INTERNATIONAL FOOD POLICY RESEARCH INSTITUTE

ETHIOPIA STRATEGY SUPPORT PROGRAM (ESSP II)

STATA TRAINING NOTES

APRIL 2009

ADDIS ABABA, ETHIOPIA

INTERNATIONAL FOOD POLICY RESEARCH INSTITUTE- ADDIS ABABA

ETHIOPIA STRATEGY SUPPORT PROGRAM (ESSP II)

P.O. Box 5689

Addis Ababa, Ethiopia

Tel: +251-6-172-000

www.ifpri.org

STATA TRAINING NOTES (USING STATA 10.0)

30, March 2009 - 3, April 2009

ETHIOPIA STRATEGY SUPPORT PROGRAM (ESSP)

EDRI/IFPRI

Table of contents

STATA TRAINING NOTES

ETHIOPIA STRATEGY SUPPORT PROGRAM (ESSP)

EDRI/IFPRI

NOTE

To all users of the STATA training materials on this CD Rom

The global macro created here is for convenience so that one doesn't have to write (type) the path of directories, which may at times be too long and susceptible to errors in typing.

In order to use the do files in the do files directory (folder), one needs to follow the same data management structure, i.e, he/she needs to have the directories named exactly the same as those given in the global macro and also be located in the same location as the global macro.

For example, for using the "data management" do file, one needs to create a directory named "EthiopianData" located in the directory D. Inside this new folder another folder another folder named "CSA Survey data" need to be created. Do the same for all directories in the global macro. The final directory should be "do". Alternatively, one can create its own global macro (directory path) and locate it in any location desired. In this case however, one must change the directory path used in the do files as well.

Introduction

This note is a compilation of training materials prepared for a STATA training organized by Ethiopian Strategy Support Program (IFPRI) and delivered by Mr. Nigussie Tefera between 30, March and 3, April 2009.

Objective of the training

The training is organized to familiarize research officers (comment) working with Ethiopia Strategy Support Program (ESSP) and Ethiopian Development Research Institute (EDRI) with STATA.

Organization of the note

The module is organized into five separate sessions of one day long each. On day one, household demographic, food commodity and non-food commodity data files were created. Some basic data cleaning was also carried out. On day two, unique household identifiers are generated and data cleaning was done for food commodities. On day three, unique household identifiers are generated and data cleaning was done for non-food commodities. On day four, unique household identifiers are generated (and data cleaning was done) for household demographics. And finally, on day five of the training household demographic, food and non-food data files were merged and data transformations required for estimation of simple demand functions (such as computing food shares, per capital expenditure etc.) were done.

DAY ONE

Description

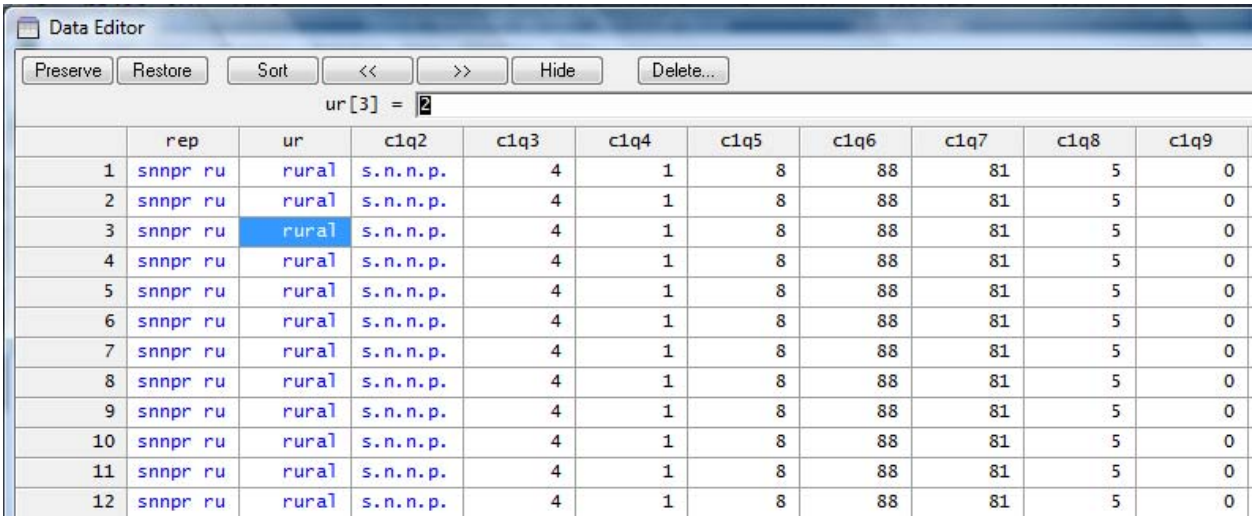
Day one's session aims at familiarizing participants with the basics of data management. In order to reduce the big (full) HICES file into manageable ones, the file is split into three separate files: household demographics, food commodity and non-food commodity data files. Household demographics data file includes key household identifier variables (key variables) such as region, rural/urban, zone, wereda etc.

The food commodity data file includes all household food expenditure items (fexp) with their corresponding key variables, while non-food commodity category includes household non-food expenditure items (nfexpe) along with their key variables. Inclusion of the key variables makes merging of these separate files easy at later stage.

`Data management.do`

***In splitting the large data file (cons2004_ict.dta) into three separate files (household demographics, food commodity and non-food commodity), we set about, in a way, to shift rows into columns. In the main data file, each row represents consumption item for a particular household, followed by a set of rows for another household. Our goal is to transform the structure of the data so that each row represents a particular household and columns represent consumption items as shown below.**

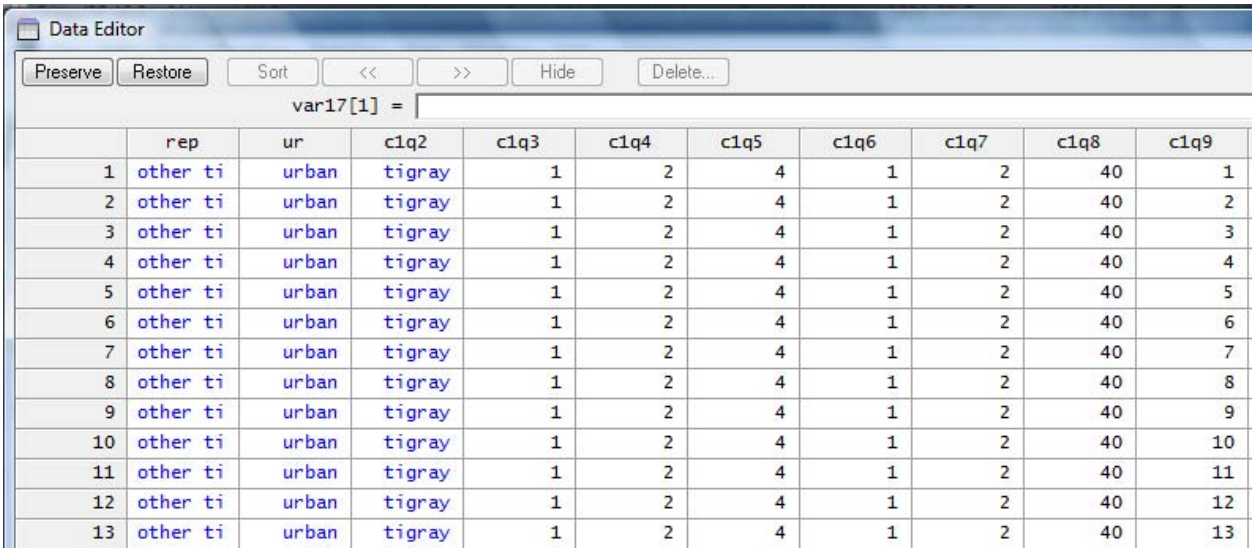
```
*"D:\EthiopiaDATA\CSA Survey data\WMS and HICES\HICES\Year 2004-05\do"
clear
set mem 500m
set more off
set type double
capture log close

gl dat "D:\EthiopiaDATA\CSA Survey data\WMS and HICES\HICES\Year 2004-05"
gl prg "D:\EthiopiaDATA\CSA Survey data\WMS and HICES\HICES\Year 2004-05\log"

*/
Note: gl= global macro
We don't not necessary need to define gl to specify our file path. One can
use the normal path to open his/her files. It is also possible to customaize
his/her gl based on the file path on his/her computer.
Alternatively, we also also use change directory
CD:\... command to open our file.
/*
log using "$prg\data.log", replace
/*
* ====================================================================
                        File details
File name "D:\EthiopiaDATA\CSA Survey data\WMS and HICES\HICES\Year 2004-
05\do

 Created by:          Nigussie Tefera
 Date first version:  27 March 2009
 Date this version:   10 April 2009

Purpose of file: Create hh demographic, food and non-food data using
 "cons2004_ict.dta"
*====================================================================
*/

** Step 1: Get the data
use "$dat\STATA\cons2004_ict.dta", clear
des
*===============Generate household level data
```



	rep	ur	c1q2	c1q3	c1q4	c1q5	c1q6	c1q7	c1q8	c1q9
1	snnpr ru	rural	s.n.n.p.	4	1	8	88	81	5	0
2	snnpr ru	rural	s.n.n.p.	4	1	8	88	81	5	0
3	snnpr ru	rural	s.n.n.p.	4	1	8	88	81	5	0
4	snnpr ru	rural	s.n.n.p.	4	1	8	88	81	5	0
5	snnpr ru	rural	s.n.n.p.	4	1	8	88	81	5	0
6	snnpr ru	rural	s.n.n.p.	4	1	8	88	81	5	0
7	snnpr ru	rural	s.n.n.p.	4	1	8	88	81	5	0
8	snnpr ru	rural	s.n.n.p.	4	1	8	88	81	5	0
9	snnpr ru	rural	s.n.n.p.	4	1	8	88	81	5	0
10	snnpr ru	rural	s.n.n.p.	4	1	8	88	81	5	0
11	snnpr ru	rural	s.n.n.p.	4	1	8	88	81	5	0
12	snnpr ru	rural	s.n.n.p.	4	1	8	88	81	5	0

*** As can be noted above, before the collapse command below, a single household (c1q9) appears in the data file a number of times since each row in the data file represent commodities.**

```
collapse (max)c1q10 (first)weight (min) c1q11,by(c1q2-c1q9 rep ur)
rename c1q10 hhsize
rename c1q11 hholdings

label var hhsize"Household size"
label var weight"Weight"
label var hholdings"Types of holdings(agricultural and non-agricultural)
order c1q2-c1q9 rep ur hhsize hholdings weight
save "$dat\STATA_RV\hhdemographic.dta", replace

use "$dat\STATA\cons2004_ict.dta", clear
```


Data Editor

Preserve | Restore | Sort | << | >> | Hide | Delete...

var17[1] =

	rep	ur	c1q2	c1q3	c1q4	c1q5	c1q6	c1q7	c1q8	c1q9
1	other ti	urban	tigray	1	2	4	1	2	40	1
2	other ti	urban	tigray	1	2	4	1	2	40	2
3	other ti	urban	tigray	1	2	4	1	2	40	3
4	other ti	urban	tigray	1	2	4	1	2	40	4
5	other ti	urban	tigray	1	2	4	1	2	40	5
6	other ti	urban	tigray	1	2	4	1	2	40	6
7	other ti	urban	tigray	1	2	4	1	2	40	7
8	other ti	urban	tigray	1	2	4	1	2	40	8
9	other ti	urban	tigray	1	2	4	1	2	40	9
10	other ti	urban	tigray	1	2	4	1	2	40	10
11	other ti	urban	tigray	1	2	4	1	2	40	11
12	other ti	urban	tigray	1	2	4	1	2	40	12
13	other ti	urban	tigray	1	2	4	1	2	40	13

***The collapse command reduces the size of the table so that a household shows up only once in the data file.**

```
*===============Generate food and non-food commodity data files
*Assume fexp=0 and fexp=. are used to represent non-food expenditure and then
recode fexp (0=.)
*88 changes made
*Similarly, assume nfexpe=0 and nfexpe=. are use to represent no non-food
expenditure and then
recode nfexpe(0=.)
*155 changes made

*Let's check items listed under food and non-food expenditures

*1: under food expenditure
*Expenditure on the following items are reported under food expenditure
column
*Decision: transfer their expenditure to non-food expenditure column

tab cq15 if fexp~=.
```

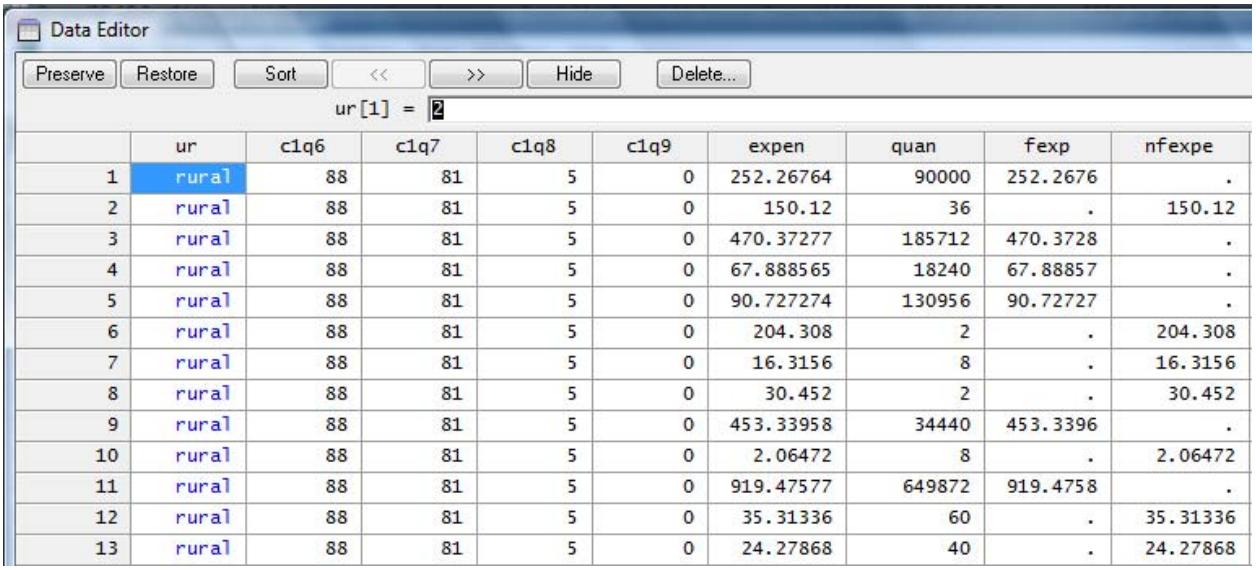
```
 replace nfexpe= fexp if ( /*
 */cq15== 20105|/*                  /*tetron*/
 */cq15== 20108|/*                  /*nylon*/
 */cq15== 20303|/*                  /*'netela'*/
 */cq15== 20375|/*                  /*pants*/
 */cq15== 20453|/*                  /*t-shirts*/
 */cq15== 20498|/*                  /*others*/
 */cq15== 20617|/*                  /*shirts*/
 */cq15== 20618|/*                  /*t-shirts*/
 */cq15== 20619|/*                  /*dress other than national*/
 */cq15== 20709|/*                  /*veil (shash)*/
 */cq15== 21101|/*                  /*tailoring and repairs*/
 */cq15== 30307|/*                  /*transportation cost(water fetc*/
 */cq15== 30402|/*                  /*firewood others*/
 */cq15== 30412|/*                  /*choping fire wood service*/
 */cq15== 40113|/*                  /*food safes*/
 */cq15== 40503|/*                  /*coffee pot*/
 */cq15== 40507|/*                  /*''mitad',traditional'*/
 */cq15== 40698|/*                  /*others*/
 */cq15== 40707|/*                  /*knives*/
 */cq15== 40801|/*                  /*cup*/
 */cq15== 40802|/*                  /*plates and dishes*/
 */cq15== 40803|/*                  /*jerry can*/
 */cq15== 41001|/*                  /*kerosine lamp & kuraz*/
 */cq15== 41002|/*                  /*flash light/torch light*/
 */cq15== 41010|/*                  /*needles & safety pins*/
 */cq15== 41298|/*                  /*others*/
 */cq15== 50501|/*                  /*doctor's fee*/
 */cq15== 60101|/*                  /*purchase of animal transport*/
 */cq15== 60102|/*                  /*purchase of private car*/
 */cq15== 60212|/*                  /*coolie charges*/
 */cq15== 72002|/*                  /*exercise books*/
 */cq15== 72003|/*                  /*papers and photocopying*/
 */cq15== 72004|/*                  /*notebooks*/
 */cq15== 72005|/*                  /*pencils*/
 */cq15== 72006|/*                  /*'pen, marker etc.'*/
 */cq15== 72009|/*                  /*files and carbones*/
 */cq15== 80109|/*                  /*belt*/
 */cq15== 80113|/*                  /*mirror*/
 */cq15== 80206|/*                  /*watches*/
 */cq15== 80301|/*                  /*hair dressing*/
 */cq15== 90308)&fexp~=.            /*religious contribution*/

*53 real changes made
* and
 recode fexp(0.00001/max=.) if ( /*
 */cq15== 20105|/*                  /*tetron*/
 */cq15== 20108|/*                  /*nylon*/
 */cq15== 20303|/*                  /*'netela'*/
 */cq15== 20375|/*                  /*pants*/
 */cq15== 20453|/*                  /*t-shirts*/
 */cq15== 20498|/*                  /*others*/
 */cq15== 20617|/*                  /*shirts*/
 */cq15== 20618|/*                  /*t-shirts*/
 */cq15== 20619|/*                  /*dress other than national*/
 */cq15== 20709|/*                  /*veil (shash)*/
```

```
 */cq15== 21101|/*                 /*tailoring and repairs*/
 */cq15== 30307|/*                 /*transportation cost(water fetc*/
 */cq15== 30402|/*                 /*firewood others*/
 */cq15== 30412|/*                 /*choping fire wood service*/
 */cq15== 40113|/*                 /*food safes*/
 */cq15== 40503|/*                 /*coffee pot*/
 */cq15== 40507|/*                 /*''mitad',traditional'*/
 */cq15== 40698|/*                 /*others*/
 */cq15== 40707|/*                 /*knives*/
 */cq15== 40801|/*                 /*cup*/
 */cq15== 40802|/*                 /*plates and dishes*/
 */cq15== 40803|/*                 /*jerry can*/
 */cq15== 41001|/*                 /*kerosine lamp & kuraz*/
 */cq15== 41002|/*                 /*flash light/torch light*/
 */cq15== 41010|/*                 /*needles & safety pins*/
 */cq15== 41298|/*                 /*others*/
 */cq15== 50501|/*                 /*doctor's fee*/
 */cq15== 60101|/*                 /*purchase of animal transport*/
 */cq15== 60102|/*                 /*purchase of private car*/
 */cq15== 60212|/*                 /*coolie charges*/
 */cq15== 72002|/*                 /*exercise books*/
 */cq15== 72003|/*                 /*papers and photocopying*/
 */cq15== 72004|/*                 /*notebooks*/
 */cq15== 72005|/*                 /*pencils*/
 */cq15== 72006|/*                 /*'pen, marker etc.'*/
 */cq15== 72009|/*                 /*files and carbones*/
 */cq15== 80109|/*                 /*belt*/
 */cq15== 80113|/*                 /*mirror*/
 */cq15== 80206|/*                 /*watches*/
 */cq15== 80301|/*                 /*hair dressing*/
 */cq15== 90308)&fexp~=.           /*religious contribution*/

*56 real changes made
*re-checking
tab cq15 if fexp~=.

*2: Non-food expenditure
tab cq15 if nfexpe~=.
*Expenditure on the following items are reported under non-food expenditure
column
*Note: the recorded values for all of them are zero except for barley for
beer
*Decision: transfer their expenditure to food expenditure column

*note:avocado/cocounat is non-food expenditure

replace fexp=nfexpe if ( /*
 */ cq15==109|/*                  /* barley for beer*/
 */ cq15==201|/*                  /* fenugreek*/
 */ cq15==208|/*                  /* fenu greek*/
 */ cq15==308|/*                  /* mutton*/
 */ cq15==802|/*                  /* cabbage*/
 */ cq15==1202|/*                 /* pepper green*/
 */ cq15==1209|/*                 /* orange powder*/
 */ cq15==1335|/*                 /* pepper whole*/
 */ cq15==1401|/*                 /* black pepper*/
 */ cq15==1403|/*                 /* long pepper*/
```

```
 */ cq15==1404|/*                    /* white cumin*/
 */ cq15==1405|/*                    /* black cumin*/
 */ cq15==1406|/*                    /* ginger*/
 */ cq15==1407|/*                    /* mixed spices*/
 */ cq15==1421|/*                    /* salt*/
 */ cq15==1701|/*                    /* barley for beer*/
 */ cq15==11106)& nfexpe~=.          /* mead/ honey wine*/

*34 real changes made

recode nfexpe(0.00001/max=.) if ( /*
 */ cq15==109|/*                     /* barley for beer*/
 */ cq15==201|/*                     /* fenugreek*/
 */ cq15==208|/*                     /* fenu greek*/
 */ cq15==308|/*                     /* mutton*/
 */ cq15==802|/*                     /* cabbage*/
 */ cq15==1202|/*                    /* pepper green*/
 */ cq15==1209|/*                    /* orange powder*/
 */ cq15==1335|/*                    /* pepper whole*/
 */ cq15==1401|/*                    /* black pepper*/
 */ cq15==1403|/*                    /* long pepper*/
 */ cq15==1404|/*                    /* white cumin*/
 */ cq15==1405|/*                    /* black cumin*/
 */ cq15==1406|/*                    /* ginger*/
 */ cq15==1407|/*                    /* mixed spices*/
 */ cq15==1421|/*                    /* salt*/
 */ cq15==1701|/*                    /* barley for beer*/
 */ cq15==11106)& nfexpe~=.                /* mead/ honey wine*/

*34 changes made
*re-checking
tab cq15 if nfexpe~=.
```

***To keep the original data file intact we save changes to a new file**

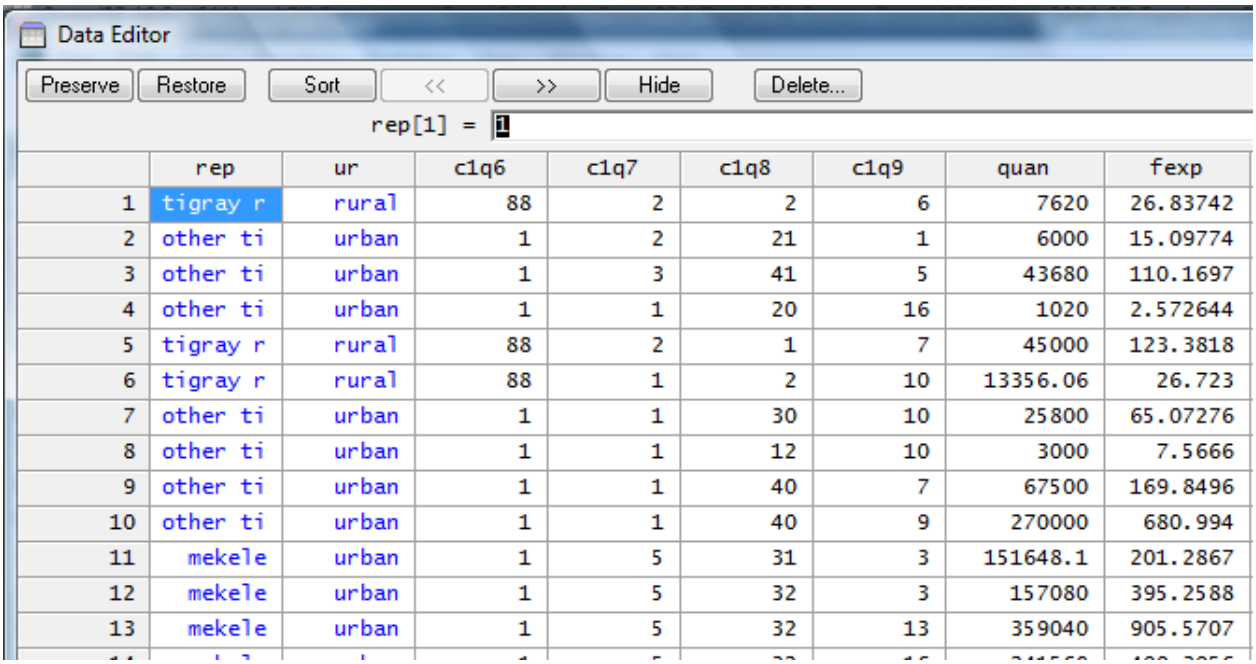
```
save "$dat\STATA\cons2004_ict_rv1.dta", replace

*Generate food commodity data
```



	ur	c1q6	c1q7	c1q8	c1q9	expen	quan	fexp	nfexpe
1	rural	88	81	5	0	252.26764	90000	252.2676	.
2	rural	88	81	5	0	150.12	36	.	150.12
3	rural	88	81	5	0	470.37277	185712	470.3728	.
4	rural	88	81	5	0	67.888565	18240	67.88857	.
5	rural	88	81	5	0	90.727274	130956	90.72727	.
6	rural	88	81	5	0	204.308	2	.	204.308
7	rural	88	81	5	0	16.3156	8	.	16.3156
8	rural	88	81	5	0	30.452	2	.	30.452
9	rural	88	81	5	0	453.33958	34440	453.3396	.
10	rural	88	81	5	0	2.06472	8	.	2.06472
11	rural	88	81	5	0	919.47577	649872	919.4758	.
12	rural	88	81	5	0	35.31336	60	.	35.31336
13	rural	88	81	5	0	24.27868	40	.	24.27868

***The command below drops all non-food items. An item can only be either food or non-food. As a result for a food item, its entry under nfexpe is missing and vice versa. All missing values are dropped, leaving only food items.**

```
use "$dat\STATA\cons2004_ict_rv1.dta", clear
keep if fexp!=.|(((cq15>=4 & cq15<=1406)|(cq15>=1421 & cq15<=11106)/*
 */ |cq15==41206)&fexp==.&nfexpe==.)
```



	rep	ur	c1q6	c1q7	c1q8	c1q9	quan	fexp
1	tigray r	rural	88	2	2	6	7620	26.83742
2	other ti	urban	1	2	21	1	6000	15.09774
3	other ti	urban	1	3	41	5	43680	110.1697
4	other ti	urban	1	1	20	16	1020	2.572644
5	tigray r	rural	88	2	1	7	45000	123.3818
6	tigray r	rural	88	1	2	10	13356.06	26.723
7	other ti	urban	1	1	30	10	25800	65.07276
8	other ti	urban	1	1	12	10	3000	7.5666
9	other ti	urban	1	1	40	7	67500	169.8496
10	other ti	urban	1	1	40	9	270000	680.994
11	mekele	urban	1	5	31	3	151648.1	201.2867
12	mekele	urban	1	5	32	3	157080	395.2588
13	mekele	urban	1	5	32	13	359040	905.5707

```
tab cq15
drop expen nfexpe c1q10 c1q11
des
*br
*tab cq15
save "$dat\STATA_RV\foodcommodity.dta", replace

*==============Generate non-food commodity
use "$dat\STATA\cons2004_ict_rv1.dta", clear
des
*br
keep if nfexpe!=.|(((cq15>=1408 & cq15<=1419)|(cq15>=11302 & cq15<=30498)/*
 */ |cq15==40403|cq15==41216|cq15==40113|cq15==90308)&fexp==.&nfexpe==.)
tab cq15
drop expen fexp c1q10 c1q11
drop  g_calo n_calo

save "$dat\STATA_RV\non-foodcommodity.dta", replace

log close
set more on
```

DAY TWO

Description

On day two of the training, unique household identifiers are created for the food commodities data file. The reason behind is that the data being commodity level, there are a number of zones in a region, a number of woredas in a zone, a number of towns in a woreda etc. therefore, in order to identify which household, in which ea, which fa etc, it is necessary to create unique household id.

