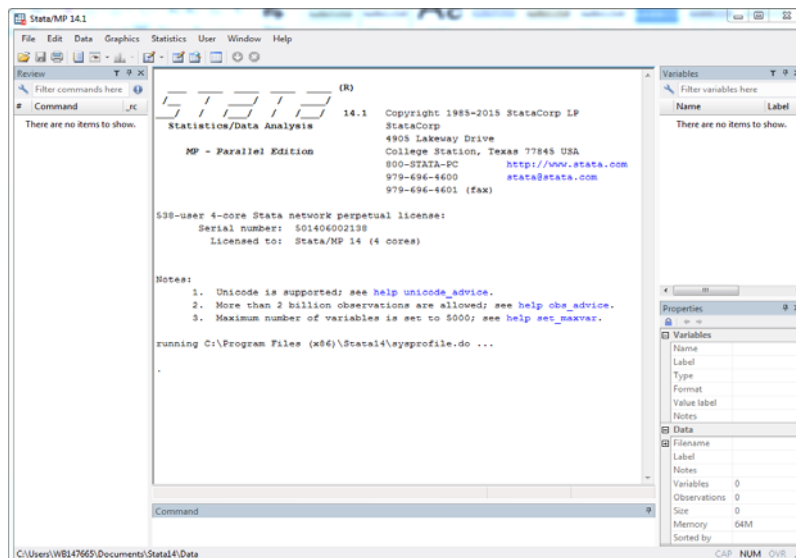


# An Introduction to Stata for Survey Data Analysis

Olivier Dupriez, World Bank

March 2017

When you launch Stata ...

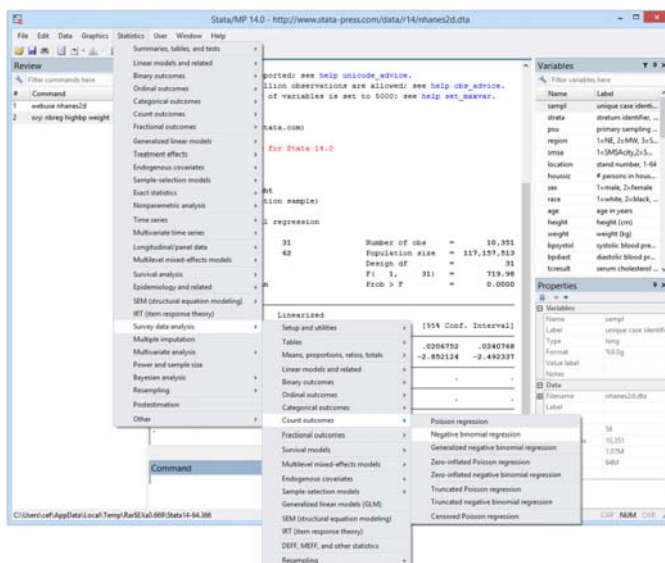


# Three ways of executing Stata commands

- Menus and dialogs (the Graphical User Interface)
- The command line
- Writing programs (do files)

3

## 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

4

## 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



5

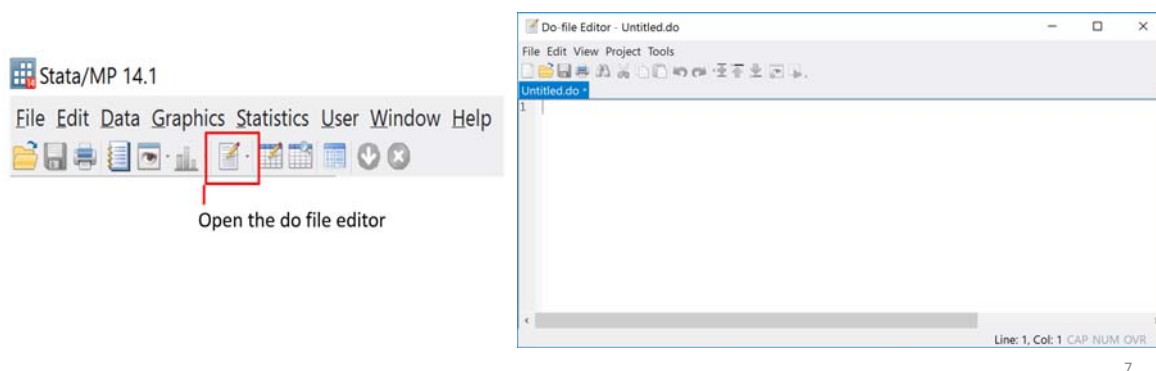
## Option 3: writing programs (do-files)

- 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.

6

# 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

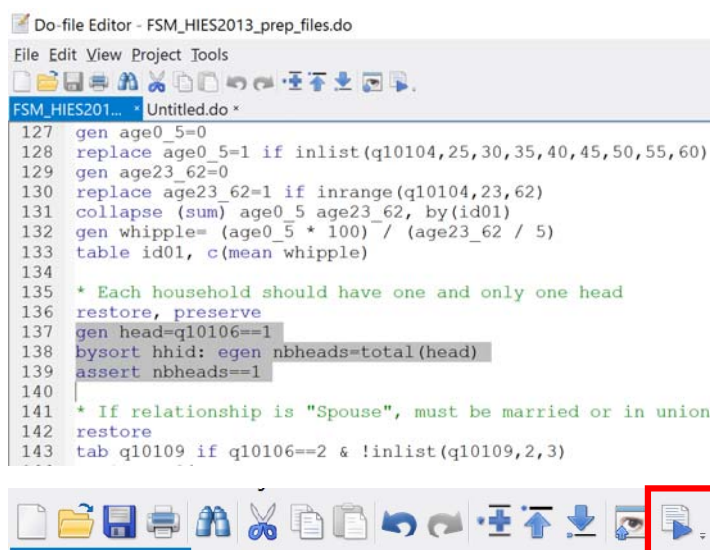


# Executing commands from the do-file editor

Type your program in the do-file editor →

Select (highlight) the commands you want to execute →

Click on the EXECUTE icon



# Ado files (contributed packages)

9

## 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`

10

## 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 (see section on maps in this presentation)
  - shp2dta, spmap
- For poverty and inequality analysis
  - povdeco, poverty, ineqdeco, inequal7, glorenz
- For you ?
  - Find out using findit

11

Before we start...

