The University of Georgia

Department of Agricultural & Applied Economics

Beginners Guide to SAS & STATA software

Developed by Vahé Heboyan

Supervised by Dr. Tim Park

Introduction

The purpose of this Guide is to assist new students in MS and PhD programs at the Department of Agricultural & Applied Economics at UGA to get started with SAS and STATA software. The guide will help beginning users to quickly get started with their econometrics and statistics classes.

This guide is not designed to be a substitute to any other official guide or tutorial, but serve as a starting point in using SAS and STATA software. At the end of this guide, several links to the official and unofficial sources for advanced use and more information will be provided.

This guide is based on the so-called pre-programmed canned procedures.

Using built-in help

Both SAS and STATA have build-in help features that provide comprehensive coverage of how to use the software and syntaxes (command codes).

- In SAS: go to HELP → Books and Training → SAS Online Tutor

- In STATA: go to HELP and use first three options for contents, keyword search and STATA command search, respectively.

SAS Tutorial

1. Working with data

a. Reading data into SAS

The most convenient way to read data into SAS for further analysis is to convert your original data file into Excel 97 or 2000. Make sure there are no multiple sheets in the file. Usually default Excel has three sheets, make sure you remove the last two. To read excel file (or other format) into SAS library, follow the path below. For your own convenience, include the names of the variables in the first row of your excel file. SAS will automatically read those as variables names, which you can use to construct command codes. For example if one of the variables if the price of a commodity, then you may chose to name it as *P* or *price*.

File → Import Data → choose data format (default is Excel) → Next → browse for the file → Next → create a name for your new file under Member (make sure to keep the same WORK folder unchanged) → Next → you may skip this step and click on Finish.

On the left hand side of the SAS window there is a vertical sub-window called Explorer and the default shows two directories: Libraries and File Shortcuts. Double click on the Libraries, then Work folder and locate your data file. Double click on it to view your loaded data. It should open in a new window and have the following name – VIEWTABLE: WORK.*name of your file* .

Remember that when you activate the SAS program. It opens there additional sub-windows that have the following function/use:

- EDITOR – for inputting your command codes;
- LOG – to see the errors if any in your code after execution;
- OUTPUT – to view the output after successful execution of your code.

After you load your data into SAS you can use the following command to read it into the Editor window. Throughout this manual, the data file will have the name **test** unless otherwise specified.

```
data test;
```

Reminder! Do not forget to put semicolons at the end! Now you may move on with your analysis!
Warning: Some users have encountered problems when they close VIEWTABLE window, i.e. the data file disappears. You may load it again, or simply leave the window open.

b. Creating the so-called 'do-files'

You input your program in the default sub-window called EDITOR. You may choose to save it for future use or editing. After you type the commands or the first line of it, simply go to File → Save As…→ give a new name and choose the directory. Anytime you need to use the command, just call it from the same directory and it will open with the information you saved the last. Remember to save your program before you close the SAS or that particular editing sub-window.

Note: *after you save it, the EDITOR sub-window will take a new name based on the name you choose*.

c. Examining the data

In SAS you can view your data as well as its summary statistics. For the beginners, this is a good point to start with, as it gives you the opportunity to see how SAS reads your data and also examine them.

To print your data on the Output menu, type the following:

```
data test;          * indicates the data file to be used ;
proc print;         * prints data found in the "test" file ;
run;                * runs and executes the program ;
```

After you type these commands, click on the "***running man***" icon to execute your commands (located on the top row of the SAS window). You can view the results in the Output window.

Hints: Always finish your command program with "**run;**" and place the cursor after it before you execute the command. You can always comment the command lines by placing the text between star(*) and a semicolon(;) as seen in the command above *(in SAS the comments are automatically turned into green and the executable command codes into blue)*.

To view summary statistics, use the command below. It will display the mean, standard deviation, min and maxima of your data.

```
data test;
proc means;
run;
```

You may customize data examination by using descriptive statistics options that are specified after the PROC MEANS statement. An example is provided below:

```
data test;
proc means max min;     * generates max and min values of ;
                        * the dataset ;
run;
```

The table below lists descriptive statistics options available in SAS.

Option	**Description**
CLM	Two-sided confidence limit for the mean
CSS	Corrected sum of squares
CV	Coefficient of variation
KURTOSIS	Kurtosis
LCLM	One-sided confidence limit below the mean
MAX	Maximum value
MEAN	Average
MIN	Minimun value
N	Number of observations with nonmissing values
NMISS	Number of observations with missing values
RANGE	Range
SKEWNESS	Skewness
STDDEV / STD	Standard Deviation
STDERR	Standard error of the mean
SUM	Sum
SUMWGT	Sum of the `Weight` variable values.
UCLM	One-sided confidence limit above the mean
USS	Uncorrected sum of squares
VAR	Variance

The following PROC statements in SAS assist in further exploration of your data. They are used in the same manner as the PROC statements discussed above (i.e. PROC PRINT and PROC MEANS).

Statements	**Description**
proc contents	Contents of a SAS dataset
proc print	Displays the data
proc means	Descriptive statistics
proc univariate	More descriptive statistics
proc boxplot	Boxplots
proc freq	Frequency tables and crosstabs
proc chart	ASCII histogram
proc corr	Correlation matrix

d. Sorting data

One can easily sort raw data in SAS using the PROC SORT statement. The default sorts in ascending order. You may also customize such that it sorts in descending order. The command below will sort your data by the values of the variable ***p***.

```
proc sort data=test;     * starts PROC SORT statement ;
      by descending p;   * specifies the order & variable ;
run;                     * executes the code ;
```

e. Creating new variables

Using your initial data set you can create new variables in SAS. For example if you want to transform your original data into logarithmical form, the code below may be used. Assume that in your original data set you had three variables (variable names in the file are provided in the parenthesis):

a) Quantity (***q***);
b) Price (***p***); and
c) Exchange rate (***ex***);

```
data test2;    * indicates the new file to be created...;
               * with the new variable(s);
set test;      * indicates the file where original data are ;
lnq=log(q);    * specifies the new variable lnq ;
lnp=log(p);    * specifies the new variable lnp ;
lnex=log(ex); * specifies the new variable lnex ;
proc print;    * prints the new data file ;
run;
```

The code above prints the original variables as well as the newly created ones. If you want to print only the new ones and delete the old ones, use the command below.

```
data test2;    * indicates the new file to be created...;
               * with the new variable(s);
set test;      * indicates the file where original data are ;
lnq=log(q);    * specifies the new variable lnq ;
lnp=log(p);    * specifies the new variable lnp ;
lnex=log(ex); * specifies the new variable lnex ;
drop q p ex;   * drops (deletes the old data)
proc print;    * prints the new data file with new variables;
               * only;
run;
```

When creating new variables you can use the basic mathematical expressions, such as multiplying (*), dividing (/), subtracting (-), adding (+), exponentiation (**), etc.

Remember: *the name of the new data file cannot be the same as the original one.*

f. Creating dummies

Dummy variables are commonly used to specify qualitative characteristics of some variables such as gender, race, and geographical location. For example, when gender of the consumer/respondent is introduced into a model, one may assign female consumers value of 1 (one) and 0 (zero) to the male consumers. Dummies may also be used to separate a variables in the original dataset based on a pre-defined formula. See more on dummy variables in your Econometrics textbook.

Assume we have a data set called ***consumer.xls*** which contains data on respondents' consumption of cheese (***q***), cheese price (***p***), household annual income (***inc***), respondent's age (***age***), and gender (***sex***). In the original data set gender is coded as 'm' for male and 'f' for female. Age is coded according to the actual age.

In order to incorporate the gender variable (***sex***) into the model we need to assign it a numeric value. SAS will not be able to use original gander data for analysis (i.e. it will not accept 'm' and 'f' as values for gender variable).

