GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Programming in Stata

Stata Self-Learning Course

Overview

1. Introduction to programming in Stata
    1. Description and use of programs
    2. Arguments
    3. Temporary objects and storage
    4. Subroutines
2. Advanced programming
    1. Parsing
    2. Options
    3. Error messages
3. Applied programming
    1. Tables
    2. Graphs

GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Introduction to Programming in Stata

Stata Self-Learning Course

A quick look at the Stata manual

- "The real power of Stata is not revealed until you program it." *Stata User's Guide Release 16, p.184*
- Two program languages:
  - ado
  - Mata
- Both languages can interact with each other
- Today, we will focus on the first, ado

What are programs?

- In Stata, programs look as follows:

```
program progname
    commands
end
```

- You have to run the program definition once[1]
- If you then type *progname*, the commands defined in the program are run
- All Stata commands you know from do-files can be used in programs

[1] or store the file with the definition as ado-file in the ado-directory

Why should you program Stata?

- Programs are routines which can be adjusted and applied to different situations
  - E.g., if you have your own table style, you could write a program for this instead of adjusting it to every dataset
- Sometimes, there are no good commands for the statistical problem you have
  - E.g., if you want to adjust your standard errors in a specific way
  - Very useful in combination with Mata
- In any case, you will understand better how Stata works, how commands are written, and how to solve errors

Passing arguments to programs

- Most programs use arguments:

  regress life_expectancy income

- Here, regress is the *progname*, and life_expectancy and income are the arguments
- The arguments are passed to the program with locals

Local	Content
`0‘	Exactly what was typed, including typed blanks etc.
`1‘	The first argument
`2‘	The second argument
`*‘	All arguments without double quotes and with proper blanks

- The arguments can also be named using args

Overview of temporary objects

- Sometimes you need intermediate variables, matrices or estimations for calculations or other purposes
- In programs, two issues might emerge:
  - You have to make sure that the name you give to the object does not already exist
  - Once you are done with the process, you don't need the object anymore, it is in your way
- Sometimes, preserve and restore can be helpful, but if the program also has permanent outputs, this might not be what you need
- Locals have traits which would solve these issues, but they can only store a certain kind of information

Overview of temporary objects

- For programming, you can use temporary objects which work similar to locals
- There are different types of temporary objects
  - tempvar
  - tempname
  - tempfile
- Used as command, all create temporary names, which then can be used to create objects which will be deleted after the program ends

What exactly is Stata doing?

- To find out how your (or any other) program is working, you can use set trace on and run the program
- Stata will then display you every single working step
- This is very time- and space-consuming, so remember to turn it off using set trace off
- The command is very useful to detect where in the routine an error occurred

Creating your own (e)return lists

- Like the standard Stata programs, results from self-written programs can be stored in r(), e(), or s()
- For this, you can specify the class of the programm as rclass, eclass, or sclass
  - rclass: return list for most commands
  - eclass: return list for estimation commands (for a recommended convention, see [P] eclass)
  - sclass: special return list for locals in subroutines (see next topic)

Running subroutines

- A topic very related to storage is the use of subroutines
- You can run not only Stata commands, but also your own programs within programs
- Note that locals are truly local to programs: You can use a local only within the program in which it was created, not in nested programs/subroutines
- Most of the time, this is very convenient: You do not have to check whether this local name was used in any other nested program
- Sometimes, you might want to transfer the content of a local to another program – use globals or sreturn for this

GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Advanced Programming

Stata Self-Learning Course

Introduction to the syntax

- Last lecture, you were familiarized with arguments and program classes
- But there is more information we can pass to programs (if, in, weights, options…), take for example

  reg income age sex if country == "France"
- How can we interpret the user‘s input in a meaningful way?
- How can we assure that the user only specifies reasonable input?
- For this, we can tell Stata which elements should be accepted by the program, using the syntax command

