STATISTICAL CAPACITY BUILDING (StatCaB) PROGRAMME

TRAINING COURSE ON

AGRICULTURAL CENSUSES AND REGISTERS

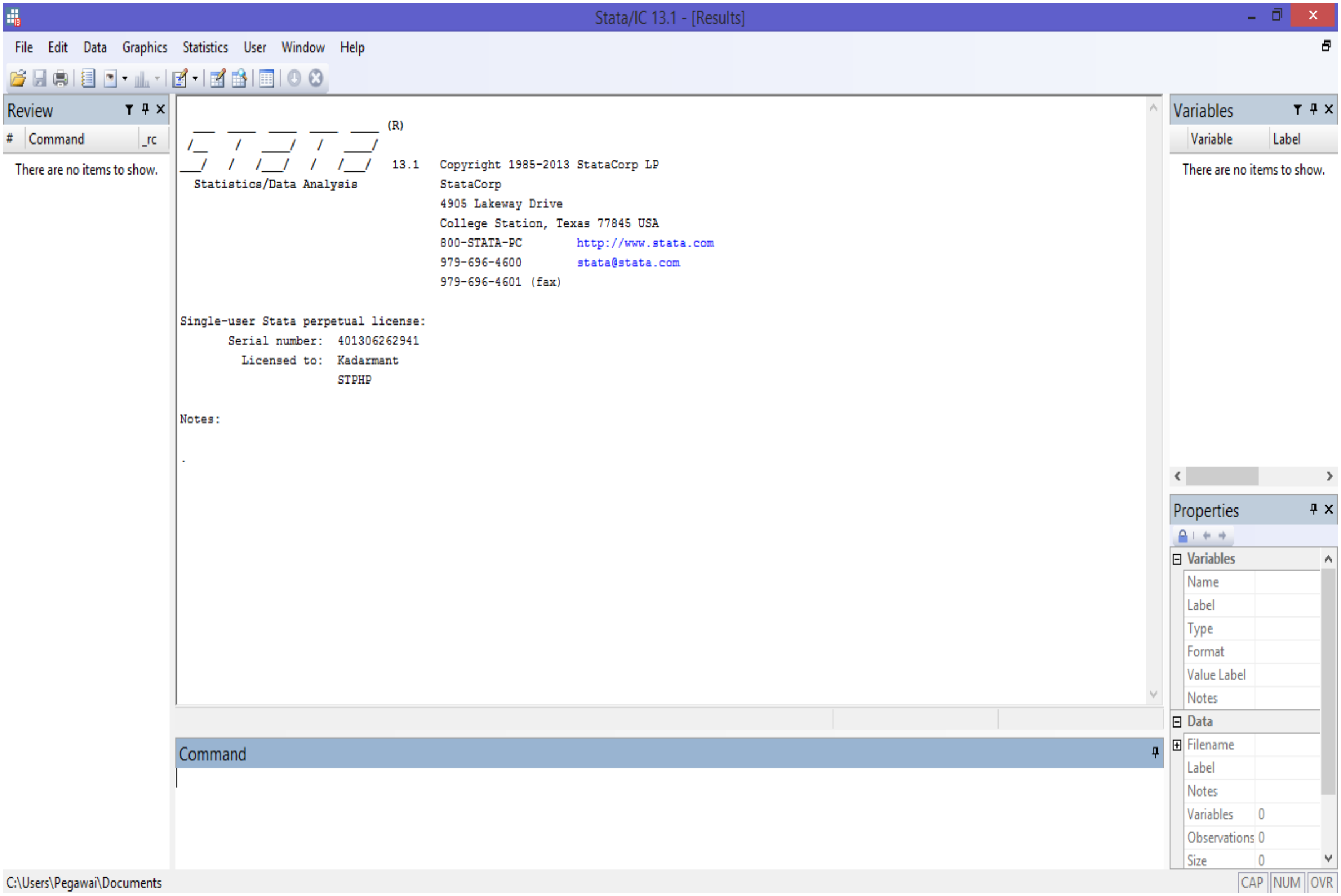
An Introduction to STATA® for Survey Data Analysis

KADARMANTO

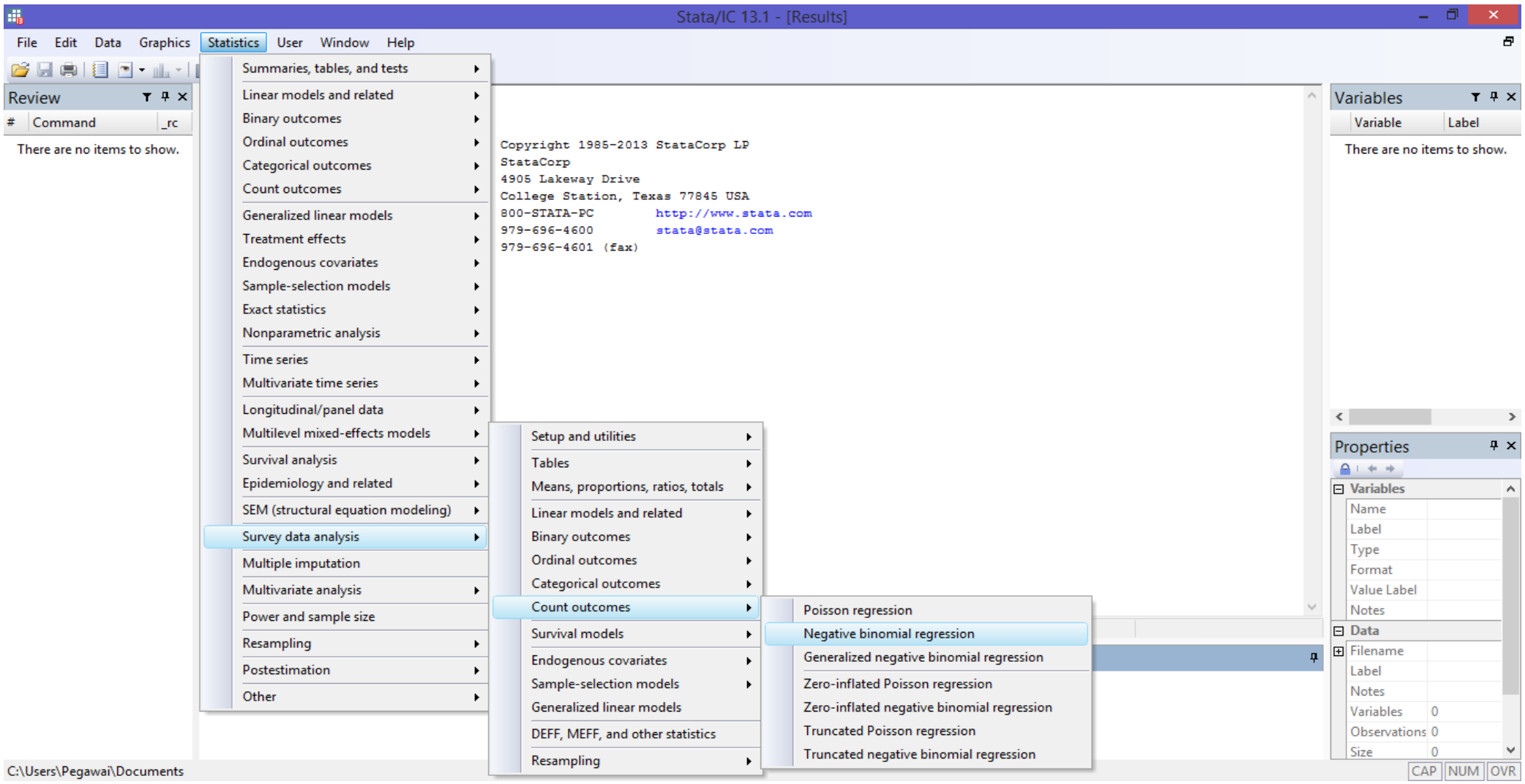
10 – 12 OCTOBER 2017, MALE – REPUBLIC OF MALDIVES

The Training **2017-StatCaB-43201** is funded by SESRIC under its Statistical Capacity Building (StatCaB) Programme.

When you launch Stata …



Option 1: the Stata Graphical User Interface



The User Interface allows for a lot of menu-driven and dialog-driven tasks

BUT this is not the way professional use Stata

Option 2: the command line

Commands are typed in the "Command" window for immediate execution.