Now we need to create a dummy variable for gender variable. Additionally, we may want to group the respondents in 2 groups according to their age; i.e. one group will include young consumers (up to 25 years of age) and older consumers (25 and above). The code below will helps to make the changes and prepare data for further analysis.

```
data consumer;      * read original data ;
proc print;         * print on screen to view data;

data consumer_2;    * name the new data-file ;
set consumer;       * indicates the file with original data ;

if sex = "m" then d1 = 1;     * define gender dummy ;
ELSE d1 = 0;

if age > 25 then d2 = 1;      * define age group dummy ;
ELSE d2 = 0;

proc print;         * print on screen to view data ;
run;                * execute the program ;
```

Note*: d1 and d2 are the news for newly created dummy variables. You may name them as you wish.*

2. Estimation

This section introduces to the Ordinary Least Squares (OLS) estimation, model diagnostics, hypothesis testing, confidence intervals, etc.

a. Linear regression

SAS PROC procedure lets to do OLS estimation using a simple command instead of writing down the entire program. The PROC REG procedure incorporates the entire command that is necessary for OLS estimation.

To estimate a regression model using OLS procedure, use the following command below.

```
proc reg data=test;      * starts OLS & specifies the data;
model q = p t;       * specifies the model to be estimated;
run;
```

When specifying the model, after the keyword MODEL, the dependent variable is specified, followed by an equal sign and the regressor variables. Variables specified here must be only numeric. If you want to specify a quadratic term for variable ***p*** in the model, you cannot use **p*p** in the MODEL statement but must create new variable (for example, ***psq=p*p***) in the DATA step discussed above.

The PROC REG and MODEL statements do the basic OLS regression. One may use various options available in SAS to customize the regression. For example, if one needs to display residual values after the regression is complete, one may use the option commands to do so. A sample list of options available in SAS are listed in the table below. Check the SAS online help for more options. Options are specified in the following way:

```
proc reg data=test;
model q = p t / option ;
run;
```

NOTE: *The default level of significance in SAS is set at 95%. To change it use the appropriate option that is listed in the table below.*

Option	Description
These options are set after the PROC REG statement with just a space between them. For example `proc reg option;`	
ALPHA = *number*	Sets the significance level used for construction of confidence intervals. The value must be between 0 and 1. The default value of 0.05 results in 95% intervals.
CORR	Displays the correlation matrix for all variables listed in the MODEL statement.
DATA=*datafile*	Names the SAS data set to be used by PROC REG.

SIMPLE	Displays the sum, mean, variance, standard deviation, and uncorrelated sum of squares for each variable used in PROC REG.
	NOTE*: this option is used with the PROC REG statement only. Will not work with the MODEL statement*. Example:
	`data test;`
	`proc reg simple;`
	`model q = p t;`
	`run;`

The table below lists the options available for MODEL statement.

Option	Description
These options are specified in the MODEL statement after a slash (/). For example, `model q = p t / option;`	
NOINT	Fits a model without the intercept term
ADJRSQ	Computes adjusted R^2
ACOV	Displays asymptotic covariance matrix of estimates assuming heteroscedasticity
COLLIN	Produces collinearity analysis
COLLINOINT	Produces collinearity analysis with intercept adjusted out
COVB	Displays covariance matrix of estimates
CORRB	Displays correlation matrix of estimates
CLB	Computes 100(1-α)% confidence limits for the parameter estimates
CLI	Computes 100(1-α)% confidence limits for an individual predicted value
CLM	Computes 100(1-α)% confidence limits for expected value of the dependent variable
DW	Computes a Durbin-Watson statistic
P	Computes predicted values
ALL	Requests the following options: ACOV, CLB, CLI, CLM, CORRB, COVB, I, P, PCORR1, PCORR2, R, SCORR1, SCORR2, SEQB, SPEC, SS1, SS@, STB, TOL, VIF, XPX. *For the options not discussed here, see SAS online help.*
ALPHA = *number*	Sets the significance level used for construction of confidence and prediction intervals and tests. The value must be between 0 and 1. The default value of 0.05 results in 95% intervals.
NOPRINT	Suppresses display of results
SINGULAR=	Sets criterion for checking for singularity

b. Testing for Collinearity

The COLLIN option performs collinearity diagnostics among regressors. This includes eigenvalues, condition indices, and decomposition of the variance of the estimates with respect to each eigenvalue. This option can be specified in a MODEL statement.

```
data test;
proc reg;
model q = p t / collin;
run;
```

NOTE: *if you use the* **collin** *option, the intercept will be included in the calculation of the collinearity statistics, which is not usually what you want. You may also use* **collinoint** *to exclude the intercept from the calculations, but it still includes it in the calculation of the regression.*

c. **Testing for Heteroskedasticity**

The SPEC option performs a model specification test. The null hypothesis for this test maintains that the errors are homoskedastic, independent of the regressors and that several technical assumptions about the model specification are valid. It performs the White test. If the null hypothesis is rejected (small p-value), then there is an evidence of heteroskedasticity. This option can be specified in a MODEL statement.

```
data test;
proc reg;
model q = p t / spec;
run;
```

d. **Testing for Autocorrelation**

DW option performs autocorrelation test. It provides the Durbin-Watson *d* statistics to test that the autocorrelation is zero.

```
data test;
proc reg;
model q = p t / dw;
run;
```

e. **Hypothesis testing**

In SAS you can easily test single or joint hypothesis after you successfully complete the estimation. For example, if we want to test the null hypothesis that the coefficient of the ***p*** variable is 1.5 (i.e. β_p=1.5), then the following command will be used.

```
proc reg data=test;
model q = p t;
test p = 1.5;           * sets up the hull hypothesis ;
run;
```

NOTE: *remember that you can always look at the t-values and p-values in the Parameter Estimation section of SAS output for the null hypothesis of coefficient is zero* $(\beta_i = 0)$.

To test the joint hypothesis of β_p=1.5 and β_t=0.8 the command below may be used.

```
proc reg data=test;
model q = p t;
test p = 1.5, t = 0.8;  * sets up the hull hypothesis ;
run;
```

Use the command below to test the hypothesis of $\beta_p + \beta_t = 2.3$.

```
proc reg data=test;
model q = p t;
test p + t = 2.3;         * sets up the hull hypothesis ;
run;
```

NOTE: *in the TEST statement the names of the variables are specified. SAS will automatically associate those with their coefficients.*

3. Creating plots

The PLOT statement in SAS enables to create scatter plots on Y-X axis (vertical-horizontal). Use the command below to create the basic plot.

```
proc reg data=test;       * starts OLS regression ;
model q = p t;
plot q*p;                 * specifies the Y and X ;
run;                      * executes the command ;
```

After executing this command, a new window will open with your ***q*** variable on vertical axis (Y) and ***p*** variable on horizontal axis (X).

You may also create multiple plots using the same command line. The code below will create various combinations of plots using the same sets of variables.

```
proc reg data=test;
plot p*q p*t q*t;
run;
```

The command above will create three separate scatter plots. One may use the code below for identical plotting. Both codes will create the same sets of scatter plots.

```
proc reg data=test;
plot (p q)*(q t);
run;
```

In many applications you will required to plot model residuals against a particular variable. Use the command below to do so.

```
proc reg data=test;
model q = p t;
plot r.*q;                * r. in SAS stands for residual ;
run;
```

The table below shows a number of other keywords that can be used with the **PLOT** statement and the statistics they display. Note that the keywords should be used in the PLOT statement line and be constructed as the one in the case with the residual above. For example,

```
plot residual.*COOKD.;
```

Keyword *	Statistics
COOKD.	Cook's D influence statistics
COVRATIO.	standard influence of observation on covariance of betas
DFFITS.	standard influence of observation on predicted value
H.	leverage
LCL.	lower bound of 100(1-α)% confidence interval for individual prediction
LCLM.	lower bound of 100(1-α)% confidence interval for the mean of the dependent variable
PREDICTED. (PRED. ; P.)	predicted values
PRESS.	residuals from refitting the model with current observation deleted
RESIDUAL. (R.)	residuals
RSTUDENT.	studentized residuals with the current observation deleted
STDI.	standard error of the individual predicted value
STDP.	standard error of the mean predicted value
STDR.	standard error of the residual
STUDENT.	residuals divided by their standard errors
UCL.	upper bound of 100(1-α)% confidence interval for individual prediction
UCLM.	upper bound of 100(1-α)% confidence interval for the mean of the dependent variables

** The keywords in the parenthesis are the alternative keywords for the same procedure. The use of either one is correct.*

NOTE: ***the dot (.) after the keyword must be specified.***

4. Weighted Least Squares Estimation

WLS is performed by adding a weight to the PROC REG statement. A WEIGHT statement names a variable in the input data set with values that are relative weights for a weighted least-squares fit. If the weight value is proportional to the reciprocal of the variance for each observation, then the weighted estimates are the best linear unbiased estimates (BLUE).

