```
1    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
2    *
3    * PROGRAM:       Stata Tutorial
4    * PROGRAMMER:    Suanna Oh (SO)
5    * INSTITUTION:   Columbia University
6    * DATE:          November 30, 2016
7    *
8    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
9
10
11   capture cd "C:\Users\suannaoh\Dropbox\Programming TA"
12   capture cd "C:\Users\NewUser\..."
13       // capture: executes following commands, suppressing all output and ignoring error
     messages
14       // for instance, it will ignore line 14 if no such directory exists on the computer
15   clear all                                       // clear all memory
16   set more off                                    // run the entire do file without
     displaying -more-
17
18   /*   Before getting started
19
20   1. Set up a project folder
21
22   Create a folder for the project materials and change directory to this folder using [cd] as
     above
23   Within this folder we can create subfolders such as:
24   - Data: for saving data output files
25   - Log files: for saving log files
26   - Raw data: for saving raw data files
27   - Do files: for saving do files
28
29
30   2. Do-file documenting tools:
31
32   // : comment out words following
33   * : comment out the entire line if placed at the beginning of a line
34   /* */ : comment out multiple lines between the two stars
35   /// : break long commands into multiple lines
36
37   */
38
39
40   ************************************************************************
41   *       Task 1
42   ************************************************************************
43
44   /*
45
46   ** Note 1a: Merging two data sets
47
48   One should follow the steps below:
49   1. Decide which type of merging needs to be performed (1:1, m:1, or 1:m)
50      1:1 indicates there is one observation per ID in both data sets
51      m:1 indicates there are m observations per ID in the master data
52      1:m indicates there are m observations per ID in the using data
53   2. Check if there will be any conflicting information in the data sets and decide how to
     deal with them
54      If the two data sets have variables with the same name, the master data information in
     kept by default
55   3. In both the master and using data, check that the unit of observation is consistent
56      Check for any duplicate ID's
57   4. Make sure that the ID variables in both data sets are named the same way
58   5. Review _merge results
59
60
61   ** Note 1b: Using macros
62
63   Macros: A macro is simply a name associated with some text.
64   Local macros
65       - Created by typing either:
66           * local macro_name1 “macro content”
67           * local macro_name1 = expression to be evaluated
```

```
    - Referenced by writing `macro_name1' in the same do-file
    - Local macros are deleted after the current do-file finishes running
    - If we reference a macro that does not exist, Stata returns an emptry string for the macro
    - One can also temporarily save files under a local macro name using [tempfile]
Global macros:
    - Global macros are only deleted after Stata is closed
    - See help macro

*/

**  1. Importing data
import delimited using "Raw data/Respondent_list.csv", clear
label variable pid "Respondent ID"
lab var name "Respondent name"
lab var village "Village name"


**  2. Merging data
    // We first check if the individual level data
duplicates report pid               // we see that the IDs are not actually unique. There are 4 observations with duplicate ids.
duplicates tag pid, gen(dup)        // this generate a var named dup, which equals 1 if id is not unique
list if dup                         // we see that 1123 is a duplicate entry so we just drop this
duplicates drop
    // this only drops the complete duplicates (all variables are the same for two observations)
    // however, there are two different respondents with the same id 1491
    // here we use the information Fake Name 394's id is actually 1495, not 1491.
replace pid=1495 if pid==1491 & name=="Fake Name 394"
isid pid            // confirm pid uniquely identifies observations
drop dup            // drop this variable since we no longer need it


    // Then, we need to create dta version of the village data
    // Instead of saving the previous data and opening new data, we will preserve the indivudal data
preserve                    // preserve the current data set
import delimited using "Raw data/Village_info.csv", clear
isid village_code
rename village_code village     // since the village id is named "village" in the other data, we should rename this var
tempfile village                // declare that [village] will be used as a macro name to save a temporary file
save `village', replace         // one uses the regular saving commands to save temporary files
restore                         // restores the preserved data


    // We can now merge the two data sets using the variable "village"
merge m:1 village using `village'
    // since there are many respodents per village, we specify m:1 that this is the case
    // we see that all respondents in our respondent list are matched to villages (there is 0 case where _merge==1)
    // we also see that there are 110 villages with no respondents (_merge==2). We don't need them in the data, so let's drop them
drop if _merge==2
drop _merge
order village, before(villagename)  // this order the variable village before villagename
tabulate treatment


**  3. Summarizing and saving
bysort subcounty: sum treatment
tab subcounty, sum(treatment)                           // more compact way to do the above
bys subcounty: egen subcounty_size=count(treatment)     // calculate number of people in each subcounty
bys subcounty: egen treatment_share=mean(treatment)     // calculate the share of treated people in each subcounty
    // bysort: run commands on subsets of data after sorting data
```

```
    // egen command can also calculate std, min, max, median, rank, total, etc.
