TRAINING COURSE ON
AGRICULTURAL CENSUSES AND REGISTERS

An Introduction to
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.

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

- 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

\[(\text{by varlist:}) \text{ command [varlist=exp]} [\text{if exp}] [\text{in range}] [\text{weight}] [, \text{options}]\]

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]
```

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>detail</td>
<td>display additional statistics</td>
</tr>
<tr>
<td>meanonly</td>
<td>suppress the display; calculate only the mean; programmer's option</td>
</tr>
<tr>
<td>format</td>
<td>use variable's display format</td>
</tr>
<tr>
<td>separator(#)</td>
<td>draw separator line after every # variables; default is separator(5)</td>
</tr>
<tr>
<td>display_options</td>
<td>control spacing, line width, and base and empty cells</td>
</tr>
</tbody>
</table>
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
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.
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 $\frac{5,000}{1,000,000} = 0.005$
- The sample weight of each household will be $\frac{1,000,000}{5,000} = 200$

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”)

Training on Agricultural Censuses and Registers in Male Republic of Maldives, 10-12 October 2017
Working on data files
The structure of a Stata data file

<table>
<thead>
<tr>
<th>month</th>
<th>Date</th>
<th>Atoll</th>
<th>Island</th>
<th>Boatname</th>
<th>Crops</th>
<th>CropsInEnglish</th>
<th>Units</th>
<th>QuantityMe-d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>jan 1/4/2015</td>
<td>AA</td>
<td>Thoddoo</td>
<td>Ranz-2</td>
<td>Falhoa</td>
<td>Papaya</td>
<td>Kg</td>
<td>1800</td>
</tr>
<tr>
<td>2</td>
<td>jan 1/4/2015</td>
<td>AA</td>
<td>Thoddoo</td>
<td>Ranz-2</td>
<td>Israel Falhoa</td>
<td>Papaya (Israel)</td>
<td>Kg</td>
<td>300</td>
</tr>
<tr>
<td>3</td>
<td>jan 1/4/2015</td>
<td>AA</td>
<td>Thoddoo</td>
<td>Ranz-2</td>
<td>Chichandaa</td>
<td>Snake Gourd</td>
<td>Kg</td>
<td>200</td>
</tr>
<tr>
<td>4</td>
<td>jan 1/4/2015</td>
<td>AA</td>
<td>Thoddoo</td>
<td>Ranz-2</td>
<td>Barabo</td>
<td>Pumpkin</td>
<td>Kg</td>
<td>200</td>
</tr>
<tr>
<td>5</td>
<td>jan 1/4/2015</td>
<td>AA</td>
<td>Thoddoo</td>
<td>Ranz-2</td>
<td>Thoraa</td>
<td>Sponge Gourd</td>
<td>Kg</td>
<td>200</td>
</tr>
<tr>
<td>6</td>
<td>jan 1/4/2015</td>
<td>AA</td>
<td>Thoddoo</td>
<td>Ranz-2</td>
<td>Hika'ndhifaiy</td>
<td>Curry Leaves</td>
<td>Easthaa</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>jan 1/4/2015</td>
<td>AA</td>
<td>Thoddoo</td>
<td>Ranz-2</td>
<td>Bashi</td>
<td>Egg Plant</td>
<td>Kg</td>
<td>300</td>
</tr>
<tr>
<td>8</td>
<td>jan 1/4/2015</td>
<td>AA</td>
<td>Thoddoo</td>
<td>Ranz-2</td>
<td>Cucumber</td>
<td>Cucumber</td>
<td>Kg</td>
<td>4000</td>
</tr>
<tr>
<td>9</td>
<td>jan 1/4/2015</td>
<td>AA</td>
<td>Thoddoo</td>
<td>Ranz-2</td>
<td>Faaga</td>
<td>Bitter Gourd</td>
<td>Kg</td>
<td>100</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Month</th>
<th>Date</th>
<th>Atoll</th>
<th>Island</th>
<th>Boatname</th>
<th>Crops</th>
<th>Crops in English</th>
<th>Units</th>
<th>Quantity</th>
<th>Me-d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>1/13/2015</td>
<td>GN</td>
<td>Fuvamulah</td>
<td>Ravaaana</td>
<td>A'nbu</td>
<td>Mango</td>
<td>Nos</td>
<td>100000</td>
<td></td>
</tr>
<tr>
<td>Feb</td>
<td>2/1/2015</td>
<td>LA</td>
<td>Hithadhoo</td>
<td>Faihaa</td>
<td>A'nbu</td>
<td>Mango</td>
<td>Nos</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td>Feb</td>
<td>2/3/2015</td>
<td>SE</td>
<td>Hithadhoo</td>
<td>Furusath</td>
<td>A'nbu</td>
<td>Mango</td>
<td>Nos</td>
<td>2500</td>
<td></td>
</tr>
<tr>
<td>Feb</td>
<td>2/8/2015</td>
<td>SE</td>
<td>Hithadhoo</td>
<td>Aagala queen</td>
<td>A'nbu</td>
<td>Mango</td>
<td>Nos</td>
<td>5000</td>
<td></td>
</tr>
<tr>
<td>Feb</td>
<td>2/11/2015</td>
<td>GN</td>