Values of the weight variable must be nonnegative. If an observation's weight is zero, the observation is deleted from the analysis. If a weight is negative or missing, it is set to zero, and the observation is excluded from the analysis. An example is provided below.

```
proc reg data=test;
model q = p t;
weight p;                * specifies the weight variable ;
run;
```

5. **GLM Regression**

PROC GLM analyzes data within the framework of General linear models. PROC GLM handles models relating one or several continuous dependent variables to one or several independent variables. The independent variables may be either *classification* variables, which divide the observations into discrete groups, or *continuous* variables.

The general GLM statement is provided below:

```
proc glm data=test;
model dependent(s) = independent(s) / options;
run;
```

For the detailed description of PROC GLM statement and options available to estimate general linear models please see the "The GLM Procedure" document available online through the SAS Institute.

http://www2.stat.unibo.it/ManualiSas/stat/chap30.pdf

6. **Seemingly Unrelated Regression**

Assume we have two regression models:

```
science = math female
write   = read female
```

It is the case that the errors (residuals) from these two models would be correlated because all of the values of the variables are collected on the same set of observations. In this situation we can use seemingly unrelated regression to estimate both models simultaneously while accounting for the correlated errors at the same time, leading to efficient estimates of the coefficients and standard errors. For this purpose we use **PROC SYSLIN** statement with option **SUR**. The **PROC SYSLIN** estimates both models simultaneously. Below is an example of SUR regression.

```
proc syslin data=test SUR;
model science = math female ;
model write = read female ;
run;
```

The first part of the output consists of the OLS estimate for each model. The second part of the output gives an estimate of the correlation between the errors of the two models. The last part of the output will have the seemingly unrelated regression estimation for our models. Note that both the estimates of the coefficients and their standard errors are different from the OLS model estimates shown above.

NOTE: *one can easily conduct SUR estimation using 3 and more models. Th procedure is the same. Just add another MODEL statement.*

7. Non-Linear Estimation

f. LOGIT

PROC LOGISTIC statement in SAS performs logistic regression. It is necessary to include descending option when a variable is coded 0/1 with 1 representing the event whose probability is being modeled. This is needed so that the odds ratios are calculated correctly.

```
proc logistic data=test descending ;
model payment = income age gender ;
run;
```

For the detailed description of PROC LOGISTIC statement and options available to conduct logistic regression please see the "The LOGISTIC Procedure" document available online through the SAS Institute.

http://www2.stat.unibo.it/ManualiSas/stat/chap39.pdf

g. PROBIT

PROC PROBIT statement in SAS computes maximum likelihood estimates of regression parameters and the natural (or threshold) response rate for quantal response data. It estimates the parameters β and C of probit equation using a modified Newton-Raphson algorithm.

The general PROBIT statement is provided below:

```
PROC PROBIT DATA=file < options > ;
   CLASS variables ;
   MODEL response=independents < / options > ;
   BY variables ;
   OUTPUT < OUT=SAS-data-set > <options > ;
   WEIGHT variable ;
```

For the detailed description of PROC PROBIT statement and options available to conduct maximum likelihood estimation please see the "The PROBIT Procedure" document available online through the SAS Institute.

http://www2.stat.unibo.it/ManualiSas/stat/chap54.pdf

8. **External Resources**
This manual contains the basic information that will be needed to start learning the SAS software. For more advanced use, I will encourage to use the resources available through the SAS software help or others that are available through other organizations. For your convenience, two sources containing one of the most comprehensive resources are listed below:

   a. **SAS/STAT User Guide** (*PDF files*). Dipartimento di Scienze Statistiche "Paolo Fortunati", Bologna, Italia. Available at: http://www2.stat.unibo.it/ManualiSas/stat/pdfidx.htm

      Contains downloadable PDF files on all procedures available in SAS (Version 8). This is a very comprehensive source and I would personally encourage using it.

   b. **SAS Learning Resources.** University of California at Los Angeles Academic Technology Services. Available at: http://www.ats.ucla.edu/stat/sas/

      Contains learning resources that help to master SAS software including text and audio/video resources. This is especially useful for those who just started to learn SAS.

1. Introduction to STATA

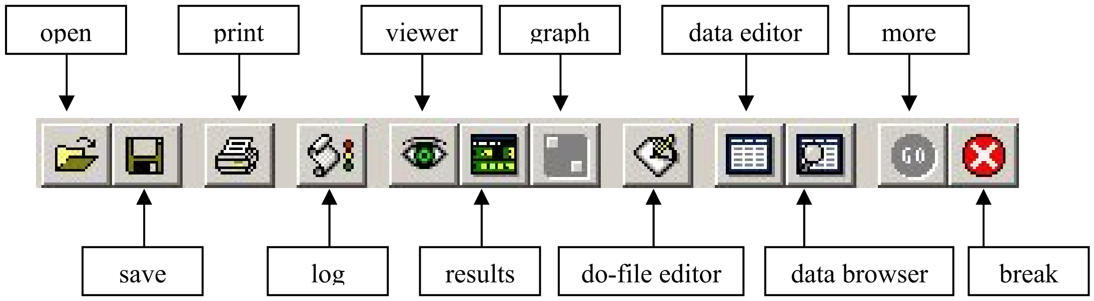
a. Limitations

The current version of STATA that is used at the Department of Agricultural and Applied Economics at UGA is the Intercooled STATA that has the following limitations:

b.	Max number of variables	- 2,047
c.	Max number of observations	- 2,147,483,647 (limited to memory)
d.	Max number of characters for a string variable	- 80
e.	Matrices	- 800 x 800