12

## 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

13

## 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`

14

## 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

15

## 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.

16



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

17

## 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

18

# Examples

## Syntax

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



Describe data in memory

```
describe [varlist] [, memory_options]
```



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



19

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

20

## 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

21

## 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”)

22

# Working on data files

23

## The structure of a Stata data file

Variables

	iid	sex	age	relation	marital	rururb	wta_hh	PROV	DIST
→ 1	05	2	12	3	2	1	2507.01	001	006
→ 2	08	2	2	5	2	1	2507.01	001	006
→ 3	02	2	42	2	1	1	2507.01	001	006
→ 4	01	1	54	1	1	1	2507.01	001	006
→ 5	03	2	6	3	2	1	2507.01	001	006
→ 6	04	2	10	3	2	1	2507.01	001	006
→ 7	06	1	32	4	1	1	2507.01	001	006
→ 8	07	2	22	3	1	1	2507.01	001	006
→ 9	02	2	37	2	1	1	2507.01	001	036

24

## 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_Fiji\Data\household.dta", clear
or
cd "C:\Stata_Fiji\Data"
use "household.dta", clear
```

25

## Sorting a data file - **sort**

**Syntax:**

```
sort varlist
```

**Example:**

```
sort hhid totexp
```

26

## 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
```

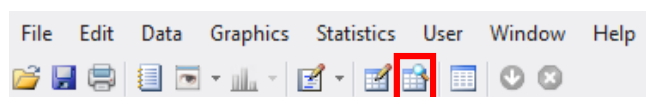
27

## 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 "household.dta", replace`
- Files saved in Stata 14 will not be readable with previous versions of the software. If you need to save data in an older format, use option `saveold`.

28

## Browsing (viewing) the data



	hhid	province	district	ea	urbrur	hhno	hhsiz	dwelling
1	532	West	11	104	Urban	0532	4	Detached House
2	380	East	8	74	Urban	0380	3	Detached House
3	950	South	20	188	Urban	0950	6	Detached House
4	533	West	11	104	Urban	0533	1	Detached House
5	568	West	12	112	Urban	0568	3	Detached House
6	865	South	18	171	Urban	0865	7	Detached House
7	746	South	16	148	Urban	0746	7	Detached House
8	806	South	17	160	Urban	0806	4	Detached House

29

## Inspecting data files – File description

describe produces a summary of the dataset in memory

`describe [varlist] [, memory_options]`

```

Contains data from C:\Stata_Manual\Data\household.dta
  obs:      1,000      Data file distributed with the Stata m
  vars:      29        17 Mar 2017 11:49
  size:     75,000     (_dta has notes)

-----
variable name  storage  display  value  variable label
               type   format   label
-----
hhid           int     %9.0g
province       byte    %10.0g  province
district       byte    %9.0g
ea             int     %9.0g
urbrur         byte    %9.0g  urbrur
hhno           str4    %9s
hhsiz         byte    %16.0g
dwelling       byte    %23.0g  dwelling
water          byte    %26.0g  water
toilet         byte    %16.0g  toilet
wall           byte    %16.0g  wall
  
```

30

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

31

## 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 "C:\Stata_Fiji\individual.dta", clear
count // Counting all observations in data file
count if sex == 1 // Counting males
count if sex == 2 & age > 12 & age < . // Counting females aged 12 +
```

32



## 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 ID, province and sex for people aged 25 or 30

```
list hhid province sex if age == 25 | age == 30
```

33

## 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 marital  
(marital status) →
```

marital: Marital status		Number of Observations		
		Total	Integers	Nonintegers
#	#	-	-	-
#	#	-	-	-
#	#	3,926	3,926	-
#	#	-	-	-
#	#	Total	3,926	3,926
#	#	Missing	-	-
1	4	3,926		
(4 unique values)				

34

## 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 sex-literate // variables sex to literate  
codebook hh* // all variables with name starting with hh
```

35

## Appending data files

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

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

36

# 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

37

# Hierarchical structure and keys

Individual-level data (1 or > rows per household (1 per household member))

PROVINCE	DISTRICT	VILLAGE	EA	HH_NUM	PERSON_NO	AGE	SEX	RELATION	MARITAL	EDUCATION	OCCUPATION
01	01	005	10	008	01	45	1	1	2	4	21
01	01	005	10	008	02	42	2	2	2	2	10
01	01	005	10	009	01	38	1	1	2	3	03
01	01	005	10	009	02	34	2	2	2	3	03
01	01	005	10	009	03	8	2	3	1	1	.
01	01	005	10	009	04	5	1	3	1	0	.
02	01	04	05	103	01	75	1	1	2	0	15
02	01	04	05	103	02	70	2	2	2	1	10

Household-level data (1 row per household)

PROVINCE	DISTRICT	VILLAGE	EA	HH_NUM	ROOF	WALLS	WATER
01	01	005	10	008	1	3	2
01	01	005	10	009	2	3	1
02	01	04	05	103	4	2	2

38

# 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)

Individual-level data (1 or > rows per household (1 per household member))

PROVINCE	DISTRICT	VILLAGE	EA	HH_NUM	PERSON_NO	AGE	SEX	RELATION	MARITAL	EDUCATION	OCCUPATION
01	01	005	10	008	01	45	1	1	2	4	21
01	01	005	10	008	02	42	2	2	2	2	10
01	01	005	10	009	01	18	1	1	2	3	03
01	01	005	10	009	02	14	2	2	2	3	08
01	01	005	10	009	03	8	2	3	1	1	...
01	02	005	10	009	04	5	1	3	1	0	...
02	01	04	05	103	01	75	1	1	2	0	15
02	01	04	05	103	02	70	2	2	2	1	10

Household-level data (1 row per household)

PROVINCE	DISTRICT	VILLAGE	EA	HH_NUM	ROOF	WALLS	WATER
01	01	005	10	008	1	3	2
01	01	005	10	009	2	1	1
02	01	04	05	103	4	2	2

Data on household consumption by product (1 row per product per household)

PROVINCE	DISTRICT	VILLAGE	EA	HH_NUM	PRODUCT	QUANTITY	UNIT	VALUE
01	01	005	10	008	101	6	KG	125
01	01	005	10	008	102	20	LIT	140
01	01	005	10	008	106	2	BAG	35
01	01	005	10	008	108	1	PRICE	20
01	01	005	10	008	201	4	KG	200
01	01	005	10	008	204	60	BUNDLE	1200
01	01	005	10	008	205	2	KG	60

Data on household consumption by product (1 row per product per household)

PROVINCE	DISTRICT	VILLAGE	EA	HH_NUM	MARKET	PRICE	QUANTITY	UNIT
01	01	005	10	101	1	60	1	KG
01	01	005	10	101	2	55	1	KG
01	01	005	10	101	3	54	1	KG
01	01	005	10	102	1	80	1	KG
01	01	005	10	102	2	78	1	KG
01	01	005	10	103	1	100	10	LIT
01	01	005	10	103	1	120	10	LIT
01	01	005	10	103	1	120	10	LIT

39

# 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.

40

## 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.

41

## 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.

42

## 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:

- 1 → observation appeared in master file only
- 2 → observation appeared in “using” file only
- 3 → match: observation appeared in both data files

43

## 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.

44

## 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
```

45

## Merging data files – Examples

- One-to-one merge on specified key variables (FSM HIES 2013 data files)

```
use "household.dta", clear  
merge 1:1 hhid using "dwelling.dta"  
tab _merge
```

- Many-to-one merge on specified key variables

```
use "individual.dta", clear  
merge m:1 hhid using "household.dta"  
tab _merge
```

46

# Working with variables

47

## 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

48



## 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

49

## 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)

50

## 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 sex "Sex"  
label define gender 1 "Male" 2 "Female"  
label value sex gender
```

51

## Modifying and eliminating value labels

**To add or modify value labels:**

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

Example:

```
label define sex 1 "Male" 2 "Girl"  
label define sex 2 "Female", modify  
label define sex 3 "Unknown", add
```

**To eliminate value labels:**

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

Example:

```
label drop sex
```

52

## 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]
```

A useful option is "no" (no label)

Examples:

```
use "individual.dta", clear
tabulate marital
tabulate marital, nol
tabulate marital, sort
tabulate marital if sex == 1
```

53

## 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

54

# 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)		

55

# 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

56

## 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 age is missing, type:  
`count if age >=.`
  - To create a new variable and assign value 1 if age is greater than 65, type:  
`generate elderly = 1 if age > 65 & age < .`

57

## Generating variables – Some examples

```
generate X = 1
generate X = age if age > 20
generate X = ln(tot_exp)
generate X = .
generate X = "Fiji" (→ create a string variable)
```

Note: if one component of the operation is missing, the result is missing (e.g., `1 + . = .`)

A shortcut to create a dummy variable (values 0 and 1):

```
generate poor = pcexp > povline
→ Will have value 1 if pcexp > povline, and 0 otherwise
```

This does the same as:

```
generate poor = 0
replace poor = 1 if pcexp > povline
```

58

## Recoding variables

Syntax:

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

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

59

## Recoding variables – Example

Creating age groups by recoding age

```
recode age (0/4 = 0) (5/9 = 5) (10/14 = 14) ...  
(90/max=90), generate(agegroup)
```

60

## The commands **encode** and **decode**

- 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.

Example: `encode prov, generate(province)`

- Use `decode` to do the opposite. Stata will generate a new (string) variable containing the label of the numeric variable

Example: `decode sex , generate(gender)`

61

## **inlist** and **inrange**

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

Examples of use:

```
generate region = 1 if inlist(province,3,4,7)
```

```
generate reprodw = 1 if inrange(age,12,49) & sex==2
```

62

## 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.)

63

## 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.
```

64



## Renaming variables

`rename` changes the name of an existing variable

Example: `rename age age_years`

Stata provides some functions for renaming groups of variables;

see `help rename group`

65

## 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 hhid q1*`

66

## 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 age == .`
  - `keep if age < .`

67

## 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:

```
use "individual.dta", clear
describe
order hhid indid ea
describe
```

hhid	int	%9.0g		Household ID	hhid	int	%9.0g	Household ID
indid	byte	%16.0g		Individual ID	indid	byte	%16.0g	Individual ID
province	byte	%10.0g	province	Province	ea	int	%9.0g	Enumeration area
district	byte	%9.0g		District	province	byte	%10.0g	Province
ea	int	%9.0g		Enumeration area	district	byte	%9.0g	District
urbrur	byte	%9.0g	urbrur	Rural/urban	urbrur	byte	%9.0g	Rural/urban

68

## 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.

69

## 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
```

70

## Use of `egen` – Some examples (cont.)

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

- `egen rank0 = rank(tot_exp), field` → assigns rank = 1 to the highest income, etc (no correction for ties; if 2 observations have the same income, they will have the same rank)
- `egen rank1 = rank(tot_exp), track` → assigns rank = 1 to the lowest income, with no correction for ties)
- `egen rank2 = rank(tot_exp), unique` → assigns rank = 1 to the lowest income; all observations have a different rank (random allocation in case of ties)

71

## 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 (e.g., quintiles of per capita expenditure - pce)

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

- Depending on the weight we use in a household survey, we would generate quintiles of households (20% of households in each quintile) or quintiles of population (20% of individuals in each quintile)
  - Use household sample weight for household quintiles
  - Create a population weight = household weight \* household size to obtain population quintiles

72

## Calculating quintiles of per capita expenditure

```
use "household.dta", clear
* To have population (not hhld) quintiles, we use population weight
generate pcexp = tot_exp / hhsize
generate popweight = hhwt * hhsize
xtile quinqpop= pcexp [pweight= popweight], nq(5)
* Check
tab quinqpop [aweight = popweight]
```

5 quintiles of pcexp	Freq.	Percent	Cum.
1	200.518997	20.05	20.05
2	200.135679	20.01	40.07
3	199.604612	19.96	60.03
4	201.964547	20.20	80.22
5	197.776166	19.78	100.00
Total	1,000	100.00	

73

## 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 household level out of data files at the individual level

74

## Use of the `collapse` command: examples

```
* Calculating household size and max/mean age from demographic data
use "individual.dta", clear // Data with demographic information
collapse (count) hh_size = sex /// Or any variable with no missing
         (mean) mean_age = age /// q10104 = age in years
         (max) max_age = age, by(hhid)
```

```
* Producing a file with number of hhlds and population by province
* Sum hhld and population sampling weights, by province
use "household.dta", clear // A file with observations at hhld level
generate popweight = hhwgt * hhsize
collapse (sum) hhwght popweight, by(province)
```

**Note:** `egen` can be used to generate the same variables without generating new files

75

## 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
```

76

# 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 male = 0  
replace male = 1 if sex == 1
```

- The short way:

```
generate male = sex==1
```

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

```
tabulate province, gen(prov)
```

This will create dummy variables `prov1`, `prov2`, `prov3`, ..., `provN` (one dummy for each province)

- One additional option is to use the `xi` command (see slides on regression).

77

# Producing tables

78

## 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` complements the Stata tabulation commands.

79

## 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.

80



## 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-way 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:

```
use "individual.dta", clear
tabulate province marital [aweight=hhwgt], row nofreq
tabulate province marital [aweight=hhwgt], column nofreq
tabulate province marital [aweight=hhwgt], cell nofreq
tab1 province sex relat marital
tab2 sex relat marital // Produces 3 tables: sex by relat, sex by marital, relat by marital
```

81

## Producing tables using command “table”

table calculates and displays tables of summary statistics.

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

### Example:

```
use "individual.dta", clear
table province marital [pweight=hhwgt], row col format(%9.0f)
table province marital [pweight=hhwgt], row col format(%9.0f)
table province marital [pweight=hhwgt], c(mean age) row col format(%9.2f)
```

82

# Producing tables using command “tabstat”

Example: Tables of summary statistics for two variables

```
use "household.dta", clear
tabstat tot_food tot_exp, by(province) stat(mean sd min max) nototal long

* Put the variables in row and the statistics in column
tabstat tot_food tot_exp, by(province) stat(mean sd min max) nototal col(stat)
```

province	stats	tot_food	tot_exp	province	variable	mean	sd	min	max
North	mean	71700.21	115795.2	North	tot_food	71700.21	40271.75	9592.185	232972.3
	sd	40271.75	83606		tot_exp	115795.2	83606	18749.05	554781.9
	min	9592.185	18749.05	East	tot_food	56311.03	35104.43	5872.188	221348.2
	max	232972.3	554781.9		tot_exp	101179.2	87828.74	10495.57	944444.3
East	mean	56311.03	101179.2						
	sd	35104.43	87828.74						