<td>Fuvahmulah</td>
<td>Naares</td>
<td>A'nbu</td>
<td>Mango</td>
<td>Nos</td>
<td>13000</td>
<td></td>
</tr>
<tr>
<td>Feb</td>
<td>2/12/2015</td>
<td>KA</td>
<td>Kaashidhoo</td>
<td>Treaser</td>
<td>A'nbu</td>
<td>Mango</td>
<td>Nos</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>Feb</td>
<td>2/13/2015</td>
<td>LA</td>
<td>Kunahandhoo</td>
<td>Soanaa</td>
<td>A'nbu</td>
<td>Mango</td>
<td>Nos</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>Feb</td>
<td>2/15/2015</td>
<td>GDH</td>
<td>Faresmaathdhaa</td>
<td>Fahiruahhi</td>
<td>A'nbu</td>
<td>Mango</td>
<td>Nos</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>Feb</td>
<td>2/26/2015</td>
<td>LA</td>
<td>Isdho/kalaidhoo</td>
<td>Falaky</td>
<td>A'nbu</td>
<td>Mango</td>
<td>Nos</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>Feb</td>
<td>2/17/2015</td>
<td>SH</td>
<td>Feevah</td>
<td>Amaaz</td>
<td>A'nbu</td>
<td>Mango</td>
<td>Nos</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>Feb</td>
<td>2/19/2015</td>
<td>NO</td>
<td>Miladho</td>
<td>Manail</td>
<td>A'nbu</td>
<td>Mango</td>
<td>Nos</td>
<td>400</td>
<td></td>
</tr>
<tr>
<td>Feb</td>
<td>2/24/2015</td>
<td>AA</td>
<td>Thoddoo</td>
<td>Feneybaa</td>
<td>A'nbu</td>
<td>Mango</td>
<td>Nos</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>Feb</td>
<td>2/26/2015</td>
<td>GN</td>
<td>Fuvahmulah</td>
<td>Havaas</td>
<td>A'nbu</td>
<td>Mango</td>
<td>Nos</td>
<td>20000</td>
<td></td>
</tr>
<tr>
<td>Mar</td>
<td>3/3/2015</td>
<td>GN</td>
<td>Fuvahmulah</td>
<td>Ravaaana</td>
<td>A'nbu</td>
<td>Mango</td>
<td>Nos</td>
<td>140000</td>
<td></td>
</tr>
<tr>
<td>Mar</td>
<td>3/4/2015</td>
<td>SH</td>
<td>Feydho</td>
<td>Shaazee-4</td>
<td>A'nbu</td>
<td>Mango</td>
<td>Nos</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>Mar</td>
<td>3/4/2015</td>
<td>SE</td>
<td>Hithadhoo</td>
<td>Aagala queen</td>
<td>A'nbu</td>
<td>Mango</td>
<td>Nos</td>
<td>6000</td>
<td></td>
</tr>
<tr>
<td>Mar</td>
<td>3/4/2015</td>
<td>AA</td>
<td>Thoddoo</td>
<td>Rani-2</td>
<td>A'nbu</td>
<td>Mango</td>
<td>Nos</td>
<td>400</td>
<td></td>
</tr>
</tbody>
</table>
describe produces a summary of the dataset in memory

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

<table>
<thead>
<tr>
<th>variable name</th>
<th>storage</th>
<th>type</th>
<th>format</th>
<th>value</th>
<th>variable label</th>
</tr>
</thead>
<tbody>
<tr>
<td>month</td>
<td>str4</td>
<td>%9s</td>
<td></td>
<td></td>
<td>month</td>
</tr>
<tr>
<td>Date</td>
<td>int</td>
<td>%td.</td>
<td></td>
<td></td>
<td>Date</td>
</tr>
<tr>
<td>Atoll</td>
<td>str3</td>
<td>%9s</td>
<td></td>
<td></td>
<td>Atoll</td>
</tr>
<tr>
<td>Island</td>
<td>str23</td>
<td>%23s</td>
<td></td>
<td></td>
<td>Island</td>
</tr>
<tr>
<td>Boatname</td>
<td>str16</td>
<td>%16s</td>
<td></td>
<td></td>
<td>Boat name</td>
</tr>
<tr>
<td>Crops</td>
<td>str18</td>
<td>%18s</td>
<td></td>
<td></td>
<td>Crops</td>
</tr>
<tr>
<td>CropsinEnglish</td>
<td>str16</td>
<td>%16s</td>
<td></td>
<td></td>
<td>Crops in English</td>
</tr>
<tr>
<td>Units</td>
<td>str7</td>
<td>%9s</td>
<td></td>
<td></td>
<td>Units</td>
</tr>
<tr>
<td>QuantityMeasured</td>
<td>long</td>
<td>%10.0g</td>
<td></td>
<td>Quantity (Measured)</td>
<td></td>
</tr>
<tr>
<td>QuantityKg</td>
<td>double</td>
<td>%10.0g</td>
<td></td>
<td>Quantity (Kg)</td>
<td></td>
</tr>
<tr>
<td>RPrice1</td>
<td>double</td>
<td>%10.0g</td>
<td></td>
<td>R.Price 1</td>
<td></td>
</tr>
<tr>
<td>RPrice2</td>
<td>double</td>
<td>%10.0g</td>
<td></td>
<td>R.Price 2</td>
<td></td>
</tr>
<tr>
<td>AvePrice</td>
<td>double</td>
<td>%10.0g</td>
<td></td>
<td>Ave.Price</td>
<td></td>
</tr>
<tr>
<td>Wsprice</td>
<td>double</td>
<td>%10.0g</td>
<td></td>
<td>Ws.price</td>
<td></td>
</tr>
<tr>
<td>Income</td>
<td>double</td>
<td>%10.0g</td>
<td></td>
<td>Income</td>
<td></td>
</tr>
</tbody>
</table>

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
```
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.