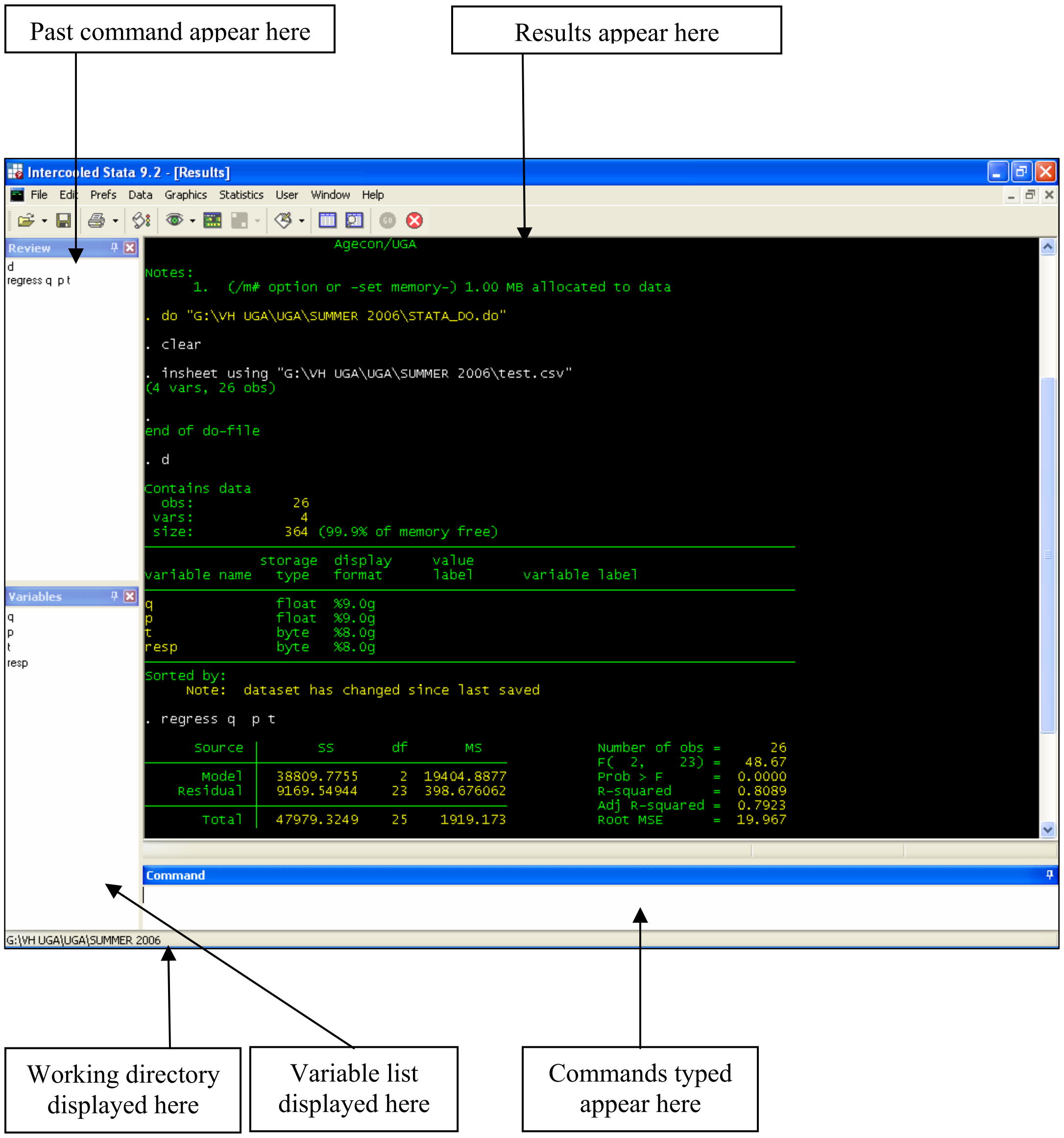
b. STATA toolbar and window

STATA toolbar consists of several buttons that have the following functions.



open:	open a STATA dataset
save:	save a dataset
print:	print contents of active window
log:	to start or stop, pause or resume a log file
viewer:	open viewer window, or bring to the front
results:	open results window, or bring to the front
graph:	open graph window, or bring to the front
do-file editor:	open do-file editor, or bring window to the front
data editor:	open data editor, or bring window to the front
data browser:	open data browser, or bring window to the front
more:	command to continue when paused in long output
break:	stop the current task. This command returns the system to as it was before you issued the command.

The default STATA working window has the following view. The descriptions of the individual components are provided below.


Past command appear here
Results appear here
Intercooled Stata 9.2 - [Results]
File Edit Prefs Data Graphics Statistics User Window Help
Review
d
regress q p t
Variables
q
p
t
resp
Command
G:\VH UGA\UGA\SUMMER 2006
Working directory displayed here
Variable list displayed here
Commands typed appear here

```
                       Agecon/UGA

Notes:
      1.  (/m# option or -set memory-) 1.00 MB allocated to data

. do "G:\VH UGA\UGA\SUMMER 2006\STATA_DO.do"

. clear

. insheet using "G:\VH UGA\UGA\SUMMER 2006\test.csv"
(4 vars, 26 obs)

.
end of do-file

. d

Contains data
  obs:            26
 vars:             4
 size:           364 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
q               float  %9.0g
p               float  %9.0g
t               byte   %8.0g
resp            byte   %8.0g
-------------------------------------------------------------------------------
Sorted by:
     Note:  dataset has changed since last saved

. regress q  p t

      Source |       SS       df       MS              Number of obs =      26
-------------+------------------------------           F(  2,    23) =   48.67
       Model |  38809.7755     2  19404.8877           Prob > F      =  0.0000
    Residual |  9169.54944    23  398.676062           R-squared     =  0.8089
-------------+------------------------------           Adj R-squared =  0.7923
       Total |  47979.3249    25    1919.173           Root MSE      =  19.967
```

c. STATA Transfer

This is a separate package that is used to convert a variety of different file-types into other formats. For example, you can easily convert Excel into STATA or vice versa.

2. Working with data

a. Creating the so called ‘do-files’

Even though you can directly type your command statements in the STATA Command window, it is advised to create a STATA do-file, which will allow you skip typing each statement line every time you need to re-run the program as well as for the use in the future. Just click on the “do-file editor” button and save it. Now you can start writing your program in do-file editor window and execute the program directly from there by selecting Tools → DO or simply using your keyboard, Ctrl+D.

NOTE: Unlike SAS, in STATA you **do not** end the statement with semicolons.

b. Loading data into STATA data editor

To read data in STATA you can either convert the original file into STATA-friendly format or simply create STATA data file (*.dta). Follow the steps below to create a STATA data file.

a) Copy data from the original file. For example notepad or Excel.
b) Open STATA data editor (see Section 1b: STATA toolbar and window) and paste copied data into the editor. If you copy the variable names from the original data file, then after pasting STATA data editor will use them as variable headings when creating a new file. Otherwise, it will name the variables according to its default procedures (e.g. var1, var2, etc.)

   NOTE: *throughout the text, var1, var2, etc. are generic variable names.*

c) Click on Preserve (in Data Editor) and close the Data Editor window.
d) Go to STATA window and select
   - File → Save As… → choose *Stata Data file*→ Save
e) Now you can use that file for your estimation.

c. Reading data into STATA

There are two primary ways of uploading data into STATA. The use of either one will depend on personal preferences. Instructions for both are provided below.

i. When using non-STATA data file, make sure to convert it into *.csv (Excel comma separated values) format or any other that is readable in STATA. For such data use the following statement:

```
insheet using “C:\MyDocuments\test.csv”
```

NOTE: Always put the file path into quotation marks. It is also suggested to use the `clear` statement before the `insheet` statement to clear the use of previous dataset(s), unless otherwise needed. Regular **.xls* file format is not accepted by STATA.

```
clear
insheet using “C:\MyDocuments\test.csv”
```

ii. When using STATA data file (see Section 2.b) use the following statement:

```
clear
use "C:\MyDocuments\test.dta"
```

d. Changing memory

There is a default memory volume set in STATA (=1m), which may not always be enough for your estimation. To change the memory assigned to STATA use the following statement:

```
set mem #k
```

where **`#`** is a number greater than the size of the dataset, and less than the total amount of memory available on your system and **`k`** defines the usnit of measurement. In this case it means kilobytes. To use megabytes, use **`m`** instead of **`k`**. Usually setting memory to **`100m`** will be adequate for most analysis.

NOTE: *To use comments in STATA, simply start your comment from a new line and a star (*) before the comment text. For example, in the statement below, the first line is a comment and will not be used by STATA as a command statement.*

```
* set new memory volume
set mem #k
```

e. Saving files

To save a data file, use the following statement;

`save, replace`	[overwrites current file]
`save `*`filename`*`, replace`	[saves file as *filename*. Replace is optional, but necessary if a file of that name already exists.]

where `filename` is the name you give to the file.

NOTE: *The statements above will save the file in the same directory where the original file is located.*

f. LOG files

You can save all output appearing in the Results window in a log file. The log file can be saved either as a STATA markup and Control Language (SMCL) or as a text (ASCII) file. SMCL is the default format in STATA. Please note that the SMCL logs cannot be read by other packages and should only be read and printed from the Viewer.

