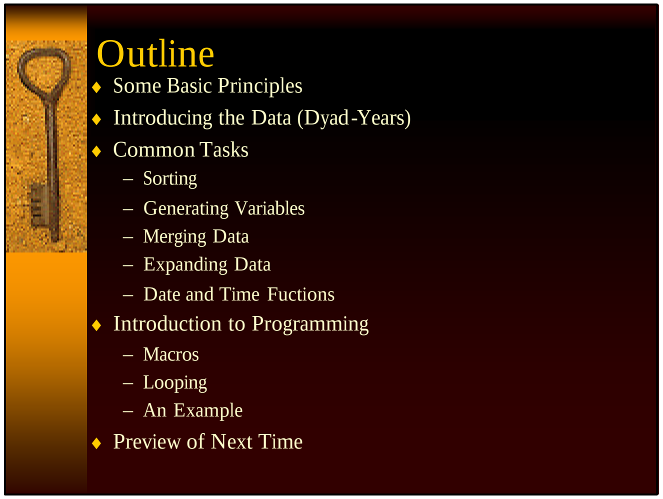


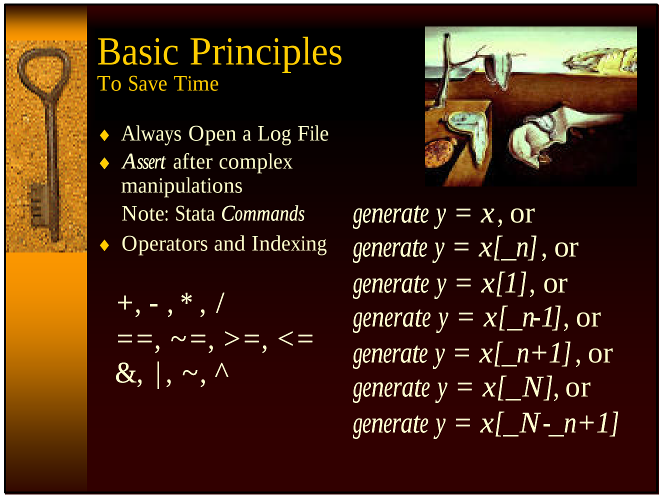
Saving Time with Prudent Data Management

Brandon Bartels &
Kevin Sweeney
Program In Statistics and Methodology



Outline

- Some Basic Principles
- Introducing the Data (Dyad-Years)
- Common Tasks
  - Sorting
  - Generating Variables
  - Merging Data
  - Expanding Data
  - Date and Time Fuctions
- Introduction to Programming
  - Macros
  - Looping
  - An Example
- Preview of Next Time



Basic Principles

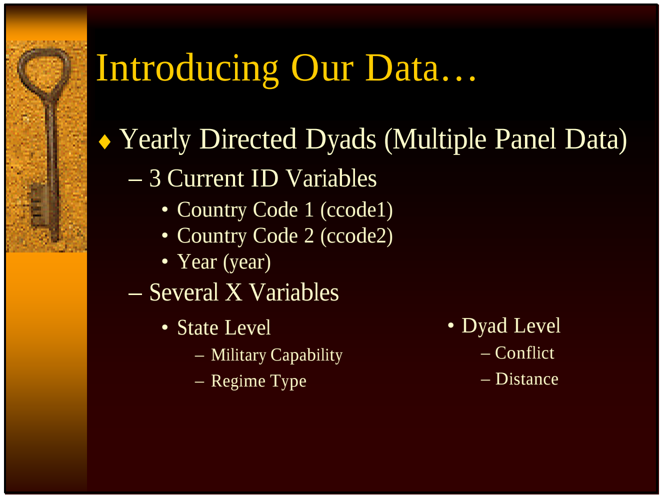
To Save Time

- Always Open a Log File
- *Assert* after complex manipulations

  Note: Stata *Commands*
- Operators and Indexing

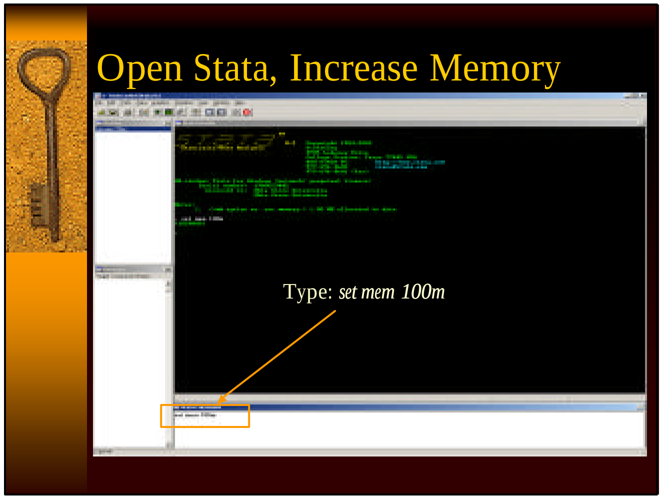
+, - , * , /
==, ~=, >=, <=
&, |, ~, ^

generate y = *x*, or
generate y = *x[_n]*, or
generate y = *x[1]*, or
generate y = *x[_n-1]*, or
generate y = *x[_n+1]*, or
generate y = *x[_N]*, or
generate y = *x[_N-_n+1]*



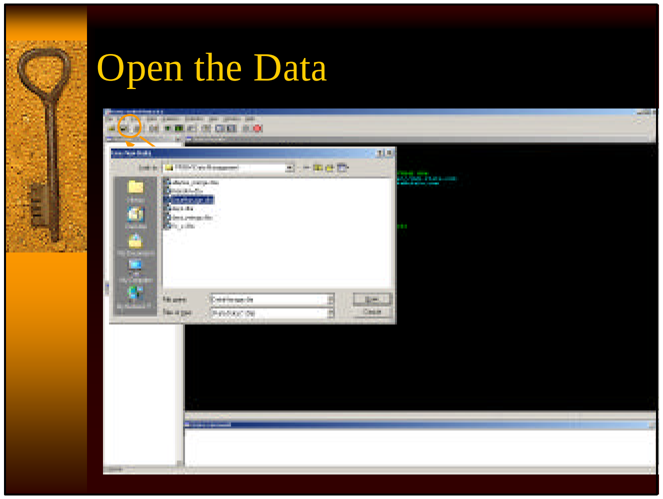
Introducing Our Data…

- Yearly Directed Dyads (Multiple Panel Data)
  - 3 Current ID Variables
    - Country Code 1 (ccode1)
    - Country Code 2 (ccode2)
    - Year (year)
  - Several X Variables
    - State Level
      - Military Capability
      - Regime Type
    - Dyad Level
      - Conflict
      - Distance