The data also suffers from duplicate cases. These duplicates are dropped and further data cleaning is done to see if the reported data has expected properties.

Foodcomodity.do

```
*"D:\EthiopiaDATA\CSA Survey data\WMS and HICES\HICES\Year 2004-05\do"
clear
set mem 500m
set more off
set type double
capture log close

gl dat "D:\EthiopiaDATA\CSA Survey data\WMS and HICES\HICES\Year 2004-05"
gl prg "D:\EthiopiaDATA\CSA Survey data\WMS and HICES\HICES\Year 2004-05\log"

log using "$prg\foodcommodity.log", replace
/*
* ====================================================================

                      File details
 File name "D:\EthiopiaDATA\CSA Survey data\WMS and HICES\HICES\Year 2004-
05\do

 Created by:          Nigussie Tefera
 Date first version:  27 March 2009
 Date this version:   10 April 2009

 Purpose of file: Check data found in "foodcommodity.dta" by
                  (1) Constructing unique hh id
                  (2) Checking for duplicate observations
                  (3) Doing range checks
                  (4) Making corrections to data
                  (5) Creating new data file with corrected values
*====================================================================
*/

** Step 1: Get the data
use "$dat\STATA_RV\foodcommodity.dta", clear

** Step 2: Generate unique household identifier
/*
We need to know the maximum digits our unique id identifier variable may
contain(has)
in Stata "double" has maximum of 16 digits of accuracy (for numerical values)
```

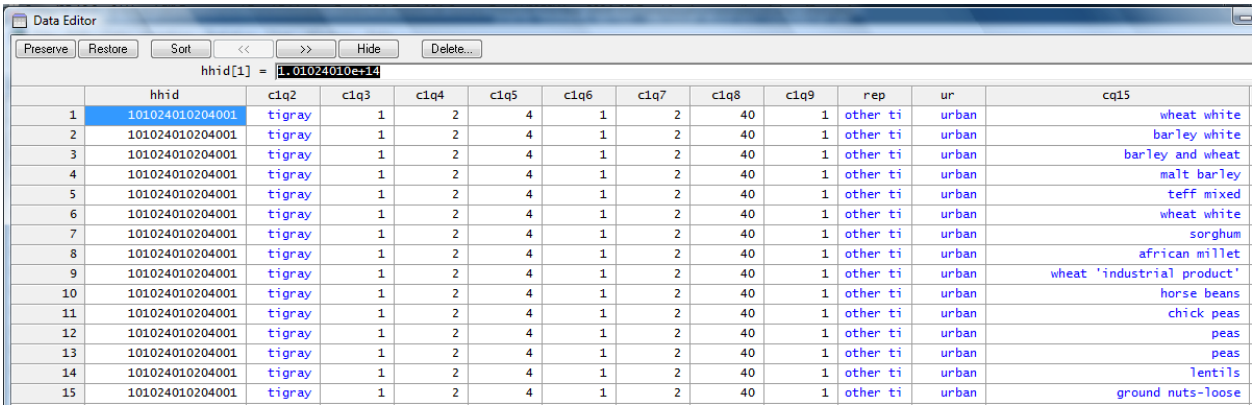
```
If we count and the total digits that our key variables have, it exceeded 16
digits
 (17 digits). so we need to recode as followes
*/

recode c1q7(101=82)(110=83)(113=84)(123=86)(126=94)

gen double hhid =c1q2*10^14+c1q3*10^12+c1q4*10^10/*
 */ +c1q5*10^9+c1q6*10^7+c1q7*10^5+c1q8*10^2+c1q9

format hhid %20.0f
format cq15 %20.0f
format quan %10.0f
format g_calo n_calo %10.0f

label var hhid"Unique household id"
order hhid c1q2-c1q9 rep ur

**Step 3:Checking for duplicated observations
egen rmiss=rmiss(c1q2-n_calo)
tab rmiss,m
*no observation for 3 cases but up to cq19
*Decision: drop them
drop if rmiss==8
*3 observations deleted
drop rmiss
*check for dupliacte observation
sort hhid cq15-n_calo
qui by hhid cq15-n_calo:gen dupobs=cond(_N==1,0,_n)
tab dupobs,m
*3 duplicate observation
*decision: drop the duplicated once
drop if dupobs>=2
*3 dublicated observation deleted
drop dupobs

** Step: 4 Data cleaning
*label items names using do file for coding
do "$dat\do\code.do"
label values cq15 code
tab cq15

/*
Initially, we thought that different values are given for the same item
listed below.
However, thanks to those who know the data structure well, we learned that
value from 1-100 are given for grains and the others (101-1000) for floor. We
decided to leave them as they were.
Moreover, if we are interested on aggregate figures we can run these commands
and once again re-check whether it has duplicate observation.

recode cq15(114=15)(40607 40739=40516)
recode cq15(110=11)(108 120 =8)(107=7)(40716 =40401)
recode cq15(302 402=202)(40220=40116)(40715=40506)
recode cq15(112=13)(40810=40609)(72503 72603=72106)(208=208)
recode cq15(40517=40113)(50607=50509)(40724=40514)(309 406=205)
recode cq15(20406=20322)(50606=50506)(301 401=201)
```

```
recode cq15(20304=40610)(50603=50503)(307 405=204)(2512=603)
recode cq15(111=12)(2605=2004)(304 1421=215)(20503 20306 20505=20303)
recode cq15(122=19)(2504=811)(50611=50511)(2101=1335)
recode cq15(20458=20624)(20375=20536)(303 403 =203)
recode cq15(20533 20623=20370)(113=14)(312=312)(211=209)(2511=601)
recode cq15(23531=20350)(102=2)(3 103=3)(101=1)(20403 20512 20606=20314)
recode cq15(305 404=207)(20909=20809)(20910=20810)
recode cq15(105=5)(703=702)(106 =6)(104=4)(20424=20353)(20429=20358)
*/
```

```
*/
All units of measurement were reported in gram and further we are not interested on source of expenditure and types of expenditure and conversion factors. Hence, we aggregate them as follows.
/*
```

***Now that unique household id is created, the commodities are labeled, and the data cleaned, running the 'collapse' command below produces a data file presents household id, labeled consumption item along with other variables.**

```
collapse (sum) quan fexp qt_epr g_calo n_calo /*
 */(mean)v_epr st_epr (first)weight, by(hhid c1q2- ur cq15)

label var quan"Total quantity (in gram)"
label var fexp"Annual food expenditure (in Birr)"
label var qt_epr"External quanitity"
label var g_calo"Gross calorie (in...)"
label var n_calo"Net calorie (in...)"
label var v_epr" External price"
label var st_epr"Standard price"
label var weight"Weight"
```



	hhid	c1q2	c1q3	c1q4	c1q5	c1q6	c1q7	c1q8	c1q9	rep	ur	cq15
1	101024010204001	tigray	1	2	4	1	2	40	1	other ti	urban	wheat white
2	101024010204001	tigray	1	2	4	1	2	40	1	other ti	urban	barley white
3	101024010204001	tigray	1	2	4	1	2	40	1	other ti	urban	barley and wheat
4	101024010204001	tigray	1	2	4	1	2	40	1	other ti	urban	malt barley
5	101024010204001	tigray	1	2	4	1	2	40	1	other ti	urban	teff mixed
6	101024010204001	tigray	1	2	4	1	2	40	1	other ti	urban	wheat white
7	101024010204001	tigray	1	2	4	1	2	40	1	other ti	urban	sorghum
8	101024010204001	tigray	1	2	4	1	2	40	1	other ti	urban	african millet
9	101024010204001	tigray	1	2	4	1	2	40	1	other ti	urban	wheat 'industrial product'
10	101024010204001	tigray	1	2	4	1	2	40	1	other ti	urban	horse beans
11	101024010204001	tigray	1	2	4	1	2	40	1	other ti	urban	chick peas
12	101024010204001	tigray	1	2	4	1	2	40	1	other ti	urban	peas
13	101024010204001	tigray	1	2	4	1	2	40	1	other ti	urban	peas
14	101024010204001	tigray	1	2	4	1	2	40	1	other ti	urban	lentils
15	101024010204001	tigray	1	2	4	1	2	40	1	other ti	urban	ground nuts-loose

```
** Step 5:Exploring the data (visiual inspection) for further data cleaning
des
count
count if ur==1
tab c1q2
tab ur
tabulate c1q2, plot
des hhid fexp
codebook ur
inspect fexp
```

```
sum fexp
sum fexp if fexp~=0==1,detail
histogram fexp if cq15==1,normal
graph box fexp if cq15==102, by(ur)
kdensity fexp if cq15==2
pnormal fexp if cq15==4
qnormal fexp if cq15==105
tabstat fexp,s(mean median sd cv min max) by(cq15)
table cq15 ur,c(mean fexp)

** step: 6 save file and close log file
save "$dat\STATA_RV\foodcommodity_rv1.dta", replace

use "$dat\STATA_RV\foodcommodity_rv1.dta", clear
```

DAY THREE

Description

On day three of the training, unique household identifiers are created for the non-food commodities data file created earlier. The rationale for doing so is that the data being commodity level, there are a number of zones in a region, a number of woredas in a zone, a number of towns in a woreda etc. therefore, in order to identify which household, in which ea, which fa etc, it is necessary to create unique household id.

The data also suffers from duplicate cases. These duplicates are dropped.

Non-foodcommodity.do

```
*"D:\EthiopiaDATA\CSA Survey data\WMS and HICES\HICES\Year 2004-05\do"
clear
set mem 500m
set more off
set type double
capture log close

gl dat "D:\EthiopiaDATA\CSA Survey data\WMS and HICES\HICES\Year 2004-05"
gl prg "D:\EthiopiaDATA\CSA Survey data\WMS and HICES\HICES\Year 2004-05\log"

log using "$prg\non-foodcommodity.log", replace
/*
* ====================================================================
                        File details
File name "D:\EthiopiaDATA\CSA Survey data\WMS and HICES\HICES\Year 2004-
05\do

 Created by:          Nigussie Tefera
 Date first version:  27 March 2009
 Date this version:   10 April 2009

 Purpose of file: Check data found in "foodcommodity.dta" by
                  (1) Constructing unique hh id
                  (2) Checking for duplicate observations
                  (3) Doing range checks
                  (4) Making corrections to data
                  (5) Creating new data file with corrected values
*====================================================================

*/

** Step 1: Get the data
use "$dat\STATA_RV\non-foodcommodity.dta", clear

** Step 2: Generate unique household identifier
*Note double has 16 digits of accuracy

recode c1q7(101=82)(110=83)(113=84)(123=86)(126=94)
```

```
gen double hhid =c1q2*10^14+c1q3*10^12+c1q4*10^10/*
 */ +c1q5*10^9+c1q6*10^7+c1q7*10^5+c1q8*10^2+c1q9

format hhid %20.0f
format hhid %20.0f
format cq15 %20.0f
format quan %10.0f

label var hhid"Unique household id"
order hhid c1q2-c1q9 rep ur

**Step 3:data cleaning
egen rmiss=rmiss(c1q2-st_epr)
tab rmiss,m
drop rmiss
*check for dupliacte observation
sort hhid cq15-st_epr
qui by hhid cq15-st_epr:gen dupobs=cond(_N==1,0,_n)
tab dupobs,m
*457 observations have duplication
*decision: drop duplicated observations
drop if dupobs>=2
*457 observation deleted
drop dupobs

** Step: 2 Data cleaning
*label values of item codes
do "$dat\do\code.do"
label values cq15 code

tab cq15

/*
The same items seem to be given different values. As we did for food items
we had discussion and agreed that the materials from which items made of
may vary. For instance, cups can be made of from plastic or stainless steel.
However, if we are not interested on that specification we can use "recode"
do file
aggregate the items
*/
*do "$dat\do\recode.do"


collapse (sum)quan nfexpe qt_epr /*
 */(mean)v_epr st_epr weight, by(hhid c1q2- ur cq15)
label var quan"Total quantity(in gram)"
label var nfexpe"Annual non-food expenditure (in Birr)"
label var qt_epr"External quanitity"
label var v_epr" External price"
label var st_epr"Standard price"
label var weight"Weight"

*step: 4 save and close log file
save "$dat\STATA_RV\non-foodcommodity_rv1.dta", replace

*aggregate all non-food-expenditure at hh level
egen tnfexpe=sum(nfexpe), by(hhid)
```

```
collapse (first)tnfexpe weight, by(hhid c1q2- ur)

label var tnfexpe"Total non-food expenditure (in Birr)"
label var weight"weight"

sort hhid
save "$dat\STATA_RV\tnonfoodexp.dta", replace

log close
set more on
```

DAY FOUR

Description

As a continuation to topics on day of the STATA training, based on the food commodity data file created then, fourteen food categories are generated. Then, weighted regional and woreda level prices are generated for use in demand estimation.
Moreover, on day three of the training, unique household identifiers are created for the household demographic data file. Checking for duplicates and data cleaning are also done. Finally, a data file with corrected values is created.

Foodcommodity.do

```
*Day 4: Stata training
*Food categories
recode cq15 (1/198=1)(201/498=2)(501/598=3)/*
 */(601/698 701/798=4)(801/898=5)/*
 */(1001/1098=6)(1101/1198=7)(1201/1398=8)(1401/1498=9)/*
 */(1501/1598=10)(1601/1698=11)( 41206 1701/1798 2101/2198=12)/*
 */(1801/1998 901/998 2501/2598 =13)/*
 */ (2001/2098 2201/2298 2601/2698 11001/11098 11101/11198=14)/*
 */ ,gen(foodcat)

*we classfied 2601/2698 under Beverages &alchols

label var foodcat"Food category by major components"
label define mcat 1"Cereals" 2"pluses" 3"Oil seeds"/*
 */ 4"prepared food &Bread" 5"Meat"/*
 */ 6"Milk, cheese and egg" 7"oils &fats"/*
 */ 8"Vegetbles&fruits" /*
 */ 9"Spcies"10"Potatoes and other tubers"/*
 */ 11"Coffe, tea &chat"/*
 */ 12"Other foods" 13"Restourants ahd hotels"/*
 */ 14"Beverages&alchols", modify
label value foodcat mcat

*Generate weighted prices at woreda and regional levels

egen con_woreda=sum(fexp), by(c1q2-c1q4 foodcat)
gen weightwp=fexp/con_woreda if con_woreda~=.
gen wprice_w=st_epr*weightwp

egen con_region=sum(fexp), by(c1q2 foodcat)
gen weightrp=fexp/con_region if con_region~=.
gen rprice_r=st_epr*weightrp

collapse (sum) quan fexp g_calo n_calo wprice_w/*
 */ rprice_r (first) weight, by(hhid-ur foodcat)

egen tfexp=sum(fexp), by(hhid)

label var quan"Total quantity (in gram)"
label var fexp"Annual food expenditure (in Birr)"
label var g_calo"Gross calorie (in...)"
```

```
label var n_calo"Net calorie (in...)"
label var weight"Weight"
label var wprice_w"Weighted price at woreda level"
label var rprice_r"Weighted price at regional level"
label var tfexp"Total food expenditure (in birr)"

sort hhid
save "$dat/STATA_RV/totalfoodexp.dta",replace

log close
set more on
```

Data Editor

Preserve | Restore | Sort | << | >> | Hide | Delete...

var14[4] =

	hhid	c1q6	c1q7	c1q8	c1q9	rep	ur	foodcat	wprice_w	rprice_r	weight	tfexp
1	101024010204001	1	2	40	1	other ti	urban	Cereals	.08351361	.00165717	1226.07	7444.3271
2	101024010204001	1	2	40	1	other ti	urban	pluses	.20442279	.00561437	1226.07	7444.3271
3	101024010204001	1	2	40	1	other ti	urban	Oil seeds	.49516825	.00215679	1226.07	7444.3271
4	101024010204001	1	2	40	1	other ti	urban	prepared food &Bread	.09162087	.00347975	1226.07	7444.3271
5	101024010204001	1	2	40	1	other ti	urban	Meat	.68360504	.02770741	1226.07	7444.3271
6	101024010204001	1	2	40	1	other ti	urban	Milk, cheese and egg	.21023908	.00966828	1226.07	7444.3271
7	101024010204001	1	2	40	1	other ti	urban	oils &fats	.4231701	.02000708	1226.07	7444.3271
8	101024010204001	1	2	40	1	other ti	urban	Vegetbles&fruits	.11548079	.00393757	1226.07	7444.3271
9	101024010204001	1	2	40	1	other ti	urban	Spcies	.27970455	.00944164	1226.07	7444.3271
10	101024010204001	1	2	40	1	other ti	urban	Potatoes and other tubers	.0312816	.00072513	1226.07	7444.3271
11	101024010204001	1	2	40	1	other ti	urban	Coffe, tea &chat	.40164659	.01322962	1226.07	7444.3271
12	101024010204001	1	2	40	1	other ti	urban	Other foods	.2540425	.01192195	1226.07	7444.3271
13	101024010204001	1	2	40	1	other ti	urban	Restourants ahd hotels	.01415791	.00143721	1226.07	7444.3271
14	101024010204001	1	2	40	1	other ti	urban	Beverages&alchols	.07192618	.00407045	1226.07	7444.3271
15	101024010204002	1	2	40	2	other ti	urban	Cereals	.02670971	.00053	1226.07	3685.2884

Hhdemographics.do

```
*"D:\EthiopiaDATA\CSA Survey data\WMS and HICES\HICES\Year 2004-05\do"
clear
set mem 500m
set more off
set type double
capture log close

gl dat "D:\EthiopiaDATA\CSA Survey data\WMS and HICES\HICES\Year 2004-05"
gl prg "D:\EthiopiaDATA\CSA Survey data\WMS and HICES\HICES\Year 2004-05\log"

log using "$prg\demographic.log", replace
/*
* ================================================================
                        File details
 File name "D:\EthiopiaDATA\CSA Survey data\WMS and HICES\HICES\Year 2004-
05\do

 Created by:          Nigussie Tefera
 Date first version:  27 March 2009
 Date this version:   10 April 2009

 Purpose of file: Check data found in "hhdemographic" by
                  (1) Constructing unique hh id
                  (2) Checking for duplicate observations
                  (3) Doing range checks
                  (4) Making corrections to data
                  (5) Creating new data file with corrected values
```

```
*==================================================================
*/

** Step 1: Get the data
use "$dat\STATA_RV\hhdemographic.dta", clear

** Step 2: Genenerate unique houshould id identifier
*Note double has 16 digits of accuracy

recode c1q7(101=82)(110=83)(113=84)(123=86)(126=94)

gen double hhid =c1q2*10^14+c1q3*10^12+c1q4*10^10/*
 */ +c1q5*10^9+c1q6*10^7+c1q7*10^5+c1q8*10^2+c1q9
format hhid %20.0f

label var hhid"Unique household id"
order hhid c1q2-c1q9 rep ur hhsize hholdings weight

** Step 3:data cleaning
egen rmiss=rmiss(hhsize-hholdings)
tab rmiss,m
drop rmiss
*check for dupliacte observation
sort hhid hhsize
qui by hhid hhsize:gen dupobs=cond(_N==1,0,_n)
tab dupobs,m
drop dupobs

save "$dat\STATA_RV\hhdemographic_rv1.dta", replace

log close
set more on
```

DAY FIVE

Description

On day five of the training the separate data files (household demographic, food commodity and non-food commodity) created and cleaned over the course of the earlier four days are merged and data transformations were done to ready the data for demand estimation. The transformations included computation food budget shares, per capita expenditure, and logarithmic transformation.

Moreover, identification of outliers in per capita food expenditure and replacement thereof of outliers by median is done. Finally, simple demand function is estimated and elasticities are computed.

Totalexpenditure.do

```
*"D:\EthiopiaDATA\CSA Survey data\WMS and HICES\HICES\Year 2004-05\do"
clear
set mem 500m
set more off
set type double
capture log close

gl dat "D:\EthiopiaDATA\CSA Survey data\WMS and HICES\HICES\Year 2004-05"
gl prg "D:\EthiopiaDATA\CSA Survey data\WMS and HICES\HICES\Year 2004-05\log"

log using "$prg\totalexp.log", replace
/*
* ================================================================
                        File details
 File name "D:\EthiopiaDATA\CSA Survey data\WMS and HICES\HICES\Year 2004-
05\do

 Created by:          Nigussie Tefera
 Date first version:  27 March 2009
 Date this version:   10 April 2009

 Purpose of file: merge data files and run simple demand function
*================================================================
*/

** Step 1: Get the data
use "$dat\STATA_RV\totalfoodexp.dta", clear
sort hhid
merge hhid using "$dat\STATA_RV\tnonfoodexp.dta"/*
 */"$dat\STATA_RV\hhdemographic_rv1.dta"

tab1 _m*
drop _m*

egen texp=rsum(tfexp tnfexp)
label var texp"total consumption expenditure"
gen tshare=tfexp/texp if texp~=0
label var tshare"Food consumption expenditure share in total expenditure"
```

```
*check for normality
histogram tfexp,normal
kdensity tfexp,normal

gen lnpc_texp=ln(texp/hhsize)
gen lnpc_tfexp=ln(tfexp/hhsize)
gen lnpc_quan=ln(quan/hhsize)
gen lnwprice_w=ln(wprice_w)
gen lnpc_fexp=ln(fexp/hhsize)

label var lnpc_texp"Log of per capita total expenditure"
label var lnpc_tfexp"Log of per capita total food expenditure"
label var lnpc_fexp"Log of per capita food expenditure"
label var lnpc_quan"Log of per capita quantity expenditure
label var lnwprice_w"Log of weight price at woreda lavel"

/*
* Once again, check for outliers
egen lnpc_median=median(lnpc_tfexp), by(ur)
egen sd=sd(lnpc_tfexp), by(ur)
gen ratio=(lnpc_tfexp-lnpc_median)/sd if sd~=0
gen outlier=(ratio>=3)

tab outlier
tabstat lnpc_tfexp[aw=weight] if outlier==0/*
 */ ,s(mean median sd cv min max) by(c1q2) format (%9.2fc)

tabstat lnpc_tfexp[aw=weight] if outlier==1/*
 */ ,s(mean median sd cv min max) by(c1q2) format (%9.2fc)

table ur [pw=weight] if outlier==1,c(n outlier) row

gen lnpc_tfexp1= lnpc_tfexp
replace lnpc_tfexp1=lnpc_median if outlier==1

tabstat lnpc_tfexp[aw=weight] if outlier==0/*
 */ ,s(mean median sd cv min max) by(c1q2) format (%9.2fc)

tabstat lnpc_tfexp[aw=weight]/*
 */ ,s(mean median sd cv min max) by(c1q2) format (%9.2fc)
*/

*Non-parametric test

twoway/*
 */(kdensity lnpc_texp[aw=weight], bw(0.2)/*
 */ legend(label(1 "All")))/*
 */(kdensity lnpc_texp[aw=weight] if ur==1, bw(0.2)/*
 */ legend(label(2 "urban")))/*
 */(kdensity lnpc_texp[aw=weight] if ur==2, bw(0.2)/*
 */ legend(label(3 "Rural")))

twoway/*
*/(lpoly share lnpc_texp [aw=weight] if ur==1,bw(0.2))/*
*/(lpoly share lnpc_texp [aw=weight] if ur==2, bw(0.2))

twoway/*
```

```
*/(lpoly tshare lnpc_texp [aw=weight] if c1q2==1,bw(0.2))/*
*/(lpoly tshare lnpc_texp [aw=weight] if c1q2==14,bw(0.2))/*
*/(lpoly tshare lnpc_texp [aw=weight] if c1q2==2,bw(0.2))/*
*/(lpoly tshare lnpc_texp [aw=weight] if c1q2==4,bw(0.2))/*
*/(lpoly tshare lnpc_texp [aw=weight] if c1q2==3, bw(0.2))
*/(lpoly tshare lnpc_texp [aw=weight] if c1q2==1,bw(0.2))/*

*simple demand function

xi:reg lnpc_quan lnpc_fexp lnwprice_w hhsize i.c1q2[aw=weight] if foodcat==1

foreach foodcat in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 {
 tab foodcat if foodcat==`foodcat'
 xi: regress lnpc_quan lnpc_fexp lnwprice_w hhsize i.c1q2[aw=weight]/*
 */ if foodcat==`foodcat'
 * Expenditure elasticity
  disp _b[lnpc_fexp ]
 * Price elasticity
 disp _b[lnwprice_w]
}

*Food items consumption expenditure share in total food consumption
expenditure

gen ishare=fexp/tfexp if tfexp~=.
egen avgishare= mean(ishare), by(foodcat)

foreach foodcat in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 {
 tab food if foodcat==`foodcat'
 quietly sum ishare if foodcat==`foodcat'
 local avgishare = r(mean)
 regress ishare lnpc_texp  lnwprice_w hhsize[aw=weight] if foodcat==`foodcat'
 * Average budget share
   disp `avgishare'
 * Expenditure elasticity
  disp 1+_b[lnpc_texp]/`avgishare'
 * Price elasticity
 disp _b[lnwprice_w]/`avgishare' - 1
}


*the end
log close
set more on
```

ANNEX

Reference Guide

for

Stata

Bingxin Yu

Development Strategy and Governance Division
International Food Policy Research Institute

Reference Guide for Stata

Introduction

Data Exploration

Analysis

Extensions

Chapter 1. Introduction

Why Stata?