\[
\text{count [if] [in]}
\]

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) →
```

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.

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

Examples:
```
codebook       // all variables in data file
codebook Crops-RPrice 1 // 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)**

<table>
<thead>
<tr>
<th>Boatname</th>
<th>Month</th>
<th>Date</th>
<th>Atoll</th>
<th>Island</th>
<th>Crops</th>
<th>CropsinEnglish</th>
<th>Units</th>
<th>Quantity</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aagala</td>
<td>Feb</td>
<td>2/26/2016</td>
<td>SE</td>
<td>Feydhoo</td>
<td>Faiykeyo</td>
<td>Ga'ndu</td>
<td>Nos</td>
<td>10</td>
<td>151.2</td>
</tr>
<tr>
<td>Aagala</td>
<td>Feb</td>
<td>2/26/2016</td>
<td>SE</td>
<td>Meedhoo</td>
<td>A'nbu</td>
<td>Ga'ndu</td>
<td>Nos</td>
<td>1000</td>
<td>250</td>
</tr>
<tr>
<td>Aagala</td>
<td>Feb</td>
<td>2/26/2016</td>
<td>SE</td>
<td>Meedhoo</td>
<td>Kaashi</td>
<td>Nos</td>
<td>600</td>
<td>64.8</td>
<td></td>
</tr>
<tr>
<td>Aagala</td>
<td>Feb</td>
<td>2/26/2016</td>
<td>SE</td>
<td>Feydhoo</td>
<td>A'nbu</td>
<td>Ga'ndu</td>
<td>Nos</td>
<td>1000</td>
<td>250</td>
</tr>
<tr>
<td>Aagala queen</td>
<td>Dec</td>
<td>12/14/2015</td>
<td>SE</td>
<td>Hithadhoo</td>
<td>Kuru'nbaa</td>
<td>Tender Coconut</td>
<td>Nos</td>
<td>2000</td>
<td>6000</td>
</tr>
<tr>
<td>Aagala queen</td>
<td>Dec</td>
<td>12/14/2015</td>
<td>SE</td>
<td>Hithadhoo</td>
<td>Ala</td>
<td>Taro</td>
<td>KG</td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td>Aagala queen</td>
<td>Dec</td>
<td>12/14/2015</td>
<td>SE</td>
<td>Hithadhoo</td>
<td>Faiykeyo</td>
<td>Banana</td>
<td>Ga'ndu</td>
<td>300</td>
<td>4500</td>
</tr>
<tr>
<td>Aagala queen</td>
<td>Dec</td>
<td>12/14/2015</td>
<td>SE</td>
<td>Meedhoo</td>
<td>Faiykeyo</td>
<td>Banana</td>
<td>Ga'ndu</td>
<td>60</td>
<td>900</td>
</tr>
<tr>
<td>Aalim</td>
<td>Sept</td>
<td>9/19/2016</td>
<td>ME</td>
<td>KolhuFushi</td>
<td>Faiykeyo</td>
<td>Ga'ndu</td>
<td>3</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>Aalim</td>
<td>Oct</td>
<td>10/6/2015</td>
<td>ME</td>
<td>KolhuFushi</td>
<td>Faiykeyo</td>
<td>Banana</td>
<td>Ga'ndu</td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>Aalim</td>
<td>Oct</td>
<td>10/6/2015</td>
<td>ME</td>
<td>KolhuFushi</td>
<td>Kuru'nbaa</td>
<td>Nos</td>
<td>69</td>
<td>207</td>
<td></td>
</tr>
</tbody>
</table>