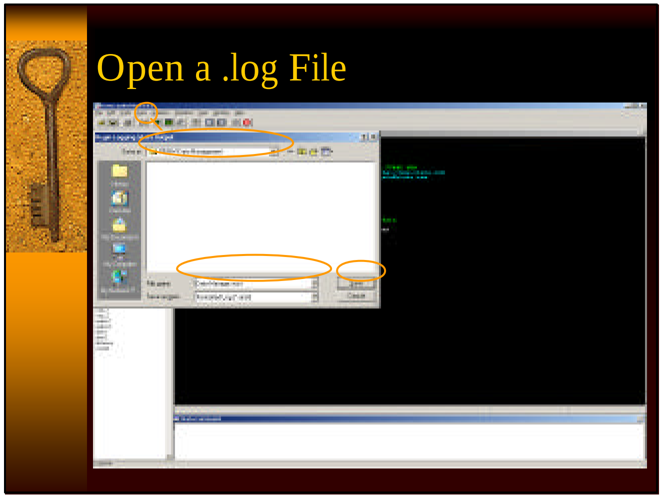


Open Stata, Increase Memory

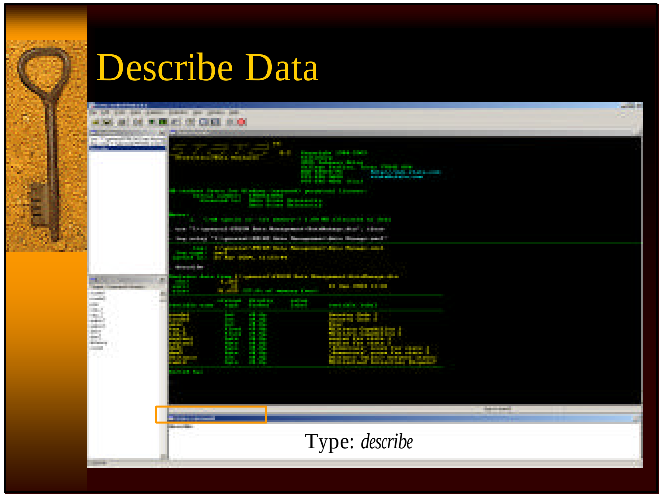
Type: *set mem 100m*



Open the Data



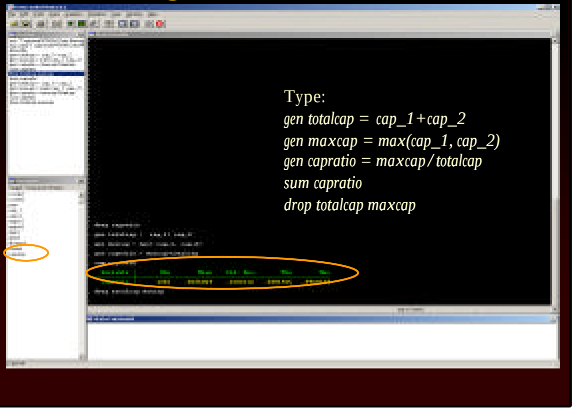
Open a .log File



Describe Data

Type: *describe*

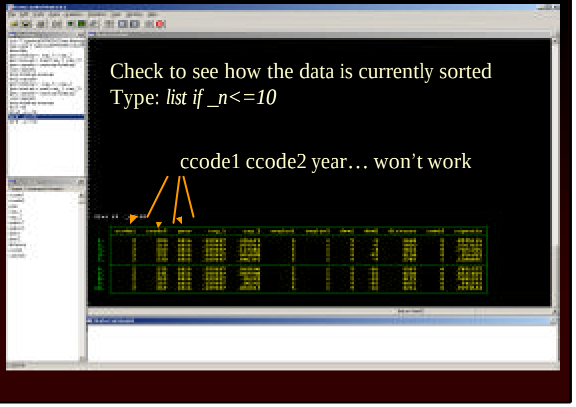
Generating New Variables



Type:

gen totalcap = cap_1+cap_2
gen maxcap = max(cap_1, cap_2)
gen capratio = maxcap/totalcap
sum capratio
drop totalcap maxcap

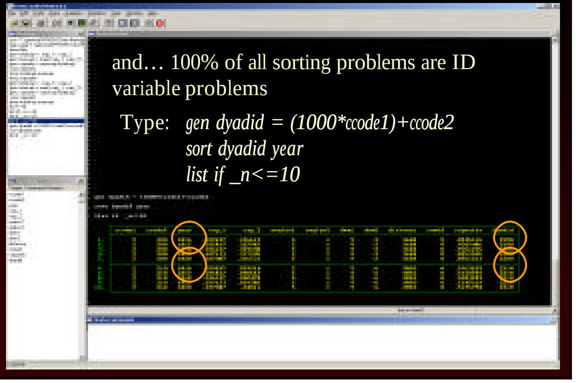
Sorting Data



Check to see how the data is currently sorted
Type: *list if _n<=10*

ccode1 ccode2 year… won't work

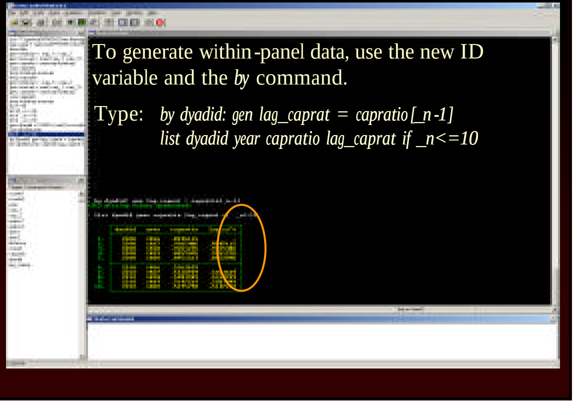
95% of all Data Management Problems are Sorting Problems



and… 100% of all sorting problems are ID variable problems

Type: *gen dyadid = (1000*ccode1)+ccode2*
sort dyadid year
list if _n<=10

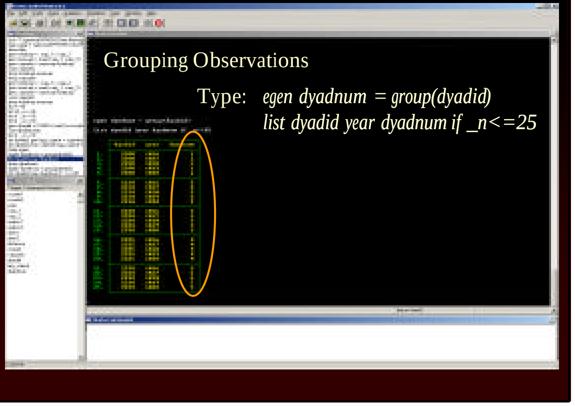
Use *by* for Panel Data



To generate within-panel data, use the new ID variable and the *by* command.

Type: *by dyadid: gen lag_caprat = capratio[_n-1]*
list dyadid year capratio lag_caprat if _n<=10

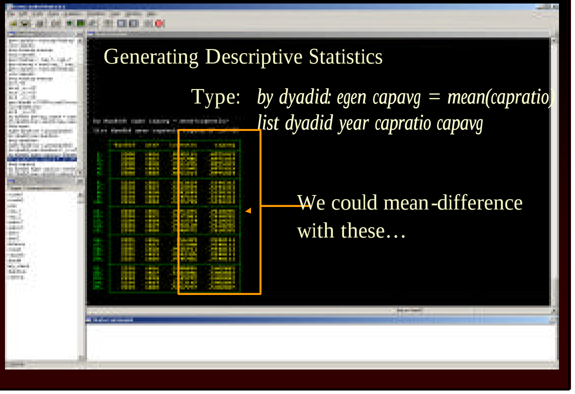
egen: extensions to *generate*



Grouping Observations

Type: *egen dyadnum = group(dyadid)*
list dyadid year dyadnum if _n<=25

egen: extensions to *generate*



Generating Descriptive Statistics

Type: *by dyadid: egen capavg = mean(capratio)*
list dyadid year capratio capavg

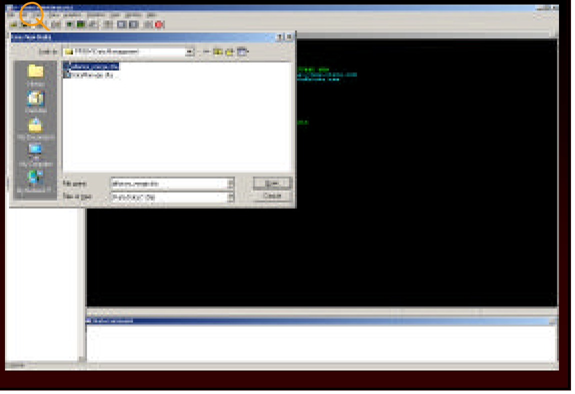
We could mean-difference with these…

Merging Data

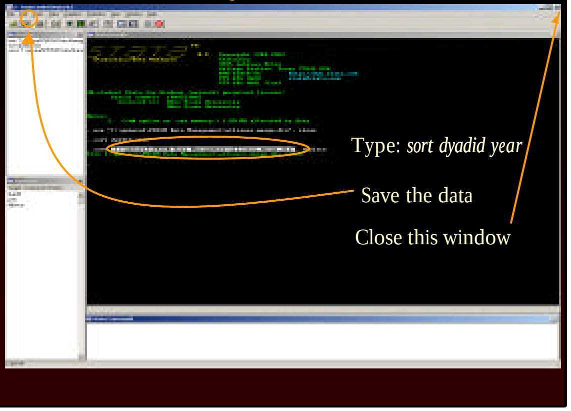
- Both data sets must contain the same ID variables.
- Both data sets must be sorted, according to those ID variables, in the same order.
- _merge, a new variable generated during the merge contains important information about the merge.