- To start a log, use the statement below:

```
log using filename                [starts an SMCL log]
log using filename, replace       [overwrites filename.smcl]
log using filename.log            [starts a text log]
```

- To translate a SMCL log file to text, go to File → Log → Translate

- If you want to create a log file that only contains the results and not command statements. You can use pause and resume options as illustrated below:

```
log off                [temporarily suspends log file]
log on                 [resumes log file]
```

- To close a log file, use the statement below:

```
log close              [closes current log file]
```

NOTE: *As in the do-file, you can add comments in a new line preceded by a star (*).*

g. Controlling output

`-more-` may appear in your Results window when the output is longer than the screen height. At anytime you can press Enter to see the next line or simply click on `-more-` to go to the end of the listing. To turn off or on the more command, use the following statement:

```
set more off
set more on
```

h. Examining the data

STATA has different alternatives for examining the datasets in STATA. Their brief description and statements are provided below:

NOTE: *Throughout the rest of the text, the underlined portion of the statement indicates that the portion may be used instead of the full statement. For example,*

- to produce summary of contents of a dataset

`describe`	[describes dataset in current memory]

NOTE: Throughout the rest of the text, the underlined portion of the statement indicates that the portion may be used instead of the full statement. For example, the statement `d` in the statement (`describe`) above serves exactly the same purpose as the full statement `describe`.

`d using` *`filename`*	[describes a stored STATA dataset]
`d` *`varlist`*	[describes a subset of a dataset]

where `varlist` is the specified variable(s). You may simply list the variables you want to be described with a space in-between. For example,

```
d var1 var2 var3
```

- To calculate and display a variety of summary statistics, use the command statement below;

`summarize`	[summarize whole dataset]
`su` *`varlist`*	[summarize subset varlist]
`su, d`	[outputs detailed summary]
`su` *`variable1,`* `d`	[outputs detailed summary of **variable1**]

- The most detailed examination of data is performed using `list` statement. It displays the values of variables by observation.

`list`	[lists **all** variables by observation]
`l` *`varlist`*	[lists specified variable(s)]

- To close a log file, use the statement below:

`log close`	[closes current log file]

NOTE: *the arguments illustrated below can be used with all descriptive statements discussed above, except otherwise stated.*

`d var4-var7`	[describes variables between var4 and var7]

HINT: *To sort a dataset in an alphabetical order, use the statement below:*

`aorder varlist`	*[if no variable is specified, it will sort the entire dataset]*

`su abc*`	[summarize all variables beginning with the string `abc*`]
`su abc?5`	[summarize all variables beginning with the string `abc` and ending with `7`]

The statements below work only with `list` command.

`list in 3`	[list the 3rd observation]
`list in -3`	[list the 3rd from last observation]
`list in 11/29`	[list observations 11 through 29]
`list in 7/-2`	[list observation 7 to 2nd from last]

- To view extra information on the variables, such as summary statistics of numerics, example data-point of strings, details of missing values, data ranges, etc. use `codebook` command statement. If not variable name is specified, the command will give information on all variables in the dataset.

i. Sorting data

The sort statement in STATA allows sorting data into ascending order of the values of the variables of *`varlist`*.

`sort var1`	[sorts data by the values of the variable `var1`]

To sort in descending order, use the following statement:

```
gsort -var1
```

NOTE: *notice that the variable listed with `gsort` statement has minus sign in front of it.*

j. Creating new variables

i. The `generate` and `egen` commands in STATA are used to create new variables. The statements below create variables that are algebraic expressions of others.

`g varNEW=var2*var3`	[creates new variable that is the product of var2 and var3]

`g varNEW=var2+var3`	[creates new variable that is the sum of var2 and var3]
`g varNEW=exp(var8/var2)`	[creates new variable that is the exponential of the ratio of var8 and var2]
`g varNEW=log(var1)`	[creates new variable that is the natural logarithm of var1]
`g varNEW=var2^3`	[creates new variable as 3rd power of var2]

NOTE: *statement* `gen` *may also be used along with* `generate` *&* `g`.

ii. The `egen` command typically creates new variables based on summary measures, such as sum, mean, max and min.

`egen varNEW=sum(var2)`	[sum of var2]
`egen varNEW=max(var1)`	[largest value in var1]
`egen varNEW=mean(var3)`	[average value of var3]
`egen varNEW=count(var1)`	[counts number of non-missing observations]

iii. The following statement replaces the old variable (varOLD) with the new one (varNEW).

`replace varOLD=var2*var3`	[replaces the varOLD with the new variable that is the product of var2 and var3]

iv. There are other two cases where `replace` statement is used.

(1) Assume that we want to create a new variable (varNEW) from the existing variable based on some specified criteria. For example, given the age of the respondents (AGE), we want to group them into 3 (three) group-ranges: 0-25; 26-60; and 61 and above. The statements below will create the new variable, say ***agerange***, based on the listed criteria.

`g agerange= .`	[creates variable ***agerange*** that has missing values]

```
replace agerange = 1 if 0<age & age<=25
```

[replace the *agerange* with 1 if the condition is met, *i.e. 0<age<25*]

```
replace agerange = 2 if 26<=age & age<=60
```

[replace the agerange with 2 if the condition is met, *i.e. 25<var2<60*]

```
replace agerange = 3 if age>60
```

[replace the agerange with 3 if the condition is met, *i.e. age>60*]

NOTE: *in STATA* **(.)** *stands for missing values. i.e. the statement above simply creates a new variable that has no values for any of the observations.*

The list of relational operators used in STATA is:

`==`	equal to
`~=`	not equal to
`>`	greater than
`>=`	greater than or equal to
`<`	less than
`<=`	less than or equal to

(2) To create a **dummy variable**. For the case above, we want to create a dummy variable for the age. The new variable, say ***dummy***, takes value of zero (0) if the respondent's age is below 35 and value of one (1), if the age is more than 35. The statement below will do it.

```
g dummy=0
replace dummy=1 if age>35
```

alternatively,

```
g dummy = .
replace dummy=0 if 0<age & age<=35
replace dummy=1 if age>35
```

NOTE: *Both statements will produce the same desired results.*

v. **String variables**. In STATA values for string variables are denoted by inverted commas – " ". For example, if we want to create a new string variable that will take the string-values of "KID" and "ADULT" for the ages of 0-18 and 19 and above, respectively. The statements below will generate the new string variable.

```
g string = “ ”        [will create string variable w/ empty values]
replace string= “kid”   if age<=18
replace string= “adult” if age>=19
```

alternatively,

```
g string = “kid”   [creates string with values kid]
replace string= “adult”   if age>=19
```

vi. **Lags and leads**

- To generate lagged variable, use the following command:

```
gen varNEW = var1[_n-1]
```

- To generate a lead variable, use the statement below:

```
gen varNEW = var1[_n+1]
```

- One can always do additional manipulation with data when creating lagged or lead variables. For example, the command below will create lagged variable for var1 and will also raise the new variable to the 2nd power.

```
gen varNEW = var1[_n-1]^2
```

NOTE: *all examples above would create lags of one period. To create other lags, simply replace 1 in [_n-1] with the desired number of periods. For example, [_n-5] will create lags of 5 periods.*

k. Keep and Drop statements

The original dataset may contain many variables that may not be of your interest. You may alter the dataset before reading into STATA or do it in STATA. You can tell STATA to either `keep` what you want or `drop` what you do not want. The results are the same. Assume dataset has 10 (ten) variables, named var1 to var10. variables of interest are only var1, var3 and var6.

```
keep var1 var3 var6
```

alternatively,

```
drop var2 var4 var5 var7 var8 var9 var10
```

alternatively,

```
drop var2 var4 var5 var7-var10
```

After execution of any of the above statements, you will be left with the variables of interest.