83

# Producing tables using package “about”

```
use "C:\Stata_Manual\Data\individual.dta", clear

* Recode age into age group
recode age (0/9=1 "0 - 9 years") (10/19=2 "10 - 19 years") ///
(20/29 =3 "20 - 29 years") (30/39=4 "30 - 39 years") ///
(40/49=5 "40 - 49 years") (50/59=6 "50 - 59 years") ///
(60/69=7 "60 - 69 years") (70/79=8 "70 - 79 years")
(80/max=9 "80 and above"), generate(agegroup)

label variable agegroup "Age group"

tabout agegroup urbrur using "C:\Stata_Manual\Data\table1.xls", ///
c(col) f(1) clab(Col_%) npos(col) style(xls) replace
```

	A	B	C	D	E
1	Rural/urban				
2	Age group	Urban	Rural	Total	N
3		Col %	Col %	Col %	
4	0 - 9 yea	19.7	21.2	20.5	805
5	10 - 19 yea	17.9	20.1	19.1	751
6	20 - 29 yea	15	13.2	14	550
7	30 - 39 yea	16.9	16.3	16.6	651
8	40 - 49 yea	14.4	11.1	12.6	494
9	50 - 59 yea	8.1	9.3	8.8	345
10	60 - 69 yea	5	5	5	197
11	70 - 79 yea	2.2	2.7	2.5	98
12	80 and abo	0.8	1	0.9	35
13	Total	100	100	100	3,926

84

# Producing graphs

85

## Graphs

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:

<http://www.stata.com/support/faqs/graphics/gph/stata-graphs/>

86

## Bar graphs

Bar graphs compare quantities in different categories of a variable.

```
graph bar yvars [if] [in] [weight] [, options]
```

where `yvars` is a list of variables.

The command has many options, and also allows to graph summary statistics of the variables (mean, median, percentiles, min, max, etc.)

87

## Bar graphs – An example

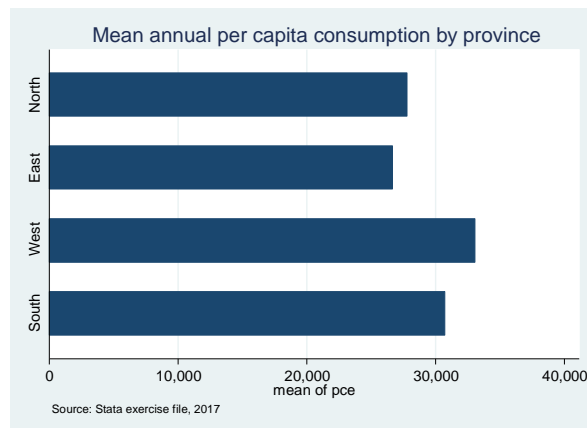
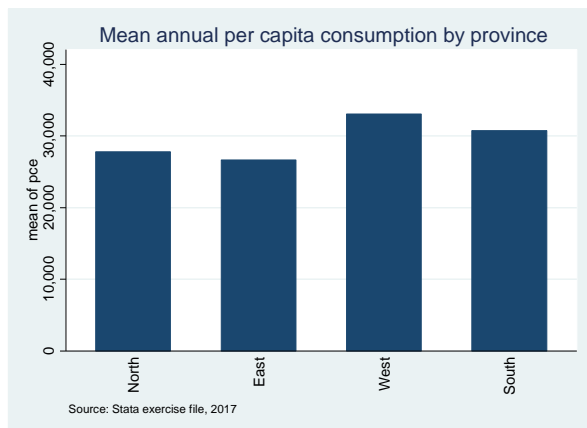
Mean per capita consumption by province

```
use "household.dta", clear
generate pce = tot_exp / hhsize
generate popweight = hgweight * hhsize
graph bar (mean) pce if pce <. [pweight = wgtpop],
over(province, label(angle(ninety))) title("Mean annual per
capita consumption by province") note("Source: Stata exercise
file, 2017")
```

\* Use "hbar" instead of "bar" for horizontal chart

88

## Bar graphs – An example



Same command, but with `hbar` instead of `bar`

89

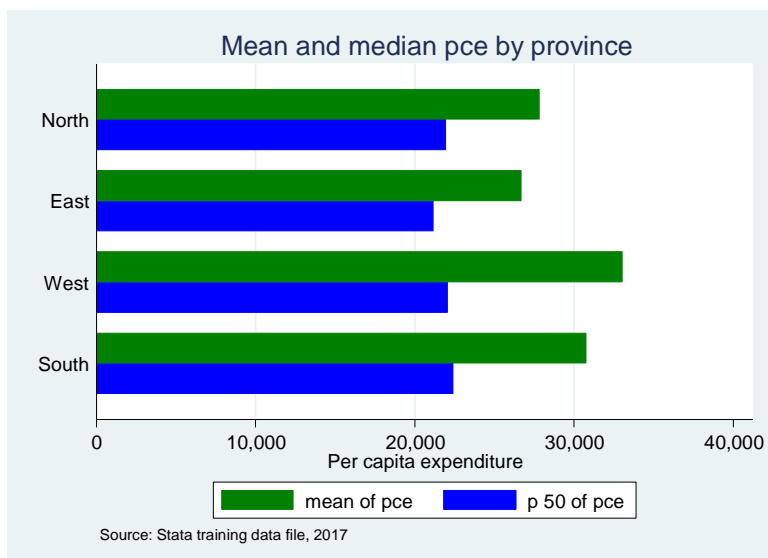
## Bar graphs – Another example

Mean and median per capita consumption by State (2 variables)

```
use "household.dta", clear
generate pce = tot_exp / hhsize
graph hbar (mean) pce (median) pce [pweight = wgtpop],
over(province) title("Mean and median pce by province")
ytitle("Per capita expenditure") note("Source: Stata
training data file, 2017") bar(1, color(green)) bar(2,
color(blue))
```

90

## Bar graphs – Another example



91

## Pie charts

- Syntax:

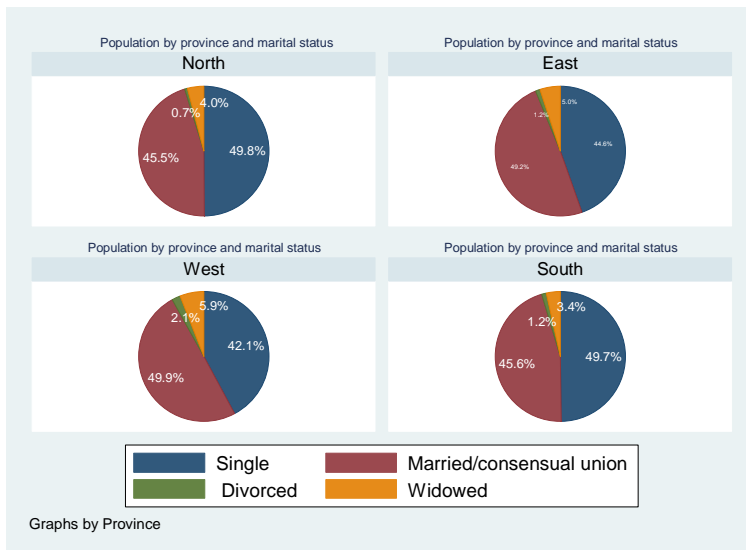
```
graph pie varlist [if] [in] [weight] [, options]
```

- Example:

```
use "individual.dta", clear
graph pie [pweight = hhwt], over(marital) plabel(_all
percent, color(white) format(%9.1f)) cw by(,
legend(on)) by(province, title(Population by province
and marital status))
```

92

# Pie charts



Notice that the title is repeated on top of each chart; this title should better be displayed only once on top of all pies, as it applies to all.