**Boat level data (1 row per boat)**

<table>
<thead>
<tr>
<th>Boatname</th>
<th>MadeIn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aagala</td>
<td>Netherlands</td>
</tr>
<tr>
<td>Aagala queen</td>
<td>India</td>
</tr>
<tr>
<td>Aalim</td>
<td>China</td>
</tr>
<tr>
<td>Aaraasth</td>
<td>Germany</td>
</tr>
<tr>
<td>Akris</td>
<td>Germany</td>
</tr>
</tbody>
</table>

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

  \[
  \text{isid varlist}
  \]

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

- Many-to-one merge on specified key variables
  
  ```stata
  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 (\texttt{label define}), then attach it to a variable (\texttt{label values}). A same set can be used for multiple variables.

For example:

\begin{verbatim}
label variable exim "Main Activity"
label define activity 1 "Importer" 2 "Exporter"
label value exim activity
\end{verbatim}
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
```
• In Stata, you can generate a new variable using the command `generate`. The general syntax is:

\[
\text{generate newvarname} = \text{expression}
\]

• 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
<table>
<thead>
<tr>
<th>Relational operators</th>
<th>Logical operators</th>
<th>Mathematical operators</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code> (less than)</td>
<td><code>l</code> (or)</td>
<td><code>+</code></td>
</tr>
<tr>
<td><code>&gt;</code> (greater than)</td>
<td><code>&amp;</code> (and)</td>
<td><code>-</code></td>
</tr>
<tr>
<td><code>==</code> (equal)</td>
<td><code>~</code> (not)</td>
<td><code>*</code></td>
</tr>
<tr>
<td><code>&lt;=</code> (less than or equal)</td>
<td></td>
<td><code>/</code></td>
</tr>
<tr>
<td><code>&gt;=</code> (greater than or equal)</td>
<td></td>
<td><code>^</code></td>
</tr>
<tr>
<td><code>!=</code> or <code>~=</code> (not equal)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Mathematical functions**

- If $x$ is a numeric variable:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs($x$)</td>
<td>the absolute value of $x$</td>
</tr>
<tr>
<td>exp($x$)</td>
<td>The exponential function of $x$</td>
</tr>
<tr>
<td>int($x$)</td>
<td>the integer obtained by truncating $x$ toward 0 (thus, int(5.2) = 5 and int(-5.8) = -5)</td>
</tr>
<tr>
<td>ln($x$) or log($x$)</td>
<td>the natural logarithm, ln($x$)</td>
</tr>
<tr>
<td>max($x_1,x_2,: : :,x_n$)</td>
<td>the maximum value of $x_1,x_2,: : :,x_n$</td>
</tr>
<tr>
<td>min($x_1,x_2,: : :,x_n$)</td>
<td>the minimum value of $x_1,x_2,: : :,x_n$</td>
</tr>
<tr>
<td>mod($x,y$)</td>
<td>the modulus of $x$ with respect to $y$</td>
</tr>
<tr>
<td>round($x,y$) or round($x$)</td>
<td>$x$ rounded in units of $y$ or $x$ rounded to the nearest integer if the argument $y$ is omitted</td>
</tr>
<tr>
<td>sqrt($x$)</td>
<td>the square root of $x$</td>
</tr>
<tr>
<td>sum($x$)</td>
<td>the running sum of $x$, treating missing values as zero</td>
</tr>
</tbody>
</table>
**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)]
```

<table>
<thead>
<tr>
<th>rule</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td># = #</td>
<td>3 = 1</td>
<td>3 recoded to 1</td>
</tr>
<tr>
<td># # = #</td>
<td>2 . = 9</td>
<td>2 and . recoded to 9</td>
</tr>
<tr>
<td>#/# = #</td>
<td>1/5 = 4</td>
<td>1 through 5 recoded to 4</td>
</tr>
<tr>
<td>nonmissing = #</td>
<td>nonmiss = 8</td>
<td>all other nonmissing to 8</td>
</tr>
<tr>
<td>missing = #</td>
<td>miss = 9</td>
<td>all other missings to 9</td>
</tr>
</tbody>
</table>
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() 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 [variblename]` 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
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
```