Merging Data: Example 1

open a new Stata Session



Sort & Save Dyadic Alliance Data



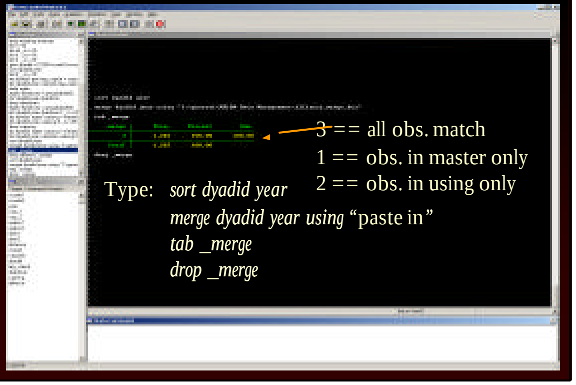
Type: *sort dyadid year*

Save the data

Close this window

Merging in the Alliance Data

go back to your original window



3 == all obs. match
1 == obs. in master only
2 == obs. in using only

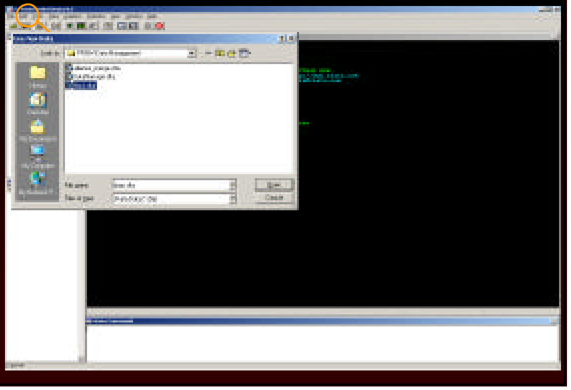
Type: *sort dyadid year*
merge dyadid year using "paste in"
tab _merge
drop _merge

Reshaping, Expanding, and Date Functions

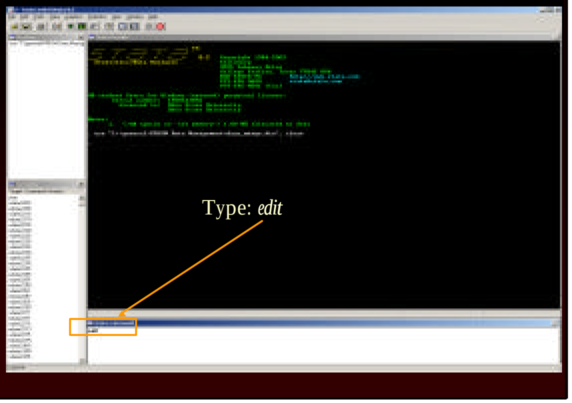
- In current Data
  - Type: *drop if dyadid>=3000*
- Reshaping moves data between "wide" and "long" forms, and vice-versa. (e.g. panels across vs. panels down)
- Expanding duplicates current observations
- Date Functions are a powerful tool to deal with the time aggregation problem…
  … if you have the data to do it.

Reshaping Data

Open a new Stata window

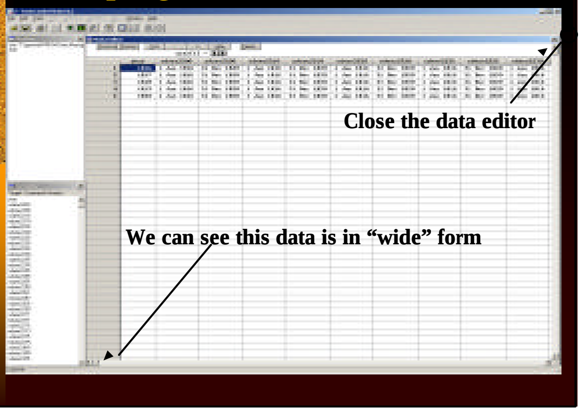


Reshaping Data



Type: *edit*

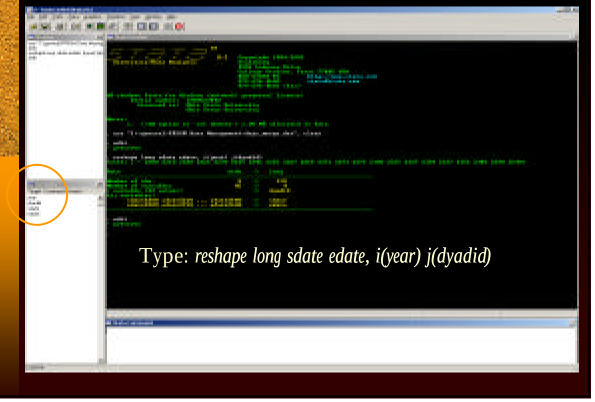
Reshaping Data



Close the data editor

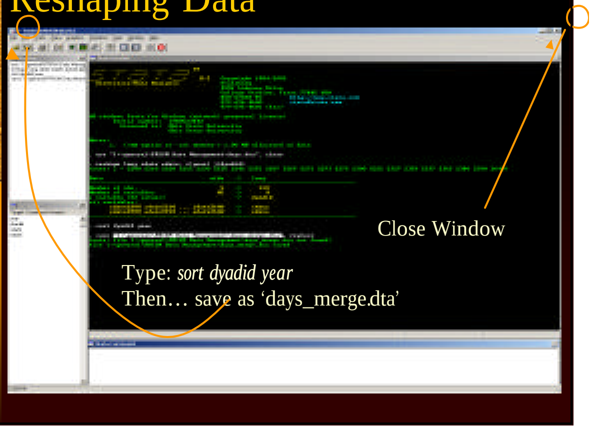
We can see this data is in “wide” form

Reshaping Data



Type: *reshape long sdate edate, i(year) j(dyadid)*

Reshaping Data

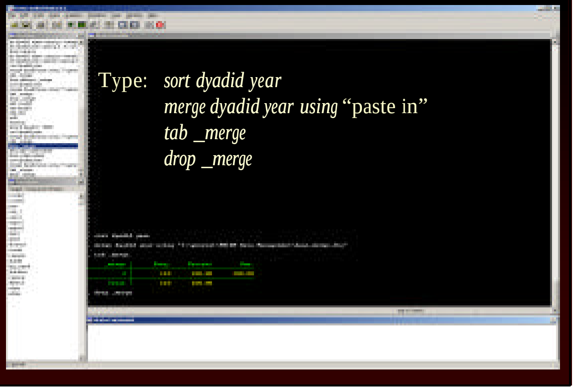


Close Window

Type: *sort dyadid year*
Then… save as ‘days_merge.dta’