Stata is a package that offers a good combination of ease to learn and power. It has numerous powerful yet simple commands for data management, which allows users to perform complex manipulations with ease. Under Stata/SE, one can have up to 32,768 in a Stata data file and 11,000 for any estimation commands.

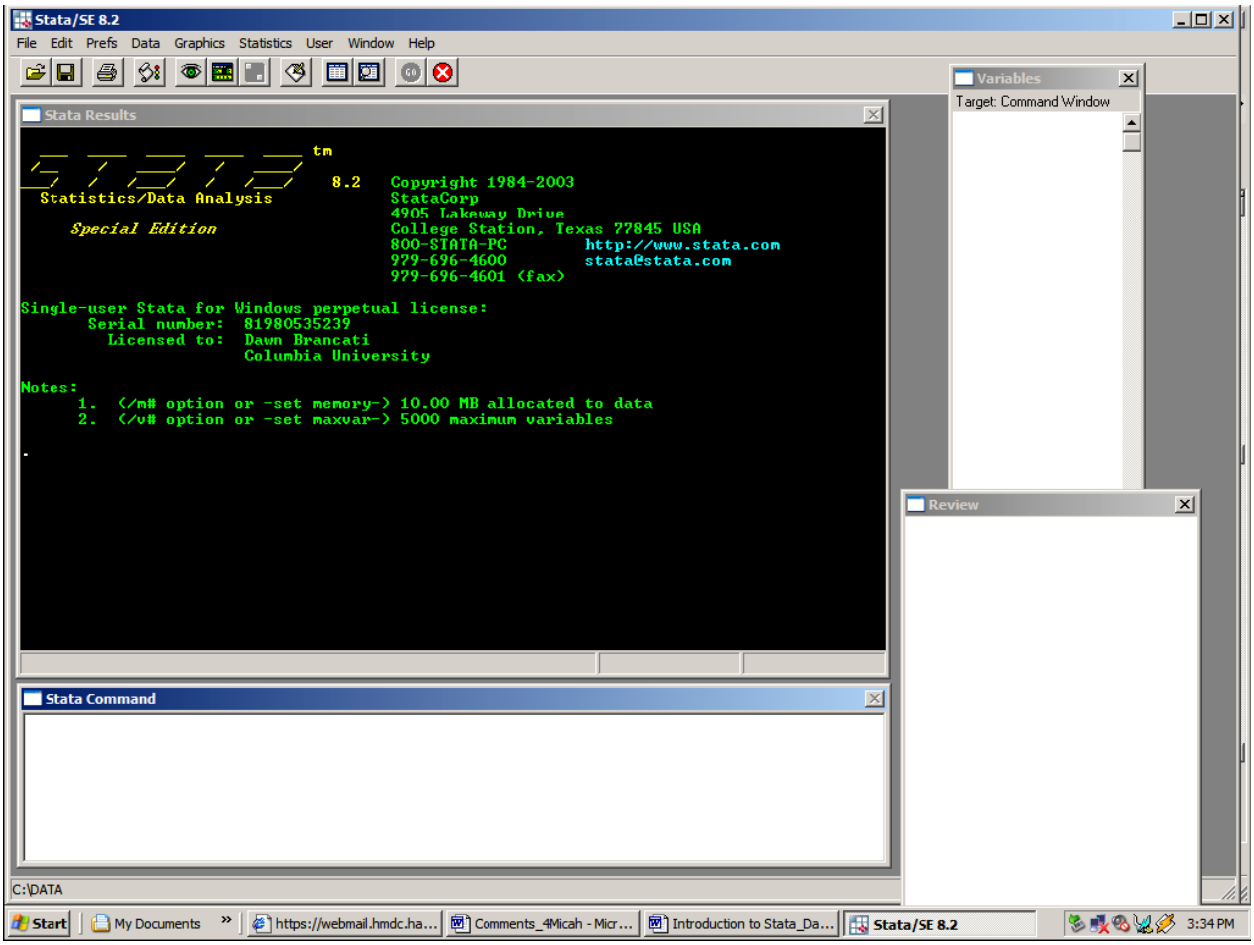
Stata performs most general statistical analyses (regression, logistic regression, ANOVA, factor analysis, and some multivariate analysis). The greatest strengths of Stata are probably in regression and logistic regression. Stata also has a very nice array of robust methods that are very easy to use, including robust regression, regression with robust standard errors, and many other estimation commands include robust standard errors as well.

Stata has the ability to easily download programs developed by other users and the ability to create your own Stata programs that seamlessly become part of Stata. One can find many cutting edge statistical procedures written by other users before and incorporate them into his/her own Stata program. Stata uses one line commands which can be entered one command at a time or can be entered many at a time in a Stata program.

The Stata interface

1. Windows

When Stata is running, there are a number of “windows” within Stata.



The **command** window on the bottom right is where you'll enter commands. When you press ENTER, they are pasted into the **Stata Results** window above, which is where you will see your commands execute and view the results. All results will be made to display in black Courier font.

On the left are two convenience windows. **Variables** window keeps a list of your current variables. If you click on one of them, its name will be pasted into the current command at the location of the cursor, which saves a little typing. The **Review** window keeps a list of all the commands you've typed this Stata session. Click on one, and it will be pasted into the command window, which is handy for fixing typos. Double-click, and the command will be pasted and re-executed. You can also export everything in the **Review** window into a .do file (more on them later) so you can run the exact same commands at any time. To do this right-click the **Review** window.

When we first open Stata, all these windows are blank except for the **Stata Results** window. You can resize these 4 windows independently, and you can resize the outer window as well. To save your window size changes, click on **Prefs** button, then Save Windowing Preferences

Entering commands in Stata works pretty much like you expect. BACKSPACE deletes the character to the left of the cursor, DELETE the character to the right, the arrow keys move the cursor around, and if you type the text is inserted at the current location of the cursor. The up

arrow does not retrieve previous commands, but you can do that by pressing PAGE UP, or CTRL-R, or by using the **Review** window.

2. Menus

Stata displays 9 drop-down menus across the top of the outer window, from left to right:
A. File
Open: open a Stata data file (use)
Save/Save as: save the Stata data in memory to disk
Do: execute a do-file
Filename: copy a filename to the command line
Print: print log or graph
Exit: quit Stata
B. Edit
Copy/Paste: copy text among the Command, Results, and Log windows
Copy Table: copy table from Results window to another file
Table copy options: what to do with table lines in Copy Table
C. Prefs - all Stata-related preferences
D. Data
E. Graphics
F. Statistics
build and run Stata commands from menus
G. User - menus for user-supplied Stata commands (download from Internet)
H. Window - bring a Stata window to the front
I. Help - Stata command syntax and keyword searches

3. Button bar

The buttons on the button bar are from left to right (equivalent command is in bold):
Open a Stata data file: **use**
Save the Stata data in memory to disk: **save**
Print a log or graph
Open a log, or suspend/close an open log: **log**
Open a new viewer
Bring Results window to front
Bring Graph window to front
New Dofile Editor: **doedit**
Edit the data in memory: **edit**
Browse the data in memory: **browse**
Scroll another page when --more-- is displayed: **Space Bar**
Stop current command or do-file: **Ctrl-Break**

Chapter 2. Getting Started

Directory commands

We begin by defining directory first.

The **pwd** command, which stands for print working directory, shows current directory you are in when Stata firs up.

```
. pwd
Y:\Stata9SE
```

The **cd** command stands for change directory, in this case, to change to the notes directory. The advantage of working from within a non-Stata directory is not only that Stata and your work are safe, but also you can use files without spelling out the full path, which can be quite handy.

```
. cd "u:\notes"
. pwd
u:\notes
```

The **log** command starts a log file called test1 that keeps a record of the commands and output during the Stata session.

```
. log using test1.txt
      log:  u:\notes\test1.txt
  log type:  smcl
 opened on:  11 Jul 2005, 14:57:25
```

The **log close** command closes and saves the current log file.

```
. log close
```

The log file, test1.txt, can be viewed with any text editor or word processor.

The **cd** command fixes current directory, but it might be inconvenient if we need to switch between Stata files under two or more different paths. We can use **global** command (which is actually a macro) to define each path and give us a shortcut in programming.

```
. global t "u:\notes"
. use $t\ethiopia.dta, clear
```

Do file

It is easier to collect all of your Stata commands used to perform a certain task together in one place, and do all the commands at once rather than one at a time. If modifications are needed,

you will have to start from scratch and try to remember you get so far in the first place. A do file allows users to keep track of pervious work and to make changes easily. Any command that you can type in on the command line can be place in a do file. There is a limit of 3500 lines to a do file.

Do files are created with the do file editor or any other text editor. Any command which can be executed from the command line can be placed in a do file. Here is a simple example of a do file:

```
cd u:\notes
log using test1.txt
pwd
log close
```

You can save this do file as e1.do in do file editor.

The **doedit** command can be used to brings up do file editor.

```
. doedit
. doedit e1.do
```

There are several ways to run a do file.

1. To use **do** command to executes the Stata commands in e1.do, display output.

```
. do e1.do
```

2. To use **run** command or press ctrl+D key to executes the Stata commands in e1.do, but display no output.

```
. run e1.do
```

If you would like to add document a do file, but do not want Stata to execute your notes, /* */ is used.

```
/* This Stata program illustrates how to read create a do file */
cd u:\notes
log using test1.txt
pwd
log close
```

Memory commands

Sometimes we might need extra memory to read a big data file.

First you can check to see how much memory is allocated to hold your data using the memory command. I am running Stata 9 under Windows, and this is what the **memory** command told me.

```
. memory
                                                    bytes
```

```
------------------------------------------------------------------------------
Details of set memory usage
    overhead (pointers)                                16        0.00%
    data                                               72        0.00%
                                          ---------------------------
    data + overhead                                    88        0.00%
    free                                       10,485,664      100.00%
                                          ---------------------------
    Total allocated                            10,485,752      100.00%
------------------------------------------------------------------------------
Other memory usage
    set maxvar usage                            1,816,666
    set matsize usage                           1,315,200
    programs, saved results, etc.                     509
                                          ---------------
    Total                                       3,132,375
-------------------------------------------------------
Grand total                                    13,618,127
```

I have 13 MB free for reading in a data file. I have a data file that's 33MB big that I want to read, which is beyond the Stata default memory allocated to me. Thus, I get the error message

```
no room to add more observations
r(901);
```

I will allocate 100 MB of memory with the **set memory** command before trying to use my file.

```
. set memory 100m

Current memory allocation

                    current                                 memory usage
    settable          value     description                 (1M = 1024k)
    ------------------------------------------------------------------
    set maxvar         5000     max. variables allowed           1.733M
    set memory          100M    max. data space                100.000M
    set matsize         400     max. RHS vars in models          1.254M
                                                             -----------
                                                               102.987M
```

Now that I have allocated enough memory, I will be able to read the file. If I want to allocate 100m (100 megabytes) every time I start Stata, I can type

```
. set memory 100m, permanently
```

And then Stata will allocate this amount of memory every time you start Stata.

Chapter 3. Input and Import Data

Create a data set

The **clear** command clears out the dataset that is currently in memory. We need to do this before we can create or read a new dataset.

```
. clear
```

One of the easiest methods for getting data into Stata is using the Stata data editor, which resembles an Excel spreadsheet. It is useful when your data is on paper and needs to be typed in, or if your data is already typed into an Excel spreadsheet. To learn more about the Stata data editor, see the edit module.

The **edit** command opens up a spreadsheet like window in which you can enter and change data. You can also get to the 'Data Editor' from the pull-down 'Window' menu or by clicking on the 'Data Editor' icon on the tool bar.

```
. edit
```

Enter values and press return. Double click on the column head and you can change the name of the variables. When you are done click the 'close box' for the 'Data Editor' window.

Another option is to use **input** command, then enter your own data set in command window or do file editor.

```
input a b c
1 2 3
4 5 6
7 8 9
end
```

Import external data set

Method 1. Use insheet command

The **insheet** command is used to read data from a comma separated or tab delimited file (.csv) created by a spreadsheet or database program. The values in the file must be either comma or tab delimited. The names are included in the file.

Consider the file comma.txt below that contains three variables, name, midterm, and final, separated by comma. The file looks like what is shown below (the variable names are indeed the first line of data.)

```
name,midterm,final
```

```
Smith,79,84
Jones,87,86
Brown,91,94
Adraktas,80,84
```

You can read this kind of file using the **insheet** command as shown below.

```
. clear
. insheet using comma.csv
```

We can issue the list command to see if the data was read properly.

```
. list
```

```
     +----------------------------+
     |     name   midterm   final |
     |----------------------------|
  1. |    Smith        79      84 |
  2. |    Jones        87      86 |
  3. |    Brown        91      94 |
  4. | Adraktas        80      84 |
     +----------------------------+
```

As you can see, the **insheet** command was pretty smart. It got the variable names from the first row of the data. It looks at the first row and can get the variable names from the first row. It also examines the file and determines for itself whether the data is separated by commas or by tabs. The exact same command could read the same file but delimited with tabs (you can try reading a file for yourself).

If the name of variables is not included in the data, you can still read data into Stata using insheet command.

```
. insheet name midterm final using comma.csv
```

Method 2. Use infile command

WE can also read data from an external ascii file using **infile** command. The names of the variables are given followed by the keyword using which in turn is followed by the name of the file. **str10** is not a variable name but indicates that name is a string variable up to 10 characters long.

The ASCII file called ascii.txt that looks like this:

```
"Smith"      79      84
"Jones"      87      86
"Brown"      91      94
"Adraktas"   80      84
```

We use commands below to check if the data is read properly:

```
. clear
. infile str10 name midterm final using ascii.txt
. list

     +---------------------------+
     |     name   midterm   final |
     |---------------------------|
  1. |    Smith        79      84 |
  2. |    Jones        87      86 |
  3. |    Brown        91      94 |
  4. | Adraktas        80      84 |
     +---------------------------+
```

To convert a SPSS file into Stata, in SPSS use “File Save As” to make a .csv file and then in Stata use the **insheet** command to read the .csv file in Stata. Another way to do it is to save an ascii text file and use **infile** command to import the data into Stata.

Method 3. use infix command

Consider a file using fixed column data like the one shown below.

```
AMC Concord   22 2930 4099
AMC Pacer     17 3350 4749
AMC Spirit    22 2640 3799
Buick Century 20 3250 4816
Buick Electra 15 4080 7827
```

Note that the variables are clearly defined by which column(s) they are located. The columns define where the make begins and ends, and the embedded spaces no longer create confusion.

This file can be read with the **infix** command as shown below.

```
. infix str make 1-13 mpg 15-16 weight 18-21 price 23-26 using
fixedcolumn.txt
 (5 observations read)
```

Here again we need to tell Stata that make is a string variable by preceding make with str. We did not need to indicate the length since Stata can infer that make can be up to 13 characters wide based on the column locations.

The **list** command confirms that the data was read correctly.

```
. list

     +-------------------------------------+
     |          make   mpg   weight   price |
     |-------------------------------------|
  1. |   AMC Concord    22     2930    4099 |
  2. |     AMC Pacer    17     3350    4749 |
```

```
  3. |    AMC Spirit    22     2640    3799 |
  4. | Buick Century    20     3250    4816 |
  5. | Buick Electra    15     4080    7827 |
     +--------------------------------------+
```

Open and Save a Stata dataset

The **use** command loads a Stata dataset into memory for use. Here we load a raw data set from HICES99 for illustration. The **clear** option allows Stata to clear the memory of previous data set in order to load the new one.

```
. use ethiopia.dta, clear
```

Stata did not want you to lose the changes that you made to the data sitting in memory. If you really want to discard the changes in memory, clear option specifies that it is okay to replace the data in memory, even though the current data have not been saved to disk.

The **save** command will save the dataset as a .dta file under the name your choose. Editing the dataset changes data in the computer's memory, it does not change the data that is stored on the computer's disk. Note that since I've defined my current directory by cd command, it is not necessary to specify the path while saving my data. But if I would like to save the data under a different directory, I have to put down the path clearly.

```
. save t1.dta
. save ..\plus\t1.dta
```

The **replace** option allows you to save a changed file to the disk, replacing the original file. Stata is worried that you will accidentally overwrite your data file. You need to use the **replace** option to tell Stata that you know that the file exists and you want to replace it.

```
. save t1.dta, replace
```

Chapter 4. Export Data

Since many different software packages (e.g. Power Point) read in Excel spreadsheets, we will focus on getting Stata results into Excel spreadsheets. Once you have your results in an Excel spreadsheet you are ready to either use Excel to format the table or create the graph, or bring the results into some other software. There are ways to create graphs and to format your results in Stata, but most people are already familiar with doing all that with one of these 4 software packages.

There are many ways to get your results out of Stata and into Excel. No matter which way you choose, it's always a good idea to check that the process did what you expected.

1. From the Stata results window, select text in Stata and paste into Excel. To copy an output table in Stata, select the table in Stata results window, right click (or Shift+Ctrl+C)to choose Copy Table, and paste into Excel.

2. From a log file, select text and paste into Excel. Or open the log file (*.log) in Excel as fixed width file. Make use of Stata's ability to start and stop logging to a log file so that your log file contains only one table of results. It's easier to have multiple log files with one table per file than multiple tables and other code in one big log file.

```
use "u:\notes\ethiopia.dta"
log using "test1.log", replace
collapse (mean) tot_exp, by(regco)
log close

log using "test2.log", replace
collapse (mean) tot_exp [pweight=weight], by(regco)
log close
```

3. From a Stata data set, use DBMScopy to convert a Stata data set to an Excel spreadsheet.

4. From a Stata data set, to use the **outsheet** command to output a spreadsheet that can be read into Excel.

```
. outsheet hhid regco tot_exp using excel1, replace
```

Outfile command exports a spreadsheet that can be read into Excel. The **wide** option causes Stata to write the data with one observation per line, which could be used to feed into other software.

```
. outfile hhid regco tot_exp using excel1, wide replace
```

Chapter 5. Examine Dataset

Stata syntax

Most Stata commands follow the same syntax:
[**by** varilist1:] **command** [varlist2] [**if** exp] [**in** range] [weight], [**options**]

Items inside of the squares brackets are either options or not available for every command. This syntax app applies to all Stata commands. In order to use **by** prefix, the dataset must first be sorted on the **by** variable(s).

The [**if** exp] can be complex, using & (and) and | (or) to join conditions together, like the example below.

```
. count if (urban==1) & missing(tot_exp)~=.
```

Logical operators used in Stata

~	not
==	equal
~=	not equal
!=	not equal
>	greater than
>=	greater than or equal
<	less than
<=	less than or equal
&	and
|	or

Note that == represents IS EQUAL TO.

Stata allows four kinds of weights in most commands:

1. **fweight**, or frequency weights, are weights that indicate the number of duplicated observations. It is used when your data set has been collapsed and contains a variable that tells the frequency each record occurred.

2. **pweight**, or sampling weights, are weights that denote the inverse of the probability that the observation is included due to the sampling design. **pweights** is correct to be used for sampling survey data. The **pweight** command causes Stata to use the sampling weight as the number of subjects in the population that each observation represents when computing estimates such as proportions, means and regressions parameters. A robust variance estimation technique will

automatically be used to adjust for the design characteristics so that variances, standard errors and confidence intervals are correct. For example,

```
. sum hhz_usu [pweight=samplewt]
```

3. **aweight**, or analytic weights, are weights that are inversely proportional to the variance of an observation; i.e., the variance of the j-th observation is assumed to be sigma^2/w_j, where w_j are the weights. Typically, the observations represent averages and the weights are the number of elements that gave rise to the average. For most Stata commands, the recorded scale of aweights is irrelevant; Stata internally rescales them to sum to N, the number of observations in your data, when it uses them.

Analytic weights are used when you want to compute a linear regression on data that are observed means. Do not use aweights to specify sampling weights. This is because the formulas that use aweights assume that larger weights designate more accurately measured observations. Conversely, one observation from a sample survey is no more accurately measured than any other observation. Hence, using the aweight command to specify sampling weights will cause Stata to estimate incorrect values of the variance and standard errors of estimates, and p-values for hypothesis tests.

4. **iweight**, or importance weights, are weights that indicate the "importance" of the observation in some vague sense. iweights have no formal statistical definition; any command that supports iweights will define exactly how they are treated. In most cases, they are intended for use by programmers who who need to implement their own analytical techniques by using some of the available estimation commands. Special care should be taken when using importance weights to understand how they are used in the formulas for estimates and variance. This information is available in the Methods and Formulas section in the Stata manual for each estimation command. In general, these formulas will be incorrect for computing the variance for data from a sample survey.

Stata accepts unambiguous abbreviations for commands and variable names. For example, we can just type

```
.des
```

Or

```
.d
```

To obtain the same output as use

```
. describe
```

Please refer to Stata manual for the exact rules of abbreviation.

Dataset examining commands

The **list** command without any variable names displays values of all the variables for all the cases. **list** command with variable names displays values of variables listed. in option gives the range of variables.

```
. list hhid urban
. list hhid tot_exp regco urban hhz_usu in 1/5
```

	hhid	tot_exp	regco	urban	hhz_usu
1.	101010888130501	5.572000027	1	0	2
2.	101010888130502	3.738346577	1	0	1
3.	101010888130503	13.01241112	1	0	5
4.	101010888130504	2.75808239	1	0	1
5.	101010888130505	4.473479271	1	0	2

. .

Here we look at *hhid, tot_exp, regco, urban, hhz_usu* for the first 5 observations.

Note about --more-- whenever it fills up the computer screen. Pressing the space bar will display the next screen, and so on, until all of the information has been displayed. To get out pf --more--, you can click on the “break” button, select “Break” from the pull-down “Tools” menu, or press the “q” key.

The if *exp* qualifier allows you to list values for those cases for which the *exp* is "true."

```
. list hhid if tot_exp==.
. list hhid urban tot_exp if regco==1
```

The first **list** displays all cases for which *tot_exp* is missing. Stata uses "." to indicate missing values. The if *regco*==1 only displays households in region 1.

The **browse** command is similar to **edit** , except that it will not allow users to change the data. The **browse** command is a convenient alternative to **list** command, but users can view the data through data editor instead of results window.