This can be done simply by including the title instruction within the "by" option. See the example provided in next slides for "dot charts".

93

# Line charts

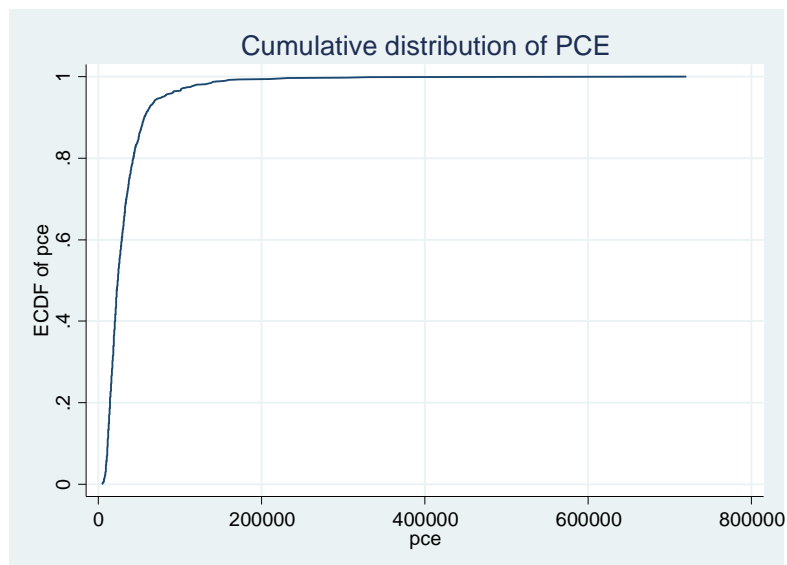
Example:

```
use "household.dta", clear
generate pce = tot_exp / hhsize
cumul pce, gen(cum)
sort cum
line cum pce, ylab(, grid) xlab(, grid) title("Cumulative
distribution of PCE")
```

Note: `cumul` creates a new variable, defined as the empirical cumulative distribution function of a numeric variable.

94

## Line charts



95

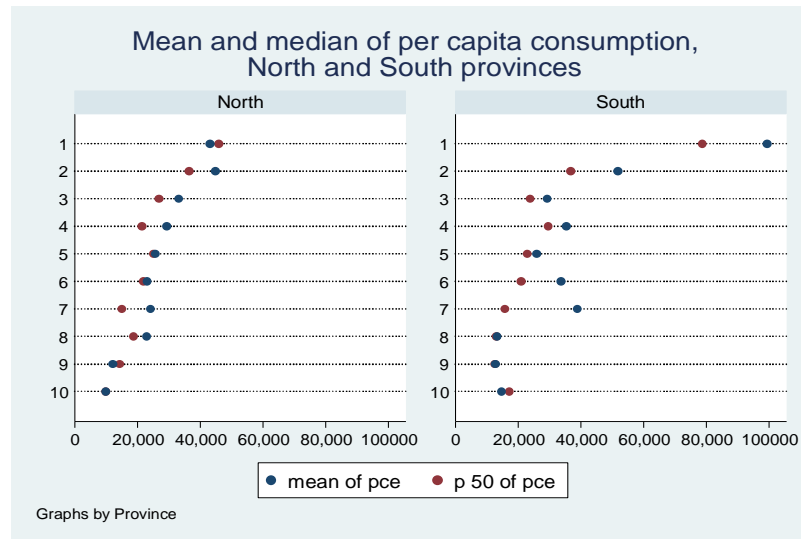
## Dot charts

```
use "household.dta", clear
generate pce = tot_exp / hhsize
recode hhsize (10/max=10), generate(hhsize2)
keep if province == 1 | province == 4 // We keep only two provinces
graph dot (mean) pce (p50) pce [pweight=wgtpop], over(hhsize2)
by(province, title("Mean and median of per capita consumption," "North
and South provinces")) // The title is within the 'by' option
```

96



## Dot charts



97

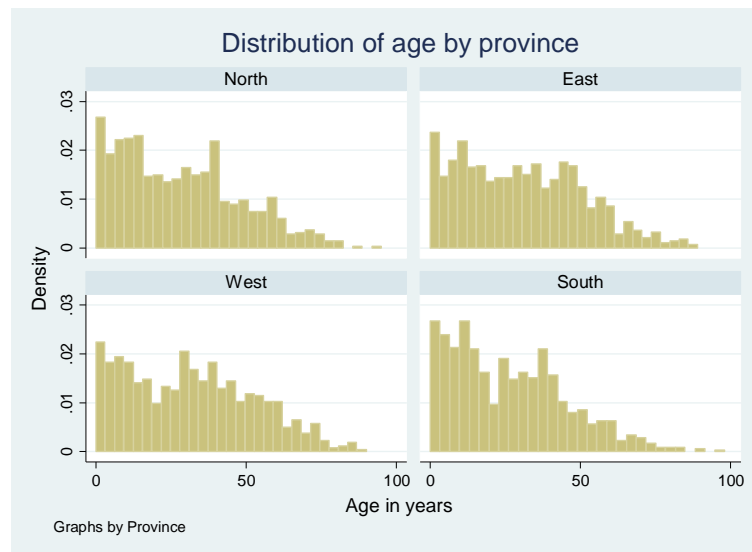
## Histograms

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

```
twoway histogram age, by(province, ///  
title("Distribution of age by province"))
```

98

# Histograms



99

# Box plots

- The box plot (a.k.a. box and whisker diagram) is a standardized way of displaying the distribution of data based on the five number summary: minimum, first quartile, median, third quartile, and maximum.
- In the simplest box plot the central rectangle spans the first quartile to the third quartile (the interquartile range or IQR).
- A segment inside the rectangle shows the median and "whiskers" above and below the box show the locations of the minimum and maximum.