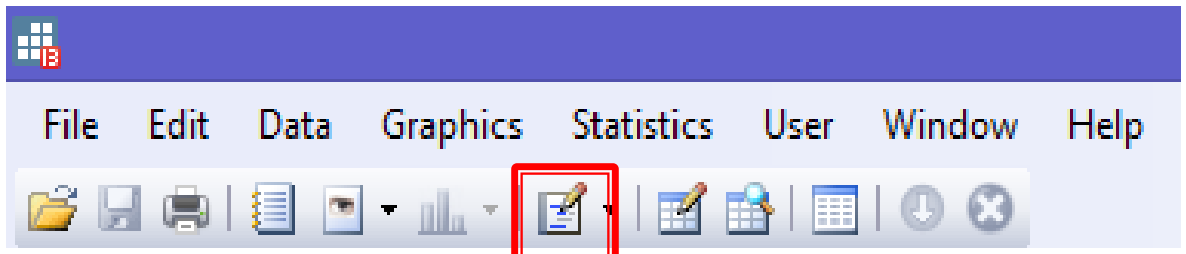
To execute a command, type it in the command line and press Enter

Option 3: writing programs (do-files)

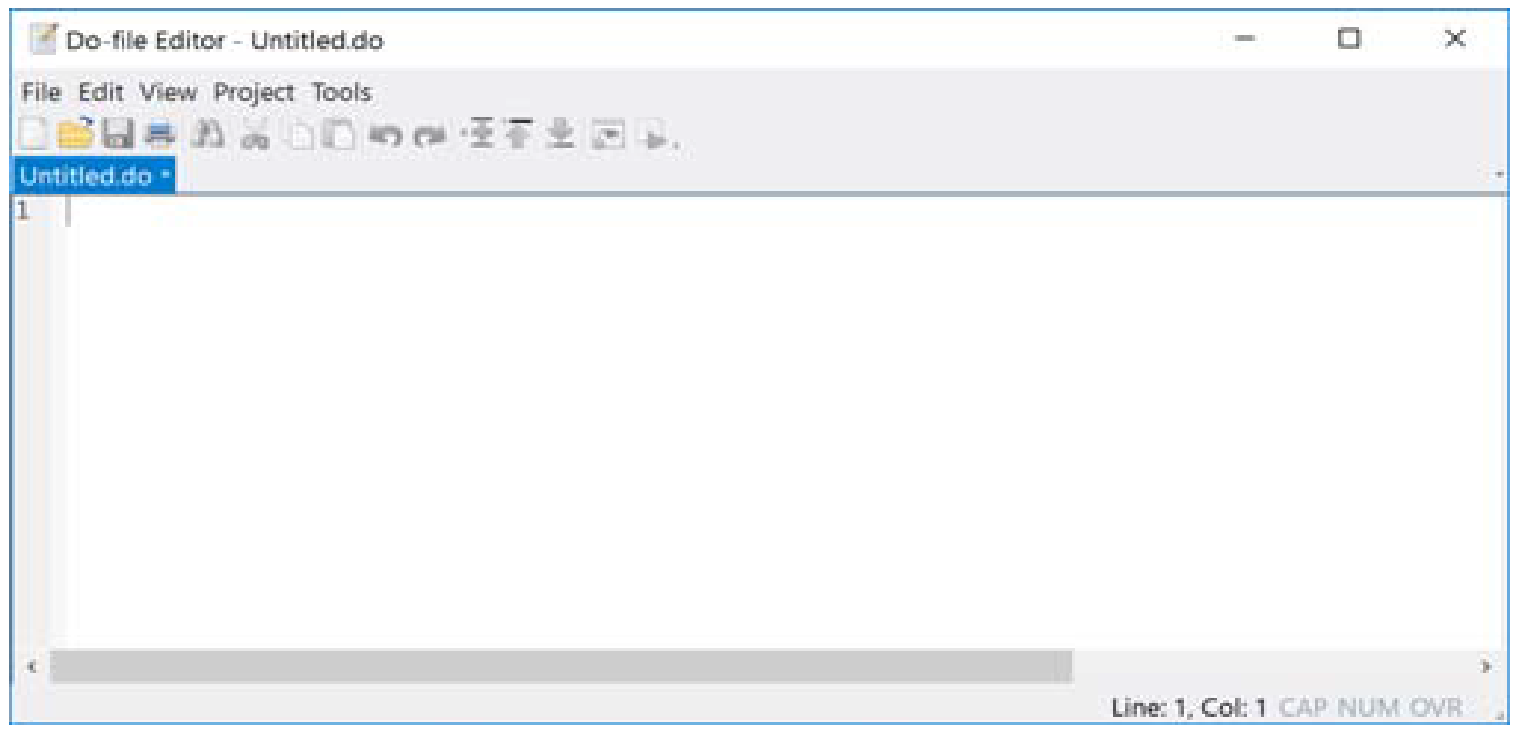
- Professionals will:
  - Write programs (do-files), not use the menu=driven or command line options
  - If relevant, write or use ado programs (specialized contributed packages)
- Why?
  - To be able to preserve, replicate, share, update, build on, re-use, and re-purpose their analysis
  - To document the analytical process
  - To automate some tasks
- Note: The menu-driven option remains useful for writing programs, as it automatically translates your selections into a command which you can copy and paste in your do files. For Stata beginners, this can help.

Accessing the do-file editor

- Do-files are text files (with .do extension) that can be produced using any text editor
- Recommendation: use the Stata do-file editor



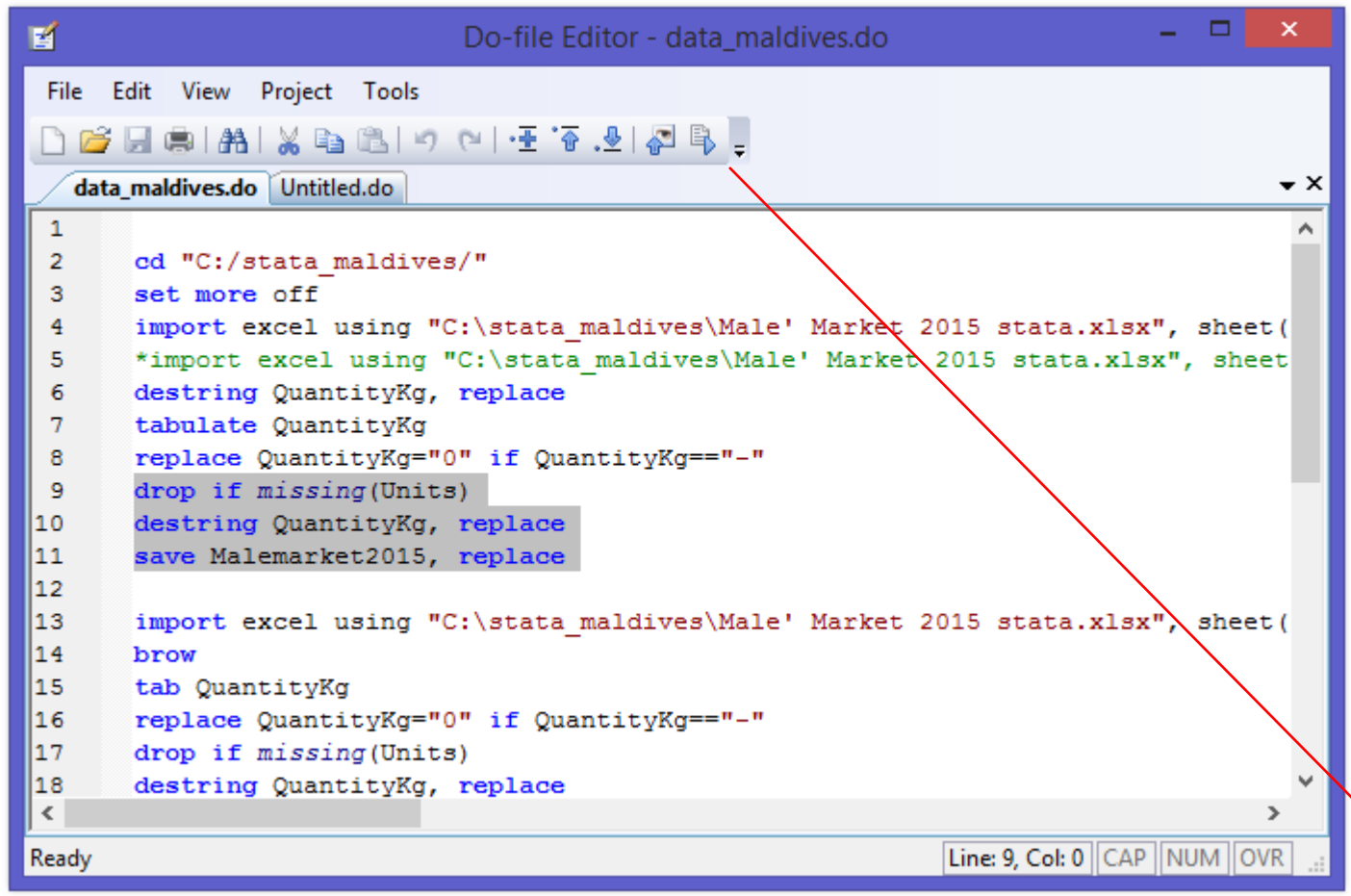
Open the do file editor



Executing commands from the do-file editor

Type your program in the do-file editor →

Select (highlight) the commands you want to execute →



Click on the EXECUTE icon

ado files

- ADO files are user-contributed packages that can be installed in Stata, to add specialized functionalities to Stata
- A large collection of ado packages is available on-line
- They can be found using the findit command in Stata
  - E.g., to find programs for inequality analysis: findit inequality
- They can also be installed from within Stata using "ssc install"
  - E.g.
    - ssc install inequal7
    - ssc install poverty

Some useful ado files

- For producing tables (in addition to Stata tabulation commands)
  - Tabout (beta version at http://tabout.net.au/docs/home.php)
- For producing maps
  - shp2dta, spmap
- For poverty and inequality analysis
  - povdeco, poverty, ineqdeco, inequal7, glorenz

- For you ?
  - Find out using findit

Before we start…

Good practice for data analysis

Some important rules to follow:

- ❑ Understand your data before you analyze them
- ❑ Document your dataset
- ❑ Protect your data – Work on a copy, not on the original dataset
- ❑ Make everything reversible and reproducible
- ❑ Document your Stata programs

Some fundamental information

- Variable names can be up to 32 characters
- Variables in a Stata file can be either numeric or alphanumeric (string variable)
- Stata is case sensitive (for commands, variable names, etc.)
  - Commands must be typed in lowercase (example: use is a valid command; but if you type USE it will not work)
  - A variable named Age is not the same as a variable named age

Getting help

- Stata has a very large number of commands. Each command has a
- syntax, and often provide multiple options.

- Users will very often rely on the on-line Help to find out how to implement a command

- The Stata command to get help on a command is help followed by the name of the command, e.g. help merge

- Understanding how to read the syntax of a command is very important
- If you do not know the name of the command, use the search
- function

Syntax of commands

- With few exceptions, the basic Stata language syntax is

[by varlist:] command [varlist=exp] [if exp] [in range] [weight] [, options]

Where:

- square brackets distinguish optional qualifiers and options from required ones.