We can use the **describe** command to displays a basic summary of a Stata dataset, describing the number of observations in the file, the number of variables, the name of variables, and variable format and label.

```
. describe regco
. describe

Contains data from ethiopia.dta
  obs:        17,332
 vars:             5                          11 Jul 2005 15:24
 size:       762,608 (99.3% of memory free)
-------------------------------------------------------------------------------
```

```
              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
hhid            double %20.0f
tot_exp         double %12.0g                 Total daily expenditures
regco           double %12.0g                 Region
urban           double %12.0g                 Urban
hhz_usu         double %12.0g                 Number of usual household
                                                members
------------------------------------------------------------------------------
```

The **codebook** command is a great tool for getting a quick overview of the variables in the data file. It produces a kind of electronic codebook form the data file, displaying information about variables' names, labels and values.

```
. codebook
. codebook hhz_usu
```

```
-------------------------------------------------------------------------------
hhz_usu
Number of usual household members
-------------------------------------------------------------------------------

                  type:  numeric (double)

                 range:  [1,18]                       units:  1
         unique values:  18                       missing .:  0/17332

                  mean:    4.7466
             std. dev:   2.39059

           percentiles:        10%       25%       50%       75%       90%
                                 2         3         4         6         8
```

Another useful command for getting a quick overview of a data file the **inspect** command. **inspect** command displays information about the values of variables and is useful for checking data accuracy.

```
. inspect
. inspect hhz_usu
```

```
hhz_usu:  Number of usual household members    Number of Observations
------------------------------------------                     Non-
                                               Total   Integers    Integers
|  #                            Negative           -          -           -
|  #                            Zero               -          -           -
|  #   #                        Positive       17332      17332           -
|  #   #                                       -----      -----       -----
|  #   #                        Total          17332      17332           -
|  #   #   #   .   .            Missing            -
+----------------------
1                    18                        17332
  (18 unique values)
```

count command can be used to show the number of observations that satisfying if options. If no conditions are specified, count displays the number of observations in the data.

```
. count
17332

. count if hhz_usu>8
 1235
```

Chapter 6. Generate and Organize Variables

Create variables

The **generate** command is used to create a new variable.

```
. gen urbanrural="RURAL"
```

For existing variables, the **replace** command is needed to replace existing value of a variable with a new value.

```
. replace urbanrural="URBAN" if urban==2
```

egen stands for extended generate and is an extremely powerful command that has many options for creating new variables. It adds summary statistics to each observation. Although **egen** and **generate** commands look a like, they produce quite different results, as showed in the example below.

```
input hid iid income
1 1 1000
1 2 250
1 3 0
2 1 600
2 2 500
3 1 20000
end
generate hhincome=sum(income), by(hid)
egen hhincome=sum(income), by(hid)
bysort hid: gen hhincome1=sum(income)
list
```

```
     +-----------------------------------------+
     | hid   iid   income   hhincome   hhinco~1 |
     |-----------------------------------------|
  1. |   1     1     1000       1250       1000 |
  2. |   1     2      250       1250       1250 |
  3. |   1     3        0       1250       1250 |
  4. |   2     1      600       1100        600 |
  5. |   2     2      500       1100       1100 |
     |-----------------------------------------|
  6. |   3     1    20000      20000      20000 |
     +-----------------------------------------+
```

Here is a list of some of the options for **egen** command:

count	number of non-missing values
diff	compares variables, 1 if different, 0 otherwise
fill	fill with a pattern

group	creates a group id from a list of variables
iqr	interquartile range
ma	moving average
max	maximum value
mean	mean
median	median
min	minimum value
pctile	percentile
rank	rank
rmean	mean across variables
sd	standard deviation
std	standardize variables
sum	sums

Another approach to generate variables is to use **recode** command. The example makes a new variable called *grade* going from 1 to 5 based on student scores (0-100).

```
. gen grade=totavg
. recode grade 0/60=0 60/70=1 70/80=2 80/90=3 90/100=4
```

Modify variables

We can use **rename** command to rename a variable.

```
. rename hhz_usu hhsize
```

The **format** command allows you to specify the display format for variables. The internal precision of the variables is unaffected.

The syntax for format command is

```
. format varlist %fmt
```

where %fmt is listed below:

```
%fmt         description                              example
------------------------------------------------------------------------------
Right-justified formats
%#.#g        general numeric format                   %9.0g
%#.#f        fixed numeric format                     %9.2f
%#.#e        exponential numeric format               %10.7e
%d           default numeric elapsed date format      %d
%d...        user-specified elapsed date format       %dM/D/Y
%#s          string format                            %15s
```

```
Right-justified, comma formats
%#.#gc       general numeric format                  %9.0gc
%#.#fc       fixed numeric format                    %9.2fc

Leading-zero formats
%0#.#f       fixed numeric format                    %09.2f
%0#s         string format                           %015s

Left-justified formats
%-#.#g       general numeric format                  %-9.0g
%-#.#f       fixed numeric format                    %-9.2f
%-#.#e       exponential numeric format              %-10.7e
%-d          default numeric elapsed date format     %-d
%-d...       user-specified elapsed date format      %-dM/D/Y
%-#s         string format                           %-15s

Left-justified, comma formats
%-#.#gc      general numeric format                  %-9.0gc
%-#.#fc      fixed numeric format                    %-9.2fc

Centered formats
%~#s         string format (special)                 %~15s
-------------------------------------------------------------------------------
```

An example for format command is to keep *tot-exp* in 2-digit decimal.

. format tot_exp %10.2f

Label variables

Now let's include some variable labels so that we know a little more about the variables. The variable urban may be confusing since it is hard to tell what the 0s and 1s mean.

```
. use ethiopia.dta, clear
. codebook urban
-------------------------------------------------------------------------------
urban
Urban
-------------------------------------------------------------------------------
                  type:  numeric (double)

                 range:  [0,1]                        units:  1
         unique values:  2                        missing .:  0/17332

            tabulation:  Freq.  Value
                         8660  0
                         8672  1
```

We will use **label** command to add a brief definition for variable urban, and clearly indicate the 0s for rural households and 1s for urban households.

The **label variable** command makes labels that help explain individual variables.

```
. label variable urban "the location of the household"
```

The **label define** command creates a definition for the values 0 and 1 called *ul*.

```
. label define ul 1 urban 0 rural
```

The **label value** command connects the values defined for *ul* with the values in variables *urban*.

```
. label values urban ul
. codebook urban
-------------------------------------------------------------------------------
urban
the location of the household
-------------------------------------------------------------------------------
                  type:  numeric (double)
                 label:  ul

                 range:  [0,1]                        units:  1
         unique values:  2                        missing .:  0/17332

            tabulation:  Freq.   Numeric  Label
                          8660         0  rural
                          8672         1  urban
```

Please notice the difference in codebook output.

In the student grading example, we the values of grade are labeled A-F by **label define** and **label values.**

```
. label define abcdf 0 “F” 1 “D” 2 “C” 3 “B” 4 “A”
. label values grade abcdf
```

Chapter 7. Descriptive Statistics

Frequency table

The **tabulate** command is useful for obtaining frequency tables. Below we make a table for *regco*.

```
. tabulate regco
```

```
     Region |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |      1,250        7.23        7.23
          2 |        785        4.54       11.77
          3 |      3,327       19.24       31.00
          4 |      3,725       21.54       52.54
          5 |        847        4.90       57.44
          6 |        916        5.30       62.73
          7 |      2,639       15.26       77.99
         12 |        742        4.29       82.28
         13 |        725        4.19       86.48
         14 |      1,499        8.67       95.14
         15 |        840        4.86      100.00
------------+-----------------------------------
      Total |     17,295      100.00
```

The **tab1** command can be used as a shortcut to request one-way frequency tables for a series of variables, instead of typing the tabulate command over and over again.

```
. tab1 regco urban
```

We can also make crosstables using tabulate. Let's look at the repair history broken down by *regco* and *urban/rural*.

```
. tabulate regco urban
```

```
           |         Urban
    Region |         0          1 |     Total
-----------+----------------------+----------
         1 |       564        688 |     1,252
         2 |       392        400 |       792
         3 |     1,740      1,600 |     3,340
         4 |     1,824      1,904 |     3,728
         5 |       372        480 |       852
         6 |       516        400 |       916
         7 |     1,872        768 |     2,640
        12 |       360        384 |       744
        13 |       360        368 |       728
        14 |       300      1,200 |     1,500
        15 |       360        480 |       840
-----------+----------------------+----------
     Total |     8,660      8,672 |    17,332
```

With the **column** option, we can request column percentages. Notice that about 21.96% of urban households lives in *regco* 4.

```
. tabulate strata urban, column
```

```
+-------------------+
| Key               |
|-------------------|
|     frequency     |
| column percentage |
+-------------------+

           |         Urban
    Region |         0          1 |     Total
-----------+----------------------+----------
         1 |       564        688 |     1,252
           |      6.51       7.93 |      7.22
-----------+----------------------+----------
         2 |       392        400 |       792
           |      4.53       4.61 |      4.57
-----------+----------------------+----------
         3 |     1,740      1,600 |     3,340
           |     20.09      18.45 |     19.27
-----------+----------------------+----------
         4 |     1,824      1,904 |     3,728
           |     21.06      21.96 |     21.51
-----------+----------------------+----------
         5 |       372        480 |       852
           |      4.30       5.54 |      4.92
-----------+----------------------+----------
         6 |       516        400 |       916
           |      5.96       4.61 |      5.29
-----------+----------------------+----------
         7 |     1,872        768 |     2,640
           |     21.62       8.86 |     15.23
-----------+----------------------+----------
        12 |       360        384 |       744
           |      4.16       4.43 |      4.29
-----------+----------------------+----------
        13 |       360        368 |       728
           |      4.16       4.24 |      4.20
-----------+----------------------+----------
        14 |       300      1,200 |     1,500
           |      3.46      13.84 |      8.65
-----------+----------------------+----------
        15 |       360        480 |       840
           |      4.16       5.54 |      4.85
-----------+----------------------+----------
     Total |     8,660      8,672 |    17,332
           |    100.00     100.00 |    100.00
```

Since we are more interested in the percentages instead of the frequencies, **nofreq** option is used to suppress the frequencies.

```
. tabulate regco urban, column nofreq
```

```
           |         Urban
    Region |         0          1 |     Total
-----------+----------------------+----------
         1 |      6.51       7.93 |      7.22
         2 |      4.53       4.61 |      4.57
         3 |     20.09      18.45 |     19.27
         4 |     21.06      21.96 |     21.51
         5 |      4.30       5.54 |      4.92
         6 |      5.96       4.61 |      5.29
         7 |     21.62       8.86 |     15.23
        12 |      4.16       4.43 |      4.29
        13 |      4.16       4.24 |      4.20
        14 |      3.46      13.84 |      8.65
        15 |      4.16       5.54 |      4.85
-----------+----------------------+----------
     Total |    100.00     100.00 |    100.00
```

We can use the **plot** option to make a plot to visually show the tabulated values.

```
. tabulate regco, plot
```

```
   Region |      Freq.
----------+------------+----------------------------------------------------
        1 |      1,252 |******************
        2 |        792 |***********
        3 |      3,340 |************************************************
        4 |      3,728 |*****************************************************
        5 |        852 |************
        6 |        916 |*************
        7 |      2,640 |*************************************
       12 |        744 |***********
       13 |        728 |**********
       14 |      1,500 |*********************
       15 |        840 |************
----------+------------+----------------------------------------------------
    Total |     17,332
```

Suppose we want to focus on just the households with household income less than 1000. We can combine **if** suffix and **tabulate** command to do it.

```
. tabulate regco urban if (tot_exp<1000), column nofreq
```

```
           |         Urban
    Region |         0          1 |     Total
-----------+----------------------+----------
         1 |      6.53       7.92 |      7.23
         2 |      4.46       4.62 |      4.54
         3 |     20.01      18.46 |     19.24
         4 |     21.10      21.97 |     21.54
         5 |      4.27       5.52 |      4.90
         6 |      5.98       4.62 |      5.30
         7 |     21.68       8.86 |     15.26
        12 |      4.17       4.41 |      4.29
```

```
        13 |      4.15       4.24 |      4.19
        14 |      3.47      13.84 |      8.67
        15 |      4.17       5.54 |      4.86
-----------+---------------------+----------
     Total |    100.00     100.00 |    100.00
```

Stata has two built-in variables called _n and _N. _n is Stata notation for the current observation number. Thus _n=1 in the first observation, and _n=2 in the second, and so on. _N is Stata notation for the total number of observations. Let's see how _n and _N work.

```
. clear
. input score group
         score      group
  1. 72 1
  2. 84 2
  3. 76 1
  4. 89 3
  5. 82 2
  6. 90 1
  7. 85 1
. end
. generate id=_n
. generate nt=_N
. list
     +-------------------------+
     | score   group   id   nt |
     |-------------------------|
  1. |    72       1    1    7 |
  2. |    84       2    2    7 |
  3. |    76       1    3    7 |
  4. |    89       3    4    7 |
  5. |    82       2    5    7 |
     |-------------------------|
  6. |    90       1    6    7 |
  7. |    85       1    7    7 |
     +-------------------------+
```

As you can see, the variable *id* contains observation number running from 1 to 7 and *nt* is the total number of observations, which is 7 for all observations.

Summary statistics

For summary statistics, **summarize** command is the mostly used. Let's generate some summary statistics on *tot_exp*.

```
. summarize tot_exp

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
     tot_exp |     17295    17.25426    18.50382   .6677808   602.4645
```

To get these values separately for urban and rural households, we could use the **by urban:** prefix as shown below. Note that we first have to sort the data by urban before using the prefix.

```
. sort urban
. by urban: summarize tot_exp
------------------------------------------------------------------------------
-> urban = 0

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
     tot_exp |      8634    13.14905    11.58237   .6677808   561.1172

------------------------------------------------------------------------------
-> urban = 1

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
     tot_exp |      8661    21.34667    22.72598   .9592329   602.4645
```

Suppose we do the summarization for each *regco*, this is not the most efficient way to do it with long output. Another concise way, which does not require the data to be sorted, is by using the **summarize()** option as part of the tabulate command.

```
. tabulate regco, summarize(tot_exp)
```

Region	Summary of Total daily expenditures Mean	Std. Dev.	Freq.
1	16.034104	14.349284	1250
2	16.21084	28.405333	785
3	15.666401	16.380004	3327
4	16.16371	13.136783	3725
5	18.827462	13.250171	847
6	15.674472	16.408756	916
7	14.505694	13.335531	2639
12	15.203531	12.665174	742
13	19.667951	11.867981	725
14	29.266701	36.231542	1499
15	18.233438	14.683283	840
Total	17.254261	18.503822	17295

We can use **if** and **by** with most Stata commands. Here, we get summary statistics for *tot_exp* for households in *regco* 1.

```
. summarize tot_exp if regco==1 & urban==0

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
     tot_exp |       564    12.20058    8.074085   1.720451   134.4393
```

ATTENTION!

Missing values are represented as . and are the highest value possible. Most commands ignore missing values by default. Some commands, such as **tabulate**, have an option to display missing if you want to see how many missing observations there are. Therefore, when values are missing, be careful with commands like **summarize** and **tabulat**e. To avoid this problem, use **missing** option to treat missing values like other values in tabulate command.

```
. tabulate regco urban, column nofreq missing
```

In **summarize** command, use !missing() option to omit missing values.

```
. summarize tot_exp if hhz_usu~=.
```

Other commands, however, may use missing values in a way that will surprise you. For example, the **replace** command does not ignore missing values. Here is a simple example to demonstrate how **replace** commands handle missing values differently. In this example, we have variable *income* with one missing value. We want to generate an index variable at cutting value of 500.

```
Clear
input hid income
1 1000
2 450
3 .
4 700
5 500
end
. gen cut1=0 if income<=500
(3 missing values generated)
. replace cut1=1 if income>500
(3 real changes made)

. gen cut2=0 if income<=500
(3 missing values generated)
. replace cut2=1 if income>500 & income<.
(2 real changes made)

. recode income (min/500 = 0) (501/max =1), generate(cut3)
(4 differences between income and cut3)

. list

     +-----------------------------------+
     | hid   income   cut1   cut2   cut3 |
     |-----------------------------------|
  1. |   1     1000      1      1      1 |
  2. |   2      450      0      0      0 |
  3. |   3        .      1      .      . |
  4. |   4      700      1      1      1 |
  5. |   5      500      0      0      0 |
     +-----------------------------------+
```

The first **replace** command changes every *income* value that's greater than 500 to 1. This command does not ignore missing values, so both *income* greater than 500 and missing values are changed to 1. This probably is not what we would normally want to do, since missing values should remain missing.

The second **replace** command changes all *income* values greater than 500 but not missing to 1. In this case no-missing values greater than 500 are changed and missing values are not changed, which is our intention.

The **recode** command automatically ignores missing values, so we don't have to think about it. The results are the same as the second **replace** command.

Advanced statistics

The **table** command calculates and displays tables of statistics, including frequency, mean, standard deviation, sum, and 1st to 99th percentile. The **row** and **col** option specifies an additional row and column to be added to the table, reflecting the total across rows and columns.

The example lists a two-way table of median of *tot_exp* (50th percentile) by *urban* and *femhead*.

```
. table urban femhead [pweight=samplewt], contents(p50 tot_exp) row col
missing

----------------------------------------
          |     Female headed household
    Urban |         0          1      Total
----------+-----------------------------
        0 | 11.713764  7.9488297  10.741507
        1 | 17.312714  11.005945  14.391151
          |
    Total | 12.156257  8.5248489  11.194192
----------------------------------------
```

The **tabstat** command displays summary statistics for a series of numeric variables in a single table.

```
. tabstat tot_exp exp_food hhz_usu agehhh, statistics(mean) by(urban) missing

Summary statistics: mean
  by categories of: urban (Urban)

   urban |   tot_exp  exp_food   hhz_usu    agehhh
---------+----------------------------------------
       0 |  13.14905  8.037849  4.953903  43.65682
       1 |  21.34667  9.699733  4.550745  43.50572
---------+----------------------------------------
   Total |  17.25426  8.870088  4.752009  43.58115
------------------------------------------------
```

Sometimes we have data files that need to be aggregated at a higher level to be useful for us. For example, we have household data but we really interested in regional data. The **collapse** command serves this purpose by converting the dataset in memory into a dataset of means, sums, medians and percentiles. Note that the **collapse** command creates a new dataset and all household information disappear and only the specified variable aggregation remain at the region level. The resulting summary table can by viewed by **edit** command.

We would like to see the mean *tot_exp* in each *regco* and urban/rural areas.

```
. collapse (mean) tot_exp [pweight=samplewt], by(regco urban)
. edit

regco urban tot_exp
1     0     12.067
```

```
1     1     14.899
2     0     13.022
2     1     17.849
3     0     11.612
3     1     16.507
4     0     13.324
4     1     17.790
5     0     15.152
5     1     22.627
6     0     11.890
6     1     18.261
7     0     12.313
7     1     18.591
12    0     10.851
12    1     19.714
13    0     19.528
13    1     20.021
14    0     21.568
14    1     30.597
15    0     16.627
15    1     19.574
```

However, this table is not easy to interpret, and we can call it a long format since the data of urban and rural are vertically listed. We will use **reshape** command to convert it into a wide form where the rural and urban are horizontally arranged in a twoway table. The **reshape wide** command tells system that we want to go from long to wide. The **i()** option records row variable while **j()** column variable.

```
. reshape wide tot_exp, i(regco) j(urban)
(note: j = 0 1)

Data                               long   ->   wide
-----------------------------------------------------------------------------
Number of obs.                       22   ->      11
Number of variables                   3   ->       3
j variable (2 values)            urban   ->   (dropped)
xij variables:
                                 tot_exp   ->   tot_exp0 tot_exp1
-----------------------------------------------------------------------------
```

The converted table is two-way table.

```
regco tot_exp0    tot_exp1
1     12.067      14.899
2     13.022      17.849
3     11.612      16.507
4     13.324      17.790
5     15.152      22.627
6     11.890      18.261
7     12.313      18.591
12    10.851      19.714
13    19.528      20.021
14    21.568      30.597
15    16.627      19.574
```

If needed, the table can be converted back into the long form by **reshape long**.

```
. reshape long tot_exp, i(regco)
```

The **collapse** and **reshape** commands are examples of the power and simplicity of Stata in its ability to shape data files.

Chapter 8. Normality and Outlier

Check for normality

An outlier is an observation that lies an abnormal distance from other values in a random sample from a population. We must be extremely mindful of possible outliers and their adverse effects during any attempt to measure the relationship between two continuous variables.

There are no official rules to identify outliers. In a sense, this definition leaves it up to the analyst (or a consensus process) to decide what will be considered abnormal. Sometimes it is obvious when an outlier is simply miscoded (for example, age reported as 230) and hence should be set to missing. But most times it is not the case.

Before abnormal observations can be singled out, it is necessary to characterize normal observations.

Skewness is a measure of symmetry, or more precisely, the lack of symmetry. A distribution, or data set, is symmetric if it looks the same to the left and right of the center point. If the coefficient of skewness is 0, the distribution is symmetric. If the coefficient is negative, the median is usually greater than the mean and the distribution is said to skewed left (the left tail is longer than the right tail). If the coefficient is positive, the median is usually less than the mean and the distribution is said to be skewed right (the right tail is longer than the left tail). The skewness for a normal distribution is zero and any symmetric data should have a skewness near zero. A value greater than 1 on skewness coefficient indicates a serious asymmetry.

Kurtosis is a measure of whether the data are peaked or flat relative to a normal distribution. The smaller the coefficient of kurtosis is, the flatter the distribution. That is, data sets with high kurtosis tend to have a distinct peak near the mean, and low kurtosis data tend to have a flat top near the mean rather than a sharp peak. A uniform distribution would be the extreme case. The standard normal distribution has a kurtosis coefficient of 3. A value of 6 or larger on kurtosis coefficient indicates a large departure from normality.

We can obtain skewness and kurtosis values by using detail option in **summarize** command. Clearly, variable *tot_exp_pc* is skewed to the right and has a peaked distribution. But both statistics indicate the distribution of *tot_exp_pc* is far from normal. Remember that missing values are represented as . and are the highest value possible. Those observations need to be dropped first.

```
. cd "u:\notes"
. use ethiopia.dta, clear
. drop if tot_exp_pc==.
(37 observations deleted)

. sum tot_exp_pc
```

```
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
  tot_exp_pc |     17295    4.123542    4.136962   .4008128   122.3926
```

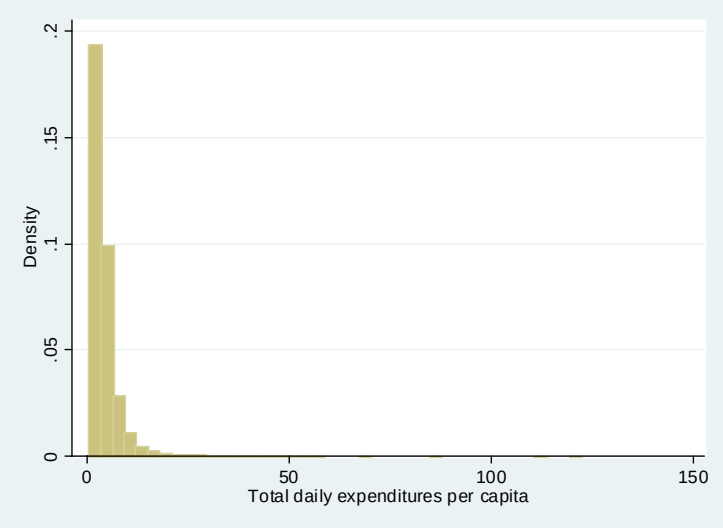
```
. summarize tot_exp_pc, detail
```

```
            Total daily expenditures per capita
-------------------------------------------------------------
      Percentiles      Smallest
 1%     .9142123       .4008128
 5%     1.265692       .4301129
10%     1.540479       .4934749       Obs                17295
25%     2.101912       .5245371       Sum of Wgt.        17295

50%     3.000105                      Mean           4.123542
                        Largest       Std. Dev.      4.136962
75%     4.630802       69.40652
90%     7.554192       86.06636       Variance       17.11445
95%     10.25375       112.2234       Skewness       6.894294
99%     20.71157       122.3926       Kurtosis       108.0724
```

Besides commands for descriptive statistics, such as **summarize**, we can also check normality of a variable visually by looking at some basic graphs in Stata, including **histograms, boxplots, kdensity, pnorm,** and **qnorm**. Let's keep using *tot_exp_pc* from ethiopia.dta file for making some graphs.