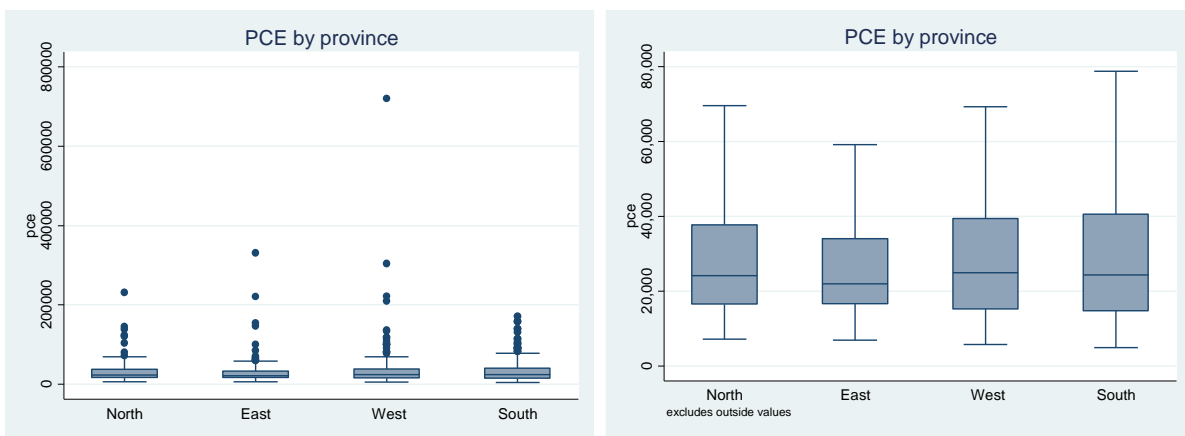
100

## Box plots – Example from FSM HIES 2013-14

```
use "household.dta", clear
generate pce = tot_exp / hhsiz
graph box pce, over(province) title("PCE by province")
graph box pce, over(province) title("PCE by province") nooutsides
```

101

## Box plots – PCE in FSM, HIES 2013-14



102

# Statistical analysis - Regressions

103

## 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`).

104

## 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.

```
use "expenditure.dta", clear
keep if itemcode==44
graph box cons_purch
```



## 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 per capita expenditure, number of rooms in the dwelling, and household size

```
. correlate pce rooms hhsize
(obs=1,000)

```

	pce	rooms	hhsize
pce	1.0000		
rooms	0.4460	1.0000	
hhsize	-0.2296	-0.2528	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

107

## 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]
```

108

## 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 *province* and *sex* into dummies, with prefix “i.”, one would simply type:

```
xi.province i.sex
```

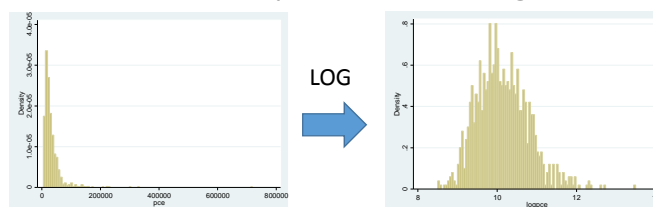
- The `xi` command and the regression command can conveniently be combined into a single command, simply by preceding the `regress` command with `xi` as shown in the code example below.

109

## Linear regression model: An example

- In this example we will predict the (log) per capita expenditure (pce) based on multiple variables:
  - Categorical: *province, dwelling, water, toilet, wall, roof, floor, electricity, car*
  - *hsize, rooms*
- The distribution of pce is skewed; we will therefore fit a model to predict its log value, which has a quasi-normal distribution. After we predict the log(pce), we will convert back to pce values using `exp`.

```
histogram pce, bin(100)  
gen logpce = log(pce)  
histogram logpce, bin(100)
```



110

# Linear regression model: An example

```
use "C:\Stata_Fiji\Data\household.dta", clear

generate pce = tot_exp/hhsize
generate logpce = log(pce)
xi: regress logpce hhsz rooms          ///
    i.province i.dwelling i.water i.toilet  ///
    i.wall i.roof i.floor i.electricity i.car  ///
    [weight=hhwgt]

predict pred_logpce
generate pred_pce = exp(pred_logpce)
summarize pce pred_pce
```

111

## Regression results (1/3)

```
. xi: regress logpce hhsz rooms          ///
> i.province i.dwelling i.water i.toilet  ///
> i.wall i.roof i.floor i.electricity i.car  ///
> [weight=hhwgt]
i.province      _Iprovince_1-4      (naturally coded; _Iprovince_1 omitted)
i.dwelling      _Idwelling_1-5      (naturally coded; _Idwelling_1 omitted)
i.water         _Iwater_1-5        (naturally coded; _Iwater_1 omitted)
i.toilet       _Itoilet_1-4       (naturally coded; _Itoilet_1 omitted)
i.wall         _Iwall_1-4         (naturally coded; _Iwall_1 omitted)
i.roof         _Iroof_1-7        (naturally coded; _Iroof_1 omitted)
i.floor        _Ifloor_1-6       (naturally coded; _Ifloor_1 omitted)
i.electricity   _Ielectrici_0-1   (naturally coded; _Ielectrici_0 omitted)
i.car          _Icar_1-2        (naturally coded; _Icar_1 omitted)
(analytic weights assumed)
(sum of wgt is 6.1871e+07)
```

Source	SS	df	MS	Number of obs	=	1,000
Model	350.330071	32	10.9478147	F(32, 967)	=	111.71
Residual	94.7644786	967	.097998427	Prob > F	=	0.0000
				R-squared	=	0.7871
				Adj R-squared	=	0.7800
Total	445.09455	999	.44554009	Root MSE	=	.31305

112



## Regression results (2/3)

logpce	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]
hhsz	-.0574795	.0067061	-8.57	0.000	-.0706396 -.0443193
rooms	.0114693	.0122005	0.94	0.347	-.0124732 .0354118
_lprovince_2	-.049774	.0336565	-1.48	0.139	-.1158221 .0162742
_lprovince_3	.0151265	.0341893	0.44	0.658	-.0519673 .0822203
_lprovince_4	.0163434	.0374566	0.44	0.663	-.0571623 .089849
_ldwelling_2	-.5513828	.0369208	-14.93	0.000	-.6238369 -.4789287
_ldwelling_3	-.906243	.0574212	-15.78	0.000	-1.018927 -.7935586
_ldwelling_4	-1.238741	.0839349	-14.76	0.000	-1.403457 -1.074026
_ldwelling_5	-1.608476	.116596	-13.80	0.000	-1.837286 -1.379665
_lwater_2	-.0953891	.0358235	-2.66	0.008	-.1656899 -.0250883
_lwater_3	-.1609466	.0286203	-5.62	0.000	-.2171117 -.1047815
_lwater_4	-.1803564	.0357671	-5.04	0.000	-.2505465 -.1101663
_lwater_5	-.0579707	.0690435	-0.84	0.401	-.1934631 .0775217
_ltoilet_2	-.0768373	.0331723	-2.32	0.021	-.1419352 -.0117393
_ltoilet_3	-.0755131	.0606584	-1.24	0.213	-.1945503 .0435241
_ltoilet_4	-.0602838	.0284143	-2.12	0.034	-.1160447 -.0045229
_lwall_2	-.0910194	.0301748	-3.02	0.003	-.150235 -.0318037
_lwall_3	-.0649203	.0537421	-1.21	0.227	-.1703849 .0405443
_lwall_4	-.1461342	.1098735	-1.33	0.184	-.3617521 .0694837
_lroof_2	-.1539876	.0632853	-2.43	0.015	-.27818 -.0297953
_lroof_3	.0410972	.1108051	0.37	0.711	-.1763489 .2585433
_lroof_4	-.0430029	.06669	-0.64	0.519	-.1738768 .087871
_lroof_5	.011435	.0711075	0.16	0.872	-.1281078 .1509777
_lroof_6	-.033504	.1165107	-0.29	0.774	-.262147 .195139
_lroof_7	.0396472	.1047088	0.38	0.705	-.1658354 .2451298
_lfloor_2	-.0153205	.0419005	-0.37	0.715	-.0975468 .0660058
_lfloor_3	-.0575531	.0283218	-2.03	0.042	-.1131324 -.0019738
_lfloor_4	-.0473296	.0394306	-1.20	0.230	-.124709 .0300499
_lfloor_5	-.0103842	.052756	-0.20	0.844	-.1139136 .0931452
_lfloor_6	-.0647265	.0966009	-0.67	0.503	-.2542981 .124845
_lelectrici_1	.0256608	.0379572	0.68	0.499	-.0488272 .1001488
_lcar_2	-.5110828	.0445085	-11.48	0.000	-.5984271 -.4237384
_cons	11.49543	.123861	92.81	0.000	11.25236 11.73849

113

## Regression results (3/3)

`summarize pce pred_pce`

Variable	Obs	Mean	Std. Dev.	Min	Max
pce	1,000	32678.03	36303.18	4906.748	719784.7
pred_pce	1,000	29989.36	16858.07	5151.454	92220.89

114

## Logistic regression, a.k.a. logit model

- Logistic regression predicts dichotomous variables, i.e. the dependent variable is binary (true/false, yes/no, poor/non-poor, etc.)
- Alternative: probit regression
- Two commands in Stata: `logit` and `logistic` (same, except that `logistic` displays estimates as odds ratios)
- Syntax (see Stata help for detail on options):