Reshaping Data

go back to original session



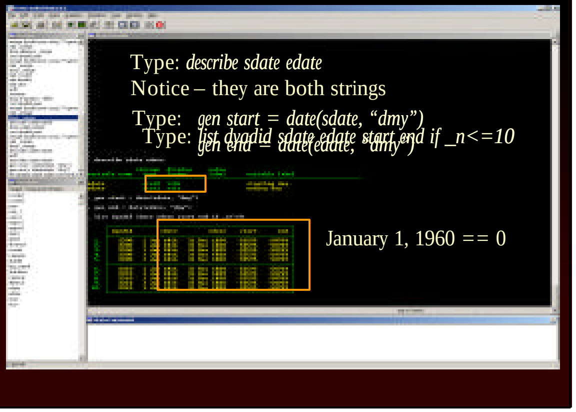
Type: *sort dyadid year*

merge dyadid year using "paste in"

tab _merge

drop _merge

Date Functions



Type: *describe sdate edate*

Notice – they are both strings

Type: *gen start = date(sdate, "dmy")*

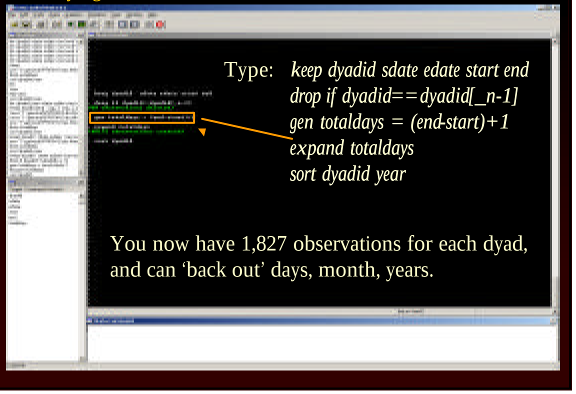
gen end = date(edate, "dmy")

Type: *list dyadid sdate edate start end if _n<=10*

January 1, 1960 == 0

Date Functions, Expanding Data

Destroying to Create



Type: *keep dyadid sdate edate start end*

drop if dyadid==dyadid[_n-1]

gen totaldays = (end-start)+1

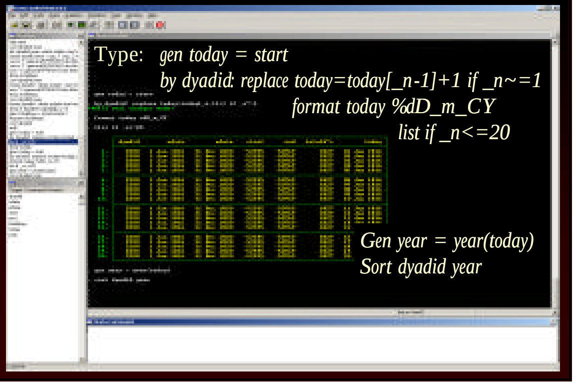
expand totaldays

sort dyadid year

You now have 1,827 observations for each dyad, and can 'back out' days, month, years.

Date Functions, Expanding Data

Destroying to Create



Type: *gen today = start*

by dyadid: replace today=today[_n-1]+1 if _n~=1

format today %dD_m_CY

list if _n<=20

Gen year = year(today)

Sort dyadid year

Recovering Your Xs

I'm Done
Type: *Clear*

Introduction to Programming in Stata

OUTLINE:

- The logic of programming in Stata
- Using saved calculated results from descriptive statistics commands.
- Macros
- Branching and Looping
- Write your own program!
- Using calculated results from estimation commands – building blocks for next PRISM session (Friday, May 7)

The Logic of Programming in Stata: The Basics

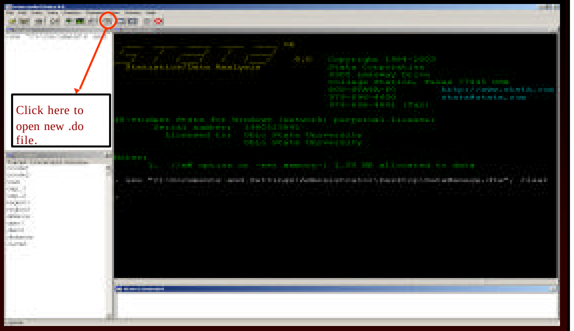
- Programming can make data management more efficient and accurate.
- Involves moving beyond canned commands in Stata and creating a more generalized set of commands designed for data management.
- Goal: Enter a program in Stata and get Stata to execute it.
- Mechanics: Using the *display* command….

```
display "Hello, world"
display 450/50
display "450/50"
display (46-467)*(789-99)/32
```

The Logic of Programming in Stata: Using .do Files

- A ".do file" is simply a plain text file containing a set of Stata commands; run commands collectively as opposed to line-by-line in the command window.
- Let's execute the commands we just ran line-by-line all at once.

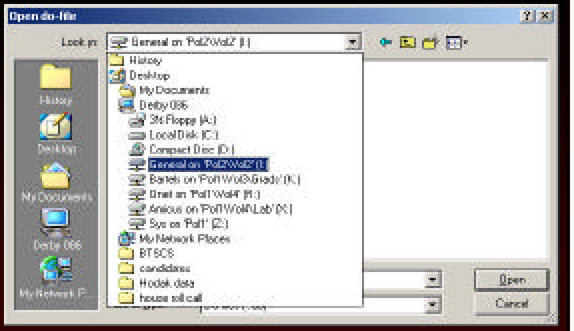
The Logic of Programming in Stata: Opening .do Files



Click here to open new .do file.

```
                                    8.0   Copyright 1984-2003
    Statistics/Data Analysis              Stata Corporation
                                          4905 Lakeway Drive
                                          College Station, Texas 77845 USA
                                          800-STATA-PC        http://www.stata.com
                                          979-696-4600        stata@stata.com
                                          979-696-4601 (fax)

Intercooled Stata for Windows (network) perpetual license:
       Serial number:  [illegible]
         Licensed to:  Ohio State University
                       Ohio State University

Notes:
      1.  (/m# option or -set memory-) 1.00 MB allocated to data
```

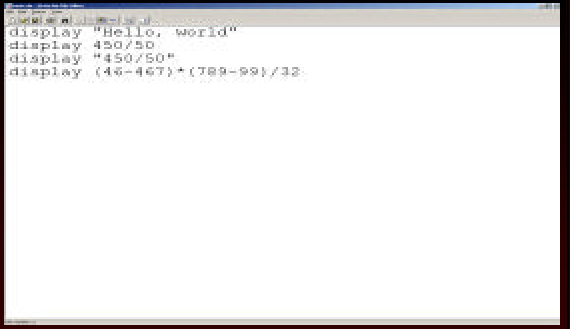
The Logic of Programming in Stata: Opening .do Files



The Logic of Programming in Stata: Opening .do Files

- Double-click on “general”
    - Double-click on “PRISM Data Management”
        - Double-click on “basic.do”

The Logic of Programming in Stata: Opening .do Files

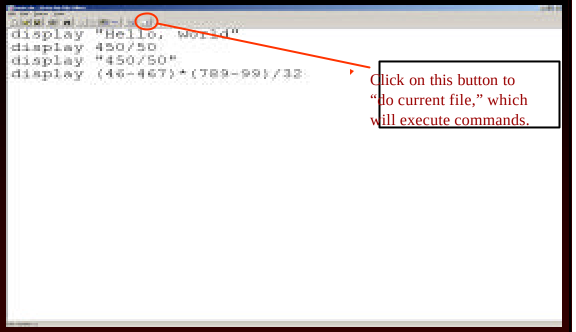


```
display "Hello, world"
display 450/50
display "450/50"
display (46-467)*(789-99)/32
```

The Logic of Programming in Stata: Running .do Files

- 3 ways to execute a .do file:
  1. “Do current file”
  2. File; Do
  3. Change home directory; “do basic”

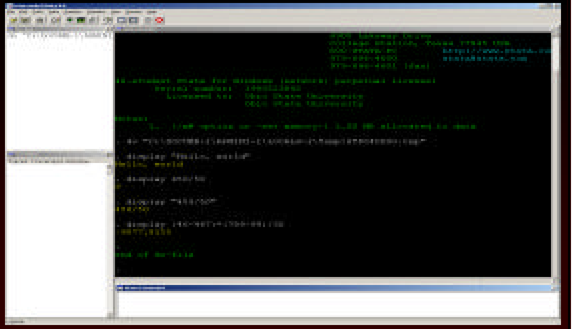
The Logic of Programming in Stata: Running .do Files



```
display "Hello, world"
display 450/50
display "450/50"
display (46-467)*(789-99)/32
```

Click on this button to “do current file,” which will execute commands.