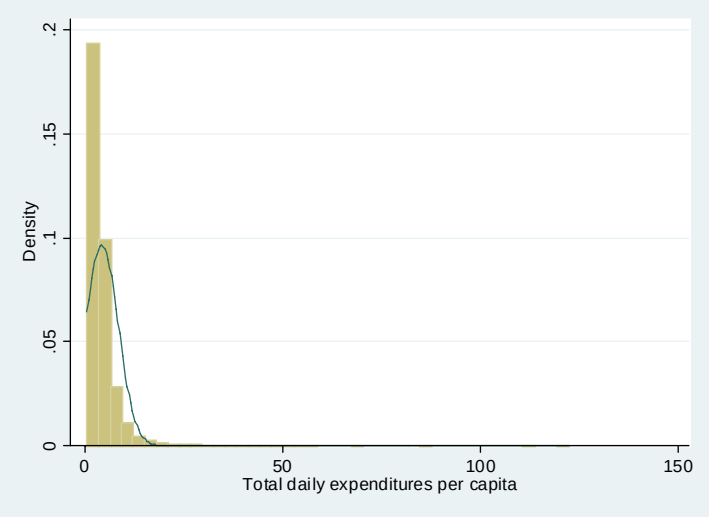
The histogram command is an effective graphical technique for showing both the skewness and kurtosis of *tot_exp_pc.*

```
. histogram tot_exp_pc
```



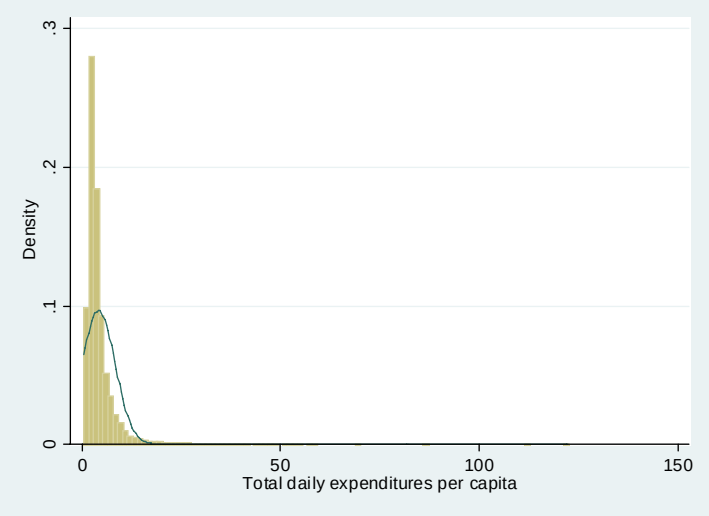
The normal option can be used to get a normal overlay. This shows the skew to the left in *tot_exp_pc*.

```
. histogram tot_exp_pc, normal
```



We can use the bin() option to increase the number of bins to 100. This better illustrates the distribution of *tot_exp_pc*. This option specifies how to aggregate data into bins. Notice that the histogram resembles a bell shape curve, but truncated at 0.

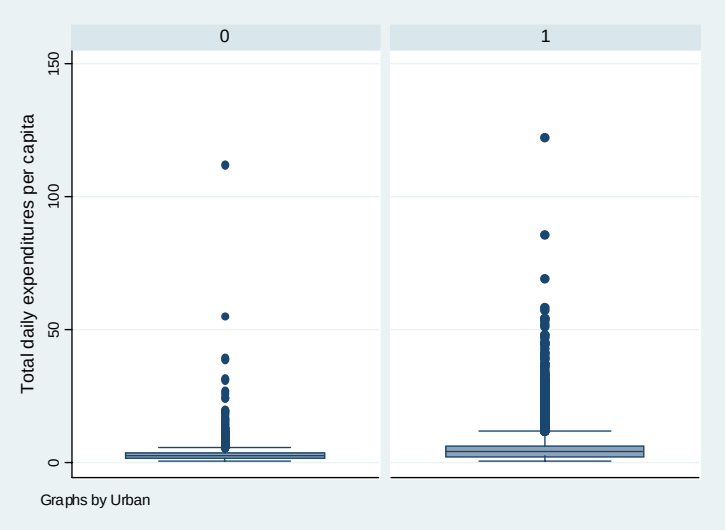
```
. histogram tot_exp_pc, normal bin(100)
```



graph box draws vertical box plots. In a vertical box plot, the y axis is numerical, and the x axis is categorical. The upper and lower bounds of box are defined by the 25th and 75th percentiles of *tot_exp_pc*, and the line within the box is the median. The ends of the whiskers are 5th and 95th percentile of *tot_exp_pc*. **graph box** command can be used to produce a boxplot which can help us examine the distribution of *tot_exp_pc*. If *tot_exp_pc* is normal, the median would be in the center of the box and the end of whiskers would be equidistant from the box.

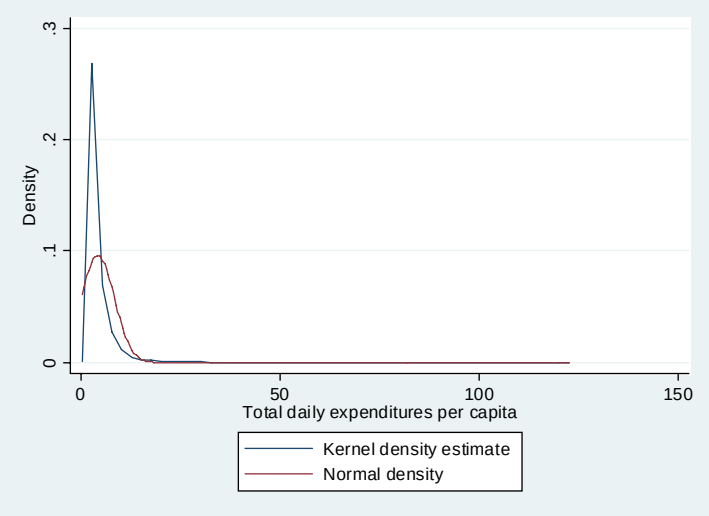
```
. graph box tot_exp_pc, by(urban)
```

The boxplot for *tot_exp_pc* shows positive skew. The median is pulled to the low end of the box, and the 95th percentile is stretched out away from the box, in both rural and urban cases.



The **kdensity** command with the normal option displays a density graph of the residual with a normal distribution superimposed on the graph. This is particularly useful in verifying that the residuals are normally distributed, which is a very important assumption for regression. The plot shows that *tot_exp_pc* is more skewed to the right and has a higher mean than that of normal distribution.

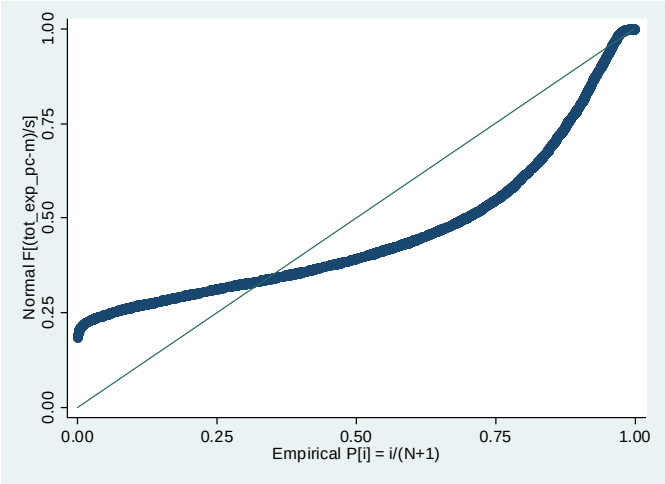
```
. kdensity tot_exp_pc, normal
```



Graphical alternatives to the **kdensity** command are the P-P plot and Q-Q plot.

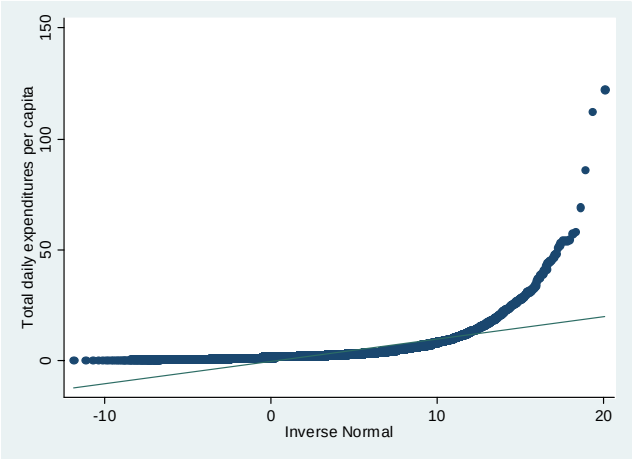
pnorm command produces a P-P plot, which graphs a standardized normal probability. It should be approximately linear if the variable follows normal distribution. The straighter the line formed by the P-P plot, the more the variable's distribution conforms to the normal distribution.

```
. pnorm tot_exp_pc
```



Qnorm command plots the quantiles of a variable against the quantiles of a normal distribution. If the Q-Q plot shows a line that is close to the 45 degree line, the variable is more normally distributed.

```
. qnorm tot_exp_pc
```



Both P-P and Q-Q plot prove that *tot_exp_pc* is not normal, with a long tail to the right. The **qnorm** plot is more sensitive to deiances from normality in the tails of the distribution, where the **pnorm** plot is more sensitive to deviances near the mean of the distribution.

From the statistics and graphs we can confidently conclude that there exists outlier, especially at the upper end of the distribution.

Deal with outliers

There are generally three ways to deal with outliers. The easiest is to delete them from analyses. The second one is to use measures that are not sensitive to them, such as median instead of mean, or transform the data to be more normal. The most complicated one is to replace them by imputation.

Since our data is heavily left-tailed, we will focus on very large outliers. A customary criterion to identify outlier is to **three times of deviation from the median**. Note that we are using the median because it is a robust statistic and if there are big outliers the mean will shift a lot but not the median.

```
/* Calculate number of standard deviations from median by urban or rural */
. use ethiopia.dta, clear
. egen median=median(tot_exp_pc), by (urban)
. egen sd=sd(tot_exp_pc), by (urban)
. gen ratio=(tot_exp_pc-median)/sd
(37 missing values generated)
. gen outlier=1 if ratio>3 & ratio~=.
(17044 missing values generated)
. replace outlier=0 if outlier==. & ratio~=.
(17007 real changes made)

. tabulate outlier, missing
    outlier |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |     17,007       98.12       98.12
          1 |        288        1.66       99.79
          . |         37        0.21      100.00
------------+-----------------------------------
      Total |     17,332      100.00
```

There are 288 observations are identified as outliers. When we compare the mean and median values from using **table** command, the mean value has dropped around 10% among urban households, while the medians are less sensitive to outliers.

```
. table urban outlier, contents(mean tot_exp_pc) row col missing

----------------------------------------
          |             outlier
    Urban |         0          1      Total
----------+-----------------------------------
        0 | 2.7359721  15.140992  2.8739015
        1 | 4.8367138  28.860726  5.3692863
          |
    Total | 3.7820815  24.287481  4.1235417
----------------------------------------

. table urban outlier, contents(median tot_exp_pc) row col missing

----------------------------------------
          |             outlier
    Urban |         0          1      Total
```

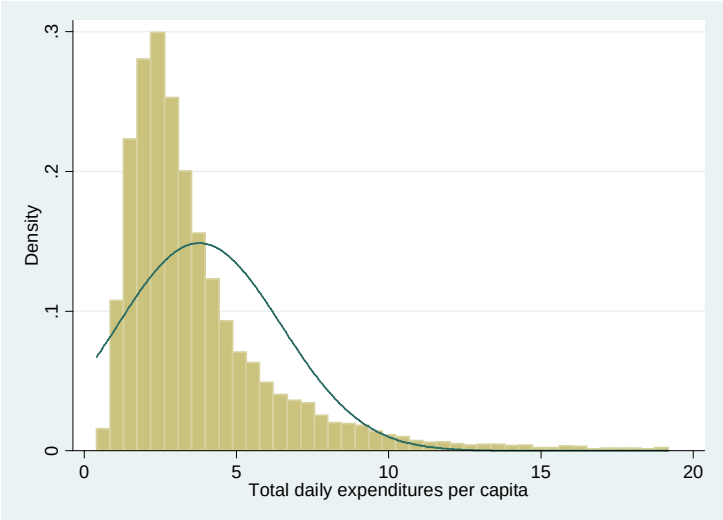
```
----------+---------------------------------
        0 |  2.453216  11.575683  2.4679672
        1 | 3.8607807  25.327541  3.9278767
          |
    Total | 2.9651103  22.549266  3.0001049
-----------------------------------------
```

Method 1. Listwise deletion

In this approach, any observation that contains a missing value for a relevant variable is dropped. Although easy to understand and to perform, it runs the risk of causing bias. Stata perform listwise deletion automatically by default in order to allow the data matrix to be inverted, a necessity for regression analysis.

Sometimes by dropping outliers we can greatly improve decrease the adverse effect of extreme values. But it does not work in our data, as indicated by the histogram below.

```
. histogram tot_exp_pc if outlier==0, normal
```



Method 2. Robust statistics

An alternative is to choose robust statistics that's not sensitive to outliers, such as median over mean, which is indicated above.

When we are concerned about outliers or skewed distributions, the **rreg** command is used for robust regression. Robust regression will result regression coefficients and standard errors from OLS, which is different from **regress** command with **robust** option.

```
. regress tot_exp hhz_usu
```

```
      Source |       SS       df       MS              Number of obs =   17295
-------------+------------------------------           F(  1, 17293) = 2195.97
       Model |  667200.753     1  667200.753           Prob > F      =  0.0000
    Residual |  5254116.66 17293  303.829102           R-squared     =  0.1127
-------------+------------------------------           Adj R-squared =  0.1126
       Total |  5921317.41 17294  342.391431           Root MSE      =  17.431
------------------------------------------------------------------------------
     tot_exp |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     hhz_usu |   2.601727   .0555198    46.86   0.000     2.492902    2.710551
       _cons |   4.890831   .2952526    16.56   0.000     4.312106    5.469556
------------------------------------------------------------------------------
```

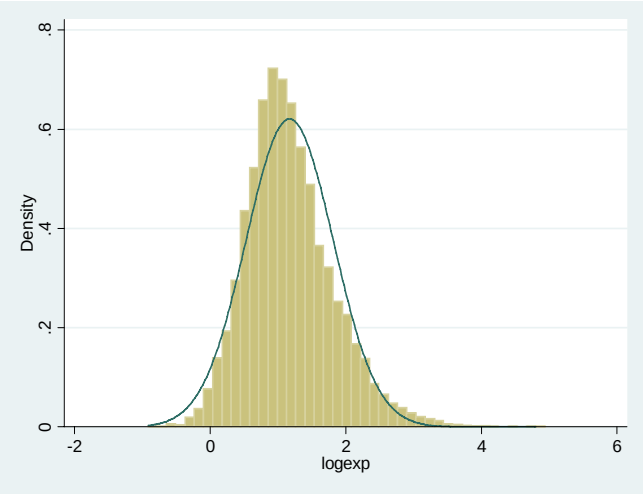
```
. rreg tot_exp hhz_usu

   Huber iteration 1:  maximum difference in weights = .98454914
   Huber iteration 2:  maximum difference in weights = .45888146
   Huber iteration 3:  maximum difference in weights = .18837666
   Huber iteration 4:  maximum difference in weights = .06859455
   Huber iteration 5:  maximum difference in weights = .02145807
Biweight iteration 6:  maximum difference in weights = .29397076
Biweight iteration 7:  maximum difference in weights = .11984779
Biweight iteration 8:  maximum difference in weights = .0403922
Biweight iteration 9:  maximum difference in weights = .0104878
Biweight iteration 10:  maximum difference in weights = .00301886

Robust regression                               Number of obs =   17295
                                                F(  1, 17293) = 5562.66
                                                Prob > F      =  0.0000

------------------------------------------------------------------------------
     tot_exp |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     hhz_usu |   1.589587   .0213129    74.58   0.000     1.547812    1.631363
       _cons |   5.948286   .1133414    52.48   0.000     5.726125    6.170447
------------------------------------------------------------------------------
```

Method 3. Data transformation

Variable *tot_exp_pc* is still skewed to the right and bounded below zero. In this case, a log transformation would be appropriate correct our dataset. The logarithm function tends to squeeze together the larger values in your data set and stretches out the smaller values, which can sometimes produce a dataset that's closer to symmetric. In addition, a log transformation can help to pull in the outliers on the high end and make them closer to the rest of the data.

Let's have a look at the distribution after the log transformation.

```
. histogram logexp if tot_exp_pc~=., normal
```



Statistics from **summarize** command also indicates an almost perfect normal distribution.

```
. sum logexp if outlier==0, detail

                          logexp
-------------------------------------------------------------
      Percentiles      Smallest
 1%    -.0896924      -.9142609
 5%     .2356188      -.8437076
10%     .4320935      -.7062833       Obs               17295
25%     .7428475      -.6452391       Sum of Wgt.       17295

50%     1.098647                      Mean           1.173457
                        Largest       Std. Dev.      .6424152
75%      1.53273       4.239981
90%     2.022103       4.455119       Variance       .4126973
95%     2.327643       4.720492       Skewness       .6641258
99%     3.030693       4.807234       Kurtosis       3.940968
```

Method 4. Imputation

After identifying outliers, usually we first denoted them as missing values. Missing data usually present a problem in statistical analyses. If missing values are correlated with the outcome of interest, then ignoring them will bias the results of statistical tests. In addition, most statistical software packages (e.g., SAS, Stata) automatically drop observations that have missing values for any variables used in an analysis. This practice reduces the analytic sample size, lowering the power of any test carried out.

Other than simply dropping missing values, there is more than one approach of imputation to fill in the cell of missing value. We will only focus on single imputation, referring to fill a missing value with one single replacement value.

The easy approach is to use arbitrary methods to impute missing data, such as mean substitution. Substitution of the simple grand mean will reduce the variance of the variable. Reduced variance can bias correlation downward (attenuation) or, if the same cases are missing for two variables and means are substituted, correlation can be inflated. These effects on correlation carry over in a regression context to lack of reliability of the beta weights and of the related estimates of the relative importance of independent variables. That is, mean substitution in the case of one variable can lead to bias in estimates of the effects of other or all variables in the regression analysis, because bias in one correlation can affect the beta weights of all variables. Mean substitution is no longer recommended.

Another approach is regression-based imputation. In this strategy, it is assumed that the same model explains the data for the non-missing cases as for the missing cases. First the analyst estimates a regression model in which the dependent variable has missing values for some observations, using all non-missing data. In the second step, the estimated regression coefficients are used to predict (impute) missing values of that variable. The proper regression model depends on the form of the dependent variable. A probit or logit is used for binary variables, Poisson or other count models for integer-valued variables, and OLS or related models for continuous variables. Even though this may introduce unrealistically low levels of noise in the data, it is a performs more robustly than mean substitution and less complex than multiple imputation. Thus it is the preferred approach in imputation.

Assuming we already coded outliers of *tot_exp_pc* as missing ., now the missing values are replaced (imputed) with predicted values.

```
. xi: regress logexp i.regco i.urban hhz_usu i.femhead agehhh, robust
. predict yhat
(option xb assumed; fitted values)

. replace logexp=yhat if tot_exp_pc==.
(325 real changes made)
```

There is another Stata command to perform imputation. The **impute** command fills in missing values by regression and put newly created variable into a new variable defined by **generate** option.

```
. xi: impute logexp i.regco i.urban hhz_usu i.femhead agehhh, generate(new1)
i.regco           _Iregco_1-15        (naturally coded; _Iregco_1 omitted)
i.urban           _Iurban_0-1         (naturally coded; _Iurban_0 omitted)
i.femhead         _Ifemhead_0-1       (naturally coded; _Ifemhead_0 omitted)
  0.21% (37) observations imputed

. xi: regress logexp i.regco i.urban hhz_usu i.femhead agehhh
. predict yhat
. replace logexp=yhat if tot_exp_pc==.
. compare logexp new1

                                         ---------- difference ----------
                           count       minimum      average     maximum
-------------------------------------------------------------------------
logexp=new1                17332
```

```
                        ----------
jointly defined              17332              0            0           0
                        ----------
total                        17332
```

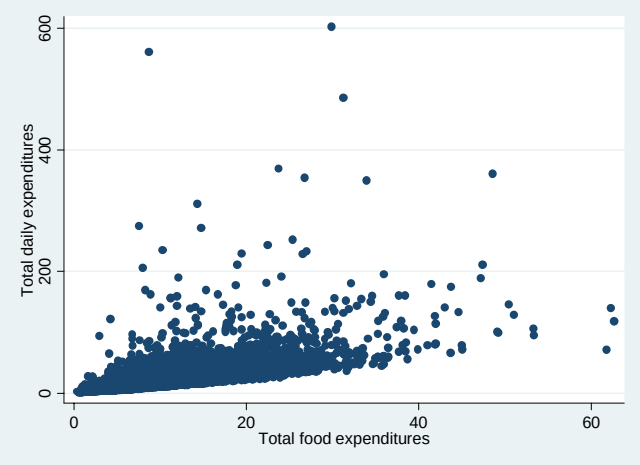
The **impute** command produces exactly the same results.

Chapter 9. Graphing Data

Graph commands to produce histogram, box plot, kdensity, P-P plot, Q-Q plot will be postponed until the introduction of normality later. But first we will get ourselves acquainted with some twoway graph commands.

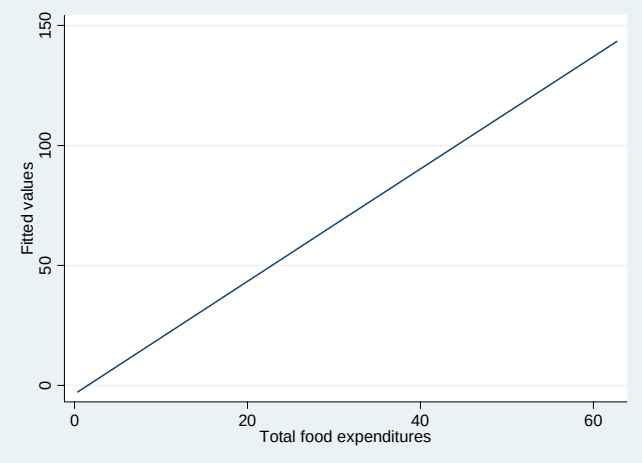
A two way scatterplot can be drawn using **(graph) twoway scatter** command to show the relationship between two variables, *tot_exp* and *exp_food*. As we would expect, there is a positive relationship between the two variables.

```
. graph twoway scatter tot_exp exp_food
```



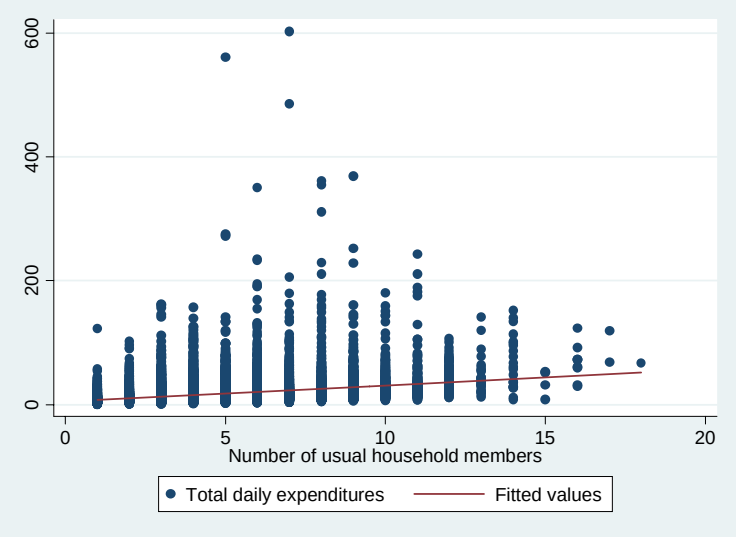
We can show the regression line predicting *tot_exp* from *exp_food* using **lfit** option.

```
. twoway lfit tot_exp exp_food
```



The two graphs can be overlapped like this

```
. twoway (scatter tot_exp hhz_usu) (lfit tot_exp hhz_usu)
```



Chapter 10. Statistical Tests

compare command

The **compare** command is an easy way to check if two variables are the same. Let's first create one variable *compare*, which equals *tot_exp* if *tot_exp* not missing, and equals 0 if *tot_exp* is missing.

```
. gen compareexp=tot_exp if tot_exp~=.
(37 missing values generated)

. replace compareexp=0 if tot_exp==.
(37 real changes made)

. compare tot_exp compareexp

                                        ---------- difference ----------
                           count        minimum      average     maximum
-------------------------------------------------------------------------
tot_exp=compareexp         17295
                         ----------
jointly defined            17295              0            0           0
tot_exp missing only          37
                         ----------
total                      17332
```

correlate command

The **correlate** command displays a matrix of Pearson correlations for the variable listed.

```
. correlate tot_exp hhz_usu
(obs=17295)

             |  tot_exp  hhz_usu
-------------+------------------
     tot_exp |   1.0000
     hhz_usu |   0.3357   1.0000
```

ttest command

We would like to see if the mean of *hhz_usu* equals to 4 by using single sample t-test, testing whether the sample was drawn from a population with a mean of 4. **ttest** command is used for this purpose.

```
. ttest hhz_usu=4

One-sample t test
```

```
------------------------------------------------------------------------------
Variable |     Obs        Mean    Std. Err.   Std. Dev.   [95% Conf.Interval]
---------+--------------------------------------------------------------------
 hhz_usu |   17332    4.746596    .0181585    2.390587    4.711003   4.782188
------------------------------------------------------------------------------
   mean = mean(hhz_usu)                                          t =  41.1155
   Ho: mean = 4                                 degrees of freedom =    17331

   Ha: mean < 4                 Ha: mean != 4                 Ha: mean > 4
Pr(T < t) = 1.0000         Pr(|T| > |t|) = 0.0000          Pr(T > t) = 0.0000
```

We are also interested that if *exp_food* is close to *tot_exp*.

```
. ttest tot_exp=exp_food

Paired t test
------------------------------------------------------------------------------
Variable |    Obs        Mean    Std. Err.   Std. Dev.   [95% Conf. Interval]
---------+--------------------------------------------------------------------
 tot_exp |  17295    17.25426    .1407023    18.50382    16.97847    17.53005
exp_food |  17295    8.870088    .0405354    5.330828    8.790635    8.949542
---------+--------------------------------------------------------------------
    diff |  17295    8.384172    .1171933    15.41215    8.154462    8.613883
------------------------------------------------------------------------------
   mean(diff) = mean(tot_exp - exp_food)                         t =  71.5414
Ho: mean(diff) = 0                           degrees of freedom =    17294

Ha: mean(diff) < 0           Ha: mean(diff) != 0           Ha: mean(diff) > 0
Pr(T < t) = 1.0000         Pr(|T| > |t|) = 0.0000          Pr(T > t) = 0.0000
```