```
logit depvar [indepvars] [if] [in] [weight] [, options]
or
logistic depvar indepvars [if] [in] [weight] [, options]
```

115

## Logistic regression model: An example

```
* We predict the poverty status of the households based on a few variables
use "C:\Stata_Fiji\Data\household.dta", clear
generate pce = tot_exp / hsize

* We create a variable poor (1) - non poor (0) using a poverty line = 18000
generate poor = pce < 18000
xi: logit poor hsize rooms i.province i.water i.toilet ///
    i.wall i.electricity i.car [pweight=hhwgt]

predict poor_pred // We apply the logistic regression model

gen poor2 = poor_pred > 0.5 // If probability > 0.5 → poor, otherwise not
table poor poor2 // Show the confusion matrix
```

116

# Logistic regression model: the results

```

Logistic regression                Number of obs   =       944
                                Wald chi2(15)      =       60.71
                                Prob > chi2       =       0.0000
Log pseudolikelihood = -7547641.9 Pseudo R2       =       0.7783
    
```

poor	Coef.	Robust Std. Err.	z	P> z	[95% Conf. Interval]	
hhsize	.1797348	.1202888	1.49	0.135	-.056027	.4154966
rooms	-2.979132	.4928465	-6.04	0.000	-3.945093	-2.01317
_lprovince_2	-.2224959	.4177789	-0.53	0.594	-1.041327	.5963357
_lprovince_3	.6037842	.4694361	1.29	0.198	-.3162936	1.523862
_lprovince_4	.3209073	.4439666	0.72	0.470	-.5492513	1.191066
_lwater_2	.3758244	.6487015	0.58	0.562	-.8956073	1.647256
_lwater_3	.9404626	.4929785	1.91	0.056	-.0257575	1.906683
_lwater_4	.8681463	.6209504	1.40	0.162	-.3488942	2.085187
_lwater_5	2.444589	1.008224	2.42	0.015	.4685063	4.420671
_ltoilet_2	-.454596	.473035	-0.96	0.337	-1.381728	.4725355
_ltoilet_3	1.356786	.7592227	1.79	0.074	-.1312629	2.844835
_ltoilet_4	.0570307	.4177907	0.14	0.891	-.761824	.8758855
_lwall_2	.0071212	.3788574	0.02	0.985	-.7354257	.7496681
_lwall_3	0	(omitted)				
_lwall_4	0	(omitted)				
_lelectrici_1	-.3253506	.4432731	-0.73	0.463	-1.19415	.5434488
_lcar_2	2.791787	.7160761	3.90	0.000	1.388304	4.195271
_lcons	7.792916	2.039737	3.82	0.000	3.795106	11.79073

Confusion matrix:

poor	poor2	
	0	1
0	656	31
1	26	287

Note: the proper way of testing a regression model and to avoid overfitting is to measure its “out of sample” performance by creating a training set and a test set.

117

# Programming in Stata

118

# 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.

119

# Header

- It is highly recommended to include a header (as “comment”) in all your programs, which describes the author, purpose, date, necessary input, and outputs of the program.
- Example:

```
*****  
* Stata program for poverty analysis using test dataset  
* Author: Olivier Dupriez, World Bank  
* Date: ...  
* Input files : ...  
*****
```

120

## 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 14
set more off
```

121

## 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.

122

## 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_TRAINING/Exercise_01.txt", replace text
```

- Syntax to close a log:

```
log close
```

- Syntax to temporarily suspend logging or resume logging:

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

123

## 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)

124

## Long commands – Example

```
#delimit ;  
recode province (17=13)(5=14)(11=15)(16=16)(7=17)  
(12=18)(3=1)(6=2)(4=3)(2=4)(14=5)(13=6)(10=7)(1=8)(8=9)  
(15=11)(9=12), gen(prov) ;  
#delimit cr
```

OR

```
recode province (17=13)(5=14)(11=15)(16=16)(7=17) ///  
(12=18)(3=1)(6=2)(4=3)(2=4)(14=5)(13=6)(10=7)(1=8)(8=9) ///  
(15=11)(9=12), gen(prov)
```

125

## 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`

126

## 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_Fiji"`

127

## 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 in a 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_Fiji"
display "The content of my global macro is " $myfolder
```

128



## 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

129

## Temporary variables

- You can also generate temporary variables (the same way you can create temporary data files) in your Stata programs. These variables are not saved; they will automatically be dropped at the end of the program execution.
- You initiate the temporary variables using the command `tempvar`. For example: `tempvar tv1 tv2 tv3`
- In your program, you refer to these variables by enclosing them in quotes like you would do with a local macro. For example:  
`gen `tv1' = income * 12`

130

## 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

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

131

## 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

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

132

## 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 hhsize // Produce summary stats of variable hhsize
display "Variable hhsize 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
```

133

## 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 choose 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).

134

## 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).

135

## 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
}
```

136

## Loops using “forvalues”

Another way of achieving a loop through numeric values is to 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

137

## 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.

138

## 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
}
```

139

## 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.

140

## Preserving and restoring data in memory

- `preserve` and `restore` deal with the programming problem where the user's data must be changed to achieve the desired result but, when the program concludes, the programmer wishes to undo the damage done to the data.
- When `preserve` is issued, the user's data are preserved. The data in memory remain unchanged. When the program or do-file concludes, the user's data are automatically restored.
- After a `preserve`, the programmer can also instruct Stata to `restore` the data now with the `restore` command. This is useful when the programmer needs the original data back and knows that no more damage will be done to the data.
- `restore`, `preserve` can be used when the programmer needs the data back but plans further damage. `restore`, `not` can be used when the programmer wishes to cancel the previous `preserve` and to have the data currently in memory returned to the user.

*(Description extracted from the Stata manual)*

141

## 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 hsize`

→ 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).

142

## 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 "individual.dta", clear
generate age = 10
variable age 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.

143

## Working with CSV and Excel files

144



## 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
```