The Logic of Programming in Stata: Running .do Files



```
[illegible]

. display "Hello, world"
Hello, world

. display 450/50
9

. display "450/50"
450/50

. display (46-467)*(789-99)/32
-9077.8125

.
end of do-file
```

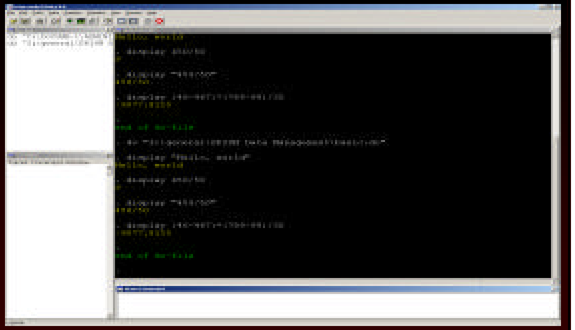
The Logic of Programming in Stata: Running .do Files

- “File; Do”
- Go to the I: drive again
- Double-click on “general”

  Double-click on “PRISM Data Management”

  Double-click on “basic.do”

The Logic of Programming in Stata: Running .do Files

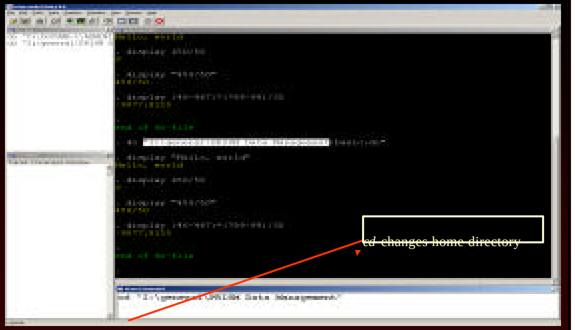


```
Hello, world

. display [illegible]
[illegible]

. display "[illegible]"
[illegible]

. display [illegible]
[illegible]

.
end of do-file

. do "[illegible]\basic.do"

. display "Hello, world"
Hello, world

. display [illegible]
[illegible]

. display "[illegible]"
[illegible]

. display [illegible]
[illegible]

.
end of do-file
```

The Logic of Programming in Stata: Running .do Files

- Change home directory; handy if you have a lot of .do files and you want to access them quickly.

The Logic of Programming in Stata: Running .do Files

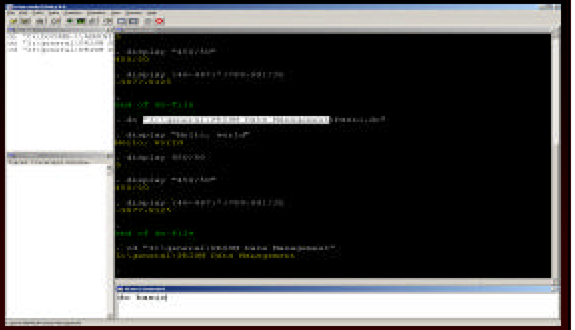


```
. do "[illegible]\basic.do"

. display "Hello, world"
Hello, world

. display [illegible]
[illegible]

. display "[illegible]"
[illegible]

. display [illegible]
[illegible]

.
end of do-file
```

```
cd "I:\general\[illegible] Data Management\"
```

cd changes home directory

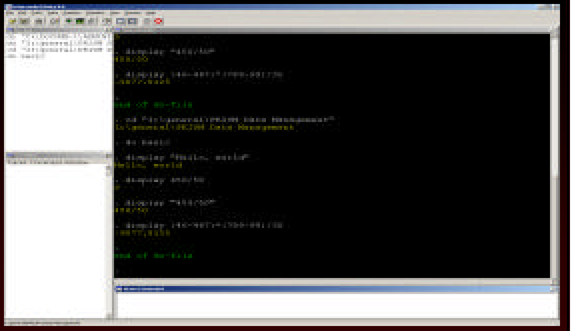
The Logic of Programming in Stata: Running .do Files



```
. display "[illegible]"
[illegible]

. display [illegible]
[illegible]

.
end of do-file

. do "[illegible]\basic.do"

. display "Hello, world"
Hello, world

. display [illegible]
[illegible]

. display "[illegible]"
[illegible]

. display [illegible]
[illegible]

.
end of do-file

. cd "I:\general\[illegible] Data Management"
I:\general\[illegible] Data Management
```

```
do basic
```

The Logic of Programming in Stata: Running .do Files



```
. display "[illegible]"
[illegible]

. display [illegible]
[illegible]

.
end of do-file

. cd "I:\general\PRISM Data Management"
I:\general\PRISM Data Management

. do hello

. display "hello, world"
hello, world

. display [illegible]
[illegible]

. display "[illegible]"
[illegible]

. display [illegible]
[illegible]

.
end of do-file
```

The Logic of Programming in Stata: Using .do Files to Run Commands

- Use .do files to archive and run multiple descriptive and estimation commands.
- First, open data. "File, Open"
- Go to the I: drive again
- Double-click on "general"

  Double-click on "PRISM Data Management"

    Double-click on "Brandon.dta"

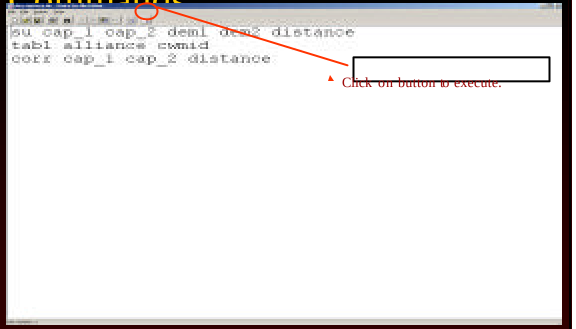
The Logic of Programming in Stata: Using .do Files to Run Commands

- Open "descriptives.do" from .do file editor.
- Go to the I: drive again
- Double-click on "general"

  Double-click on "PRISM Data Management"

    Double-click on "descriptives.do"

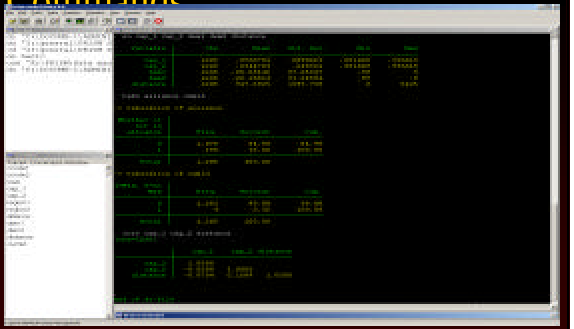
The Logic of Programming in Stata: Using .do Files to Run Commands



```
su cap_1 cap_2 dem1 dem2 distance
tab1 alliance cwmid
corr cap_1 cap_2 distance
```

Click on button to execute.