Introduction to the syntax

- In this example, regress needs at least one variable, and we want to allow the user to specify if/in specifications
- Thus, the syntax command would read as follows*:

```
syntax varlist(min=1) [if] [in]
```

- This specification ***demands*** at least one variable and ***allows*** if and in specifications optionally
- If the user now types

```
reg income age sex if country == "France"
```

syntax creates the following locals:

`varlist'	income age sex
`if'	if country == "France"
`in'	*(empty)*

* The real regress command also allows for weights and options, this is a simplified version

Introduction to the syntax

```
syntax varlist(min=1) [if] [in]
```

- In this example, varlist is the input/argument
- There are three types of arguments which can be used for the syntax command: varlist, namelist, and anything
- Both varlist and namelist have subtypes, but the respective local will always be `varlist‘/`namelist‘

Type	Subtype	Comment	Local
varlist	varlist	List of existing variables	varlist
	varname	Abbrev. for varlist(max=1)	
	newvarlist	List of names for new variables	
	newvarname	Abbrev. for newvarlist(max=1)	
namelist	namelist	List of names for matrices/locals/variables	namelist
	name	Abbrev. for name(max=1)	
anything		Any input (commas need to be in quotes)	anything

Parsing

- The syntax command is part of the ***parsing*** process
- This process describes the break-down of the user‘s entry into meaningful elements and conversion into a meaningful structure
- For example, syntax stores the elements passed to the command before if/in or other special elements in `varlist‘
- You can define your own parsing rules using gettoken and tokenize

Helpful options and commands

- The marksample command can be used after syntax to generate a temporary indicator variable marking the observations which should be used (e.g. according to if)
- Remember that quietly suppresses the Stata output but still stores the results in r() etc. if applicable

Options with the syntax command

- Another type of input are options, e.g.

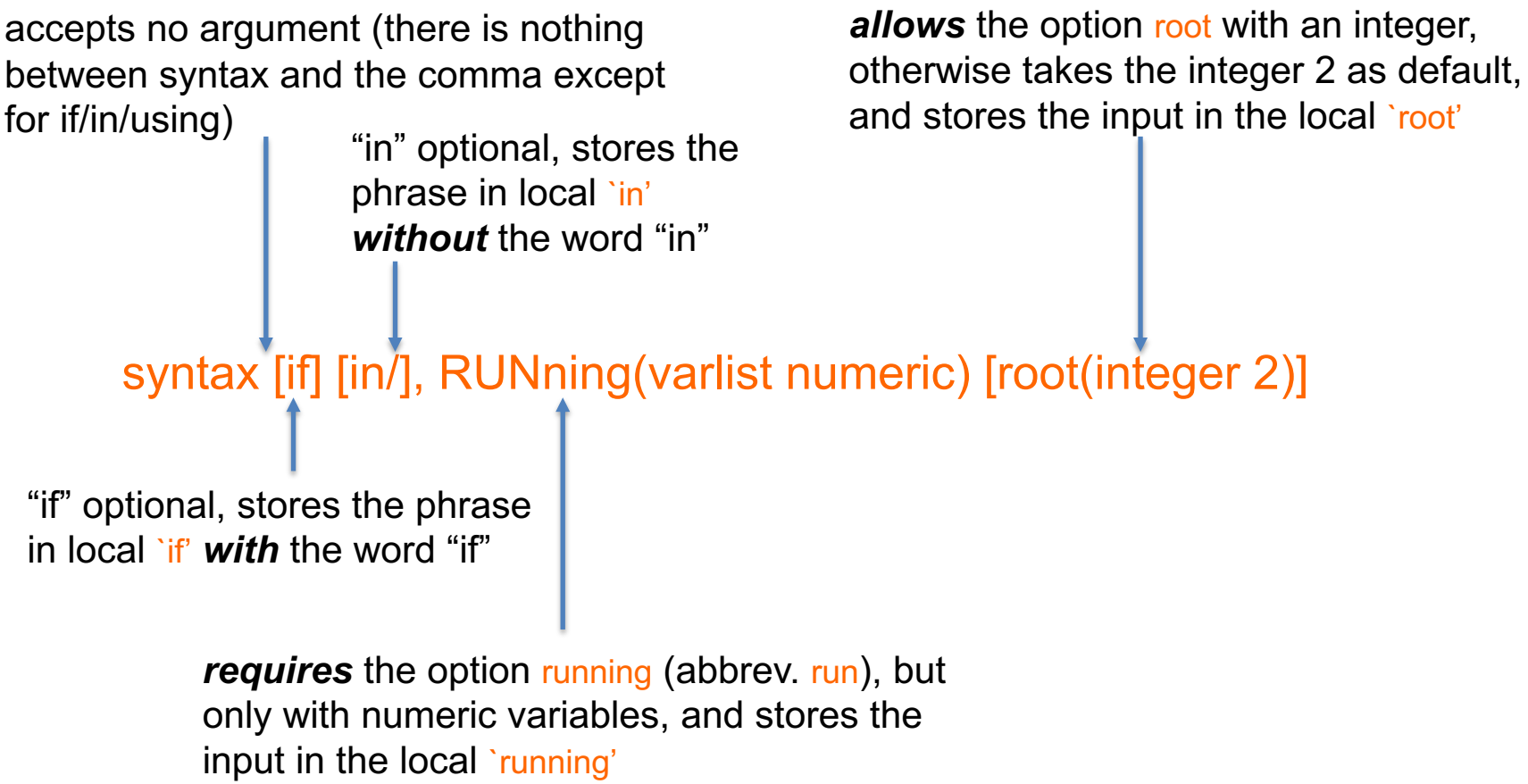
  reg income age sex if country == "France", ***vce(cluster district)***