    // see help egen
save "Data/Respondent_village_data.dta", replace




*************************************************************************
*       Task 2
*************************************************************************

/*

Note 2: Working with time variables

- Human readable forms (HRFs) is the readable date form usually stored as string variables.
  While easy to read, Stata cannot work with these variables.
  ex ) "2015-11-17", "2010 Jul 12"
- Stata internal form (SIF) is the date form Stata can understand.
  These are numeric variables usually calculated with respect to 01jan1960 midnight.
  One can convert string time variables into SIF forms using various datetime functions.
- To use any ts- commands, one must declare the time variable using [tsset]

*/


**  1. Importing data and converting time variables
import delimited using "Raw data/GDP.csv", clear
generate time_day = date(date, "YMD")           // days since 01jan1960
format time_day %td                             // format the variable so that it is readable
generate time_quarter = qofd(time_day)          // quaters since 1960q1
format time_quarter %tq
label var time_quarter "Quarter"


**  2. Regress GDP on 1 period lagged var and 2 period lagged var.
tsset time_quarter                              // declare data to be time series
reg gdp L.gdp L2.gdp                            // L. (F.): create or use a lag (lead) variable


**  3. Detrend GDP using the BK filter and plot the series
tsfilter bk gdp_bk=gdp                          // de-trend gdp using a filter
label var gdp_bk "Detrended GDP"
tsline gdp_bk if time_quarter>tq(1990q1), lwidth(medthick) graphregion(color(white))   // plot gdp_bk against time
    // I used thicker lines and white background to make the graph look nicer
graph export Graphs/fig_gdp.pdf, replace




*************************************************************************
*       Task 3
*************************************************************************


**  1. Manipulating string variables
import excel "Raw data\Data_Extract_From_World_Development_Indicators.xlsx", sheet("Data") firstrow clear
drop if mi(CountryName)
gen ShortName=substr(SeriesName,1,strpos(SeriesName,"(")-1)
    // substr helps us get a portion of a string
    // strpos returns position of a particular string in another string
    // go to 'help functions' -> 'string functions' to see other string functions available
split SeriesCode, gen(Code) parse(".")
    // we can split a string based on a specific character
rename Code3 Code


**  2. How many countries have St. in their names?
```

```
encode ShortName, gen(SeriesNum)        // create a new variable that assigns a unique number to each series
count if regexm(CountryName,"St.") & SeriesNum==1
    // regexm returns 1 if the first argument (string) contains the second argument (regular expression)


**  3. Reshaping
keep SeriesNum CountryName CountryCode YR*
order SeriesNum CountryName CountryCode YR*
reshape long YR, i(SeriesNum CountryName)           // converts wide data to long data
rename _j year
rename YR value
replace value="" if value==".."
destring value, replace
    // destring converts a string variable to a numeric variable
    // tostring performs the opposite function
    // note the difference from encode and decode
sort SeriesNum CountryName year
by SeriesNum CountryName: gen growth_rate=(value-value[_n-1])/value[_n-1]
by SeriesNum CountryName: egen avg_growth=mean(growth_rate)
save "Data/Dev_indicators.dta", replace



*******************************************************************************
*       Task 4
*******************************************************************************

/*

Note 4a: Using a loop

Using a loop, we can repeat a set of commands to each element in a specified list: efficient coding!