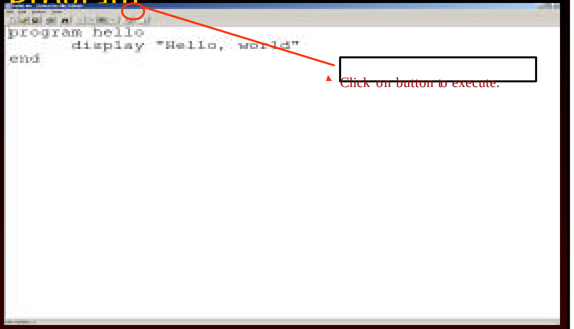
The Logic of Programming in Stata: Using .do Files to Run Commands



The Logic of Programming in Stata: Writing and Executing a Program

- Write a program in a .do file to execute commands; comes in handy for more complex data management tasks, especially ones for which there are many variables that need transforming and/or generating.
- Basics of programming: Let’s program “Hello, world”
- Open “hello.do” from the .do file editor.
- Go to the I: drive again
- Double-click on “general”
  - Double-click on “PRISM Data Management”
    - Double-click on “hello.do”

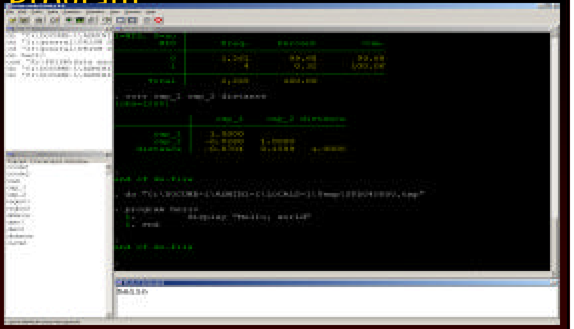
The Logic of Programming in Stata: Writing and Executing a Program



```
program hello
        display "Hello, world"
end
```

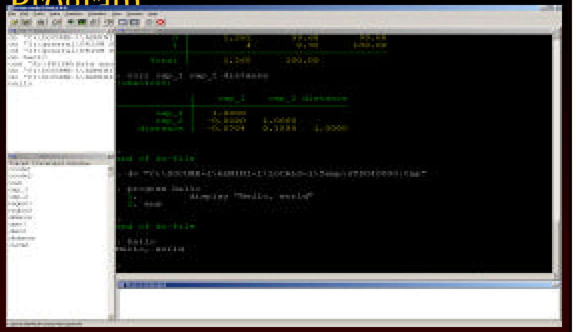
Click on button to execute.

The Logic of Programming in Stata: Writing and Executing a Program



```
. corr emp_1 emp_2 distance

end of do-file

. do "[illegible]"

. program hello
  1.         display "Hello, world"
  2. end

end of do-file
```

The Logic of Programming in Stata: Writing and Executing a Program



```
[illegible]

. program hello
  1.         display "Hello, world"
  2. end

end of do-file

. hello
Hello, world
```

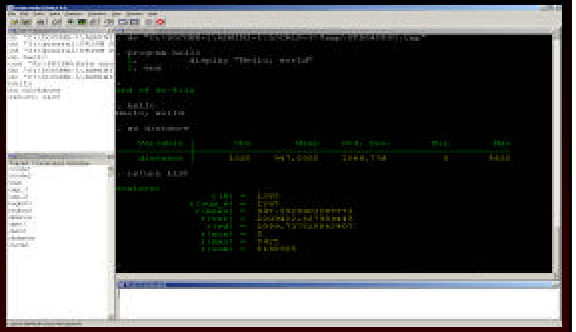
Using Saved Calculated Results

- For both descriptive and estimation commands, Stata saves calculations such as the mean, sd, min, max, coefficients, se's, etc.
- Use *return list* after a descriptive command (such as *summary* or *tab*) to list all saved results.

  su distance

  return list

Using Saved Calculated Results

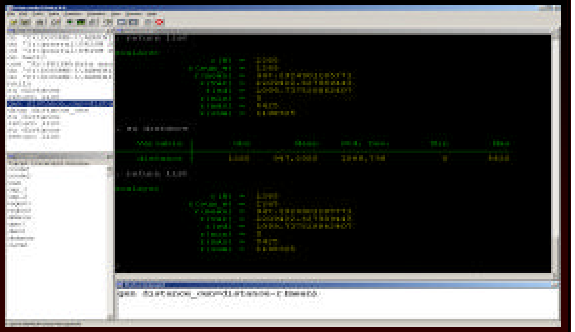


```
. program hello
  1.         display "Hello, world"
  2. end

end of do-file

. hello
Hello, world

. su distance

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
    distance | [illegible]

. return list

scalars:
[illegible]
```

Using Saved Calculated Results

- Use these saved calculated results for quick variable generation.
- For instance, generating mean-centered variables…

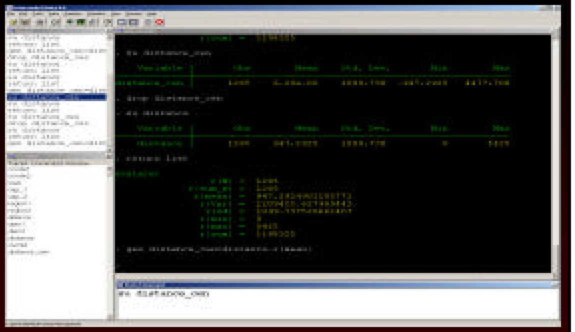
Using Saved Calculated Results



```
. return list

scalars:
                  r(N) =  [illegible]
              r(sum_w) =  [illegible]
               r(mean) =  [illegible]
                r(Var) =  [illegible]
                 r(sd) =  [illegible]
                r(min) =  0
                r(max) =  [illegible]
                r(sum) =  [illegible]

. su distance

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
    distance |  [illegible]

. return list

scalars:
[illegible]
```

```
gen distance_cen=distance-r(mean)
```

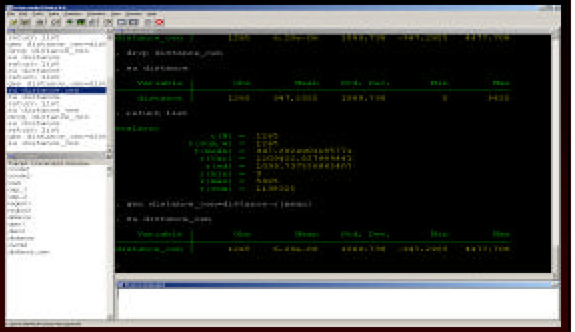
Using Saved Calculated Results



```
. su distance_cen

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
distance_cen |  [illegible]

. drop distance_cen

. su distance

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
    distance |  [illegible]

. return list

scalars:
[illegible]

. gen distance_cen=distance-r(mean)
```

```
su distance_cen
```

Using Saved Calculated Results

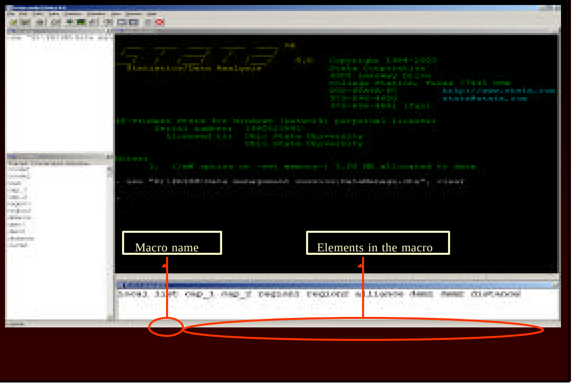


```
distance_cen |  [illegible]

. drop distance_cen

. su distance

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
    distance |  [illegible]

. return list

scalars:
[illegible]

. gen distance_cen=distance-r(mean)

. su distance_cen

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
distance_cen |  [illegible]
```

Macros

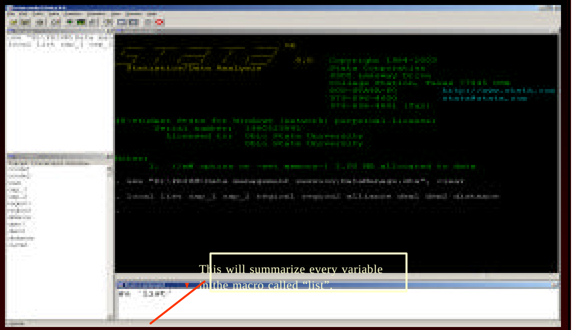
- Macros in Stata are VERY powerful and crucial for advanced programming.
- They allow one to condense a group of variables or a complicated numeric or string expression into a shorthand macro name.
- Basic syntax:
  - *local macroname expression*
- To recall macro, use: `*macroname*'
  - Note: left quote is above the tab key, right quote is the normal single quote.
- Examples….