NOTE: *the hyphen sign (-) in the third statement indicates that all variables between var7 and var10 should be dropped. It is very useful when you deal with many variables and do not want to type all names individually.*

You can also apply those two statements to observations. For example, we have annual observations for 1900-2006, but the observations of interest are only 1945-2006. Using the statements below, we will eliminate all undesired observations. Assume we have a time variable called `year` which takes four-digit values for years.

```
keep if year >= 1945
```

alternatively,

```
drop if year < 1945
```

NOTE: *refer to the list of <u>relational operators</u> used in STATA discussed in the previous sections.*

To keep the observations between years 1915 and 1923, use the statement below.

```
keep if year>=1915 & year<=1923
```

alternatively,

```
drop if year<1915 | year>1923
```

To keep the observations between 1915-1923 and 1988-1994, use the following statement.

```
keep if (year>=1915 & year<=1923) | (year>=1988 & year<=1994)
```

NOTE: *<u>logical</u> operators in STATA are:*

&	*and*
|	*or*
~	*not*

You may `keep` or `drop` observations with specific values (numeric and/or string).

`drop if var3 == .`	[drops observations with missing values]
`drop if var3 == “young”`	[drops observations for which var3 takes string-value *young*]
`drop if _n <= 10`	[drops observations 1 to 10]
`drop if _n == _N`	[drops the last observation in the dataset]

There are cases when working with databases where the same observation type occurs more than once you are interested in the first one only. For example, suppose that the data has each store manager stacked one on top of the other and you only want to keep the first observation of each store manager. The statement below will eliminate undesired observations.

```
drop if manager[_n]==manager[_n-1]
```

alternatively,

```
drop if manager==manager[_n-1]
```

3. **Creating graphs, plots & histograms**
The statements in this section will help to successfully illustrate your data using graphs, plots and histograms.

NOTE: *you can always learn more by exploring STATA help, which contains additional info on the statements in this section as well as options available for them. For example, exploring STATA help for graphs (simply type* `help graph` *in the command window) will help you to customize graphs, plots and histograms, such as adding/changing legends, labels, titles, color schemes, etc.*

a. `twoway` is a family of plots, all of which fit on numeric y and x scales. The basic command statement for it is:

```
twoway plottype varlist
```

where, *`plottype`* indicates the type of plot requested and `varlist` lists variables of interest. Table below lists plot-types available in STATA.

Plot type	**Description**
`scatter`	scatter plot
`line`	line plot
`connected`	connected-line plot
`scatteri`	scatter with immediate arguments
`area`	line plot with shading
`bar`	bar plot
`spike`	spike plot
`dropline`	dropline plot
`dot`	dot plot
`rarea`	range plot with area shading
`rbar`	range plot with bars
`rspike`	range plot with spikes
`rcap`	range plot with capped spikes
`rcapsym`	range plot with spikes capped with symbols
`rscatter`	range plot with markers
`rline`	range plot with lines
`rconnected`	range plot with lines and markers
`pcspike`	paired-coordinate plot with spikes
`pccapsym`	paired-coordinate plot with spikes capped with symbols
`pcarrow`	paired-coordinate plot with arrows
`pcbarrow`	paired-coordinate plot with arrows having two heads
`pcscatter`	paired-coordinate plot with markers
`pci`	pcspike with immediate arguments
`pcarrowi`	pcarrow with immediate arguments
`tsline`	time-series plot
`tsrline`	time-series range plot
`mband`	median-band line plot
`mspline`	spline line plot
`lowess`	LOWESS line plot
`lfit`	linear prediction plot
`qfit`	quadratic prediction plot
`fpfit`	fractional polynomial plot
`lfitci`	linear prediction plot with CIs
`qfitci`	quadratic prediction plot with CIs
`fpfitci`	fractional polynomial plot with CIs
`function`	line plot of function
`histogram`	histogram plot
`kdensity`	kernel density plot

b. `graph matrix` draws scatter plot matrices. The basic command statement for it is:

```
graph matrix varlist
```

c. `graph bar` draws vertical bar charts. In a vertical bar chart, the y axis is numerical, and the x axis is categorical. The basic statement for it is:

```
graph bar numeric_var, over(cat_var)
```

where, `numeric_var` must be numeric; statistics of it are shown on the y axis. `cat_var` may be numeric (such as time horizon) or string (such as group names); it is shown on the categorical x axis.

NOTE: *to draw horizontal bar charts, change* `bar` *syntax to* `hbar`*. Such as* *`graph hbar numeric_var, over(cat_var)`*

d. `graph dot` draws horizontal dot charts. In a dot chart, the categorical axis is presented vertically, and the numerical axis is presented horizontally. Even so, the numerical axis is called the y axis, and the categorical axis is still called the x axis:

```
graph dot numeric_var, over(cat_var)
```

e. `graph box` draws vertical box plots. In a vertical box plot, the y axis is numerical, and the x axis is categorical.

```
graph box varlist, over(cat_var)
```

f. `graph pie` draws pie charts. `graph pie` has three modes of operation. The first corresponds to the specification of two or more variables. The statement below will draw four pie slices using listed variables.

```
graph pie var1 var2 var3 var4
```

The second mode of operation corresponds to the specification of a single variable and the `over( )` option. The statement below will draw pie slices for each value of `cat-var`; i.e. the first slice corresponds to the sum of `var1` for the first `cat-var`, the second to the sum of `var1` for the second `cat-var`, and so on.

```
graph pie var1, over(cat_var)
```

The third mode of operation corresponds to the specification of `over( )` with no variables. Pie slices will be drawn for each value of variable `var2`. The number of slices corresponds to the number of observations in each group.

```
graph pie, over(var2)
```

g. `histogram` draws histograms of `varname`. The statement takes only one variable at a time.

```
histogram var1
```
[draws a histogram for var1]

4. Estimation

a. Linear regression

STATA uses `regress` or `reg` statements to fit a model of the dependent variable on the independent variables using linear regression.

The basic statement has the following form:

```
regress depvar indepvars, options
```

where,

`depvar`	- dependent variable
`indepvars`	- independent variables
`options`	- various options available for `regress` command (see table on the next page)

For example, to fit the model $y_t = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \varepsilon$, the following statement will be used.

```
regress y x1 x2 x3
```

With this statement STATA automatically adds a constant term or intercept to the regression.

To estimate standard errors using the Huber-White sandwich estimator, use the `robust` option.

```
regress y x1 x2 x3, robust
```

Weighted least squares

Most STATA statement can deal with weighted data. To weight your data simply add the weight statement after the independent variables as shown in the statement below.

```
regress y x1 x2 x3 [weighttype=varname]
```

Four kinds of weights are permitted in STATA:

1. **frequency weights** (`fweights`) indicate the number of duplicated observations.

2. **sampling weights** (`pweights`) denote the inverse of the probability that the observation is included due to the sampling design.
3. **analytic weights** (`aweights`) 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.
4. **importance weights** (`iweights`) 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 want to produce a certain computation.

Assuming var3 is the variable in our data that contains the weight and we want to use frequency weights, the statement will be:

```
regress y x1 x2 x3 [fweights = var3]
```

Table below lists commonly used options available in STATA.

Options	Description
`noconstant`	suppress constant term
`hascons`	Has user-supplied constant
`tsscons`	compute total sum of squares with constant; seldom used
`vce(vcetype)`	`vcetype` may be `robust`, `bootstrap`, or `jackknife`
`robust`	synonym for `vce (robust)`
`cluster(varname)`	adjust standard errors for intragroup correlation
`mse1`	force mean squared error to 1
`hc2`	use u^2_j/(1-h_jj) as observation's variance
`hc3`	use u^2_j/(1-h_jj)^2 as observation's variance
`level(#)`	set confidence level; default is `level(95)`
`beta`	report standardized beta coefficients
`eform(string)`	report exponentiated coefficients and label as *string*
`noheader`	suppress the table header
`Plus`	make table extendable
`depname(varname)`	substitute dependent variable name; programmer's option

The `correlate` statement will show the correlations among specified variables.

```
correlate varlist
```

b. Testing for Collinearity

We can use **vif** statement command available in STATA to check for multicollinearity. **vif** stands for variance inflation factors for the independent variables. In general, a variable that has VIF value of greater than 10 may require further investigation. Many use the level of tolerance, defined as 1/VIF, to check on the degree of collinearity. If a variable has a tolerance value lower than 0.1 (i.e. **vif** greater than 10) it means that the variable could be considered as a linear combination of other independent variables. Code below will perform linear regression and check for the presence of multicollinearity.

```
reg y x1 x2 x3
vif
```

Alternatively, we can use `collin` command to perform collinearity diagnostics. Unlike the `vif` command that follows a `regress` command, `collin` command does not need to be run with a `regress` command but requires variable indication.