- varlist denotes a list of variable names, command denotes a Stata command, exp denotes an algebraic expression, range denotes an observation range, weight denotes a weighting expression, and options denotes a list of options.

Example of syntax

- Type help summarize in the command line. The summarize command calculates and displays a variety of univariate summary statistics. We syntax is:

summarize [varlist] [if] [in] [weight] [, options]

Options	Description
detail	display additional statistics
meanonly	suppress the display; calculate only the mean; programmer's option
format	use variable's display format
separator(#)	draw separator line after every # variables; default is separator(5)
display_options	control spacing, line width, and base and empty cells

Short and abbreviated name of commands

- Command (and variable) names can generally be abbreviated to save typing.
- As a general rule, command, option, and variable names may be abbreviated to the shortest string of characters that uniquely identifies them.
- For instance, typing su (or summ) instead of summarize will work.
- This rule is violated if the command or option does something that cannot easily be undone; the command must then be spelled out in its entirety.
- The syntax underlines the minimum set of characters needed

Examples

Syntax

regress *depvar* [*indepvars*] [*if*] [*in*] [*weight*] [, *options*]

Describe data in memory

describe [*varlist*] [, *memory_options*]

recode *varlist* (*rule*) [(*rule*) ...] [, **generate**(*newvar*)]

Analysis of sample survey data: Survey design, sample weights, and the svy commands

Training on Agricultural Censuses and Registers in Male Republic of Maldives, 10-12 October 2017

A brief reminder on sampling design

- We are interested in using Stata for survey data analysis
- Survey data are collected from a sample of the population of interest

- Each observation in the dataset represents multiple observations in the total population
- Sample can be drawn in multiple ways: simple random, stratified, etc.

- For example: randomly select N villages in each province first, then 15 households in each village

- Sample weights are variables that indicate how many units in the population each observation represents

Sampling weights

- Sample weights are typically the inverse of the probability for an observation of being selected

- Example: in a simple random selection, if the total population has 1,000,000 households and we draw a sample of 5,000:
- The probability of being selected is 5,000 / 1,000,000 = 0.005
- The sample weight of each household will be 1,000,000 / 5,000 = 200

- In more complex sample designs, the sample weight will be different for each region, or enumeration area, etc.

- When we produce estimates (of totals, means, ratios, etc.) we need to apply these weights to have estimates that represent the population and not the sample (i.e. we need "weighted estimates")

Working on data files

The structure of a Stata data file

Variables

Observations

	month	Date	Atoll	Island	Boatname	Crops	CropsinEnglish	Units	QuantityMe~d
1	jan	1/4/2015	AA	Thoddoo	Ramz-2	Falhoa	Papaya	Kg	1800
2	jan	1/4/2015	AA	Thoddoo	Ramz-2	Israel Falhoa	Papaya (Israel)	Kg	300
3	jan	1/4/2015	AA	Thoddoo	Ramz-2	Chichandaa	Snake Gourd	Kg	200
4	jan	1/4/2015	AA	Thoddoo	Ramz-2	Baraboa	Pumpkin	Kg	200
5	jan	1/4/2015	AA	Thoddoo	Ramz-2	Thoraa	Sponge Gourd	Kg	200
6	jan	1/4/2015	AA	Thoddoo	Ramz-2	Hika'ndhifaiy	Curry Leaves	Basthaa	20
7	jan	1/4/2015	AA	Thoddoo	Ramz-2	Bashi	Egg Plant	Kg	300
8	jan	1/4/2015	AA	Thoddoo	Ramz-2	Cucumber	Cucumber	Kg	4000
9	jan	1/4/2015	AA	Thoddoo	Ramz-2	Faaga	Bitter Gourd	Kg	100

Opening a data file

- Syntax:

  use filename, clear

- If no path is specified, Stata will look in the default directory. You can find what is the default data directory by typing "cd" or "pwd" in the command line. You can change the directory by typing cd "path".

- Example:

  use "C:\Stata_Maldives\Data\Malemarket2015.dta", clear

  or

  cd "C:\Stata_Maldives\Data"

  use "Malemarket2015.dta", clear

Sorting a data file - sort

- Syntax:

  sort varlist

- Example:

  sort Crops Date

Sorting a data file - gsort

- The sort command will sort by ascending value of the selected variable(s)
- To sort in descending order, use the gsort command
- Syntax:

gsort [+|-] varname [[+|-] varname ...] [, generate(newvar) mfirst]

- The options allow you, among other things, to generate a variable with a sequential number of the ordered records.

Example: to sort a data file by decreasing order of variable income:

gsort -tot_exp hhid

Compressing and saving data files

- Compressing
  - compress attempts to reduce the amount of memory used by your data.
  - It never results in loss of precision
  - Note: this is not the same as zipping files.
- Saving Stata data files
  - save [filename] [, save_options]
  - E.g.,
  - save “Malemarket2015.dta", replace

- Files saved in current stata will not be readable with previous versions of the software. If you need to save data in an older format, use option saveold.

Browsing (viewing) the data

	month	Date	Atoll	Island	Boatname	Crops	CropsinEnglish	Units	QuantityMe~d
1	jan	1/13/2015	GN	Fuvamulah	Ravaanaa	A'nbu	Mango	Nos	100000
2	feb	2/1/2015	LA	Hithadhoo	Faihaa	A'nbu	Mango	Nos	130
3	feb	2/3/2015	SE	Hithadhoo	Furusath	A'nbu	Mango	Nos	2500
4	feb	2/8/2015	SE	Hithadhoo	Aagala queen	A'nbu	Mango	Nos	5000
5	feb	2/8/2015	TH	Hirilandhoo	Nafaa	A'nbu	Mango	Nos	500
6	feb	2/11/2015	GN	Fuvahmulah	Naares	A'nbu	Mango	Nos	13000
7	feb	2/12/2015	KA	Kaashidhoo	Treaser	A'nbu	Mango	Nos	800
8	feb	2/12/2015	LA	Kunahandhoo	Soanaa	A'nbu	Mango	Nos	2000
9	feb	2/15/2015	GDH	Faresmaathodaa	Fahiroalhi	A'nbu	Mango	Nos	2000
10	feb	2/15/2015	LA	Isdhoo/kalaidhoo	Falaky	A'nbu	Mango	Nos	500
11	feb	2/17/2015	SH	Feevah	Amaaz	A'nbu	Mango	Nos	1000
12	feb	2/19/2015	NO	miladhoo	Manzil	A'nbu	Mango	Nos	400
13	feb	2/24/2015	AA	Thoddoo	Feneybaa	A'nbu	Mango	Nos	200
14	feb	2/26/2015	GN	Fuvahmulah	Havaas	A'nbu	Mango	Nos	20000
15	mar	3/3/2015	GN	Fuvahmulah	Ravaanaa	A'nbu	Mango	Nos	140000
16	mar	3/4/2015	SH	Feydhoo	Shaazee-4	A'nbu	Mango	Nos	500
17	mar	3/4/2015	SE	Hithadhoo	Aagala queen	A'nbu	Mango	Nos	6000
18	mar	3/4/2015	AA	Thoddoo	Ramz-2	A'nbu	Mango	Nos	400

Inspecting data files – File description

`describe` produces a summary of the dataset in memory

`describe [varlist] [, memory_options]`

```
Contains data from Malemarket2015.dta
  obs:          4,840
 vars:             15                          11 Oct 2017 19:11
 size:        682,440
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
month           str4    %9s                   month
Date            int     %td..                 Date
Atoll           str3    %9s                   Atoll
Island          str23   %23s                  Island
Boatname        str16   %16s                  Boat name
Crops           str18   %18s                  Crops
CropsinEnglish  str16   %16s                  Crops in English
Units           str7    %9s                   Units
QuantityMeasu~d long    %10.0g                Quantity (Measured)
QuantityKg      double  %10.0g                Quantity (Kg)
RPrice1         double  %10.0g                R.Price 1
RPrice2         double  %10.0g                R.Price 2
AvePrice        double  %10.0g                Ave.Price
Wsprice         double  %10.0g                Ws.price
Income          double  %10.0g                Income
-------------------------------------------------------------------------------
Sorted by:  Crops  Date
```

Inspecting data files – Summary statistics

`summarize` calculates and displays a variety of univariate summary statistics. If no `varlist` is specified, summary statistics are calculated for all the variables in the dataset.

```
summarize [varlist] [if] [in] [weight] [, options]
```

Examples:

```
summarize [weight=hhwgt]
summarize [weight=hhwgt]
summarize if province==1
```

Inspecting data files – Counting records

`count` counts the number of observations that satisfy the specified conditions. If no conditions are specified, `count` displays the number of observations in the data.

```
count [if] [in]
```

Examples:

```
use Malemarket2016sheet3, clear
Count // Counting all observations in data file
count if Boatname=="Faihaa“// Counting Faihaa
count if Boatname=="Faihaa" & QuantityKg>500 // Counting Faihaa with Quantity more than 500 kg
```

Inspecting data files – Listing observations

`list` allows you to view the values in selected observations

```
list [varlist] [if] [in] [, options]
```

Examples:

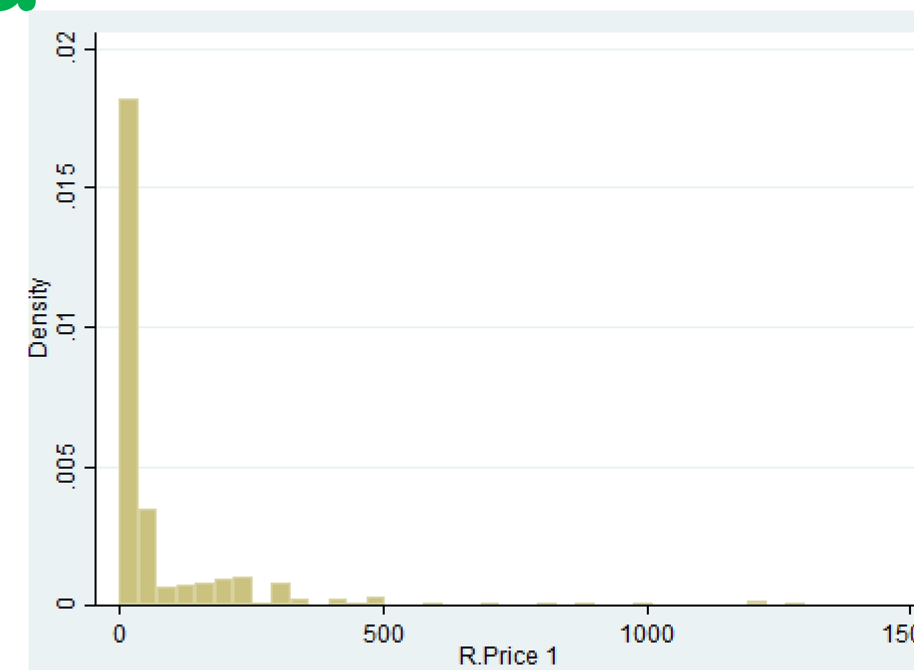
List of top 5 observations:

```
list in 1/5
```

Display Island and Crops for Boat named Faihaa or Barubaree

```
list Island Crops if Boatname=="Faihaa" | Boatname=="Barubaree"
```

Inspecting data files – Inspect command



The `inspect` command provides a quick summary of a numeric variable, different from the `summarize` command.

```
inspect [varlist] [if] [in]
```

Example:

`inspect RPrice1`

(marital status) →

```
RPrice1:   R.Price 1                          Number of Observations
-------------------------------          ---------------------------------------
                                             Total    Integers   Nonintegers
|  #                          Negative          -          -             -
|  #                          Zero              8          8             -
|  #                          Positive       4276       4252            24
|  #                                        -----      -----         -----
|  #                          Total          4284       4260            24
|  #   .   .   .   .          Missing           2
+------------------------                   -----
0                  1300                      4286
   (65 unique values)
```

We discover that the variable has 2 missing value ; 4284 observations in the dataset have some value for RPrice1. Moreover, the values are all non negative and 4260 are integers while 24 are nonintegers. Among those 4284 observations are 65 unique (different) values. The variable ranges from 0 to 1300, and we are provided with a small histogram that suggests that the variable appears to be what it claims.

Inspecting data files – Produce a codebook

codebook examines the variable names, labels, and data to produce a codebook describing the dataset.

```
codebook [varlist] [if] [in] [, options]
```

Examples:

```
codebook      // all variables in data file
codebook   Crops-RPrice1    // variables Crops to RPrice1
codebook   RP*    // all variables with name starting with RP
```

Appending data files

`append` appends Stata-format datasets stored on disk to the end of the dataset in memory.

```
append using filename [filename ...] [, options]
```

Hierarchical structure of survey datasets

- Survey datasets are typically made of multiple related data files
- For example, in a household survey, one file may contain:
  - Demographic information (1 observation per person)
  - Data on education (1 observation per person aged 4+)
  - Data on employment (1 observation per person aged 15+)
  - Data on births (1 observation per woman aged 12 to 49)
  - Data on dwelling characteristics (1 observation per household)
  - Data on expenditures (1 observation per product/service per household) Etc.
- We need "keys" (common variables) to merge these files

Hierarchical structure and keys

Individual level data (1 or > rows per boat)

	Boatname	month	Date	Atoll	Island	Crops	CropsinEnglish	Units	QuantityMe~d	QuantityKg
1	Aagala	feb	2/28/2016	SE	Feydhoo	Faiykeyo		Ga'ndu	20	300
2	Aagala	feb	2/28/2016	SE	Meedhoo	A'nbu		Nos	1400	151.2
3	Aagala	feb	2/28/2016	SE	Feydhoo	Faiykeyo		Ga'ndu	10	150
4	Aagala	feb	2/28/2016	SE	Meedhoo	Kaashi		Nos	1000	250
5	Aagala	feb	2/28/2016	SE	Feydhoo	A'nbu		Nos	600	64.8
6	Aagala	feb	2/28/2016	SE	Feydhoo	Kaashi		Nos	1000	250
7	Aagala queen	dec	12/14/2015	SE	Hithadhoo	Kuru'nbaa	Tender Coconut	Nos	2000	6000
8	Aagala queen	dec	12/14/2015	SE	Hithadhoo	Ala	Taro	Kg	300	300
9	Aagala queen	dec	12/14/2015	SE	Hithadhoo	Faiykeyo	Banana	Ga'ndu	300	4500
10	Aagala queen	dec	12/14/2015	SE	Meedhoo	Faiykeyo	Banana	Ga'ndu	60	900
11	Aalim	sept	9/19/2016	ME	Kolhufushi	Faiykeyo		Ga'ndu	3	45
12	Aalim	oct	10/6/2015	ME	Kolhufushi	Faiykeyo	Banana	Ga'ndu	2	30
13	Aalim	oct	10/6/2015	ME	Kolhufushi	Kuru'nbaa		Nos	69	207

Boat level data (1 row per boat)

	Boatname	madein
1	Aagala	Netherland
2	Aagala queen	India
3	Aalim	China
4	Aaraasth	Germany
5	Akris	Germany

Merging data files

- Merging data files is a crucial operation for survey data analysis and it is important to fully master it.
- The objective is to merge observations found in 2 different data files based on "key variables" (variables common to both datasets)
- Key variables are the identifiers of the observations (e.g., identifier of the household)

Merging data files

- The relationship between 2 data files can be of different types. The most important for survey data analysts are:

  - The one-to-one relationships (where one observation from the source file has only one observation in the merged file)
    - For example: One file contains the demographic information about individuals; the other one contains the employment variables for the same sample.

- The many-to-one relationships (where multiple observations in the source file correspond to one observation in the merged file)
  - For example: One file contains the information on individuals (age, sex, etc.) and the other one contains information on dwelling characteristics. For all members of a same household, there will be one and only one observation about the dwelling characteristics.

Merging data files

- To merge observations, we need key variables which are variables common to both data files being merged.

- In the exercise data files, each household has a unique identifier (variable hhid) and each household member is uniquely identified by a combination of two variables: hhid (which identifies the household) and indid which identifies the person within the household.

- In principle, hhid is unique to each household in the household- level file, and the combination of hhid and indid is unique to each individual in the person-level data file.
- If that is not the case, the merging will not be successful.

Merging data files

- Merging data files – The syntax

- One-to-one merge on specified key variables

  merge 1:1 varlist using filename [, options]

- Many-to-one merge on specified key variables

  merge m:1 varlist using filename [, options]

- IMPORTANT: Data files must be sorted by the key variables for merge to work. If the data are not sorted, you will get an error message.

Merging data files

- Merging data files – The _merge variable

- The merge command generates a new variable named _merge that reports on the outcome of the merging. The variable can take 5 possible values. Values 1 to 3 are particularly relevant:

  - observation appeared in master file only
  - observation appeared in "using" file only
  - match: observation appeared in both data files

Checking unicity of key(s)

- We can easily check that the key variable(s) provide(s) a unique identification of each observation, using the isid command.

```
isid varlist
```

- If there are duplicates, it means that you did not identify the right variables as keys, or that there are problems in the data files
- Duplicates can be identified and listed using the duplicates command.

Tagging duplicates (an example)

- To find duplicates → Use "tag" option of duplicates command

duplicates tag [varlist] [if] [in] , generate(newvar)

- Example:

duplicates tag hhid indid, generate(isdup)

tabulate isdup

Merging data files – Examples

- One-to-many merge on specified key variables (Male Market 2016 data files)

```
use Boatname, clear
sort Boatname
merge 1:m Boatname using Malemarket2016sort
tab _merge
```

- Many-to-one merge on specified key variables

```
use “Malemarket2016sort.dta", clear
merge m:1 hhid using “Boatname.dta"
tab _merge
```

Working with variables

Variables – The basics

- Variable names can be up to 32 characters
- Stata is case sensitive
- Variables in a Stata file can be either numeric or alphanumeric (string)
- Variable names can be abbreviated (like commands)
- Use of * and ?
- List of variables: v3-v7

Labeling variables and values

Variables should be documented.

- All variables should have a label. A variable label is a description (up to 80 characters) of the variable.

- All categorical variables should also have value labels. Value labels are the descriptions of the codes used in categorical variables (e.g., for variable sex, 1 = "Male" and 2 = "Female")

- Labels help you identify variables, and will be used by Stata when tables or other outputs are produced

Labeling variables

To add a label to a variable:

```
label variable varname ["label"]
```

To change or modify a variable label: same command (will overwrite the existing label)

Labeling values

Add value labels is a two-step process: we first define a set of labels (`label define`), then attach it to a variable (`label values`).

A same set can be used for multiple variables.