The t-test for independent groups comes in two varieties: pooled variance and unequal variance. We want to look at the differences in *tot_exp* between rural and urban households. We will begin with the **ttest** command for independent groups with pooled variance and compare the results to the **ttest** command for independent groups using unequal variance.

```
. ttest tot_exp, by(urban)

Two-sample t test with equal variances
------------------------------------------------------------------------------
   Group |     Obs        Mean    Std. Err.   Std. Dev.   [95% Conf. Interval]
---------+--------------------------------------------------------------------
       0 |    8634    13.14905    .1246497    11.58237    12.90471     13.3934
       1 |    8661    21.34667     .244196    22.72598    20.86799    21.82535
---------+--------------------------------------------------------------------
combined |   17295    17.25426    .1407023    18.50382    16.97847    17.53005
---------+--------------------------------------------------------------------
    diff |           -8.197619    .2744217               -8.735513   -7.659724
------------------------------------------------------------------------------
   diff = mean(0) - mean(1)                                      t = -29.8723
Ho: diff = 0                                 degrees of freedom =    17293

   Ha: diff < 0                 Ha: diff != 0                 Ha: diff > 0
Pr(T < t) = 0.0000         Pr(|T| > |t|) = 0.0000          Pr(T > t) = 1.0000
```

```
. ttest tot_exp, by(urban) unequal

Two-sample t test with unequal variances
------------------------------------------------------------------------------
   Group |     Obs        Mean    Std. Err.   Std. Dev.   [95% Conf. Interval]
---------+--------------------------------------------------------------------
       0 |    8634    13.14905    .1246497    11.58237    12.90471     13.3934
       1 |    8661    21.34667     .244196    22.72598    20.86799    21.82535
---------+--------------------------------------------------------------------
combined|   17295    17.25426    .1407023    18.50382    16.97847    17.53005
---------+--------------------------------------------------------------------
    diff|           -8.197619    .2741701               -8.735033   -7.660205
------------------------------------------------------------------------------
   diff = mean(0) - mean(1)                                      t = -29.8998
Ho: diff = 0                     Satterthwaite's degrees of freedom =  12883.4

   Ha: diff < 0                 Ha: diff != 0                 Ha: diff > 0
Pr(T < t) = 0.0000         Pr(|T| > |t|) = 0.0000          Pr(T > t) = 1.0000
```

The **by()** option can be extended to group mean comparison test.

```
. ttest tot_exp, by(regco)
. ttest tot_exp, by(regco) unequal
```

Other statistical test

The **hotelling** command performs Hotelling's T-squared test of whether the means are equal between two groups.

```
 . hotel tot_exp, by(urban)

-------------------------------------------------------------------------------
-> urban = 0

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
     tot_exp |      8634    13.14905    11.58237   .6677808   561.1172
-------------------------------------------------------------------------------
-> urban = 1

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
     tot_exp |      8661    21.34667    22.72598   .9592329   602.4645

2-group Hotelling's T-squared = 892.35653
F test statistic: ((17295-1-1)/(17295-2)(1)) x 892.35653 = 892.35653

H0: Vectors of means are equal for the two groups
              F(1,17293) =  892.3565
       Prob > F(1,17293) =    0.0000
```

The tabulate command performs a chi-square test to see if two variables are independent.

```
. tabulate urban femhead, chi2

           |     Female headed
           |       household
     Urban |         0          1 |     Total
-----------+----------------------+----------
         0 |     6,621      2,013 |     8,634
         1 |     5,116      3,545 |     8,661
-----------+----------------------+----------
     Total |    11,737      5,558 |    17,295

          Pearson chi2(1) = 615.2195   Pr = 0.000
```

Chapter 11. Data Management

Subset data

We can subset data by keeping or dropping variables, or by keeping and dropping observations.

1. keep and drop variables

Suppose our data file have many variables, but we only care about just a handful of them. We can subset our data file to keep just those variables to our intereste. The **keep** command is used to keep variables in the list while dropping other variables.

```
. keep hhid exp_food tot_exp
```

Instead of wanting to keep just a handful of variables, it is possible that we might want to get rid of just one or two variables in the data file. The **drop** command is used to drop variables in the list while keeping other variables.

```
. drop tot_exp
```

2. keep and drop observations

The **keep if** command is used to keep observations if condition is met.

```
. keep if urban==0
(8660 observations deleted)
```

We want to focus on rural households in the data set, which means 8660 urban households are dropped from the data set.

Similar concepts can be found in **drop if** command. We eliminate the observations with missing values with **drop if** command. The portion after the drop if specifies which observations that should be dropped.

```
. drop if missing(tot_exp)
(37 observations deleted)
```

3. Use **use** command to drop variables and observations

You can eliminate both variables and observations with the **use** command. Let's read in just *hhid, tot_exp, regco, urban, hhz_usu* from ethiopia.dta file.

```
. use hhid tot_exp regco urban hhz_usu using ethiopia.dta
```

We can also limited read in data for urban households.

```
. use hhid tot_exp regco urban hhz_usu using ethiopia.dta if urban==1
```

Organize data

The **sort** command arranges the observations of the current data into ascending order based on the values of the variables listed. There is no limit to the number of variables in the variable list. Missing numeric values are interpreted as being larger than any other number, so they are placed last. When you sort on a string variable, however, null strings are placed first.

```
.  sort hhid regco urban
```
Variable ordering

The **order** command helps us to organize variables in a way that makes sense by changing the order of the variables. While there are several possible orderings that are logical, we usually put the id variable first, followed by the demographic variables, such as region, zone, gender, urban/rural. We will put the variables regarding the household total expenditure as follows.

```
. order hhid regco zone urban hhz_usu tot_exp
```

Using _n and _N in conjunction with the **by** command can produce some very useful results. When used with **by** command, _N is the total number of observations within each group listed in **by** command, and _n is the running counter to uniquely identify observations within the group. To use the **by** command we must first sort our data on the **by** variable.

```
. sort group
. by group: generate n1=_n
. by group: generate n2=_N
. list
     +-----------------------------------+
     | score   group   id   nt   n1   n2 |
     |-----------------------------------|
  1. |    72       1    1    7    1    4 |
  2. |    85       1    7    7    2    4 |
  3. |    76       1    3    7    3    4 |
  4. |    90       1    6    7    4    4 |
  5. |    84       2    2    7    1    2 |
     |-----------------------------------|
  6. |    82       2    5    7    2    2 |
  7. |    89       3    4    7    1    1 |
     +-----------------------------------+
```

Now n1 is the observation number within each group and n2 is the total number of observations for each group. This is very useful in programming, especially in identifying duplicate observations.

To use _n to find out duplicated observations, we can type:

```
. sort group
. list if id == id[_n+1]
```

To use _N to identify duplicated observations, use:

```
. sort group score
. by group score: generate ngroup=_N
. list if ngroup>1
```

If there are a lot of variables in the data set, it could take a long time to type them all out twice. We can make use of the "*" and "?" wildcards to indicates that we wish to use all the variables.

Further we can combine **sort** and **by** commands into a single statement. Below is a simplified version of the code and will yield the same results as above.

```
. bysort *:generate nn=_N
. list if nn>1
```

Create one data set from two or more data sets

Appending data files

We can create a new data by **append** command, which concatenates two datasets, that is, stick them together vertically, one after another.

Supposing we are given one file with data for the rural households (called rural.dta) and a file for the urban households (called urban.dta). We need to combine these files together to be able to analyze them.

```
. use rural.dta, clear
. append using urban.dta
```

The **append** command does not require that the two datasets contain the same variables, even though this is typically the case. But it highly recommended to use the identical list of variables for **append** command to avoid missing values from one dataset.

One-to-one match merging

Another way of combining data files is match merging. The **merge** command sticks two datasets horizontally, one next to the other. Before any merge, both datasets must be sorted by identical merge variable.

Assuming we are working on our household expenditure data, and we have been given two files. One file has all the demographic information (called hhinfo.dta) and the other file with total expenditure by household (called hhexp.dta). Both data sets have been cleaned and sorted by hhid. We would like to merge the two households together by hhid.

```
. use hhinfo.dta, clear
. list
. sort hhid
. save h1.dta, replace

. use hhexp.dta, clear
. list
. sort hhid
. save h2.dta, replace

. use h1.dta, clear
. merge hhid using h2.dta
```

After **merge** command, a _merge variable appears. The _merge variable indicates, for each observation, how the merge go. This is especially useful in identifying mismatched records. _merge can have one of three values in merging file A using file B:
_merge==1 the records contains information from master data file A
_merge==2 the records contains information from using data file B
_merge==3 the records contains information from both files

When there are many records, **tabulating _merge** is very useful to summarize how many mismatched observations you have. In this case, all of the records match so the value for _merge is always 3.

```
. tab _merge
     _merge |      Freq.     Percent        Cum.
------------+-----------------------------------
          2 |      8,660       49.97       49.97
          3 |      8,672       50.03      100.00
------------+-----------------------------------
      Total |     17,332      100.00
```

One-to-many match merging

Another kind of merge is called a one to many merge. Say, we have one data file household.dta contains household information, and another data file individual.dta contains information of each individual in the household. If we merge households.dta with individual.dta, there can be multiple individuals per household and hence it is a one to many merge.

The strategy for the one to many merge is really the same as the one to one match merge.

```
. use household.dta, clear
. list
. sort hhid
. save h1.dta, replace

. use individual.dta, clear
. list
. sort hhid
```

```
. save h2.dta, replace

. use h1.dta, clear
. merge hhid using h2.dta
```

There is no difference in the order of files to be merged and the results are the same. The only difference is the order of the records after the merge.

Label data

Besides giving labels to variables, we can also label the data set itself so that we will remember what the data are. The **label data** command places a label on the whole dataset.

```
. label data “relabeled household”
```

We can also add some notes to the data set. The **note:** (note the colon, “:”) command allows you to place notes into the dataset.

```
. notes hhsize: the variable hhz_usu was renamed to hhsize
```

The **notes** command display all notes in the data set.

```
. notes
hhsize:
the variable hhz_usu was renamed to hhsize
```

Chapter 12. Linear Regression

Regression commands

This is an example of ordinary linear regression by using **regress** command.

```
. regress tot_exp hhz_usu
```

```
     Source |       SS       df       MS              Number of obs =   17295
------------+------------------------------           F(  1, 17293) = 2195.97
      Model |  667200.753     1  667200.753           Prob > F      =  0.0000
   Residual |  5254116.66 17293  303.829102           R-squared     =  0.1127
------------+------------------------------           Adj R-squared =  0.1126
      Total |  5921317.41 17294  342.391431           Root MSE      =  17.431

------------------------------------------------------------------------------
    tot_exp |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
------------+----------------------------------------------------------------
    hhz_usu |   2.601727   .0555198    46.86   0.000     2.492902    2.710551
      _cons |   4.890831   .2952526    16.56   0.000     4.312106    5.469556
------------------------------------------------------------------------------
```

This regression tells us that for every extra person (*hhz_usu*) added to a household, total daily expenditure (*tot_exp*) will increase by 2.6 Ethiopia Birr. This increase is statistically significant as indicated by the 0.000 probability associated with this coefficient.

The other important piece of information is the r-squared (r2) which equals to 0.1127. In essence, this value tells us that by our independent variable (*hhz_usu*) accounts for approximately 11% of the variation of dependent variable (*tot_exp*).

We can run the regression with robust standard errors, which can tolerate a non-zero percentage of outliers, i.e., when the residuals are not iid. This is very useful when there is heterogeneity of variance. The **robust** option does not affect the estimates of the regression coefficients.

```
. regress tot_exp hhz_usu, robust
```

```
Linear regression                               Number of obs =   17295
                                                F(  1, 17293) = 1351.39
                                                Prob > F      =  0.0000
                                                R-squared     =  0.1127
                                                Root MSE      =  17.431
------------------------------------------------------------------------------
             |               Robust
    tot_exp |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
------------+----------------------------------------------------------------
    hhz_usu |   2.601727   .0707735    36.76   0.000     2.463003     2.74045
      _cons |   4.890831   .2804312    17.44   0.000     4.341157    5.440505
------------------------------------------------------------------------------
```

The **regress** command without any arguments redisplays the last regression analysis.

Extract results

Stata stores results from estimation commands in e(), and you can see a list of what exactly is stored using the **ereturn list** command.

```
. ereturn list

scalars:
                  e(N) =  17295
              e(df_m) =  1
              e(df_r) =  17293
                 e(F) =  2195.973818320767
               e(r2) =  .112677755062528
              e(rmse) =  17.43069424443639
               e(mss) =  667200.7528912034
               e(rss) =  5254116.658171482
              e(r2_a) =  .1126264439976499
                e(ll) =  -73972.67623316433
              e(ll_0) =  -75006.45947850558

macros:
             e(title) : "Linear regression"
            e(depvar) : "tot_exp"
               e(cmd) : "regress"
        e(properties) : "b V"
           e(predict) : "regres_p"
             e(model) : "ols"
         e(estat_cmd) : "regress_estat"

matrices:
                 e(b) :  1 x 2
                 e(V) :  2 x 2

functions:
            e(sample)
```

Using the **generate** command, we can extract those results, such as estimated coefficients and standard errors, to be used in other Stata commands.

```
. regress tot_exp hhz_usu
. gen intercept=_b[_cons]

. display intercept
4.890831
. gen slope=_b[hhz_usu]
. display slope
2.6017268
```

The **estimates table** command displays a table with coefficients and statistics for one or more estimation sets in parallel columns. In addition, standard errors, t statistics, p-values, and scalar statistics may be listed by **b, se, t, p** options.

```
. estimates table, b se t p

----------------------------
    Variable |   active
-------------+--------------
     hhz_usu |  2.6017268
             |  .05551983
             |      46.86
             |     0.0000
       _cons |   4.890831
             |  .29525262
             |      16.56
             |     0.0000
----------------------------
```

Prediction commands

The **predict** command computes predicted value and residual for each observation. The default shown below is to calculate the predicted *tot_exp*.

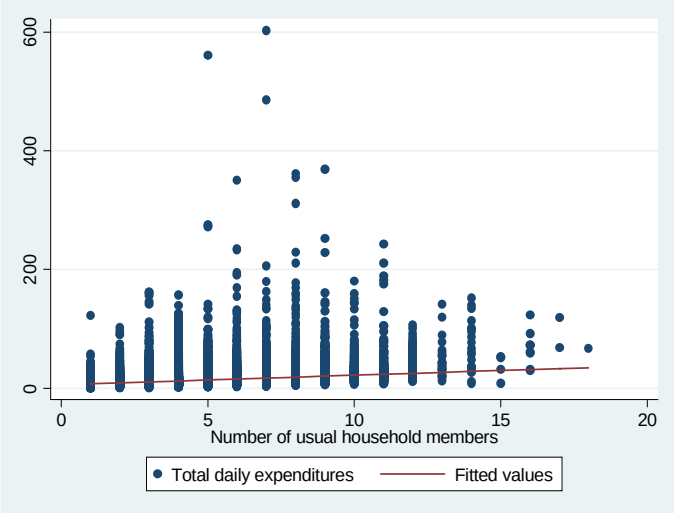
```
. predict pred
(option xb assumed; fitted values)
```

When using the **resid** option the predict command calculates the residual.

```
. predict e, residual
```

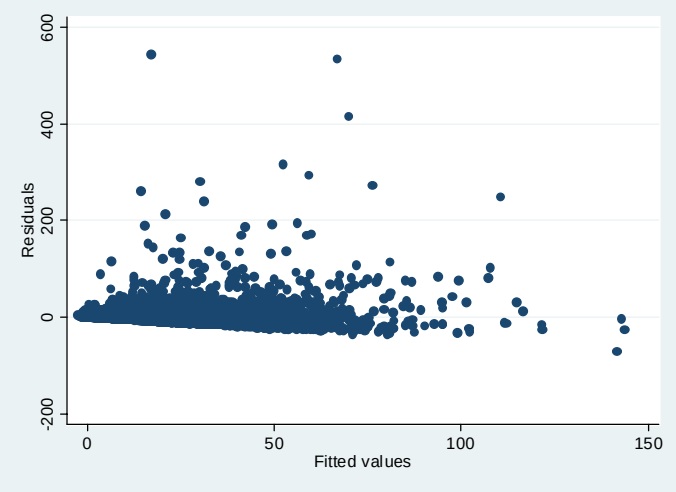
We can plot the predicted value and observed value using **graph twoway** command.

```
. regress tot_exp exp_food
. predict pred
. graph twoway (scatter tot_exp hhz_usu) (line pred hhz_usu)
```



The **rvfplot** command is a convenience command that generates a plot of the residual versus the fitted values. It is used after **regress** command.

```
. regress tot_exp exp_food
. rvfplot
```



The **rvpplot** command is another convenience command which produces a plot of the residual versus a specified predictor and it is also used after regress. In this example, it produces the same graph as above.

```
. regress tot_exp exp_food
. rvpplot exp_food
```

Hypothesis tests

The **test** command performs Wald tests for simple and composite linear hypotheses about the parameters of estimation.

```
. gen regco1=0
. replace regco1=1 if regco==1
(1252 real changes made)
. gen regco2=0
. replace regco2=1 if regco==2
(792 real changes made)
. gen regco3=0
. replace regco3=1 if regco==3
(3340 real changes made)
. gen regco4=0
. replace regco4=1 if regco==4
(3728 real changes made)

. regress tot_exp hhz_usu regco1 regco2 regco3 regco4
```

```
     Source |       SS       df       MS              Number of obs =   17295
------------+------------------------------           F(  5, 17289) =  453.41
      Model |  686427.054     5  137285.411           Prob > F      =  0.0000
   Residual |  5234890.36 17289  302.787342           R-squared     =  0.1159
------------+------------------------------           Adj R-squared =  0.1157
      Total |  5921317.41 17294  342.391431           Root MSE      =  17.401

------------------------------------------------------------------------------
    tot_exp |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
------------+----------------------------------------------------------------
    hhz_usu |   2.586416   .0556412    46.48   0.000     2.477353    2.695478
     regco1 |  -1.656508   .5287445    -3.13   0.002      -2.6929   -.6201148
     regco2 |   -1.59704    .650351    -2.46   0.014    -2.871794   -.3222864
     regco3 |  -1.721974   .3587041    -4.80   0.000    -2.425071   -1.018878
     regco4 |  -2.516142   .3437652    -7.32   0.000    -3.189956   -1.842327
      _cons |   6.028982    .333079    18.10   0.000     5.376113     6.68185
------------------------------------------------------------------------------
```

```
. test regco1=0

 ( 1)  regco1 = 0

       F(  1, 17289) =    9.82
            Prob > F =    0.0017

. test regco1=regco2=regco3=regco4

 ( 1)  regco1 - regco2 = 0
 ( 2)  regco1 - regco3 = 0
 ( 3)  regco1 - regco4 = 0

       F(  3, 17289) =    1.66
            Prob > F =    0.1726
```

test and **predict** are commands that can be used in conjunction with all of the above estimation procedures.

The **suest** command combines the estimation results from regressions (including parameter estimates and associated covariance matrices) into a single parameter vector and simultaneous covariance matrix of the sandwich/robust type.

Typical applications of **suest** command are tests for intra-model and cross-model hypotheses using **test** or **testnl** command, such as a generalized Hausman specification test, or Chow test for structural break.

Before we perform any test using suest command, it is important we first keep estimation results by **estimates store** command.

```
. reg  tot_exp hhz_usu if urban==1
. estimates store urban
. reg  tot_exp hhz_usu if urban==0
. estimates store rural
```

```
. suest urban rural

Simultaneous results for urban, rural
                                                  Number of obs   =      17295

------------------------------------------------------------------------------
             |               Robust
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
urban_mean   |
      hhz_usu|    3.43175   .1170849    29.31   0.000     3.202268    3.661232
        _cons|   5.729652   .4250396    13.48   0.000      4.89659    6.562714
-------------+----------------------------------------------------------------
urban_lnvar  |
        _cons|   6.092408   .1238756    49.18   0.000     5.849617      6.3352
-------------+----------------------------------------------------------------
rural_mean   |
      hhz_usu|   1.932295   .0492704    39.22   0.000     1.835727    2.028863
        _cons|   3.576649   .2157359    16.58   0.000     3.153814    3.999484
-------------+----------------------------------------------------------------
rural_lnvar  |
        _cons|   4.748265   .3268713    14.53   0.000     4.107609    5.388921
------------------------------------------------------------------------------
```

We would like to test if the *hhz_usu* coefficients are zeros by using **test** command.

```
. test hhz_usu

 ( 1)  [urban_mean]hhz_usu = 0
 ( 2)  [rural_mean]hhz_usu = 0

           chi2(  2) = 2397.14
         Prob > chi2 =    0.0000
```

Next we want to see if the same *hhz_usu* coefficient holds for rural and urban households. We can type

```
. test [urban_mean]hhz_usu=[rural_mean]hhz_usu

 ( 1)  [urban_mean]hhz_usu - [rural_mean]hhz_usu = 0

           chi2(  1) =  139.33
         Prob > chi2 =    0.0000
```

Or we can test if coefficients between equations are equal, or a Chow test.

```
. test ([urban_mean]hhz_usu=[rural_mean]hhz_usu)
([urban_mean]_cons=[rural_mean]_cons)

 ( 1)  [urban_mean]hhz_usu - [rural_mean]hhz_usu = 0
 ( 2)  [urban_mean]_cons - [rural_mean]_cons = 0

           chi2(  2) = 1414.59
         Prob > chi2 =    0.0000
```

This is equivalent to have **accumulate** options in test command, which tests hypothesis jointly with previously tested hypotheses

```
. test ([urban_mean]hhz_usu=[rural_mean]hhz_usu)
. test ([urban_mean]_cons=[rural_mean]_cons) , accumulate

 ( 1)  [urban_mean]hhz_usu - [rural_mean]hhz_usu = 0
 ( 2)  [urban_mean]_cons - [rural_mean]_cons = 0

           chi2(  2) = 1414.59
         Prob > chi2 =    0.0000
```

Heteroskedasticity

We can always visually check how well the regression surface fits the data y plotting residuals versus fitted values, like **rvfplot** or **rvpplot** commands. In addition, there are a bunch of statistical tests to test heteroskedasticity in regression errors.

We can use the **hettest** command to run an auxiliary regression of $\ln e_i^2$ on the fitted values.

```
. hettest

Breusch-Pagan / Cook-Weisberg test for heteroskedasticity
         Ho: Constant variance
         Variables: fitted values of tot_exp

         chi2(1)      = 33821.20
         Prob > chi2  =   0.0000
```

We can also use information matrix test by **imtest** command, which provides a summary test of violations of the assumptions on regression errors.

```
. imtest

Cameron & Trivedi's decomposition of IM-test

---------------------------------------------------
              Source |       chi2     df      p
---------------------+-----------------------------
  Heteroskedasticity |     216.69      2    0.0000
            Skewness |      49.16      1    0.0000
            Kurtosis |       4.31      1    0.0379
---------------------+-----------------------------
               Total |     270.17      4    0.0000
---------------------------------------------------
```

For the next two tests first we need to download the programs from internet at
http://econpapers.repec.org/software/bocbocode/s390601.htm
and

http://econpapers.repec.org/software/bocbocode/s390602.htm

The **bpagan** command computes the Breusch-Pagan Lagrange multiplier test for heteroskedasticity in the error distribution, conditional on a set of variables which are presumed to influence the error variance. The test statistic, a Lagrange multiplier measure, is distributed Chi-squared(p) under the null hypothesis of homoskedasticity.

```
. gen food2=exp_food^2
. gen lgfood=log(exp_food)
. regress tot_exp exp_food
. bpagan exp_food food2 lgfood

Breusch-Pagan LM statistic:   47406.9  Chi-sq( 3)  P-value =      0
```

The **whitetst** command computes White's test for heteroskedasticity following regression. This test is a special case of the Breusch-Pagan test (bpagan). The White test does not require specification of a list of variables, as that list is constructed from the explanatary list. Alternatively, **whitetst** can perform a specialized form of the test which economizes on degrees of freedom.

```
. whitetst

White's general test statistic :  216.6935  Chi-sq( 2)  P-value =  8.8e-48
```

Both tests rejects the null hypothesis of homoskedasticity.

xi command for categorical data

When there is categorical data, it could be inefficient to generate a series of dummy variables. The **xi** prefix is used to dummy code categorical variables, and we tag these variables with an “i.” in front of each target variable.