<table>
<thead>
<tr>
<th>5 quantiles of Income</th>
<th>Freq.</th>
<th>Percent</th>
<th>Cum.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>885</td>
<td>20.66</td>
<td>20.66</td>
</tr>
<tr>
<td>2</td>
<td>847</td>
<td>19.77</td>
<td>40.43</td>
</tr>
<tr>
<td>3</td>
<td>841</td>
<td>19.63</td>
<td>60.06</td>
</tr>
<tr>
<td>4</td>
<td>869</td>
<td>20.28</td>
<td>80.35</td>
</tr>
<tr>
<td>5</td>
<td>842</td>
<td>19.65</td>
<td>100.00</td>
</tr>
</tbody>
</table>

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
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-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:

```
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)
```

<table>
<thead>
<tr>
<th>Island</th>
<th>stats</th>
<th>QuantityKg</th>
<th>Income</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baarah</td>
<td>mean</td>
<td>1016.675</td>
<td>11570.83</td>
</tr>
<tr>
<td></td>
<td>sd</td>
<td>1667.984</td>
<td>11603.35</td>
</tr>
<tr>
<td></td>
<td>min</td>
<td>37.2</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>max</td>
<td>7500</td>
<td>40000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Island</th>
<th>variable</th>
<th>mean</th>
<th>sd</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baarah</td>
<td>QuantityKg</td>
<td>1016.675</td>
<td>1667.984</td>
<td>37.2</td>
<td>7500</td>
</tr>
<tr>
<td></td>
<td>Income</td>
<td>11570.83</td>
<td>11603.35</td>
<td>200</td>
<td>40000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Island</th>
<th>variable</th>
<th>mean</th>
<th>sd</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bilehfa</td>
<td>QuantityKg</td>
<td>243.3333</td>
<td>326.277</td>
<td>30</td>
<td>900</td>
</tr>
<tr>
<td></td>
<td>Income</td>
<td>3166.667</td>
<td>1091.177</td>
<td>2000</td>
<td>4500</td>
</tr>
</tbody>
</table>
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(pricelgroup)

label variable pricelgroup "RPrice1 Group"

tabout pricelgroup exim using "tabout.xls", replace c(col) f(1)
clab(Col_%) npos(col) style(xls)
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 in detail
STATISTICAL ANALYSIS - REGRESSION
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

OK ??
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
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: \texttt{regress, predict}

• \texttt{regress} performs ordinary least-squares linear regression.
• The syntax is:
  \begin{verbatim}
  regress depvar [indepvars] [if] [in] [weight] [, options]
  \end{verbatim}

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

For a single-equation model, the syntax is:

\begin{verbatim}
  predict [type] newvar [if] [in] [, options]
\end{verbatim}
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)

Training on Agricultural Censuses and Registers in Male Republic of Maldives, 10–12 October 2017
Regression results (3/3)

```
. summarize Income pred_income

<table>
<thead>
<tr>
<th>Variable</th>
<th>Obs</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Income</td>
<td>4286</td>
<td>16141.02</td>
<td>45231.28</td>
<td>0</td>
<td>1600000</td>
</tr>
<tr>
<td>pred_income</td>
<td>4251</td>
<td>16154.42</td>
<td>35250.34</td>
<td>-18997.28</td>
<td>1429934</td>
</tr>
</tbody>
</table>
```
Programming in Stata
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.
  
  o Begin the line with a ‘ * ’; Stata ignores such lines.
  
  o Place the comment in /* ... */ delimiters.
  
  o Place the comment after two forward slashes, that is, //. Everything after the // to the end of the current line is considered a comment.
  
  o 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.,

  ```stata
  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 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_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:

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

- 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

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

```plaintext
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 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 $#t - #1$
Loops using “foreach”

foreach is used in conjunction with strings.

```plaintext
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
  }
  ```
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.
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
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*

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

- Example:
  
  ```stata
  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:

```stata
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:

```stata
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.

```stata
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
```

Training on Agricultural Censuses and Registers in Male Republic of Maldives, 10-12 October 2017
Specific Commands for Survey Data Tabulation and Analysis
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`