Macros



```
local list emp_1 emp_2 region1 region2 alliance dem1 dem2 distance
```

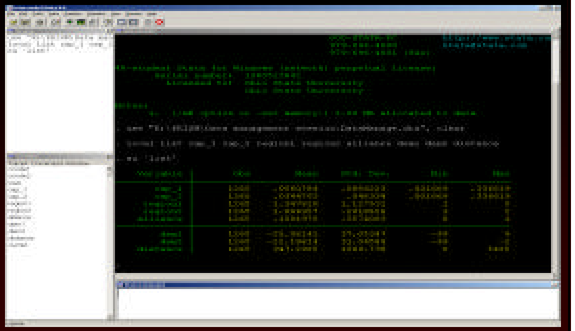
Macros



```
. local list emp_1 emp_2 region1 region2 alliance dem1 dem2 distance
su `list'
```

This will summarize every variable in the macro called “list”.

Macros



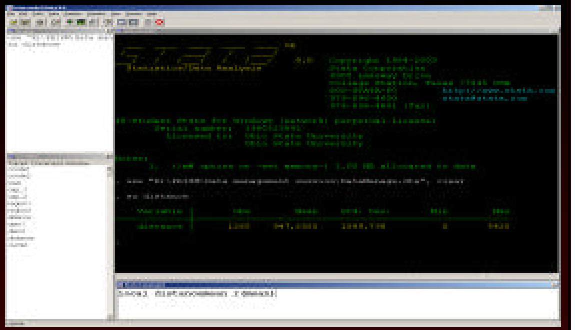
```
. local list emp_1 emp_2 region1 region2 alliance dem1 dem2 distance
. su `list'
```

Macros

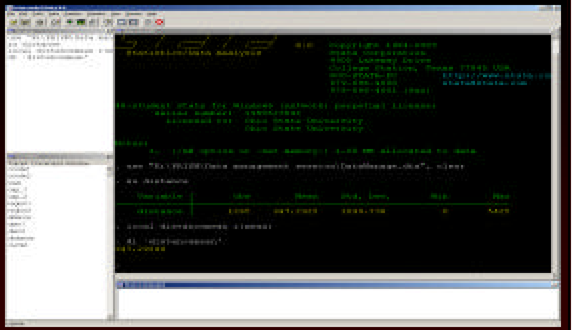
- Use macros to save descriptive stats from “summarize”.

  su distance

Macros



Macros



Macros

- Other examples:

  local a 5.67

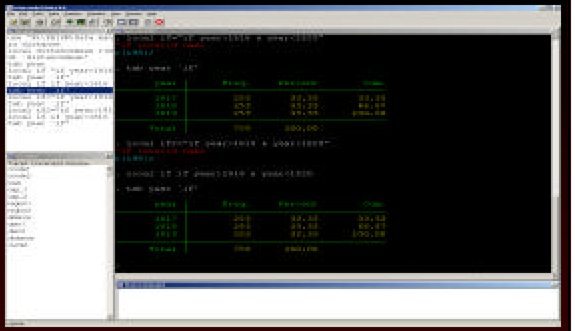
  local b 96.34

  local c 6.65

  di `a'*`b' - `c'/`a'

  local if if year>1816 & year<1820

Macros



Macros

- Macros are very useful for writing generalizable programs. Stata has some nice built-in features.
- For instance, when running a program in Stata, words included in the command after the program name are understood to be macros, named “1”, “2”, etc.
- For instance, in our “Hello, world” program.

  hello distance
- Stata would assume that in the program, distance is a macro named “1”. So anything in this program that referred to `1’ would now be distance.
- Subsequent variables after distance would be macros “2”, “3”, etc.

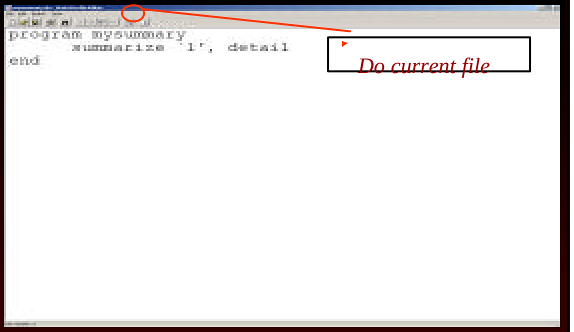
Macros

- Example: Open “mysummary.do” from the .do file editor.
- Go to the I: drive again
- Double-click on “general”

  Double-click on “PRISM Data Management”

  Double-click on “mysummary.do”

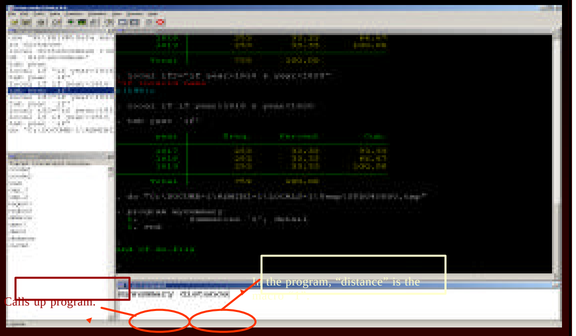
Macros



```
program mysummary
        summarize `1', detail
end
```

Do current file

Macros



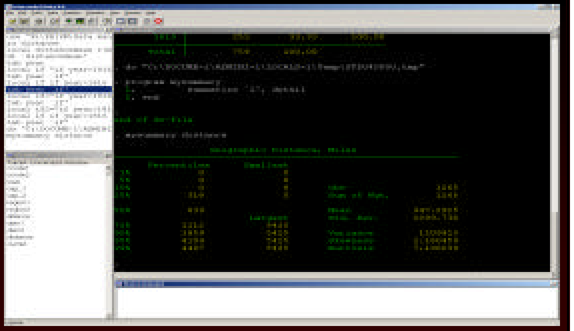
```
. program mysummary
  1.          summarize `1', detail
  2. end

end of do-file
```

```
mysummary distance
```

Calls up program.

In the program, “distance” is the macro “1”.

Macros



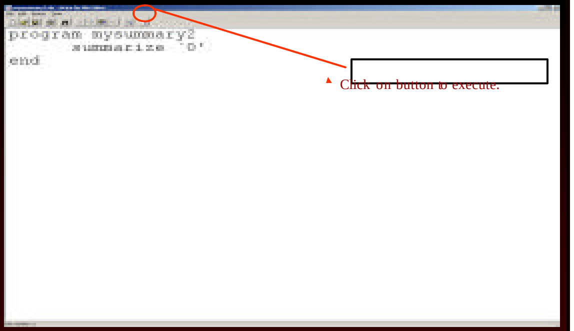
Macros

Same thing as entering *su distance, detail*

Macros

- The macro “0” indicates all of the variables included after program.
- Open “mysummary2.do” from the .do file editor.
- Go to the I: drive again
- Double-click on “general”
  - Double-click on “PRISM Data Management”
    - Double-click on “mysummary2.do”

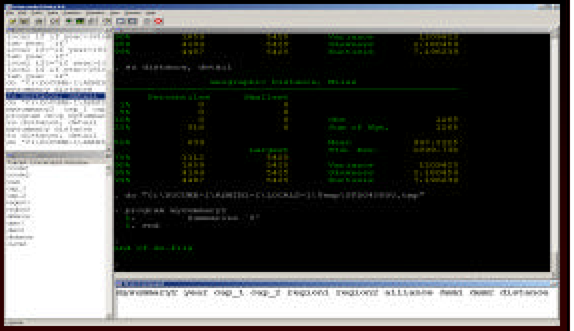
Macros



```
program mysummary2
        summarize `0'
end
```

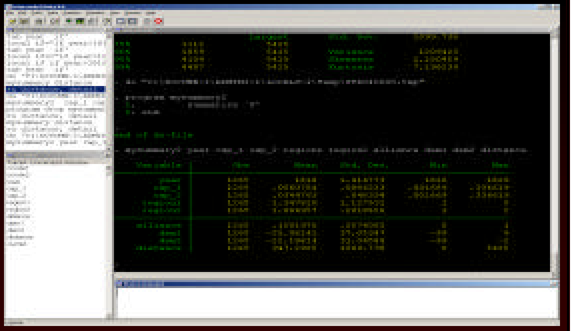
Click on button to execute.