- The syntax command allows you to program your own options, e.g.

  syntax varlist(min=1) [if] [in], vce(namelist)

- You can specify options to be mandatory (no brackets) or optional (squared brackets)
- You can define abbreviations (abbreviation in caps)
- You can have options which are only words (e.g. replace) and options which require input (e.g. vce)
- For the latter, an input type is needed (varlist, numlist etc.) which can be amended by constraints (numeric, min/max etc.)

Options with the syntax command



More helpful options and commands

- Program options
  - The byable option lets the program accept the by prefix
  - The sortpreserve option tells Stata to restore the previous sorting after the program ends
- Remember that quietly suppresses the Stata output but still stores the results in r() etc. if applicable

A remark on error messages

- The syntax command comes with its own error messages for misspecification of the program syntax
- However, it might be useful to write your own error messages or warnings to prevent mistakes
- We have done this before using display and exit, but you can also include the pre-defined error codes using error
- To have the output printed red, type display in red

GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Applied Programming

Stata Self-Learning Course

From the code to the program

- If you write your own program, you will seldomly write it „top to bottom“
- Most of the time, you will already have some code which you want to generalize
- Hence, you need to think how you make it adaptable to a wider setting, e.g.
  - change variable names to locals/tempvars
  - run loops over variables/groups/levels
- Also think about how the user will use the program
  - make it as general as possible (allow as many options/variable types as reasonable)
  - still, only allow input/options which make sense
  - write meaningful error checks and messages

Useful thoughts/checks

If you write a program, think about:

- Do you want to put something to the return/ereturn list?
- What input do you need? What should be specified or chosen by the user?
- Can the syntax command take care of it?
- Do you need error checks?
- Where might if-branches be needed?
- After running the program, will the dataset/working space be the same except for the required changes?

Useful commands/procedures

- Confirm types/classes etc.

  confirm
- Count words/elements/arguments

  wordcount() local w: word count
- Count distinct levels

  unique levelsof `var', local(w)
- String functions and extended macro functions

  help string function help macro
- Expand local lists during loops
- Use temporary objects