Structure of a foreach loop:
    foreach [lname] of(in) [list] {
        commands referring to `lname'
    }
"foreach" repeatedly sets local macro `lname' to each element of the [list] and executes the commands enclosed in braces {}.
Allowed lists include:
- foreach lname in any_list {
- foreach lname of varlist varlist {
- foreach lname of numlist numlist {
- forvalues lname = range {
        forvalues is the fastest way to loop over consecutive values


Note 4b: if programming command

"if" executes a set of commands if exp is true (nonzero). Otherwise the statement following [else] is executed
    - not to be confused with the if qualifier
Basic structure:
    if exp {
        multiple commands
    }
    else {
        multiple commands
    }

*/


**  1. Reshaping data
use "Data/Dev_indicators.dta", clear
bys CountryName SeriesNum year: keep if _n==1       // keep one observation per country*series*year
keep SeriesNum CountryName CountryCode year growth_rate
reshape wide growth_rate, i(CountryName year) j(SeriesNum)
drop if year==2010
```

```
262
263
264   **  2. Looping
265   foreach x of numlist 1/3{                                        // let `x' take the value 1, 2,
      and then 3 for each series
266       bys CountryName: egen count`x'=count(growth_rate`x')         // calculate the number of
      observations per country for each series
267   }
268   egen min_count=rowmin(count*)                                    // calculate the minimum number
      of observations
269
270   capture log close                                                // close any log files open
271   log using "Log files/correlation.log", replace                   // start a log file
272   levelsof CountryName                                             // get the list of all country
      names for the country loop
273
274   foreach x in `r(levels)'{                                        // here `x' becomes each
      country name
275       di "`x'"                                                     // print country name
276       quietly sum min_count if CountryName=="`x'"                  // count the minimum number of
      observations for this country
277       if r(mean)<4 di "Missing values"                             // if the number is less than
      4, print "Missing values"
278       else pwcorr growth_rate? if CountryName=="`x'"               // otherwise calculate
      correlations
279       di ""                                                        // creates a line gap between
      each country result for readability
280   }
281   log close
282
283
284   ********************************************************************************
285   *       Task 5
286   ********************************************************************************
287
288
289       // First we can clean the animals data
290   use "Raw data/animals.dta", clear
291   replace type="NA" if mi(type)
292   isid village hhn id animal type
293
294
295       // need to impute values of animals by animal type
296   destring number price, replace force
297   egen category=group(animal type)
298   levelsof category if mi(price)
299   foreach y in `r(levels)'{
300       qui sum price if category==`y'
301       replace price=`r(mean)' if mi(price) & category==`y'
302   }
303   gen total_value=number*price
304   collapse (sum) total_value, by(village hhn id animal)
305   rename animal asset_type
306   tempfile animal
307   save `animal', replace
308
309
310       // Next we can clean other asset data
311   use "Raw data/cash.dta", clear
312   foreach x of varlist balance balance2 cash jewelry cloth{
313       if "`x'"=="jewelry" gen `x'_DK=`x'=="DK" | `x'=="0"
314       else gen `x'_DK=`x'=="DK"
315       destring `x', replace force
316       sum `x'
317       replace `x'=`r(mean)' if `x'_DK==1
318       drop `x'_DK
319   }
320   egen saving=rowtotal(balance balance2), m
321
322
323       // To create an asset level data, we can append partial data sets instead of reshaping
324   local k=1       // we can increase the value of this macro by 1 for each loop iteration
```

```
foreach x of varlist cash jewelry cloth saving{
    preserve
    keep village hhn id `x'
    drop if mi(`x')
    rename `x' total_value
    gen asset_type="`x'"
    tempfile data`k'
    save `data`k'', replace
    restore
    local `k++'     // increase the value of macro `k' by 1
}

use `data1', clear
append using `data2' `data3' `data4'
append using `animal'

sort village hhn id asset_type total_value
save "Data/asset_data.dta", replace

/*

Some useful websites for learning Stata:
http://data.princeton.edu/stata/
http://www.ats.ucla.edu/stat/stata/
http://www.cpc.unc.edu/research/tools/data_analysis/statatutorial/
http://staskolenikov.net/stata/

*/
exit
```