Macros



```
mysummary year cap_1 cap_2 region1 region2 alliance dem1 dem2 distance
```

Macros



Branching and Looping

- “Real programs branch and loop.”
- Branching: *if* and *else*.
- Looping: *foreach*, *forvalues*, and *while*.
- Basic syntax for *foreach*:

```
foreach macroname [in | of listtype] list {
      commands
}
```

- Basic syntax for *forvalues*:

```
forvalues macroname = range {
    commands
}
```

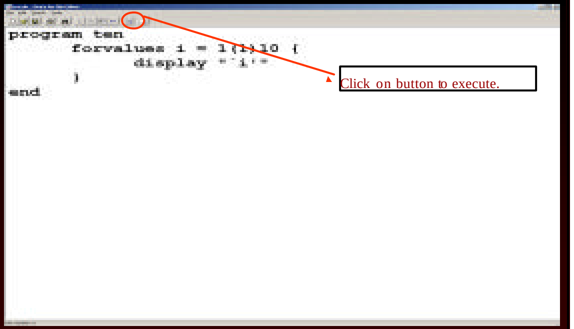
Branching and Looping

- Open “ten.do” from the .do file editor.
- Go to the I: drive again
- Double-click on “general”

    Double-click on “PRISM Data Management”

        Double-click on “ten.do”

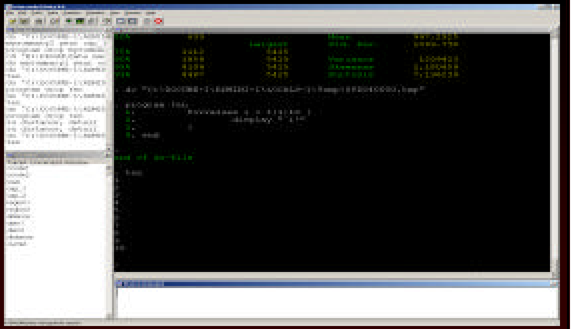
Branching and Looping



```
program ten
        forvalues i = 1(1)10 {
                display "`i'"
        }
end
```

Click on button to execute.

Branching and Looping

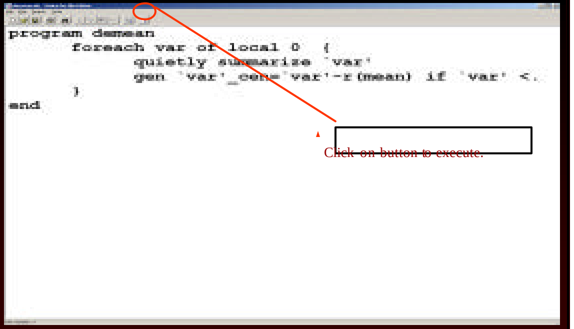


```
. do "[illegible]"

. program ten
  1.         forvalues i = 1(1)10 {
  2.                 display "`i'"
  3.         }
  4. end

end of do-file

. ten
```

Branching and Looping

- Use *foreach* to issue command on multiple variables at a time.
- “Demean” example: Powerful program for mean-centering variables; efficient and foolproof.
- Open “demean.do” from the .do file editor.
- Go to the I: drive again
- Double-click on “general”
  - Double-click on “PRISM Data Management”
    - Double-click on “demean.do”

Branching and Looping



```
program demean
        foreach var of local 0  {
                quietly summarize `var'
                gen `var'_cen=`var'-r(mean) if `var' <.
        }
end
```

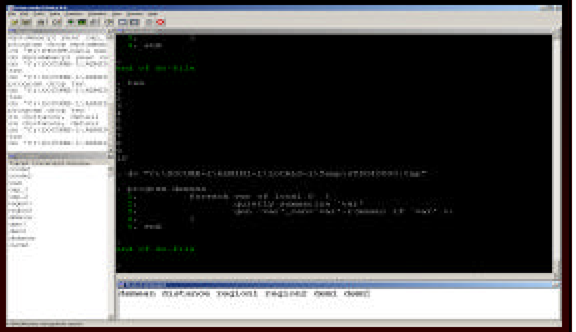
Click on button to execute.

Branching and Looping

- Housekeeping detour:

  drop distance_cen

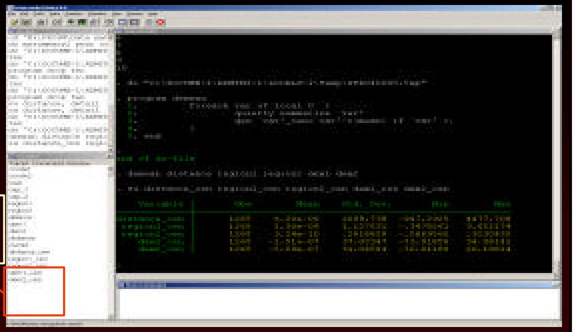
Branching and Looping



```
. do "[illegible]"

. program demean
  1.         foreach var of local 0 {
  2.                 quietly summarize `var'
  3.                 gen `var'_cen=`var'-r(mean) if `var'<.
  4.         }
  5. end

end of do-file
```

```
demean distance region1 region2 dem1 dem2
```

Branching and Looping

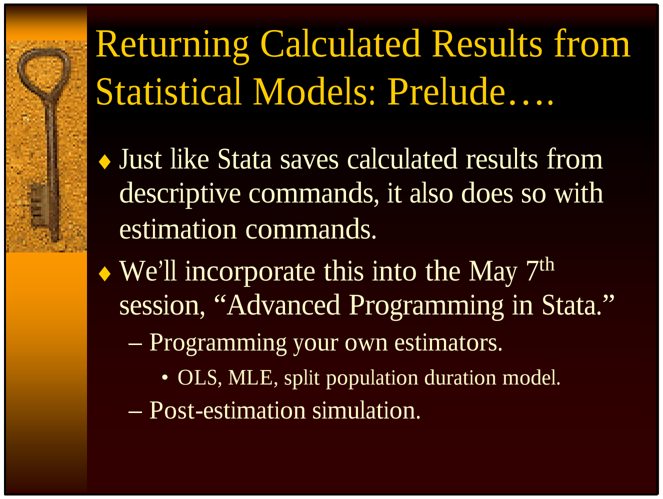


```
. do "[illegible]"

. program demean
  1.         foreach var of local 0 {
  2.                 quietly summarize `var'
  3.                 gen `var'_cen=`var'-r(mean) if `var'<.
  4.         }
  5. end

end of do-file

. demean distance region1 region2 dem1 dem2

. su distance_cen region1_cen region2_cen dem1_cen dem2_cen

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
distance_cen |  [illegible]
 region1_cen |  [illegible]
 region2_cen |  [illegible]
    dem1_cen |  [illegible]
    dem2_cen |  [illegible]
```

Creates five mean-centered variables.

You Can Write Your Program!

- Use returned calculated results, macros, branching and looping to write you own program to make your own data management more efficient and powerful.



Returning Calculated Results from Statistical Models: Prelude….

- Just like Stata saves calculated results from descriptive commands, it also does so with estimation commands.
- We'll incorporate this into the May 7th session, "Advanced Programming in Stata."
  - Programming your own estimators.
    - OLS, MLE, split population duration model.
  - Post-estimation simulation.