For example:

```
label variable exim "Main Activity"
label define activity 1 "Importer" 2 "Exporter"
label value exim activity
```

Modifying and eliminating value labels

To add or modify value labels:

label define lblname # "label" [# "label" ...] [, add modify replace]

Example:

```
label define activity 1 "Importer" 2 "Exporter"
label define activity 2 "Exporter to Europe", modify
label define activity 3 "Exporter to Non-Europe", add
```

To eliminate value labels:

label drop {lblname [lblname ...] | _all}

Example:

```
label drop activity
```

Tabulating values of a variable

- Note: we will see later how to produce cross-tables of summary statistics.

```
tabulate varname [if] [in] [weight] [, tabulate1_options]
```

Examples:

```
save Malemarket2016merge, replace
use Malemarket2016merge, clear
tabulate Island
tabulate Island, sort // sort frequencies
tabulate Island if exim==3
```

Generating new numeric variables

- In Stata, you can generate a new variable using the command `generate`. The general syntax is:

```
generate newvarname = expression
```

- You cannot generate a variable if a variable with the same name already exists
- Use the command `replace` to assign new values to an existing variable

Operators

Relational operators	Logical operators	Mathematical operators
< (less than)	| (or)	+
> (greater than)	& (and)	-
== (equal)	~ (not)	*
<= (less than or equal)		/
>= (greater than or equal)		^
!= or ~= (not equal)		

Mathematical functions

- If x is a numeric variable:

abs(x)	the absolute value of x
exp(x)	The exponential function of x
int(x)	the integer obtained by truncating x toward 0 (thus, int(5.2) = 5 and int(-5.8) = -5)
ln(x) or log(x)	the natural logarithm, ln(x)
max(x1,x2,: : :,xn)	the maximum value of x1; x2; : : : ; xn
min(x1,x2,: : :,xn)	the minimum value of x1; x2; : : : ; xn
mod(x,y)	the modulus of x with respect to y
round(x,y) or round(x)	x rounded in units of y or x rounded to the nearest integer if the argument y is omitted
sqrt(x)	the square root of x
sum(x)	the running sum of x, treating missing values as zero

Missing values

- Missing values in Stata are indicated by a dot (.)
- Stata has the possibility to create different types of missing values
  - . / .a / .b / etc. until .z
  - By default, the simple dot is used (.)

- IMPORTANT: . Is considered by Stata as the largest positive value (infinity). This means that the "value" of . Is greater than any number.
- This has important implications when we work with variables:
  - To count the number of observations for which variable RPrice1 is missing, type:

```
count if RPrice1 >=.
```

  - To create a new variable and assign value 1 if RPrice1 is greater than 1000, type:

```
generate highprice1=1 if RPrice1>1000 & RPrice1<.
```

Recoding variables

Syntax:

```
recode varlist (rule)[(rule)...] [, generate(newvar)]
```

rule	Example	Meaning
# = #	3 = 1	3 recoded to 1
# # = #	2 . = 9	2 and . recoded to 9
#/# = #	1/5 = 4	1 through 5 recoded to 4
nonmissing = #	nonmiss = 8	all other nonmissing to 8
missing = #	miss = 9	all other missings to 9

Recoding variables – Example

Creating price1 groups by recoding RPrice1

```
recode RPrice1 (0/200=1) (201/400=2)
(401/600=3) (601/800=4) (801/1000=5)
(1001/max=6), generate(price1group)
```

The commands encode and decode

- Use decode to convert numeric into strings variables. Stata will generate a new (string) variable containing the label of the numeric variable

Example: `decode exim, generate(sexim)`

- Use `encode` to convert strings into numeric variables. Stata will create a new (numeric) variable by automatically assigning numeric codes and create the corresponding value labels.

inlist and inrange

`inlist()` and `inrange()` are useful programming functions associated with commands that are often used.

Examples of use:

```
generate europe=1 if inlist(madein,"Netherland","Germany")
generate middleprice=1 if inrange(RPrice1,750,1000)
```

Operations on string variables

- In some cases, numeric variable may have been imported as string variables (e.g., 1 will not be considered as value 1, but as an alphanumeric character)
  - You cannot perform mathematical operations on string variables
  - Note: in the Stata browser, string variables will be displayed in red

- You can convert a variable from string to numeric type by using the `destring [variablename]` command. This will only work if the variable only contains numbers, not letters.

- Stata provides many functions for working with string variables (including functions to subset strings, concatenate, etc.)

Operations on string variables – Some functions

- `abbrev(s,n)` → returns s (=text) abbreviated to a length of n
- `substr(s,n1,n2)` → returns the substring of s, starting at position n1, for a length of n2
- `strlower(s) / strupper(s)` → converts to lower (upper) case
- Functions can be combined (nested) into one command
- Strings can be combined using " + "
- Example:

```
generate staff = "Pierre"
generate staff2 = strupper(substr(staff,1,4))+ "."
// → staff2 = PIER.
```

Renaming variables

`rename` changes the name of an existing variable

Example: `rename price1group price1class`

Stata provides some functions for renaming groups of variables; see `help rename group`

Deleting (or keeping) variables

- `drop` eliminates variables from the data file in memory.
- `keep` works the same as drop, except that you specify the variables to be kept rather than the variables to be deleted.
- Warning: `drop` and `keep` are not reversible (there is no “undo”). Once you have eliminated variables, you cannot read them back in again. You would need to go back to the original dataset and read it in again.
- Examples:
  - `drop _merge`
  - `keep date \\ there will be only date in your data`

Deleting (or keeping) observations

- The same commands drop and keep can be used to select observations
- `drop` eliminates observation; `keep` works the same as drop, except that you specify the observations to be kept rather than the ones to be deleted.
- Warning: `drop` and `keep` are not reversible. Once you have eliminated observations, you cannot read them back in again. You would need to go back to the original dataset and read it in again.
- Examples:
  - `drop if RPrice1 ==.`
  - `keep if RPrice1 < .`

Ordering variables

`order` changes the sequence in which the variables are listed in a data file. It does not change the value of the data. This will typically be done to ensure that some key variables are displayed on top of the list.
You only have to list the variables you want to be displayed first.

For example:

```
describe
order europe,before(madein)
describe
```

Boatname	str16	%16s		Boat name
madein	str10	%10s		
exim	float	%22.0g	activity	Main Activity
month	str4	%9s		month
Date	int	%td..		Date

Boatname	str16	%16s		Boat name
europe	float	%9.0g		
madein	str10	%10s		
exim	float	%22.0g	activity	Main Activity
month	str4	%9s		month
Date	int	%td..		Date

Generating new variables with egen

- `egen` creates new variables representing summary statistics (calculated in rows or columns)
- `egen` uses functions specifically written for it
- The syntax is:

```
egen [type] newvar = fcn(arguments) [if] [in] [, options]
```

- The functions include `count()`, `iqr()`, `min()`, `max()`, `mean()`, `median()`, `mode()`, `rank()`, `pctile()`, `sd()`, and `total()`.

- These functions take a `by . . . :` prefix which allow calculation of summary statistics within each by-group.

Use of egen – Some examples

```
use "individual.dta", clear
```

*Add a variable with the age of the oldest hhld member for each hhld

```
egen oldest = max(age), by(hhid)
```

*Add the number of members declared as “spouse”

```
generate spouse= 1 if relat == 2
egen numsp = sum(spouse), by(hhid)
tabulate numsp
```

Use of egen – Some examples (cont.)

egen = rank() creates a variable assigning the rank of a variable. For example, with a variable `RPrice1`:

- `egen rank0 = rank(RPrice1), field` → assigns rank = 1 to the highest RPrice1, etc (no correction for ties; if 2 observations have the same RPrice1, they will have the same rank)

- `egen rank1 = rank(RPrice1), track` → assigns rank = 1 to the lowest RPrice1, with no correction for ties)
- `egen rank2 = rank(RPrice1), unique` → assigns rank = 1 to the lowest RPrice1; all observations have a different rank (random allocation in case of ties)

Producing deciles or quintiles using xtile

- The command `xtile` is used for example to generate quintiles or deciles based on the values of a variable

```
xtile newvar = exp [if] [in] [weight] [, xtile_options]
```

Calculating quintiles of per capita expenditure

```
xtile quinincome=Income, nq(5)
*Check
tab quinincome
```

5 quantiles of Income	Freq.	Percent	Cum.
1	885	20.66	20.66
2	847	19.77	40.43
3	841	19.63	60.06
4	869	20.28	80.35
5	842	19.65	100.00
Total	4,284	100.00	

Collapsing variables

- `collapse` converts the dataset in memory into a dataset of means, sums, medians, etc.

```
collapse clist [if] [in] [weight] [, options]
```

- Collapsing data files is a very useful tool, which needs to be well understood

- It will be used for example to produce data files at the boat level out of data files at the individual level

Use of the collapse command: examples

```
collapse (count) freq = Income ///
         (mean)  mean_income = Income ///
          (max)   max_income  = Income ///
          (min)   min_income  = Income, by(Boatname)
```

Use of duplicates drop

One way to keep only one observation per group (e.g., per household) is to use `collapse`. Another way is to remove all duplicates of the key variables using the `duplicates drop` command.

```
duplicates drop varlist [if] [in],
force
```

Generating dummy variables

- Dummy variables are variables with values 0 (false) and 1 (true). We already saw how to generate a dummy variable using the `generate` command, e.g.
  - The long way:

```
generate importer = 0
replace importer = 1 if exim == 1
```

  - The short way:

```
generate importer=exim==1
```

- When you have multiple categories, this method is tedious. You can use the `tabulate` command instead. For example:

```
tabulate exim, gen(ex_im)
```

  This will create dummy variables ex_im1, ex_im2, ex_im3, …, ex_imN (one dummy for each province)
- One additional option is to use the `xi` command

Producing tables

Tabulation

- We saw in a previous slide that frequency tables can easily be produced using the tabulate command (see also tab1 and tab2).
- For producing multi-dimension tables with summary statistics, we will use the table commands.
- Stata also provides the command tabstat for producing tables with summary statistics for a series of numeric variables.
- A user-contributed package (ado file) named tabout complement the Stata tabulation commands.

A note on copy/pasting tables

- To copy and paste tables from the Stata results window, use the copy table option, not copy. The formatting of the table will then be preserved, and cells will be properly distinguished when pasting to Excel.