145

## 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)

146

## 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
```

147

## 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]
```

148

## 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 = ("$$DATE")
putexcel F14 = ("$$TIME")
putexcel K20 = (`poverty_headcount')
```

149

## Interacting with Excel: an example

In this example, we will extract the value of various poverty lines from an Excel sheet, use these poverty lines to calculate poverty indicators, and save selected poverty indicators back in the XLSX file.

Note 1: to run the example shown in the next slide, the package `poverty` must have been installed (`ssc install poverty`).

Note 2: the package `poverty` saves the various results it produces in global macros named `S_1` to `S_27` (see package help file). Global macros are referred to as `$$_1` to `$$_27` in Stata programs.

150

## 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_Fiji\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/ hhszise
    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 = ("$_DATE") // We save the date in cell C6
```

151

## Interacting with the Operating System

152

## Interaction with the operating system

In some programs, you may want to execute some commands from the operating command prompt, for example to erase a file or to obtain a list of files in a directory.

You can execute such commands by preceding them with a !

Examples:

```
!dir C:\FSM
```

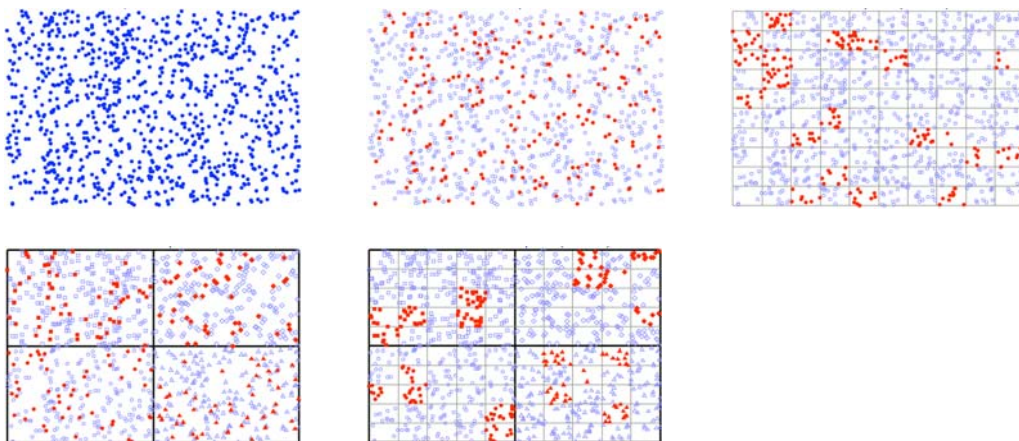
```
!erase "C:\FSM\temporary_file.dta"
```

153

## Specific commands for survey data tabulation and analysis

154

## Some example of sample designs



Source: Jeff Pitblado, Associate Director, Statistical Software at StataCorp LP. 2009 Canadian Stata Users Group Meeting. Available at [http://www.stata.com/meeting/canada09/ca09\\_pitblado\\_handout.pdf](http://www.stata.com/meeting/canada09/ca09_pitblado_handout.pdf)

155

## 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.

156

## 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]
```

157

## 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`

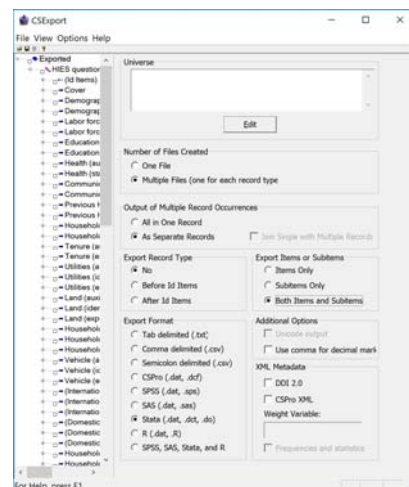
158

# Importing data from CsPro

159

## To export a CsPro dataset to Stata

- Create a new folder in which you will save the exported materials
- Open the CsPro data dictionary corresponding to the file to be exported, then select **Tools > Export Data**.
  - The Export dialog box will be opened
  - Enter the options as shown in the next slide
  - CsPro will generate a collection of files (to be saved in the new folder). These files contain the materials needed to produce the Stata data files (not yet the Stata data files themselves).
  - You will have to run all [.do] files to produce the data files in Stata format, and save them.



160



# The export options in CsPro

- Select the options as follows:

Number of Files Created

One File

Multiple Files (one for each record type)

Output of Multiple Record Occurrences

All in One Record

As Separate Records  Join Single with Multiple Records

Export Record Type

No

Before Id Items

After Id Items

Export Items or Subitems

Items Only

Subitems Only

Both Items and Subitems

Export Format

Tab delimited (.txt)

Comma delimited (.csv)

Semicolon delimited (.csv)

CsPro (.dat, .dcf)

SPSS (.dat, .sps)

SAS (.dat, .sas)

Stata (.dat, .dct, .do)

R (.dat, .R)

SPSS, SAS, Stata, and R

161

## CsPro export to Stata

- CsPro export to Stata will generate, for each record type in the CsPro dictionary:
  - One do file (extension DO)
  - One dictionary file (extension DCT)
  - One data file (extension DAT)
- CsPro does not generate the Stata data files; it generates the materials needed to produce the Stata data files.
- This can involve executing many do files (one per record type).
- They can be run one by one, or a do file can be produced to run them in one batch

162

## Executing the do files one by one

- For each record type in the CsPro data dictionary, CsPro will have produce a DAT file (an ASCII fixed-format file containing the data for each specific record type), a DCT file that contains the information on the position of each variable in the DAT file and the variable and value labels, and a DO file that applies the DCT information to the DAT file.
- For each do file, you will have to run (in Stata) the following code:

```
clear
do filename.DO
compress
save "filename.dta", replace
```

163

## Executing all do files in a batch using a do file

```
clear *
set more off
cd "C:\FSM_HIES_2013\CsPro" // Where the CsPro export files (DCT, DAT, DO) are stored

local outdir = "C:\FSM_HIES_2013\Stata" // Where we will save the Stata data files

capture !erase listDCT.txt // Delete the list of CsPro .DCT files if it exists
!dir *.dct /B -> listDCT.txt // Create a text file containing the list of CsPro .DCT files

file open Recs using listDCT.txt, read // Open that text file containing the list of files
file read Recs line

while r(eof)=0 { // We will read the lines one by one, until we reach the end of file (eof)
    local filenm = substr("`line'",1,length("`line')-4) // Remove ".DCT" to keep only the file name
    clear
    do "`filenm'.do" // Run the do file (convert data from ASCII to Stata, and add labels)
    compress
    save "`outdir'\`filenm'.dta", replace
    file read Recs line // If not last line of the text file, read next line
}

file close Recs // Job completed; we can close the text file
```

164