To use the White test, first you will need to download and install **whitetst** within STATA. To find the source for downloading follow the steps below:

1) type **findit collin** in the STATA command window
2) click on:
   collin from http://www.ats.ucla.edu/stat/stata/ado/analysis
3) click on **(click here to install)**
4) wait till it reports that installation is complete
5) now type **help collin** in the STATA command window to see more info on collinearity diagnostics.

The following statement will perform liniar regression and collinearity diagnostics.

```
reg y x1 x2 x3
collin varlist
```

c. Testing for Heteroskedasticity

There are two statements in STATA that test for heteroscedasticity - `hettest` and `whitetst`.

`hettest` is available in STATA as a default, whereas the `whitetst` command needs to be downloaded within STATA from the internet. Both test the null hypothesis that the variance of the residuals is not heteroscedastic. Therefore, if the p-value is very small, we would have to reject the hypothesis and accept the alternative hypothesis that the variance is heteroscedastic.

The statement below will perform liniar regression and test for heteroscedasticity using Breusch-Pagan / Cook-Weisberg test.

```
reg y x1 x2 x3
hettest
```

To use the White test, first you will need to download and install **whitetst** within STATA. To find the source for downloading follow the steps below:

1) type **findit whitetst** in the STATA command window
2) click on **sg137** next to STB-55
3) click on **(click here to install)**
4) wait till it reports that installation is complete
5) now type **help whitetst** in the STATA command window to see more info on White's test.

The following statement will perform liniar regression and test for heteroscedasticity using White's test.

```
reg y x1 x2 x3
whitetst
```

d. Testing for Autocorrelation

In STATA serial autocorrelation can be tested using Durbin-Watson d-statistics. There are two steps for performing DW test. First, using `tsset` you will need to declare data to be time series and specify the time variable, and second, using `dwstat` statement test for the evidence of serial autocorrelation. The general statement for performing DW test is provided below.

```
tsset timevar, options
```

where, `timevar` specifies the time variable and `options` indicate how `timevar` will be displayed, i.e. daily, monthly, quarterly, or else.

Table below lists all `options` available in STATA for `timevar`. If no option is listed, it is identical to the `generic`.

Option	Description
daily	display time scales as daily (%td, 0 = 1jan1960)
weekly	display time scales as weekly (%tw, 0 = 1960w1)
monthly	display time scales as monthly (%tm, 0 = 1960m1)
quarterly	display time scales as quarterly (%tq, 0 = 1960q1)
halfyearly	display time scales as half-yearly (%th, 0 = 1960h1)
yearly	display time scales as yearly (%ty, 0 = 1960 = 1960)
generic	display time scales as generic (%tg, 0 = ?)
format(%fmt)	indicate how timevar will be displayed

The statement below declares that data are time-series, sets variable `year` as the time variable (option=`yearly`) and generates DW d-statistics.

```
tsset year, yearly
dwstat
```

e. Hypothesis testing

STATA statement `test` performs Wald tests for simple and composite linear hypothesis about the parameters of the most recently fitted model.

The full syntax for test statement is:

```
test spec, options
```

where, `spec` specifies the testable hypothesis and `options` indicate the nature of the test.

Other tests:

- likelihood ratio test:	see `lrtest`
- Wald-type test of nonlinear hypothesis:	see `testnl`

Below is the list of options:

Options	Description
`mtest[(opt)]`	test each condition separately
`coef`	report estimated constrained coefficients
`accumulate`	test hypothesis jointly with previously tested hypotheses
`notest`	suppress the output
`common`	test only variables common to all the equations
`constant`	include the constant in coefficients to be tested
`nosvyadjust`	carry out the Wald test as W/k ~ F(k,d); for use with svy estimation commands
`minimum`	perform test with the constant, drop terms until the test becomes nonsingular, and test without the constant on the remaining terms; highly technical
`matvlc(matname)`	save the variance-covariance matrix; programmer's option

Below are some examples of hypothesis testing. All tests should be performed after regression. For example,

```
regress y x1 x2 x3 x4
```

1. Ho: $\beta_{x1}= \beta_{x2}= \beta_{x3}$

   ```
   test x1 = x2 = x3
   ```

2. Ho: $(\beta_{x1}+ \beta_{x3})/2= \beta_{x2}$

   ```
   test (x1 + x2)/2 = x3
   ```

3. Ho: $\beta_{x1}= \beta_{x3}$ and $\beta_{x2}= \beta_{x4}$

   ```
   test (x1 = x3) (x2=x4)
   ```

4. Ho: $\beta_{x1}= 0.6$ and $\beta_{x4}= -1.1$

   ```
   test (x1 = 0.6) (x4=-1.1)
   ```

5. Ho: $\beta_{x1}=0$, $\beta_{x2}=0$, $\beta_{x3}=0$, $\beta_{x1}=0$ [test jointly]

   ```
   test x1 x2 x3 x4, mtest
   ```

6. Ho: β_{x1}=0, β_{x2}=0, β_{x3}=0, β_{x1}=0 [test each separately]

```
test x1 x2 x3 x4, mtest
```

7. Ho: β_{x1}=0, β_{x2}=0.7, and β_{x3}=(-1.2) [test each separately]

```
test (x1=0) (x2=0.7) (x3=-1.2), mtest
```

8. Ho: $2\times\beta_{x3} + 0.5 = \beta_{x1}$

```
test 2*x3 + 0.5 = x1
```

9. Same as no. 8 but we also want to see what the constrained parameter estimates look like.

```
test 2*x3 + 0.5 = x1, coef
```

f. Confidence Intervals

STATA statement `ci` computes standard errors and confidence intervals for each of the variables in `varlist`. The full syntax for `ci` statement is:

```
ci varlist , options
```

List of options is provided below.

Options	Descriptions
`binomial`	binomial 0/1 variables; compute exact confidence intervals
`poisson`	Poisson variables; compute exact confidence intervals
`exposure(varname)`	exposure variable; implies poisson
`exact`	calculate exact confidence intervals; the default
`wald`	calculate Wald confidence intervals
`wilson`	calculate Wilson confidence intervals
`agresti`	calculate Agresti-Coull confidence intervals
`jeffreys`	calculate Jeffreys confidence intervals
`total`	output all groups combined (for use with by only)
`separator(#)`	draw separator line after every # variables; default is separator(5)
`level(#)`	set confidence level; default is level(95)

NOTE: *`exact`, `wald`, `agresti`, `wilson` and `jeffreys` options require the `binomial` option to be specified first*

Examples:

1. `ci x1 x2`
2. `ci x1 x2 x5 , level(99)`
3. `ci x4, binomial Wilson level(90)`

STATA statement `lincom` computes point estimates, standard errors, t or z statistics, p-values, and confidence intervals for linear combinations of coefficients **after any estimation command**. Results can optionally be displayed as odds ratios, hazard ratios, incidence-rate ratios, or relative risk ratios.

Full syntax for this statement is:

```
lincom exp, options
```

where, `exp` is a combination of algebraic and/or string expressions that are specified in a natural way using the standard rules of hierarchy. Parentheses are used to force a different order of evaluation.

Options	**Descriptions**
`eform`	generic label; exp(b); the default
`or`	odds ratio
`hr`	hazard ratio
`irr`	incidence-rate ratio
`rrr`	relative-risk ratio

Examples:

```
regress y x1 x2 x3 x4
```

1. `lincom x2 - x1`
2. `lincom 3*x3 + 1.25*x1 – 1.36`

g. Prediction

STATA statement `predict` calculates predictions, residuals, influence statistics, and the like after estimation. Exactly what predict can do is determined by the previous estimation command; command-specific options are documented with each estimation command. Regardless of command-specific options, the actions of predict share certain similarities across estimation commands. The detailed use of `predict` statement outlined in STATA manual is provided below.