Producing tables using command "tabulate"

- tabulate produces one-way or two-way tables. It can be used to produce tables showing frequencies in percentages. tab1 and tab2 will produce one-wan and two-way tables for multiple variables in one batch (tab2 will produce tables for all combinations of the specified variables).

  tabulate varname1 varname2 if in weight , options

- Example:

  tabulate Island month, row nofreq

  tabulate Island month, column nofreq

  tabulate Island month, cell nofreq

  tab1 Island month exim

  tab2 month exim madein// Produces 3 tables: month by exim, month by madein, exim by madein

Producing tables using command “table”

table calculates and displays tables of summary statistics.

table rowvar [colvar [supercolvar]] [if] [in] [weight] [, options]

Example:

use Malemarket2016merge, clear

table Island month, row col format(%9.0f)

table Island month, c(mean Income) row col format(%9.2f)

Producing tables using command "tabstat"

Example: Tables of summary statistics for two variables

tabstat QuantityKg Income, by(Island) stat(mean sd min max) nototal long

* Put the variables in row and the statistics in column

tabstat QuantityKg Income, by(Island) stat(mean sd min max) nototal col(stat)

Island	stats	Quanti~g	Income
Baarah	mean	1016.675	11570.83
	sd	1667.984	11603.35
	min	37.2	200
	max	7500	40000
Bilehfahi	mean	243.3333	3166.667
	sd	326.277	1091.177
	min	30	2000
	max	900	4500

Island	variable	mean	sd	min	max
Baarah	QuantityKg	1016.675	1667.984	37.2	7500
	Income	11570.83	11603.35	200	40000
Bilehfahi	QuantityKg	243.3333	326.277	30	900
	Income	3166.667	1091.177	2000	4500

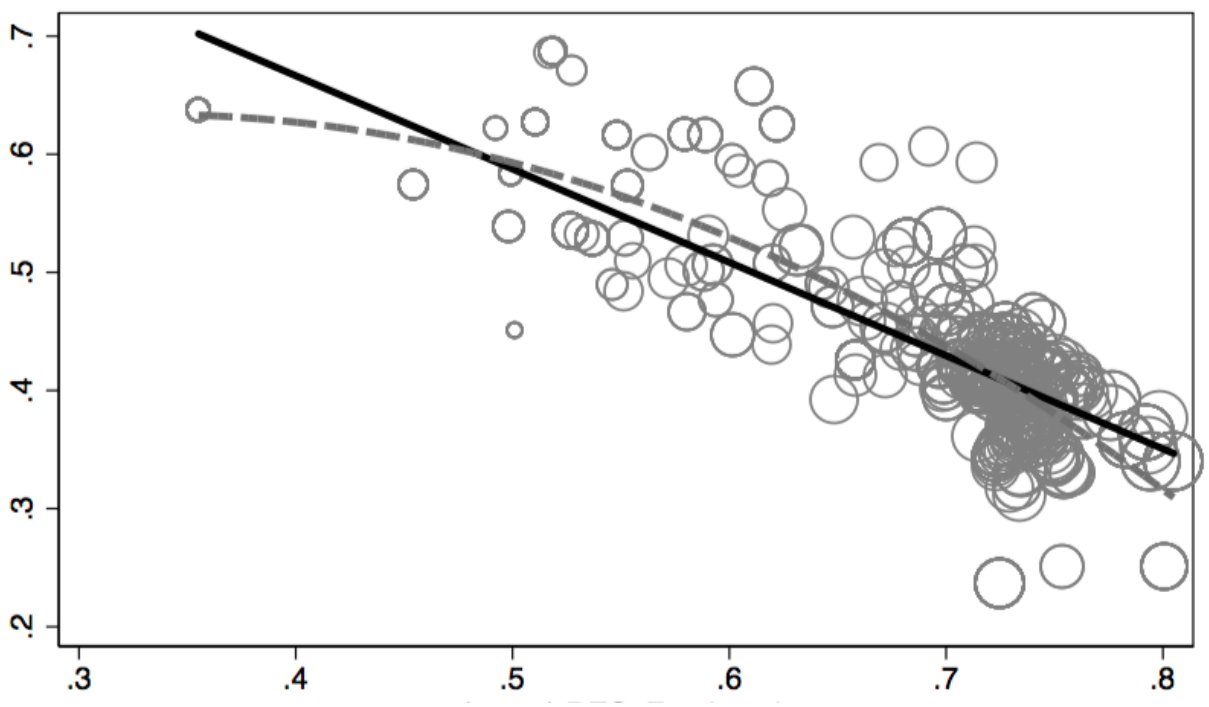
Producing tables using package "tabout"

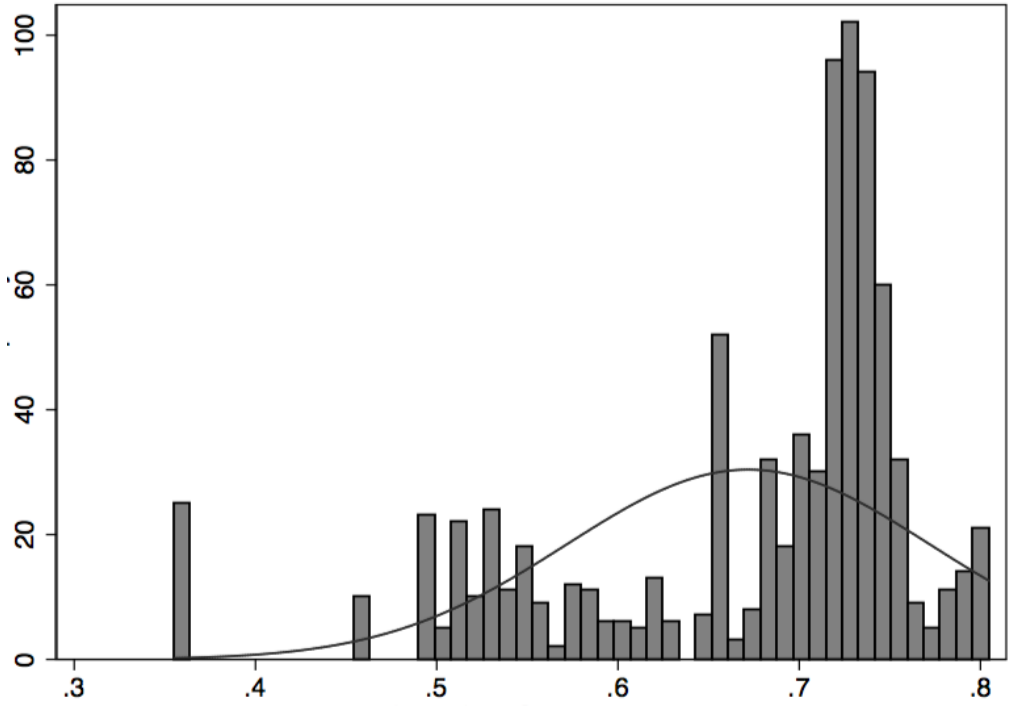
```
recode RPrice1 (0/200=1 "0-200") (201/400=2 "201-400") (401/600=3 "401-600") ///
(601/800=4 "601-800") (801/1000=5 "801-1000") ///
(1001/max=6 "1001 and above"), generate(price1group)

label variable price1group "RPrice1 Group"
tabout price1group exim using "tabout.xls", replace c(col) f(1) clab(Col_%) npos(col) style(xls)
```

	A	B	C	D	E	F
1		Main Activity				
2	RPrice1 Group	Importer	Exporter t	Exporter t	Total	N
3		Col %	Col %	Col %	Col %	
4	0-200	86.5	90.7	84.6	89.4	3,829
5	201-400	10.7	7.1	10	8.1	348
6	401-600	1.1	1.4	2.3	1.4	58
7	601-800	0.3	0.2	0.8	0.3	11
8	801-1000	0.7	0.2	1.5	0.4	17
9	1001 and above	0.7	0.4	0.8	0.5	21
10	Total	100	100	100	100	4,284
11						

Producing Graphs in Stata





KADARMANTO

BPS-STATISTICS INDONESIA

Graph

Stata has powerful graph capabilities.

Producing simple charts is very easy. But Stata offers many options that allows you to generate complex ones, and to customize about every aspect of your charts. A full manual is dedicated to it.

Tip: Use the menu-driven tools, which will produce the code for you.

We only show here some basic, common commands. Once you master these commands, read the Stata manual for more. Or visit Stata's on- line "Visual overview for creating graphs" at:

