

Programming with Stata 14.1 Cheat Sheet

For more info see Stata's reference manual (stata.com)

1 Scalars both r- and e-class results contain scalars

scalar x1 = 3
create a scalar x1 storing the number 3
scalar a1 = "I am a string scalar"
create a scalar a1 storing a string

Scalars can hold numeric values or arbitrarily long strings

2 Matrices e-class results are stored as matrices

matrix a = (4 \ 5 \ 6)
create a 3 x 1 matrix
matrix b = (7, 8, 9)
create a 1 x 3 matrix
matrix d = b' transpose matrix b; store in d
matrix ad1 = a \ d
row bind matrices
matrix ad2 = a , d
column bind matrices
matselrc b x, c(1 3) findit matselrc
select columns 1 & 3 of matrix b & store in new matrix x
mat2txt, **matrix**(ad1) **saving**(textfile.txt) **replace**
export a matrix to a text file ssc install mat2txt

DISPLAYING & DELETING BUILDING BLOCKS

[scalar | matrix | macro | estimates] [list | drop] b
list contents of object b or drop (delete) object b
[scalar | matrix | macro | estimates] dir
list all defined objects for that class

matrix list b list contents of matrix b
matrix dir list all matrices
scalar drop x1 delete scalar x1

3 Macros public or private variables storing text

GLOBALS available through Stata sessions **PUBLIC**

global pathdata "C:/Users/SantasLittleHelper/Stata"
define a global variable called pathdata
cd \$pathdata — add a \$ before calling a global macro
change working directory by calling global macro
global myGlobal price mpg length
summarize \$myGlobal
summarize price mpg length using global

LOCALS available only in programs, loops, or .do files **PRIVATE**

local myLocal price mpg length
create local variable called myLocal with the strings price mpg and length
summarize `myLocal' add a ' before and a ' after local macro name to call
summarize contents of local myLocal
levelsof rep78, **local**(levels)
create a sorted list of distinct values of rep78, store results in a local macro called levels
local varLab: **variable label** foreign can also do with value labels
store the variable label for foreign in the local varLab

TEMPVARS & TEMPFILES special locals for loops/programs

tempvar temp1 — initialize a new temporary variable called temp1
generate `temp1' = mpg^2 — save squared mpg values in temp1
summarize `temp1' — summarize the temporary variable temp1
tempfile myAuto create a temporary file to be used within a program see also tempname
save `myAuto'

Building Blocks basic components of programming

R- AND E-CLASS: Stata stores calculation results in two* main classes:

r return results from general commands such as **summary** or **tabulate**
e return results from estimation commands such as **regress** or **mean**

To assign values to individual variables use:

- 1 SCALARS **r** individual numbers or strings
 - 2 MATRICES **e** rectangular array of quantities or expressions
 - 3 MACROS **e** pointers that store text (global or local)
- * there's also s- and n-class

4 Access & Save Stored r- and e-class Objects

Many Stata commands store results in types of lists. To access these, use **return** or **ereturn** commands. Stored results can be scalars, macros, matrices or functions.

summarize price, detail
r **return** list
returns a list of scalars

mean price
e **ereturn** list
returns list of scalars, macros, matrices and functions

scalars:
r(N) = 74
r(mean) = 6165.25...
r(Var) = 86995225.97...
r(sd) = 2949.49...
...

Results are replaced each time an r-class / e-class command is called

scalars:
e(df_r) = 73
e(N_over) = 1
e(N) = 73
e(k_eq) = 1
e(rank) = 1

generate p_mean = r(mean)
create a new variable equal to average of price

generate meanN = e(N)
create a new variable equal to obs. in estimation command

preserve create a temporary copy of active dataframe

restore restore temporary copy to original point set restore points to test code that changes data

ACCESSING ESTIMATION RESULTS

After you run any estimation command, the results of the estimates are stored in a structure that you can save, view, compare, and export

regress price weight
estimates store est1
store previous estimation results est1 in memory

Use **estimates store** to compile results for later use

eststo est2: **regress** price weight mpg ssc install estout
eststo est3: **regress** price weight mpg foreign
estimate two regression models and store estimation results
estimates table est1 est2 est3
print a table of the two estimation results est1 and est2

EXPORTING RESULTS

The **estout** and **outreg2** packages provide numerous, flexible options for making tables after estimation commands. See also **putexcel** command.

esttab est1 est2, se star(* 0.10 ** 0.05 *** 0.01) label
create summary table with standard errors and labels

esttab using "auto_reg.txt", replace plain se
export summary table to a text file, include standard errors

outreg2 [est1 est2] using "auto_reg2.txt", see replace
export summary table to a text file using outreg2 syntax

Additional Programming Resources

- bit.ly/statacode** download all examples from this cheat sheet in a .do file
- adoupdate** Update user-written .ado files ssc install adolist
- adolist** List/copy user-written .ado files
- net install package, from** (https://raw.githubusercontent.com/username/repo/master) install a package from a Github repository
- https://github.com/andrewheiss/SublimeStataEnhanced** configure Sublime text for Stata 11-14

Loops: Automate Repetitive Tasks

ANATOMY OF A LOOP

see also **while**

Stata has three options for repeating commands over lists or values: **foreach**, **forvalues**, and **while**. Though each has a different first line, the syntax is consistent:

```
foreach x of varlist var1 var2 var3 {  
    // temporary variable used only within the loop  
    // requires local macro notation  
    command `x', option  
    ...  
}
```

objects to repeat over
open brace must appear on first line
close brace must appear on final line by itself
command(s) you want to repeat can be one line or many

FOREACH: REPEAT COMMANDS OVER STRINGS, LISTS, OR VARIABLES

foreach x inof [local, global, varlist, newlist, numlist] {
Stata commands referring to 'x'
} list types: objects over which the commands will be repeated

STRINGS
foreach x in auto.dta auto2.dta {
sysuse `x', clear
tab rep78, missing
} same as...
append using file1.dta
append using file2.dta

LISTS
foreach x in "Dr. Nick" "Dr. Hibbert" {
display length `x'
} display length("Dr. Nick")
display length("Dr. Hibbert")
When calling a command that takes a string, surround the macro name with quotes.

VARIABLES
foreach x in mpg weight {
summarize `x'
} must define list type
foreach x of varlist mpg weight {
summarize `x'
} summarize mpg
summarize weight
foreach in takes any list as an argument with elements separated by spaces
foreach of requires you to state the list type, which makes it faster

FORVALUES: REPEAT COMMANDS OVER LISTS OF NUMBERS

forvalues i = 10(10)50 {
display `i'
} Use display command to show the iterator value at each step in the loop
display 10
display 20
...
ITERATORS
i = 10/50 → 10, 11, 12, ...
i = 10(10)50 → 10, 20, 30, ...
i = 10 20 50 → 10, 20, 30, ...

DEBUGGING CODE

set trace on (off) trace the execution of programs for error checking see also **capture** and **scalar _rc**

PUTTING IT ALL TOGETHER

generate car_make = word(make, 1) — pull out the first word from the make variable
levelsof car_make, **local**(cmake) — calculate unique groups of car_make and store in local cmake
local i = 1
local cmake_len : word count `cmake' — store the length of local cmake in local cmake_len
foreach x of local cmake {
display in yellow "Make group `i' is `x'"
if `i' == `cmake_len' {
display "The total number of groups is `i'"
}
local i = ++i — increment iterator by one
}

define the local i to be an iterator
tests the position of the iterator, executes contents in brackets when the condition is true
increment iterator by one