`predict newvar` creates `newvar` containing "predicted values"--numbers related to the E(y|x). For instance, after linear regression, predict newvar creates xb and, after probit, creates the probability F(xb).

1) `predict newvar`, xb creates newvar containing xb. This may be the same result as (1) (e.g., linear regression) or different (e.g., probit), but regardless, option xb is allowed.
2) `predict newvar, stdp` creates `newvar` containing the standard error of the linear prediction xb.
3) `predict newvar, other_options` may create `newvar` containing other useful quantities; see help for the particular estimation command to find out about other available options.
4) `nooffset` added to any of the above commands requests that the calculation ignore any offset or exposure variable specified by including the `offset(varname)` or `exposure(varname)` options when you fitted the model.

`predict` can be used to make in-sample or out-of-sample predictions:

5) `predict` calculates the requested statistic for all possible observations, whether they were used in fitting the model or not. predict does this for the standard options (1) through (3) and generally does this for estimator-specific options (4).
6) `predict newvar if e(sample), ...` restricts the prediction to the estimation subsample.
7) Some statistics make sense only with respect to the estimation subsample. In such cases, the calculation is automatically restricted to the estimation subsample, and the documentation for the specific option states this. Even so, you can still specify if `e(sample)` if you are uncertain.

The full syntax for `predict` statement is:

```
predict varNEW , options
```

where, varNEW is the name of the variable that does not exist in the dataset yet. For the more complete list of options and their detailed descriptions simply type `help predict` in the STATA command window.

NOTE: *this syntax is used for single-equation models. See help predict for multiple-equation syntax and related options.*

Example:

```
regress y x1 x2 x3 x4
predict yhat
```

h. Extracting estimated parameters and standard errors
At any time to extract estimated parameters, standard errors and other so-called built-in system variables use the following specifications.

`_b[varname]`	for parameter estimates
`_se[varname]`	for standard errors
`_cons`	is equal to the number 1 when used directly and refers to the intercept term when used indirectly, as in `_b[_cons]`.
`_n`	contains the number of the current observation.
`_N`	contains the total number of observations in the dataset.
`_pi`	contains the value of pi to machine precision.

See `help system variables` for more and detailed info on this section.

5. IV regression (2SLS)

`ivreg` statement fits a linear regression model using instrumental variables (or two-stage least squares) of `depvar` on `varlist1` and `varlist2`, using `varlist_iv` (along with `varlist1`) as instruments for `varlist2`.

In the language of two-stage least squares, `varlist1` and `varlist_iv` are the exogenous variables, and `varlist2` are the endogenous variables.

Full syntax for `ivreg` statement is:

```
ivreg depvar varlist1 (varlist2 = varlist_iv) [weight] , options
```

where,

`weight` is used to weight the data (see Weighted least squares in Section 4), `options` are listed in the table on the next page, and the rest are explained above.

Examples:
```
1) ivreg y1 (y2 = z1 z2 z3) x1 x2 x3
2) ivreg y1 x1 x2 x3 (y2 = z1 z2 z3)
3) ivreg y1 x1 x2 (y2 y3 = z1 z2 z3)
4) ivreg y1 x1 x2 (y2 = z1 z2 z3) x3
5) ivreg y1 (y2 = z1 z2 z3) x1 x2 x3 [fw=pop]
6) ivreg y1 (y2 = z1 z2 z3) x1 x2 x3, robust
7) ivreg y1 (y2 = z1 z2 z3) x1 x2, robust cluster(var2)
```

Options	Descriptions
`noconstant`	suppress constant term
`hascons`	has user-supplied constant
`vce(vcetype)`	`vcetype` may be `robust`, `bootstrap`, or `jackknife`
`robust`	synonym for `vce(robust)`
`cluster(varname)`	adjust standard errors for intra-group correlation
`level(#)`	set confidence level; default is level(95)
`first`	report first-stage estimates
`beta`	report normalized beta coefficients
`noheader`	display only the coefficient table
`depname(varname)`	substitute dependent variable name
`eform(string)`	report exponentiated coefficients and use string to label them
`+mse1`	force MSE to be 1

6. Seemingly Unrelated Regression

To read details of SUR and find useful example, please type `help suest` in the STATA command window.

7. Non-Linear Estimation

a. LOGIT

`logit` statement in STATA fits a maximum-likelihood logit model. `depvar=0` indicates a negative outcome; `depvar!=0 & depvar!=.` (`typically depvar=1`) indicates a positive outcome.

STATA `logistic` statement displays estimates as odds ratios. Many users prefer this to `logit`. Results are the same regardless of which you use; both are the maximum-likelihood estimator.

Full syntax for logit command is:

```
logit depvar indepvars [weighttype=varname] , options
```

where, `depvar` is the dependent variable, `indepvars` are the independent variables, `weighttype` specifies the type of weight to be used (see Section 4 for details), `varname` specifies the weight variable, and `options` are listed below.

Options	Descriptions
`noconstant`	suppress constant term
`offset(varname)`	include `varname` in model with coefficient constrained to 1
`asis`	retain perfect predictor variables
`vce(vcetype)`	`vcetype` may be `robust`, `bootstrap`, or `jackknife`
`robust`	synonym for `vce(robust)`
`cluster(varname)`	adjust standard errors for intragroup correlation
`level(#)`	set confidence level; default is level(95)
`or`	report odds ratios
`maximize_options`	control the maximization process
`+nocoef`	do not display coefficient table

b. <u>PROBIT</u>

`probit` statement in STATA fits a maximum-likelihood probit model.

Full probit syntax is:

```
probit depvar indepvars [weighttype=varname] , options
```

Syntax for probit regression with marginal effects reporting is:

```
dprobit depvar indepvars [weighttype=varname], options
```

probit options	description
`noconstant`	suppress constant term
`offset(varname)`	include varname in model with coefficient constrained to 1
`asis`	retain perfect predictor variables
`vce(vcetype)`	`vcetype` may be `robust`, `bootstrap`, or `jackknife`
`robust`	synonym for `vce(robust)`
`cluster(varname)`	adjust standard errors for intragroup correlation
`level(#)`	set confidence level; default is level(95)
`maximize_options`	control the maximization process
`+nocoef`	do not display the coefficient table

dprobit options	**description**
`offset(varname)`	include varname in model with coefficient constrained to 1
`at(matname)`	point at which marginal effects are evaluated
`asis`	retain perfect predictor variables
`classic`	calculate mean effects for dummies like those for continuous variables
`robust`	compute standard errors using the robust/sandwich estimator
`cluster(varname)`	adjust standard errors for intragroup correlation
`level(#)`	set confidence level; default is level(95)
`maximize_options`	control the maximization process
`+nocoef`	do not display the coefficient table

8. External resources

This manual contains the basic information that will be needed to start learning the STATA software. For more advanced use, I will encourage to use the resources available through the STATA software help or others that are available through other organizations. For your convenience, two sources containing one of the most comprehensive resources are listed below:

a. **SAS/STAT User Guide** (*PDF files*). Dipartimento di Scienze Statistiche "Paolo Fortunati", Bologna, Italia. Available at: http://www2.stat.unibo.it/ManualiSas/stat/pdfidx.htm

   Contains downloadable PDF files on all procedures available in SAS (Version 8). This is a very comprehensive source and I would personally encourage using it.

b. **STATA Learning Resources.** University of California at Los Angeles Academic Technology Services. Available at: http://www.ats.ucla.edu/stat/stata
   Contains learning resources that help to master STATA software including text and audio/video resources. This is especially useful for those who just started to learn STATA.

c. In STATA command window type `help keyword` or `search keyword` to explore STATA built in manuals. The keywords can be anything that may be directly related to the information you are looking for. For example, to find syntax for weighted least squares estimation, you may use `search weighted`.