https://www.stata.com/support/faqs/graphics/gph/stata-graphs/

Stata Graphics

Stata Graphics in detail

Statistical Analysis-Regression

KADARMANTO

BPS-STATISTICS INDONESIA

Regressions in Stata

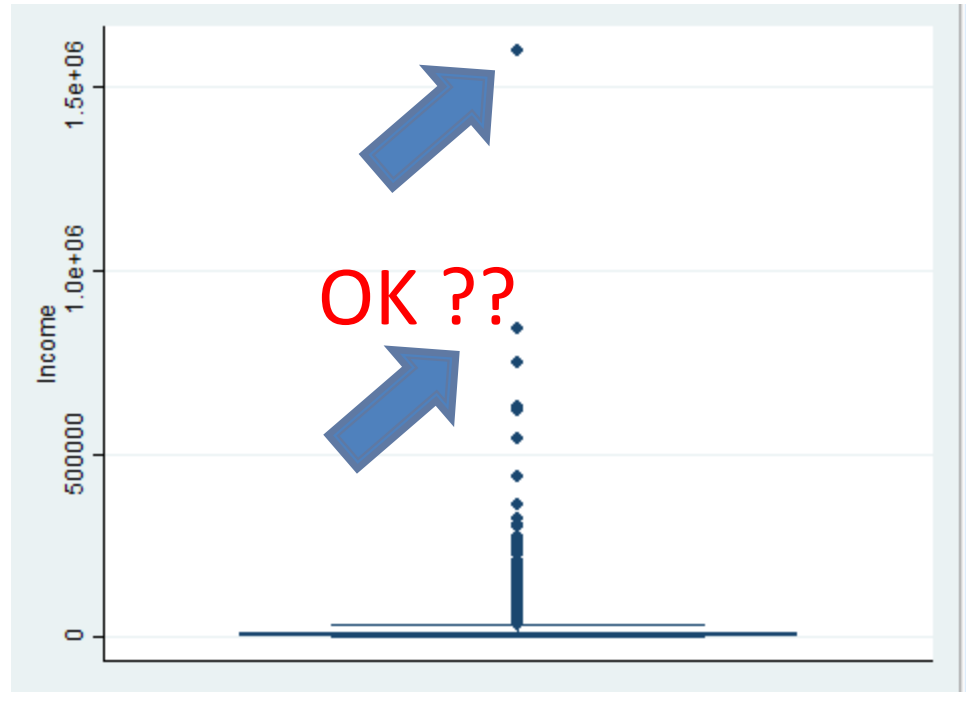
- Stata provides commands for running many types of regressions (linear, non-linear, logistic, probit, quantile, etc.)
- The most common types are the linear and the logistic models.
- The linear model used to predict the value of a continuous variable based on the value of one or more independent variables
- The logistic model used to predict the value of a binary variable (e.g., poor / non-poor) or a categorical variable with more than 2 categories (multinomial regression)

- Some specific commands allow taking complex survey designs into consideration (command svyreg).

A quick look at the data before regressing: outliers

- Before running a regression, make sure your data do not have outliers, invalid values, or a large number of missing cases. You can do that by producing various types of tables and charts. For example, before regressing the rent on dwelling characteristics, you could produce box plots of some variables.

```
graph box income
```



A quick look at the data before regressing: correlations among variables

- You can also look at the correlations of variables that you plan to use in the regression model, using command correlate

- Syntax:

```
correlate [varlist] [if] [in] [weight] [, correlate_options]
```

Example: correlation between Income, QuantityMeasured, QuantityKg, and Wsprice

```
. correlate Income QuantityMeasured QuantityKg Wsprice
(obs=4251)
```

	Income	Quanti~d	Quanti~g	Wsprice
Income	1.0000			
QuantityMe~d	0.6564	1.0000		
QuantityKg	0.5790	0.3794	1.0000	
Wsprice	0.0987	-0.1254	-0.0228	1.0000

The linear regression model

- All variables used in the model must be numeric (no string variables).
- The dependent variable must be a real-number variable (a continuous variable, for example "household income" or "rental value").
- The independent variables can be continuous or categorical variables. Prior to being used in a linear regression model, variables can - and in some cases must - be transformed, e.g.:
- the log value of continuous variables can be used instead of the original value (for dependent variables and predictors)
- categorical variables used as predictors must be transformed into dummy variables

Linear regression: regress, predict

Linear regression: **regress, predict**

- `regress` performs ordinary least-squares linear regression.
- The syntax is:

  regress depvar [indepvars] [if] [in] [weight] [, options]

- Once a model has been fit using the `regress` command, it can be applied to data to predict values of the dependent variable using the `predict` command. This command will make prediction using the latest regression model run by Stata.

  For a single-equation model, the syntax is:

  predict [type] newvar [if] [in] [, options]

Creating dummies for categorical variables

- The best option to convert categorical values into dummies is to use the xi command. The command only requires the choice of a prefix to indicate the dummy version of the variables to be converted. For example, to convert variables Island into dummies, with prefix "i.", one would simply type:

```
i.Island
```

- The xi command and the regression command can conveniently be combined into a single command, simply by preceding the regress command with xi as shown in the code example below.

Linear regression model: An example

```
xi:regress Income QuantityMeasured QuantityKg ///
Wsprice i.Island
predict pred_income
summarize income pred_income
```

Regression results (1/3)

```
.  xi:regress Income QuantityMeasured QuantityKg Wsprice ///
>       i.Island
i.Island          _IIsland_1-79       (_IIsland_1 for Island==Baarah omitted)
```

Source	SS	df	MS
Model	5.2810e+12	81	6.5197e+10
Residual	3.4818e+12	4169	835153471
Total	8.7627e+12	4250	2.0618e+09

Number of obs	4251
F(81, 4169)	78.07
Prob > F	0.0000
R-squared	0.6027
Adj R-squared	0.5949
Root MSE	28899

Income	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
QuantityMeasured	6.393724	.1268445	50.41	0.000	6.145041	6.642407
QuantityKg	7.859194	.2179045	36.07	0.000	7.431985	8.286403
Wsprice	95.12001	5.18423	18.35	0.000	84.95615	105.2839
_IIsland_2	4477.713	13191.74	0.34	0.734	-21385.13	30340.56
_IIsland_3	9077.659	14212.98	0.64	0.523	-18787.36	36942.68
_IIsland_4	-6617.935	10523.89	-0.63	0.529	-27250.37	14014.5
_IIsland_5	-8535.337	29504.23	-0.29	0.772	-66379.36	49308.69
_IIsland_6	5629.646	15608.81	0.36	0.718	-24971.94	36231.23
_IIsland_7	6144.104	21273.55	0.29	0.773	-35563.39	47851.6
_IIsland_8	11032.9	29499.11	0.37	0.708	-46801.08	68866.88
_IIsland_9	-10842.04	14209.21	-0.76	0.445	-38699.66	17015.59
_IIsland_10	7383.445	8043.348	0.92	0.359	-8385.805	23152.69
_IIsland_11	3231.911	6980.84	0.46	0.643	-10454.26	16918.08
_	_					
_IIsland_72	4240.268	21270	0.20	0.842	-37460.28	45940.81
_IIsland_73	8046.391	21274.84	0.38	0.705	-33663.64	49756.42
_IIsland_74	-1170.966	29498.93	-0.04	0.968	-59004.6	56662.66
_IIsland_75	158.0786	21274.06	0.01	0.994	-41550.42	41866.58
_IIsland_76	3091.39	21270.13	0.15	0.884	-38609.4	44792.18
_IIsland_77	-7723.369	29506.67	-0.26	0.794	-65572.17	50125.43
_IIsland_78	984.7915	15609.55	0.06	0.950	-29618.25	31587.83
_IIsland_79	-738.0196	15608.54	-0.05	0.962	-31339.08	29863.04
_cons	-12285.08	5922.196	-2.07	0.038	-23895.74	-674.4188

Regression results (3/3)

```
. summarize Income pred_income
```

Variable	Obs	Mean	Std. Dev.	Min	Max
Income	4286	16141.02	45231.28	0	1600000
pred_income	4251	16154.42	35250.34	-18997.28	1429934

Programming in Stata

KADARMANTO

BPS-STATISTICS INDONESIA

Programming

- Including comments in your programs is crucial !

- Commands can be used to describe the program, explain the purpose of some components, etc.
- There are four ways to include comments in a do-file.
  - Begin the line with a ' * '; Stata ignores such lines.
  - Place the comment in /* … */ delimiters.
  - Place the comment after two forward slashes, that is, //. Everything after the
  - // to the end of the current line is considered a comment.
  - Place the comment after three forward slashes, that is, ///. Everything after the /// to the end of the current line is considered a comment.

version, and set more off

- The first commands that you will include in your programs will often be version and set more off
- version indicates which version of Stata you are writing the program for (Stata evolves, and some commands can change)
- set more off is a parameter that controls the display of the results

- Example:

```
version 13
set more off
```

Logging the output

- In some cases, you may want to produce a log of the results.
- The log can be produced as a text file, or as a formatted Stata file.

- You have to provide in your program the filename and location where the log will be saved.

- At the beginning of your program, you will "open" the log file. You will close it at the end (note: you can set the log on and off within programs if you do not want to log all results).
- You can only have one log file open at a time.
- You can replace the content of an existing log file, or append to it.

Logging the output – Syntax and example

- Syntax to open a log:

```
log using filename [, append replace [text|smcl] name(logname)]
```

  - Example:

```
log using "C:/Stata_Maldives/Exercise_01.txt", replace text
```

- Syntax to close a log:

```
log close
```

- Syntax to temporarily suspend logging or resume logging:

```
log [off|on]
```

Long commands – The continuation line

- Some of your commands will be too long to fit on one line
- By default, Stata considers that each line contains one command
- If a command is provided on more than one line, you need to inform Stata about it. This can be done by:
  - Using a special character to inform Stata where the end of the command is #delimit (return to default by using #delimit cr)
  - Typing /// at the end of each line (except the last)

Record number and number of records

- Stata has two macro variables that you can use any time in your programs
- One is named _N and indicates the total number of observations in the file

- The other one indicates the sequential number of each observation in the data file and is named n

Macros

- In many Stata programs, you will make use of macro variables. These are variables that are not saved in data files, but are available for use during the execution of your programs.

- Macros can be local (in which case they only exist within a specific do file) or global (in which case they can be used across programs).

- You create a macro variable simply by declaring its type and giving it a value (numeric or string), e.g.,

```
local i = 1
global myfolder = "C:₩Stata_Maldives"
```

Macros

- Once a macro has been created and contains some value or text, you can use it in your programs.
- To refer to a local variables ina program, put the name of the macro between quotes as follows `macroname' . For global macros, put the character $ before the name (e.g., $macroname)
- Example:

```
local i = 10
display "The value of my local macro is " `i'
global myfolder = "C:₩Stata_Maldives"
display "The content of my global macro is " $myfolder
```

Temporary files

- In some programs, you may want to generate data files that are needed only for the execution of that program. You can create such temporary files using the `tempfile` command. Temporary files are automatically erased at completion of the program's execution.