In out example, the explanatory variable *regco* has 11 levels and requires 10 dummy variables. The **test** command is used to test the collective effect of the 10 dummy-coded variables. In other words, it tests the main effect of variable *regco*. Note that the dummy-coded variables name is written in exactly the same one as it appears in the regression results, including the uppercase I.

```
. xi: regress tot_exp hhz_usu i.regco
i.regco           _Iregco_1-15        (naturally coded; _Iregco_1 omitted)

     Source |       SS       df       MS              Number of obs =   17295
------------+------------------------------           F( 11, 17283) =  279.97
      Model |   895547.87    11  81413.4427           Prob > F      =  0.0000
   Residual |  5025769.54 17283   290.79266           R-squared     =  0.1512
------------+------------------------------           Adj R-squared =  0.1507
      Total |  5921317.41 17294  342.391431           Root MSE      =  17.053
------------------------------------------------------------------------------
    tot_exp |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
```

```
------------+----------------------------------------------------------------
    hhz_usu |   2.548534   .0546996    46.59   0.000     2.441317    2.655751
  _Iregco_2 |    .061185   .7765802     0.08   0.937    -1.460991    1.583361
  _Iregco_3 |  -.0698934   .5657551    -0.12   0.902    -1.178831    1.039044
  _Iregco_4 |  -.8451452   .5577965    -1.52   0.130    -1.938483    .2481925
  _Iregco_5 |    1.39709   .7595084     1.84   0.066     -.091623    2.885804
  _Iregco_6 |  -.3928507   .7416831    -0.53   0.596    -1.846625    1.060923
  _Iregco_7 |  -2.886479   .5862374    -4.92   0.000    -4.035564   -1.737395
  _Iregco_12|  -.8356235   .7902772    -1.06   0.290    -2.384647    .7133998
  _Iregco_13|   3.721305   .7960722     4.67   0.000     2.160923    5.281688
  _Iregco_14|   11.42991   .6543105    17.47   0.000     10.14739    12.71242
  _Iregco_15|   1.545186    .760929     2.03   0.042     .0536878    3.036684
       _cons|   4.543275   .5417198     8.39   0.000      3.48145    5.605101
------------------------------------------------------------------------------
```

```
. test _Iregco_2 _Iregco_3 _Iregco_4 _Iregco_5 _Iregco_6 _Iregco_7 _Iregco_12
_Iregco_13 _Iregco_14 _Iregco_15

 ( 1)  _Iregco_2 = 0
 ( 2)  _Iregco_3 = 0
 ( 3)  _Iregco_4 = 0
 ( 4)  _Iregco_5 = 0
 ( 5)  _Iregco_6 = 0
 ( 6)  _Iregco_7 = 0
 ( 7)  _Iregco_12 = 0
 ( 8)  _Iregco_13 = 0
 ( 9)  _Iregco_14 = 0
 (10)  _Iregco_15 = 0

       F( 10, 17283) =   78.53
            Prob > F =    0.0000
```

We reject the null hypothesis of no regional effects since p-value is small.

The **xi** prefix can also be used to create dummy variables for *regco* and for the interaction term of *regco* and *hhz_usu*. The first **test** command tests the overall interaction and the second **test** command test the main effect of urban.

```
. xi: regress tot_exp hhz_usu i.regco*hhz_usu
. test _IregXhhz__2 _IregXhhz__3 _IregXhhz__4 _IregXhhz__5 _IregXhhz__6
_IregXhhz__7 _IregXhhz__12 _IregXhhz__13 _IregXhhz__14 _IregXhhz__15
. test _Iregco_2 _Iregco_3 _Iregco_4 _Iregco_5 _Iregco_6 _Iregco_7 _Iregco_12
_Iregco_13 _Iregco_14 _Iregco_15
```

By default, Stata selects the first category in the categorical variable as the reference category. If we would like to declare a certain category as reference category, the **char** command is needed.

In the model above, we would like to use region 5 as reference region, and the commands are

```
. char regco[omit] 5

. xi: regress tot_exp hhz_usu i.regco
```

```
i.regco           _Iregco_1-15        (naturally coded; _Iregco_5 omitted)

     Source |       SS       df       MS              Number of obs =   17295
------------+------------------------------           F( 11, 17283) =  279.97
      Model |   895547.87    11  81413.4427           Prob > F      =  0.0000
   Residual |  5025769.54 17283   290.79266           R-squared     =  0.1512
------------+------------------------------           Adj R-squared =  0.1507
      Total |  5921317.41 17294  342.391431           Root MSE      =  17.053
------------------------------------------------------------------------------
    tot_exp |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
------------+-----------------------------------------------------------------
    hhz_usu |   2.548534   .0546996    46.59   0.000     2.441317    2.655751
  _Iregco_1 |   -1.39709   .7595084    -1.84   0.066    -2.885804     .091623
  _Iregco_2 |  -1.335905   .8452886    -1.58   0.114    -2.992757    .3209459
  _Iregco_3 |  -1.466984   .6573026    -2.23   0.026    -2.755363   -.1786041
  _Iregco_4 |  -2.242236   .6492055    -3.45   0.001    -3.514744   -.9697271
  _Iregco_6 |  -1.789941     .81341    -2.20   0.028    -3.384307    -.195575
  _Iregco_7 |   -4.28357   .6734329    -6.36   0.000    -5.603566   -2.963573
  _Iregco_12|  -2.232714   .8579721    -2.60   0.009    -3.914426   -.5510015
  _Iregco_13|   2.324215   .8633818     2.69   0.007     .6318993    4.016531
  _Iregco_14|   10.03282   .7330675    13.69   0.000      8.59593     11.4697
  _Iregco_15|   .1480954   .8305156     0.18   0.858    -1.479799     1.77599
       _cons|   5.940366   .6479407     9.17   0.000     4.670336    7.210395
------------------------------------------------------------------------------
```

Some estimation procedures in Stata are included here:

anova	analysis of variance and covariance
arch	autoregressive conditional heterosce. family of estimators
arima	autoregressive integrated moving average models
bsqreg	quantile regression with bootstrapped standard errors
cnreg	censored-normal regression
cnsreg	constrained linear regression
ereg	maximum-likelihood exponential distribution models
glm	generalized linear models
ivreg	instrumental variable and two-stage least squares regression
lnormal	maximum-likelihood lognormal distribution models
mvreg	multivariate regression
nl	nonlinear least squares
poisson	maximum-likelihood poisson regression
qreg	quantile regression
reg3	three-stage least squares regression
regress	linear regression
rreg	robust regression using IRLS

sureg	seemingly unrelated regression
tobit	tobit regression
vwls	variance-weighted least squares regression
zinb	zero-inflated negative binomial model
zip	zero-inflated poisson models

Chapter 13. Logistic Regression

Logistic regression

We are not going to talk the theory behind logistic regression, per se, but focus on how to perform logistic regression analyses and interpret the results using Stata. It is assumed that users are familiar with logistic regression.

We will use the ethiopia1.dta dataset. It added one binary response variable called *poverty*. The **logistic** command by default produces the output in odds ratios but can display the coefficients if the **coef** options is used.

```
. logistic poverty hhz_usu agehhh, coef

Logistic regression                                     Number of obs   =      17295
                                                        LR chi2(2)      =     980.69
                                                        Prob > chi2     =     0.0000
Log likelihood = -10398.765                             Pseudo R2       =     0.0450
------------------------------------------------------------------------------
    poverty |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
------------+-----------------------------------------------------------------
    hhz_usu |   .2142649   .0072011    29.75   0.000     .2001511    .2283787
     agehhh |   .0019468   .0011607     1.68   0.093    -.0003281    .0042218
      _cons |  -1.879701   .0610683   -30.78   0.000    -1.999392   -1.760009
------------------------------------------------------------------------------
```

The exact same results can be obtained by using the **logit** command.

```
. logit poverty hhz_usu agehhh

Iteration 0:   log likelihood = -10889.111
Iteration 1:   log likelihood = -10401.544
Iteration 2:   log likelihood = -10398.765
Iteration 3:   log likelihood = -10398.765

Logistic regression                                     Number of obs   =      17295
                                                        LR chi2(2)      =     980.69
                                                        Prob > chi2     =     0.0000
Log likelihood = -10398.765                             Pseudo R2       =     0.0450
------------------------------------------------------------------------------
    poverty |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
------------+-----------------------------------------------------------------
    hhz_usu |   .2142649   .0072011    29.75   0.000     .2001511    .2283787
     agehhh |   .0019468   .0011607     1.68   0.093    -.0003281    .0042218
      _cons |  -1.879701   .0610683   -30.78   0.000    -1.999392   -1.760009
------------------------------------------------------------------------------
```

The **xi** prefix can also be used in logistic model to include categorical variables.

```
. xi: logit poverty hhz_usu agehhh i.urban
```

```
.urban           _Iurban_0-1         (naturally coded; _Iurban_0 omitted)

Iteration 0:   log likelihood = -10889.111
Iteration 1:   log likelihood = -9795.1014
Iteration 2:   log likelihood = -9770.7937
Iteration 3:   log likelihood = -9770.7226

Logistic regression                               Number of obs   =      17295
                                                  LR chi2(3)      =    2236.78
                                                  Prob > chi2     =     0.0000
Log likelihood = -9770.7226                       Pseudo R2       =     0.1027
------------------------------------------------------------------------------
    poverty |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
------------+-----------------------------------------------------------------
    hhz_usu |   .2127718   .0074852    28.43   0.000      .198101    .2274426
     agehhh |   .0026375   .0011946     2.21   0.027     .0002961    .0049788
  _Iurban_1 |   -1.22581   .0357395   -34.30   0.000    -1.295858   -1.155762
      _cons |  -1.362829   .0643528   -21.18   0.000    -1.488958     -1.2367
------------------------------------------------------------------------------
```

Extract results

We can use **ereturn** or **estat** command to retrieve results from estimation, same as with other regression commands.

```
. xi: logit poverty hhz_usu agehhh i.urban
. ereturn list

scalars:
                  e(N) =  17332
              e(ll_0) =  -10903.56206621753
                e(ll) =  -9782.301403136013
              e(df_m) =  3
              e(chi2) =  2242.521326163038
              e(r2_p) =  .102834344984885

macros:
             e(title) : "Logistic regression"
            e(depvar) : "poverty"
               e(cmd) : "logit"
          e(crittype) : "log likelihood"
           e(predict) : "logit_p"
        e(properties) : "b V"
         e(estat_cmd) : "logit_estat"
          e(chi2type) : "LR"

matrices:
                  e(b) :  1 x 4
                  e(V) :  4 x 4

functions:
            e(sample)
```

```
. estat summarize

  Estimation sample logit                 Number of obs =  17332

  ------------------------------------------------------------
      Variable |        Mean     Std. Dev.       Min        Max
  -------------+----------------------------------------------
       poverty |    .3229864      .467631          0          1
       hhz_usu |    4.746596     2.390587          1         18
        agehhh |    43.60322     14.90114         13         99
     _Iurban_1 |    .5003462     .5000143          0          1
  ------------------------------------------------------------

. estat ic

-------------------------------------------------------------------------------
      Model |    Obs    ll(null)   ll(model)     df          AIC         BIC
------------+------------------------------------------------------------------
          . |  17332   -10903.56   -9782.301      4      19572.6    19603.64
-------------------------------------------------------------------------------
```

Marginal effects

We use mfx command to numerically calculates the marginal effects or the elasticities and their standard errors after estimation. Several options are available for the calculation of marginal effects.
dydx is the default.
eyex specifies that elasticities be calculated in the form of d(lny)/d(lnx)
dyex specifies that elasticities be calculated in the form of d(y)/d(lnx)
eydx specifies that elasticities be calculated in the form of d(lny)/d(x)

```
. use "U:\notes\poverty.dta", clear
. xi: logit poverty hhz_usu agehhh i.urban
. mfx, dydx

Marginal effects after logit
      y  = Pr(poverty) (predict)
         =  .29852704
-------------------------------------------------------------------------------
variable |     dy/dx    Std. Err.      z    P>|z|  [    95% C.I.   ]      X
---------+---------------------------------------------------------------------
 hhz_usu |   .044831      .00155   28.99   0.000     .0418  .047862    4.7466
  agehhh |  .0005173      .00025    2.07   0.038   .000028  .001007   43.6032
_Iurba~1*| -.2522575      .00693  -36.40   0.000   -.26584 -.238675   .500346
-------------------------------------------------------------------------------
(*) dy/dx is for discrete change of dummy variable from 0 to 1
```

Hypothesis Tests

Likelihood-ratio test

The **lrtest** command performs a likelihood-ratio test for the null hypothesis that the parameter vector of a statistical model satisfies some smooth constraint. To conduct the test, both the unrestricted and the restricted models must be fitted using the maximum likelihood method (or some equivalent method), and the results of at least one must be stored using estimates store.

The **lrtest** command provides an important alternative to Wald testing for models fitted by maximum likelihood. Wald testing requires fitting only one model (the unrestricted model). Hence, it is computationally more attractive than likelihood-ratio testing. Most statisticians, however, favor using likelihood-ratio testing whenever feasible since the null-distribution of the LR test statistic is often "more closely" chi-square distributed than the Wald test statistic.

We would like to see if the introduction of regional dummy will help our estimation. We perform a likelihood ratio test using **lrtest** command.

```
. xi: logit poverty hhz_usu agehhh i.urban
. estimates store m1
. logit poverty hhz_usu agehhh
. lrtest m1
Likelihood-ratio test                                 LR chi2(1)  =   1256.08
(Assumption: . nested in m1)                          Prob > chi2 =    0.0000
```

The null hypothesis is firmly rejected.

Other hypothesis tests for parameters are the same as described in OLS.

Other related commands

In addition to the built-in Stata commands, we will be demonstrating the use of a number of user-written ado's, in particular, **listcoef, fitstat, prchagne, prtab, prgen**, etc. Those ado files streamline the process and presents the results in a nice looking format. To find out more about these programs or to download them type **findit** followed by the program name in the Stata command window.

```
. findit listcoef

Web resources from Stata and other users
(contacting http://www.stata.com)

3 packages found (Stata Journal and STB listed first)
-----------------------------------------------------

sg152 from http://www.stata.com/stb/stb57
```

```
    STB-57 sg152.  Listing and interpreting transformed coef. from ... /
    Listing and interpreting transformed coefficients from certain /
    regression models / STB insert by J. Scott Long, Indiana University /
    Jeremy Freese, University of Wisconsin-Madison / Support:

spostado from http://www.indiana.edu/~jslsoc/stata
   spostado: Stata 8 & 7 commands for the post-estimation interpretation of /
   regression models. Based on Long's Regression Models for Categorical / and
   Limited Dependent Variables. / Support: www.indiana.edu/~jslsoc/spost.htm
   / Scott Long & Jeremy Freese (spostsup@indiana.edu)

spost9_ado from http://www.indiana.edu/~jslsoc/stata
   spost9_ado Stata 9 commands for the post-estimation interpretation of /
   regression models. Install spostado.pkg for Stata 8. / Based on Long's
   Regression Models for Categorical and Limited / Dependent Variables.
   Second Edition. / Support www.indiana.edu/~jslsoc/spost.htm /Scott Long  &
```

These add-on programs ease the running and interpretation or ordinal logistic models. Or, you can install the complete spostado package by clicking on one of the links under web resources.

One useful command is **prchange,** which computes discrete and marginal change for regression models for categorical and count variables. Marginal change is the partial derivative of the predicted probability or predicted rate with respect to the independent variables. Discrete change is the difference in the predicted value as one independent variable changes values while all others are held constant at specified values.

The discrete change is computed when a variable changes from its minimum to its maximum (Min->Max), from 0 to 1 (0->1), from its specified value minus .5 units to its specified value plus .5 (-+1/2), and from its specified value minus .5 standard deviations to its value plus .5 standard deviations (-+sd/2).

```
. xi: logit poverty hhz_usu agehhh i.urban
. prchange

logit: Changes in Probabilities for poverty

           min->max      0->1     -+1/2    -+sd/2  MargEfct
  hhz_usu    0.7162    0.0267    0.0446    0.1062    0.0446
   agehhh    0.0482    0.0005    0.0006    0.0082    0.0006
_Iurban_1   -0.2530   -0.2530   -0.2529   -0.1280   -0.2571

              0       1
Pr(y|x)  0.7007  0.2993
```

After estimating a regression model, the **prtab** command presents a one- to four-way table of the predicted values (probabilities, rate) for different combinations of values of independent variable.

```
. prtab hhz_usu
. prtab hhz_usu, x(agehhh=50 _Iurban_1==0)

logit: Predicted probabilities of positive outcome for poverty
```

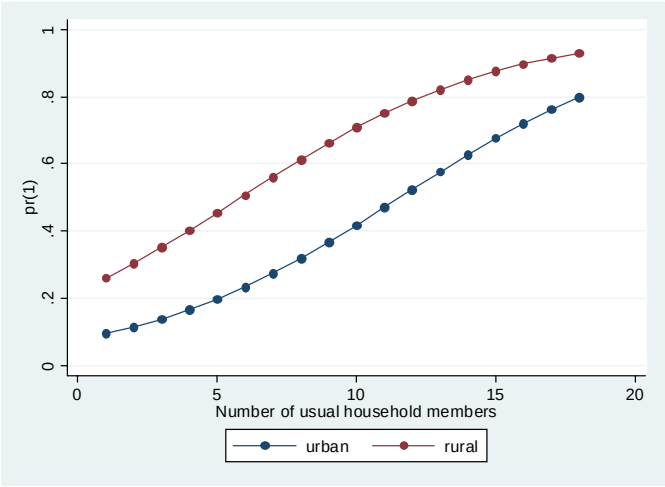
```
----------------------
Number of |
usual     |
household |
members   | Prediction
----------+-----------
        1 |     0.2634
        2 |     0.3070
        3 |     0.3543
        4 |     0.4047
        5 |     0.4571
        6 |     0.5106
        7 |     0.5637
        8 |     0.6155
        9 |     0.6647
       10 |     0.7107
       11 |     0.7526
       12 |     0.7903
       13 |     0.8236
       14 |     0.8526
       15 |     0.8775
       16 |     0.8987
       17 |     0.9166
       18 |     0.9316
----------------------
```

We interpret the results from **prtab** command in this way, given a rural household with household head ages 50, the probability of the household stays in poverty increases from 0.2634 to 0.9316 as the number of household member goes up.

The **prgen** command computes predicted values and confidence intervals for regression with continuous, categorical, and count outcomes in a way that is useful for making plots. Predicted values are computed for the case in which one independent variable varies over a specified range while the others are held constant. You can request variables containing upper and lower bounds for these variables. You can also create a variable containing the marginal change in the outcome with respect to the specified variable, holding other variables constant. New variables are added to the existing dataset that contain these predicted values that can be plotted.

In first **prgen** command, we generate variable *urbanpovertyp1* for predicted probability of being poor and *urbanpovertyp1* for being not poor. Since we know from **prtab** command that *hhz_usu* takes value from 1 to 18, we use **ncases**() option to define the number of predicted values as *urbanpovertyx* varies from the start value 1 to the end value 18.

```
. prgen hhz_usu, gen(urbanpoverty) x(_Iurban_1=1) ncases(18)

logit: Predicted values as hhz_usu varies from 1 to 18.

. prgen hhz_usu, gen(ruralpoverty) x(_Iurban_1=0) ncases(18)

logit: Predicted values as hhz_usu varies from 1 to 18.

. graph twoway (connected urbanpovertyp1 urbanpovertyx) (connected
ruralpovertyp1 ruralpovertyx), legend(label(1 "urban") label(2 "rural"))
```



Stata has a variety of commands for performing estimation when the dependent variable is dichotomous or polychotomous. Here is a list of some estimation commands for discrete dependent variable. See estimation commands for a complete list of all of Stata's estimation commands.

asmprobit	alternative-specific multinomial probit regression
binreg	GLM models for the binomial family
biprobit	bivariate probit regression
blogit	logit regression for grouped data
bprobit	probit regression for grouped data
clogit	conditional logistic regression
cloglog	complementary log-log regression
glogit	weighted least squares logit on grouped data
gprobit	weighted least squares probit on grouped data
heckprob	probit model with selection
hetprob	heteroskedastic probit model
ivprobit	probit model with endogenous regressors
logistic	logistic regression
logit	maximum-likelihood logit regression
mlogit	maximum-likelihood multinomial logit models
mprobit	multinomial probit regression
nbreg	maximum-likelihood negative binomial regression
nlogit	nested logit regression
ologit	maximum-likelihood ordered logit
oprobit	maximum-likelihood ordered probit

probit	maximum-likelihood probit estimation
rologit	rank-ordered logistic regression
scobit	skewed logistic regression
slogit	stereotype logistic regression
xtcloglog	random-effects and population-averaged cloglog models
xtgee	GEE population-averaged generalized linear models
xtlogit	fixed-effects, random-effects, and population-averaged logit models
xtprobit	random-effects and population-averaged probit models

Chapter 14. Simulation

Class

Before we get started, let's first classify Stata commands into three categories:

r-class: general commands such as **summarize**. Results are returned in r() and generally must be used before executing more commands.

The **return list** command lists results stored in r().

```
. summarize tot_exp

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
     tot_exp |     17295    17.25426    18.50382   .6677808   602.4645

. return list

scalars:
                  r(N) =  17295
              r(sum_w) =  17295
               r(mean) =  17.25426066783716
                r(Var) =  342.3914311936321
                 r(sd) =  18.50382206987606
               r(min) =  .6677808165550232
               r(max) =  602.4645385742188
               r(sum) =  298412.4382502437
```

We can save the results from **summarize** command into our dataset.

```
. gen mean=r(mean)
```

e-class: estimation commands such as **regress, logistic**, that fit statistical models. Such estimation results stay around until the next model is fitted. Results are returned in e().

The **ereturn list** command lists results stored in e(). We've seen examples of **ereturn** in regression and logistic models before.

There are also **s-class, n-class**, and **c-class** commands, but we will skip them for now.

Program command

The **program** command defines and manipulates programs, and return results in r() if using the **return** command. Or the program can be defined as e-class and returns results in e() by using the **ereturn** command. In the program, return commands are used to return saved results defined as e- or r-class. We tell Stata that the program is finished by using the **end** command.

Let's write a very simple program called *t*, to display "hello" on the screen.

```
program t, rclass
display "hello"
end
```

When we would like to use program t to display a "hello" on the screen, we simply type

```
. t
hello
```

And "hello" will appear on the screen.

The **program** command allows users to write their own programs and use them later, which provides great flexibilities and efficiency. Let's look at another example of program to calculate per capita expenditure and return the results as a scalar. We name the program *myprog*.

```
program define myprog, rclass
   version 9
   summarize `1'
   local exp = r(mean)
   summarize `2'
   local people =r(mean)
   return scalar pcexp = `exp'/`people'
end

use "u:\notes\ethiopia.dta", clear
myprog tot_exp hhz_usu
```

This program *myprog* executes like this: first our program calls **summarize** and stores the mean of the variable *tot_exp* in a local macro *exp*. The program then repeats this procedure for the second variable *hhz_usu* and stores the mean in another local macro *people*. Finally, the ratio of the two means is computed and returned as a scalar by our program in the saved result we call *r(pcexp)*.

```
return list
display "Average per capita expenditure is " r(pcexp)
Average per capita expenditure is 3.635081
```

Simulation

Bootstrap command

The **bootstrap** command handles repeatedly drawing a sample with replacement, running the user-written program, collecting the results into a new dataset, and presenting the bootstrap results. It allows user to supply an expression that is a function of the saved results of existing commands, or they can write a program to calculate the statistics of interest. The user-written

calculation program is easy to write because every Stata command saves the statistics it calculates.

For instance, assume that we wish to obtain the bootstrap estimate of the standard error of the mean of variable *tot_exp*. Stata has a built-in command, **summarize**, that calculates and displays summary statistics including means. In addition to displaying the calculated results, the **summarize** command saves them in the form of *r()*.

```
. sum tot_exp

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
     tot_exp |     17295    17.25426    18.50382   .6677808   602.4645

. return list

scalars:
                 r(N) =  17295
             r(sum_w) =  17295
              r(mean) =  17.25426066783716
               r(Var) =  342.3914311936321
                r(sd) =  18.50382206987606
               r(min) =  .6677808165550232
               r(max) =  602.4645385742188
               r(sum) =  298412.4382502437
```

In order to get a bootstrap estimate of standard error of mean, all we need to do is to type the **bootstrap** command and Stata will do the work. The **reps()** option is required since it specifies the number of bootstrap replications to be performed. The default number is 50. It is recommended to choose a large but tolerable number of replications to obtain the bootstrap estimates.

```
. bootstrap r(mean), rep(50): sum tot_exp
(running summarize on estimation sample)

Warning:  Since summarize is not an estimation command or does not set
e(sample), bootstrap has no way to determine which observations are used in
calculating the statistics and so assumes that all observations are used.
This means no observations will be excluded from the resampling due to
missing values or other reasons.

If the assumption is not true, press Break, save the data, and drop the
observations that are to be excluded.  Be sure that the dataset in memory
contains only the relevant data.

Bootstrap replications (50)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
..................................................    50

Bootstrap results                               Number of obs      =     17332
                                                Replications       =        50

      command:  summarize tot_exp
```

```
       _bs_1:  r(mean)

------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Normal-based
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      _bs_1 |   17.25426    .124953   138.09   0.000     17.00936    17.49916
------------------------------------------------------------------------------
```

The output tells us that if we would like to specify the size of the sample to be drawn, instead of all observations by default, we can set sample size by **size()** option. Note that sample size should always be less than or equal to the number of observations.

```
. bootstrap r(mean), rep(200) size(10000): sum tot_exp
(running summarize on estimation sample)
Bootstrap results                                 Number of obs      =     17332
                                                  Replications       =       200

      command:  summarize tot_exp
        _bs_1:  r(mean)

------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Normal-based
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      _bs_1 |   17.25426   .1963464    87.88   0.000     16.86943    17.63909
------------------------------------------------------------------------------
```

Still we would like to have a look at the resulting mean for each bootstrap replicates, which is saved as a Stata data file. If we add **saving()** option, the results will be saved in specified file that we can open and look into it.

```
. bootstrap _b, rep(200) size(10000) saving(bootstrap1, replace): reg tot_exp
hhz_usu
Linear regression                                 Number of obs      =     17295
                                                  Replications       =       200
                                                  Wald chi2(1)       =    727.49
                                                  Prob > chi2        =    0.0000
                                                  R-squared          =    0.1127
                                                  Adj R-squared      =    0.1126
                                                  Root MSE           =   17.4307
------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Normal-based
    tot_exp  |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
    hhz_usu  |   2.601727   .0964602    26.97   0.000     2.412668    2.790785
       _cons |   4.890831   .3839908    12.74   0.000     4.138223    5.643439
------------------------------------------------------------------------------
```

Let’s compare the bootstrap estimates to OLS, and see the difference.

```
. reg tot_exp hhz_usu

      Source |       SS       df       MS              Number of obs =   17295
```

```
------------+------------------------------           F(  1, 17293) = 2195.97
      Model |  667200.753     1  667200.753           Prob > F      =  0.0000
   Residual |  5254116.66 17293  303.829102           R-squared     =  0.1127
------------+------------------------------           Adj R-squared =  0.1126
      Total |  5921317.41 17294  342.391431           Root MSE      =  17.431

------------------------------------------------------------------------------
    tot_exp |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
------------+----------------------------------------------------------------
    hhz_usu |   2.601727   .0555198    46.86   0.000     2.492902    2.710551
      _cons |   4.890831   .2952526    16.56   0.000     4.312106    5.469556
------------------------------------------------------------------------------
```

Another option **seed()** sets the random-number seed for bootstrapping. We can change the random number seed and obtain the bootstrap estimates again, using the same number of replications. If the results change dramatically, the number of replications we choose is too small and let's pick a larger number. If results are similar enough, we probably have a large enough number.

```
. bootstrap r(mean), rep(50) seed (123456): sum tot_exp
------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Normal-based
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      _bs_1  |   17.25426   .1315345   131.18   0.000     16.99646    17.51206
------------------------------------------------------------------------------

. bootstrap r(mean), rep(50) seed (987654): sum tot_exp
------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Normal-based
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      _bs_1  |   17.25426   .1374904   125.49   0.000     16.98478    17.52374
------------------------------------------------------------------------------
```

We might agree that 50 replicates are good enough to serve our purpose.

For an example of a situation where we need to write a program, consider the case of bootstrapping the per capita expenditure. We first define the calculation routine using **program** command, which we name *myprog*, defined at the beginning of this chapter.

With our program written, we can now obtain the bootstrap estimate by simply typing

```
use "u:\notes\ethiopia.dta", clear
bootstrap pcexpd=r(pcexp), rep(100) seed(123456) size(1000): myprog tot_exp
hhz_usu
```

to execute bootstrap with our *pcexp* program for 100 replications.

```
Bootstrap results                               Number of obs      =     17332
                                                Replications       =       100
      command:  myprog tot_exp hhz_usu
       pcexpd:  r(pcexp)
------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Normal-based
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     pcexpd |   3.635081   .1065683    34.11   0.000     3.426211    3.843951
------------------------------------------------------------------------------
```

Jackknife command

The **jackknife** command calculates estimates by leaving out one observation from the sample at one time, and no sampling method is needed for jackknife estimation. The jackknife is a reliable method for estimating standard error nonparametrically. The method is easy to use, but it could be extremely computationally intensive.

The **eclass, rclass**, and **n()** options specify where the number of observations on which it based the calculated results.

eclass specifies that command save the number of observations in e(N).
rclass specifies that command save the number of observations in r(N).
n() specifies an expression that evaluates to the number of observations used.

We can estimate the standard deviation of the standard deviation of *tot_exp*, so we type

```
jackknife sd=r(sd), rclass: summarize tot_exp
```

It takes at least 15 minutes for my computer to execute this command, and the output is

```
Jackknife results                               Number of obs      =     17295
                                                Replications       =     17295
      command:  summarize tot_exp
           sd:  r(sd)
         n():  r(N)

------------------------------------------------------------------------------
             |             Jackknife
             |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
          sd |   18.50382   .9552704    19.37   0.000      16.6314    20.37625
------------------------------------------------------------------------------
```

Simulate command

The syntax of **simulate** command is

```
Simulate exp_list, reps(#): command
```

The **simulate** command performs Monte Carlo type simulations by running specified *commands* for # replications and save the results in *exp_list*. Most Stata commands and user-defined programs can be used with simulate. The **reps()** option is required to specify the number of replications to be performed.

Let's make a dataset containing means and variances of 100-observation samples from a standard normal distribution by performing the experiment 10,000 times:

```
capture program drop normalsim
program define normalsim, rclass
       version 9
       syntax [, obs(integer 1) mu(real 0) sigma(real 1) ]
       drop _all
       set obs `obs'
       tempvar z
       gen `z' = `mu' + `sigma'*invnorm(uniform())
       summarize `z'
       return scalar mean = r(mean)
       return scalar Var  = r(Var)
end

simulate mean=r(mean) var=r(Var), reps(10000): normalsim, obs(100)
```

The resulting dataset contains means and variances of 100 observations from 10,000 samples.

```
.  sum

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        mean |     10000    -.001763    .0998963   -.346022   .3886282
         var |     10000    1.001679    .1408844   .5014969   1.698629
```

To make a dataset containing means and variances of 50-observation samples from a normal distribution with a normal mean of and standard deviation of 7. Perform the experiment 10,000 times:

```
. simulate mean=r(mean) var=r(Var), reps(10000): normalsim, obs(50) mu(-3)
sigma(7)
```

Chapter 15. System Equations

The most common problem in estimating system equations are seemingly unrelated regressions (SURE). The **sureg** command fits SURE models.

Let's borrow an example from internet on student scores (details can be find at http://www.ats.ucla.edu/stat/stata/notes/hsb2.dta) and we are interested in estimating a SURE model with two equations:

read=f(write, math, science)
socst=f(write, math, science)

We will use sureg command with two equations defined by parentheses:

```
. use http://www.ats.ucla.edu/stat/stata/notes/hsb2.dta, clear
(highschool and beyond (200 cases))

. describe

Contains data from http://www.ats.ucla.edu/stat/stata/notes/hsb2.dta
  obs:           200                          highschool and beyond (200
                                                cases)
 vars:            11                          20 Jun 2000 14:13
 size:         9,600 (99.9% of memory free)
------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
id              float  %9.0g
female          float  %9.0g       fl
race            float  %12.0g      rl
ses             float  %9.0g       sl
schtyp          float  %9.0g       scl        type of school
prog            float  %9.0g       sel        type of program
read            float  %9.0g                  reading score
write           float  %9.0g                  writing score
math            float  %9.0g                  math score
science         float  %9.0g                  science score
socst           float  %9.0g                  social studies score
------------------------------------------------------------------------------

. sureg (read write math science) (socst write math science)

Seemingly unrelated regression
------------------------------------------------------------------------------
Equation          Obs  Parms        RMSE    "R-sq"       chi2        P
------------------------------------------------------------------------------
read              200      3    6.930412    0.5408     235.54   0.0000
socst             200      3    8.180626    0.4164     142.73   0.0000
------------------------------------------------------------------------------

------------------------------------------------------------------------------
```

```
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
read         |
       write |   .2376706   .0689943     3.44   0.001     .1024443    .3728968
        math |   .3784015   .0738838     5.12   0.000     .2335919    .5232111
     science |   .2969347   .0669546     4.43   0.000     .1657061    .4281633
       _cons |   4.369926   3.176527     1.38   0.169    -1.855954    10.59581
-------------+----------------------------------------------------------------
socst        |
       write |   .4656741   .0814405     5.72   0.000     .3060536    .6252946
        math |   .2763008   .0872121     3.17   0.002     .1053682    .4472334
     science |   .0851168   .0790329     1.08   0.281    -.0697848    .2400185
       _cons |   8.869885   3.749558     2.37   0.018     1.520886    16.21888
------------------------------------------------------------------------------
```

Now, say that we would like to constrain the write coefficient to be the same for the *read* and *socst* dependent variable. The **constraint** command is used to define a constraint named *1*.

```
. constraint define 1 [read]write = [socst]write
. sureg (read write math science) (socst write math science), constraint(1)
```

Extract results

We can first use **estat** command to check covariance matrix.

```
. estat vce

Covariance matrix of coefficients of sureg model

             | read                                                | socst
        e(V) |      write        math     science       _cons |      write
math     science       _cons
-------------+-----------------------------------------------------+-------------
-----------------------------------
read         |                                                     |
       write |  .00476021                                          |
        math | -.00206069   .00545882                              |
     science | -.00136975  -.00213253   .00448292                  |
       _cons |  -.0717133  -.06805451   -.0478838   10.090327 |
-------------+-----------------------------------------------------+-------------
-----------------------------------
socst        |                                                     |
       write |  .00181246  -.00078461  -.00052153  -.02730498 |  .00663256
        math | -.00078461   .00207846  -.00081197  -.02591189 | -.00287123
.00760595
     science | -.00052153  -.00081197   .00170688  -.01823185 | -.00190852  -
.00297133    .0062462
       _cons | -.02730498  -.02591189  -.01823185   3.8419121 | -.09992051
-.0948226  -.06671808   14.059187
```

Chapter 16. Simultaneous Equations

Sometimes we need to consider the issue of simultaneous equations, where the model equations are jointly determined. Variables that depend on the model are endogenous variables and variables determined outside of the model are exogenous.

Here we will estimate a simultaneous equation model from Greene's "Econometric Analysis" (2000, pp 655). This is Klein's small, dynamic model of consumption, investment, private wages, equilibrium demand, private profits, and capital stock.

$$
\begin{aligned}
C_t &= \alpha_0 + \alpha_1 P_t + \alpha_2 P_{t-1} + \alpha_3 (W_t^p + W_t^g) + \epsilon_{1t} && \text{(consumption)},\\
I_t &= \beta_0 + \beta_1 P_t + \beta_2 P_{t-1} + \beta_3 K_{t-1} + \epsilon_{2t} && \text{(investment)},\\
W_t^p &= \gamma_0 + \gamma_1 X_t + \gamma_2 X_{t-1} + \gamma_3 A_t + \epsilon_{3t} && \text{(private wages)},\\
X_t &= C_t + I_t + G_t && \text{(equilibrium demand)},\\
P_t &= X_t - T_t - W_t^p && \text{(private profits)},\\
K_t &= K_{t-1} + I_t && \text{(capital stock)}.
\end{aligned}
$$

In this model, *c, I, w* are the endogenous variables, while *g, t, wg, yr* are exogenous variables.

The dataset is obtained from internet as

```
.  use http://www.ats.ucla.edu/stat/stata/examples/greene/TBL16-2, clear
.  generate w = wg+wp
.  generate k = k1+i
.  generate yr=year-1931
.  generate p1 = p[_n-1]
(1 missing value generated)
.  generate x1 = x[_n-1]
(1 missing value generated)
. save u:\notes\table16-2.dta, replace
(note: file u:\notes\table16-2.dta not found)
file u:\notes\table16-2.dta saved
```

There is more than one way to estimate this model. We usually use instrumental variables to estimate simultaneous equations, to obtain whether a set of consistent estimates by least squares.

Method 1. use ivreg demand

We can use **ivreg** command to perform 2SLS on single equation of consumption function. The initial **ivreg** command produces the correct coefficients but the standard errors are wrong. Additional code is added to obtain the correct standard errors.

```
. ivreg c p1 (p w = t wg g yr p1 x1 k1)

Instrumental variables (2SLS) regression
```

```
      Source |       SS       df       MS              Number of obs =      21
-------------+------------------------------           F(  3,    17) =  225.93
       Model |  919.504138     3  306.501379            Prob > F      =  0.0000
    Residual |  21.9252518    17  1.28972069            R-squared     =  0.9767
-------------+------------------------------           Adj R-squared =  0.9726
       Total |  941.429389    20  47.0714695            Root MSE      =  1.1357

------------------------------------------------------------------------------
           c |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
           p |   .0173022   .1312046     0.13   0.897    -.2595153    .2941197
           w |   .8101827   .0447351    18.11   0.000        .7158    .9045654
          p1 |   .2162338   .1192217     1.81   0.087    -.0353019    .4677696
       _cons |   16.55476   1.467979    11.28   0.000     13.45759    19.65192
------------------------------------------------------------------------------
Instrumented:  p w
Instruments:   p1 t wg g yr x1 k1
------------------------------------------------------------------------------

/* additional code to get correct standard errors, thanks to Kit Baum */
. mat vpr=e(V)*e(df_r)/e(N)
. mat se=e(b)
. local nc=colsof(se)
. forv i=1/`nc' { mat se[1,`i']=sqrt(vpr[`i',`i']) }
. mat list se

se[1,4]
            p          w         p1      _cons
y1  .11804942  .04024972  .10726797  1.3207925
```

Method 2. use reg3 command

We can also run a 2SLS use **reg3** command for single equation estimation of the consumption function and obtain correct standard errors.

```
. reg3 (c p p1 w), 2sls nodfk inst(t wg g yr p1 x1 k1)

Two-stage least-squares regression
----------------------------------------------------------------------
Equation          Obs  Parms        RMSE    "R-sq"     F-Stat        P
----------------------------------------------------------------------
c                  21      3    1.135659    0.9767     279.09   0.0000
----------------------------------------------------------------------

------------------------------------------------------------------------------
             |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
c            |
           p |   .0173022   .1180494     0.15   0.885    -.2317603    .2663647
          p1 |   .2162338    .107268     2.02   0.060    -.0100818    .4425495
           w |   .8101827   .0402497    20.13   0.000     .7252632    .8951022
       _cons |   16.55476   1.320793    12.53   0.000     13.76813    19.34139
------------------------------------------------------------------------------
Endogenous variables:  c p w
```

```
Exogenous variables:   t wg g yr p1 x1 k1
------------------------------------------------------------------------------
```

Method 3. use reg3 for 2SLS

We can run a 2SLS use **reg3** command to estimate limited-information estimates for system of equations of the consumption, investment, and wage functions.

```
. reg3 (c p p1 w) (i p p1 k1) (wp x x1 yr), 2sls nodfk inst(t wg g yr p1 x1
k1)

Two-stage least-squares regression
----------------------------------------------------------------------------
Equation          Obs  Parms        RMSE    "R-sq"     F-Stat        P
----------------------------------------------------------------------------
c                  21      3    1.135659    0.9767   279.0941   0.0000
i                  21      3    1.307149    0.8849   50.89437   0.0000
wp                 21      3    .7671548    0.9874    524.005   0.0000
----------------------------------------------------------------------------

------------------------------------------------------------------------------
             |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
c            |
           p |   .0173022   .1180494     0.15   0.884    -.2196919    .2542963
          p1 |   .2162338    .107268     2.02   0.049     .0008844    .4315833
           w |   .8101827   .0402497    20.13   0.000      .729378    .8909874
       _cons |   16.55476   1.320793    12.53   0.000     13.90316    19.20636
-------------+----------------------------------------------------------------
i            |
           p |   .1502219   .1732292     0.87   0.390    -.1975503    .4979941
          p1 |   .6159434   .1627853     3.78   0.000     .2891382    .9427486
          k1 |  -.1577876   .0361262    -4.37   0.000    -.2303141   -.0852612
       _cons |   20.27821   7.542704     2.69   0.010     5.135599    35.42082
-------------+----------------------------------------------------------------
wp           |
           x |   .4388591   .0356319    12.32   0.000     .3673251    .5103931
          x1 |   .1466739   .0388361     3.78   0.000     .0687071    .2246406
          yr |   .1303956    .029141     4.47   0.000     .0718927    .1888985
       _cons |   1.500296   1.147779     1.31   0.197    -.8039674    3.804559
------------------------------------------------------------------------------
Endogenous variables:  c p w i wp x
Exogenous variables:   t wg g yr p1 x1 k1
------------------------------------------------------------------------------
```

Method 4. use reg3 for 3SLS

We can run a 3SLS use **reg3** command to estimate full--information estimates for system of equations of the consumption, investment, and wage functions.

```
. reg3 (c p p1 w) (i p p1 k1) (wp x x1 yr), 3sls inst(t wg g yr p1 x1 k1)
```

```
Three-stage least squares regression
------------------------------------------------------------------------------
Equation           Obs  Parms        RMSE    "R-sq"       chi2        P
------------------------------------------------------------------------------
c                   21      3    .9443305    0.9801   864.5909   0.0000
i                   21      3    1.446736    0.8258   162.9808   0.0000
wp                  21      3    .7211282    0.9863   1594.751   0.0000
------------------------------------------------------------------------------

------------------------------------------------------------------------------
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
c            |
           p |   .1248904   .1081291     1.16   0.248    -.0870387    .3368194
          p1 |   .1631439   .1004382     1.62   0.104    -.0337113    .3599992
           w |    .790081   .0379379    20.83   0.000      .715724    .8644379
       _cons |   16.44079   1.304549    12.60   0.000     13.88392    18.99766
-------------+----------------------------------------------------------------
i            |
           p |  -.0130791   .1618962    -0.08   0.936    -.3303898    .3042316
          p1 |   .7557238   .1529331     4.94   0.000     .4559805    1.055467
          k1 |  -.1948482   .0325307    -5.99   0.000    -.2586072   -.1310893
       _cons |   28.17785   6.793768     4.15   0.000     14.86231    41.49339
-------------+----------------------------------------------------------------
wp           |
           x |   .4004919   .0318134    12.59   0.000     .3381388     .462845
          x1 |    .181291   .0341588     5.31   0.000     .1143411    .2482409
          yr |    .149674   .0279352     5.36   0.000      .094922    .2044261
       _cons |   1.797216   1.115854     1.61   0.107    -.3898181    3.984251
------------------------------------------------------------------------------
Endogenous variables:  c p w i wp x
Exogenous variables:   t wg g yr p1 x1 k1
------------------------------------------------------------------------------
```

Chapter 17. Troubleshooting and Update

The **help** command followed by a Stata command brings up the on-line help system for that command. It can be used from the command line or from the help window. With help you must spell the full name of the command completely and correctly.

```
. help regress
```

The **help** contents will list all commands that can be accessed using help command.

```
. help contents
```

The **search** command looks for the term in help files, Stata Technical Bulletins and Stata FAQs. It can be used from the command line or from the help window.

```
. search logit
```

The findit command can be used to search the Stata site and other sites for Stata related information, including ado files. Say that we are interested in panel data, so we search for this program from within Stata by typing

```
. findit panel data
```

The Stata viewer window appears and we are shown a number of resources related to this key word.

Stata is composed of an executable file and official ado files. Ado stands for automatically loaded do file. An ado file is a Stata command that created by users like you. Once installed in your computer, they work pretty much the same way so Stata commands. Stata files are regularly updated. It is important to make sure that you are always running the most up to date Stata, and please do so regularly.

The **update** command reports on the current update level and installs official updates to Stata. It helps users to be up to date with the latest Stata ado and executable file, and copy and installs the ado files into the directory specified.

```
. update
. update ado, into(d:\ado)
```

You can keep track of all the users ado files that you have added to your package over time by **ado** command, which will list all of them, with information on where you got it from and what it does.

```
. ado
[1] package spost9_ado from http://www.indiana.edu/~jslsoc/stata
```

```
      spost9_ado Stata 9 commands for the post-estimation interpretation of

[2] package st0081 from http://www.stata-journal.com/software/sj5-1
      SJ5-1 st0081.  Visualizing main effects and interactions...
```

These ado files can be deleted by **ado uninstall** command.

```
. ado uninstall st0081
package st0081 from http://www.stata-journal.com/software/sj5-1
      SJ5-1 st0081.  Visualizing main effects and interactions...

(package uninstalled)
```

Chapter 18. Advanced Programming

Besides simple one-line commands, we can always get more from Stata by more sophisticated programming.

Looping

Consider the sample program below, which reads in income data for twelve months.

```
input famid inc1-inc12
1 3281 3413 3114 2500 2700 3500 3114 3319 3514 1282 2434 2818
2 4042 3084 3108 3150 3800 3100 1531 2914 3819 4124 4274 4471
3 6015 6123 6113 6100 6100 6200 6186 6132 3123 4231 6039 6215
end
```

Say that we wanted to compute the amount of tax (10%) paid for each months, which means to compute 12 variables by multiplying each of the inc* variable by 0.10.

There is more than one way to execute part of your do file more than once.

1. The simplest way is to use 12 **generate** commands.

```
generate taxinc1 = inc1 * .10
generate taxinc2 = inc2 * .10
generate taxinc3 = inc3 * .10
generate taxinc4 = inc4 * .10
generate taxinc5 = inc5 * .10
generate taxinc6 = inc6 * .10
generate taxinc7 = inc7 * .10
generate taxinc8 = inc8 * .10
generate taxinc9 = inc9 * .10
generate taxinc10= inc10 * .10
generate taxinc11= inc11 * .10
generate taxinc12= inc12 * .10
```

2. Another way to computer 12 variables is to use the **foreach** command.

In the example below, we use the **foreach** command to cycle through the variables *inc1* to *inc12* and compute the taxable income as *taxinc1-taxinc12*.

```
foreach var of varlist inc1-inc12 {
  generate tax`var' = `var' * .10
}
```

The initial **foreach** statement tells Stata that we want to cycle through the variables *inc1* to *inc12* using the statements that are surrounded by the curly braces. Note the curly braces must be open at the end of **foreach** command line. The first time we cycle through the statements, the value of *var* will be *inc1* and the second time the value of *var* will be *inc2* and so on until the final iteration where the value of *var* will be *inc12*. Each statement within the loop (in this case, just

the one generate statement) is evaluated and executed. When we are inside the **foreach** loop, we can access the value of *var* by surrounding it with the funny quotation marks like this `var' . The ` is the quote right below the ~ on your keyboard and the ' is the quote below the " on your keyboard. The first time through the loop, `var' is replaced with *inc1*, so the statement

```
generate tax`var' = `var' * .10
```

becomes

```
generate taxinc1 = inc1 * .10
```

This is repeated for *inc2* and then *inc3* and so on until *inc12*. So, this **foreach** loop is the equivalent of executing the 12 **generate** commands manually, but much easier and less error prone.

3. The third way is to use **while** loop.

First we define a Stata local variable that is going to be the loop increment. Similar to the **foreach** command, codes are in terms of local variable `var'.

```
local i=1
while `i'<=12 {
generate taxinc`i'=inc`i'*0.10
local i=`i'+1
}
```

Local variable *i* can be seen as a counter, and the **while** command states how many times the commands within the **while** loop are going to be replicated. This statement basically says do until counter value reaches the limit 12. Note the curly braces must be open at the end of **while** command line. All commands between curly braces will be executed each time the system go through the **while** loop. So first the statement

```
generate taxinc`i'=inc`i'*0.10
```

becomes

```
generate taxinc1=inc1*0.10
```

The counter value is increased by 1 unit afterwards. Note that the fourth line means the value of local variable i will be increased by 1 from its current value stored in `i'.

Create variable containing percentiles

If we are interested in divide a dataset by its percentile, say, quartile, there are at least two ways to realize it.

1. use **gen** and **egen** command to create a running counter, then using **generate** and **replace** command to create percentile code.

Here is an example from my colleague.

```
gen pop=hhsize*weight
gen popsum=sum(pop)
egen totpop=sum(pop)
gen cut=0
replace cut=1 if popsum<(totpop/5)
replace cut=2 if popsum<(2*totpop/5) & popsum>=(totpop/5)
replace cut=3 if popsum<(3*totpop/5) & popsum>=(2*totpop/5)
replace cut=4 if popsum<(4*totpop/5) & popsum>=(3*totpop/5)
replace cut=5 if popsum<=(totpop) & popsum>=(4*totpop/5)
```

2. use **pctile** and **xtile** command

The **pctile** command creates a new variable containing the percentiles of another variable. The **xtile** command creates a new variable that categorizes exp by its quantiles. Let's try those two commands and check the output. The pctile command produces three cutting points at 25^{th}, 50^{th} (meidan), and 75^{th} percentile. The xtile command assign a new variable to recode which quartile the observation belongs to.

```
. pctile pct1=tot_exp [pweight=samplewt], nq(4)
. tab pct1

percentiles |
of tot_exp  |      Freq.     Percent        Cum.
------------+-----------------------------------
   7.678307 |          1       33.33       33.33
   11.19419 |          1       33.33       66.67
   16.24392 |          1       33.33      100.00
------------+-----------------------------------
      Total |          3      100.00

. xtile pct2=tot_exp [pweight=samplewt], nq(4)
. tab pct2

4 quantiles |
of tot_exp  |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |      3,414       19.74       19.74
          2 |      3,575       20.67       40.41
          3 |      4,131       23.89       64.30
          4 |      6,175       35.70      100.00
------------+-----------------------------------
      Total |     17,295      100.00
```

Chapter 19. Helpful Sources

http://www.stata.com/
http://www.stata.com/statalist/archiv
Statalist is hosted at the Harvard School of Public Health, and is an email listserver where Stata users including experts writing Stata programs to users like us maintain a lively dialogue about all things statistical and Stata. You
can sign on to statalist so that you can receive as well as post your own questions through email.

http://ideas.repec.org/s/boc/bocode.html
http://www.princeton.edu/~erp/stata/main.html
http://www.cpc.unc.edu/services/computer/presentations/statatutorial/
http://www.ats.ucla.edu/stat/stata/