- You can create multiple temporary files in a program.

- You create them by giving them a name before putting content in them.
- Example: to create 2 temporary files named t0 and t1, type: `tempfile t0 t1`

- The command tempfile can be put anywhere in your program.

- To refer to a tempfile, enclose its name into single quotes (like local macros).
- Example:

```
save `t0', replace
```

Stored results

- Commands that return an output often store results in memory, which can be used in programs
- For example, in addition to displaying summary statistics on screen, the command summarize stores the following results

r(N)	number of observations	r(p1)	1st percentile (detail only)
r(mean)	mean	r(p5)	5th percentile (detail only)
r(skewness)	skewness (detail only)	r(p10)	10th percentile (detail only)
r(min)	minimum	r(p25)	25th percentile (detail only)
r(max)	maximum	r(p50)	50th percentile (detail only)
r(sum_w)	sum of the weights	r(p75)	75th percentile (detail only)
r(Var)	variance	r(p90)	90th percentile (detail only)
r(kurtosis)	kurtosis (detail only)	r(p95)	95th percentile (detail only)
r(sum)	sum of variable	r(p99)	99th percentile (detail only)
r(sd)	standard deviation		

- The command `mean` stores results in various e( ) macros/scalars/matrices (see help of `mean` command)
- Note: some packages (e.g., `poverty`) store results in global macro variables.

Use of stored results: An example

- Commands that return an output often store results in memory, which can be used in programs
- See the command's help for a list of stored results (when available)
- For example, in addition to displaying summary statistics on screen, the command summarize stores the following results

r(N)	number of observations	r(p1)	1st percentile (detail only)
r(mean)	mean	r(p5)	5th percentile (detail only)
r(skewness)	skewness (detail only)	r(p10)	10th percentile (detail only)
r(min)	minimum	r(p25)	25th percentile (detail only)
r(max)	maximum	r(p50)	50th percentile (detail only)
r(sum_w)	sum of the weights	r(p75)	75th percentile (detail only)
r(Var)	variance	r(p90)	90th percentile (detail only)
r(kurtosis)	kurtosis (detail only)	r(p95)	95th percentile (detail only)
r(sum)	sum of variable	r(p99)	99th percentile (detail only)
r(sd)	standard deviation		

The display command

- `display` displays strings and values of scalar expressions. It produces output from the programs that you write. It can be used for example to display a result of a command, or the value of a macro.

- Example 1:

```
summarize Income   // Produce summary stats of variable
Income
display "Variable Income has a mean of " r(mean) " and a
max of " r(max)
```

- Example 2:

```
display "Today is the: " c(current_date) //
c(current_date) = the system date
```

Loops

- Many programs will contain commands or sets of commands that need to be repeated (e.g., you may need to calculate values for each year in a range of years).

- Stata provides various methods for looping or repeating commands in a do-file.

- Depending on the purpose of the loop, you may want to chose one of the methods over another one (in some cases, more than one method may achieve the same result, but one may be more "elegant" or efficient than another one).

Loops using “while”

- A first option to create a loop in a do-file is to use the while command.

- Stata will repeat the commands specified in the loop as long as the while condition is met.

- Typically, this will be used when the set of commands must be repeated a fixed number of times (e.g. 5 loops).

Loops using “while” - Example

- We run a command displaying the value of calendar year, from 2000 to 2020, by increment of 5.

```
local year = 2000
while `year' <= 2020 {
   display "Calendar year is now: " `year'
   local year = `year' + 5
}
```

Loops using “forvalues”

Another way of achieving a loop through numeric values is top use “forvalues”.

```
forvalues lname = range {
    commands referring to `lname'
}
```

where range is

- #1(#d)#2 meaning #1 to #2 in steps of #d
- #1/#2 meaning #1 to #2 in steps of 1
- #1 #t to #2 meaning #1 to #2 in steps of #t - #1
- #1 #t : #2 meaning #1 to #2 in steps of #t - #1

Loops using “foreach”

foreach is used in conjunction with strings.

```
foreach country in KIR FSM FJI {
   display "The selected country is " "`country'"
}
```

This command can be used with variable names, numbers, or any string of text.

Loops using “levelsof”

- levelsof displays a sorted list of the distinct values of a categorical variable. Using this command, you can generate a macro containing a list of these values, and use this list to loop through the values.

- Example:

```
levelsof ethnicgrp, local(ethnic)
foreach l of local ethnic {
… some commands to be run for each value of ethnic


}
```

Branching

- We may want to execute some commands when a particular condition is met, and another set of commands when the condition is not met. This is done by "branching" using the "if" and "else" commands. The implementation in Stata is as follows:

```
if [condition] {
… execute these commands …
}
else {
… execute these other commands …
}
```

- Notice the use of curly brackets { and }. The set of commands to be implemented under each condition must be listed in their own set of brackets.

Quietly or noisily executing commands

- In some cases, you may want to run a command but not show the terminal output. This can be done using the quietly command. Syntax: quietly [:] command
- Example: quietly regress pce province industry hhsize
  - No output is presented, but the e() results are available.

- Note: You can combine quietly with { } to quietly run a block of commands (and use noisily to make a command within this block run non-quietly if needed).

Debugging a program

Your program may crash out half-way through for some reason. For example, if you are trying to create a new variable called age but there is already a variable named age.

```
use "use Malemarket2016merge, clear", clear
generate Income = 10
variable Income already defined
```

When the program is simple, detecting the cause of the problem is easy. With complex programs, it is not always so obvious. The set trace command, which traces the execution of the program line-by-line, may help identify the problem.

Working With CSV Excel Files

KADARMANTO

BPS-STATISTICS INDONESIA

Importing data from a CSV file

Use `import delimited` to import data from a CSV file. You have the option to treat the first row of CSV data as Stata variable names, and to select a specific range of rows/columns.

Syntax: **import delimited** [*varlist*] *filename* [, *options*]

Example:

```
*Importing a CSV file, where the first row contains variable
 names
import delimited "household.csv", clear
*We do the same, but for a selection of columns and rows of
 the CSV file
*(we keep the first 5 variables, and the top 50 observations)
import delimited "household.csv", rowrange(1:50) colrange(1:5)
clear
```

Importing data from an Excel worksheet

- Use import excel to import any worksheet (or a custom cell range) from an XLS or XLSX file. You have the option to treat the first row of Excel data as Stata variable names.

- Syntax:

```
import excel [using] filename [, import_excel_options]
```

- Example:

```
import excel "household.csv", clear
```

- (see Stata manual for more options)

Reading specific cells from an Excel worksheet

- You can read specific cells from an Excel worksheet and save the values as macro variables for use in Stata programs. For example:

```
import excel using "C:₩poverty_lines.xlsx", cellrange(B1:C1) clear local ctry = B
local year = C
```

Saving a Stata data file in Excel format

- Use export excel to save your Stata data file (all variables or a subset) in an Excel sheet. You have the option to replace an entire workbook, or to save the data as a new worksheet in an existing workbook. You can save the Stata variable names or variable labels as first row of the worksheet. You can chose to export the values or the corresponding value labels.

- Syntax:

  export excel [using] filename [if] [in] [, export_excel_options]

- or (to export only a subset of variables)

  export excel [varlist] using filename [if] [in] [, export_excel_options]

Saving values in Excel sheets

- To save the results of Stata calculations in specific cells of an Excel file, you will use putexcel. The command putexcel set indicates the Excel file to be used and some formatting options. The command putexcel writes values (from a Stata macro or matrix) in the Excel file.
- For example:

```
putexcel set "poverty_lines", sheet("Sheet1") modify keepcellformat
putexcel B27 = matrix(WI) // B27 = top right corner of matrix
putexcel F13 = ("$S_DATE")
putexcel F14 = ("$S_TIME")
putexcel K20 = (`poverty_headcount')
```

Interacting with Excel: an example

```
* We read values of poverty lines in Excel, calculate poverty in Stata, and save results in
Excel.
set more off
cd "C:\Stata_Data"

local myXLS = "Test_poverty_lines.xlsx" // Excel file containing poverty lines

putexcel set "`myXLS'", modify // Will save results in that same file

forvalues i = 10(1)18 {       // Poverty lines are stored in cells B10 to B18
        import excel using "`myXLS'", cellrange(B`i') clear  // Read poverty line value

        local pline = B  // Store it in a macro
        use "household.dta", clear
        gen pce = tot exp/ hhsize
        poverty pce [aweight = wgtpop], line(`pline') all // Calculate poverty indic.
        putexcel C`i' = ($S_6) // Package poverty saves output in global macros
        putexcel D`i' = ($S_8) // We save two of the results in Excel (cols C and D)
}
putexcel C6 = ("$S_DATE") // We save the date in cell C6
```

SPECIFIC COMMANDS FOR SURVEY DATA TABULATION AND ANALYSIS

KADARMANTO

BPS-STATISTICS INDONESIA

Defining the survey design

- Sample design can affect the standard errors from results of statistical analyses. Analysis must take survey design features into account.
- To do so, we must issue the svyset command to tell Stata about the sample design. You use svyset to designate variables that contain information about the survey design, such as the sampling units and weights.
- Once this command has been issued, you can use the svy: prefix before each command.

Defining the survey design - Syntax

- For single-stage design:

```
svyset [psu] [weight] [, design_options options]
```

- For multiple-stage design

```
svyset psu [weight] [, design_options] [|| ssu , design_options] ... [options]
```

Using svy: commands

- After svyset, you can use many commands with prefix svy: and you will get more accurate results.
- Some commands that can use svy:
  - Descriptive statistics: mean
  - Estimate means proportion: proportion
  - Estimate proportions ratio: ratio
  - Estimate ratios total: total
  - Linear regression: regress

THANK YOU