A GENERAL OVERVIEW OF STATA AND COURSE MATERIAL

Maria Elena Bontempi e.bontempi@economia.unife.it

Roberto Golinelli roberto.golinelli@unibo.it

this version: 26/09/2007[§]

1. Introduction

The present file and the others listed below were prepared as a guide for a course of applied econometric; empirical exercises make use of the statistical-econometric software STATA.

1.1. A guide to the lectures

This note presents the main STATA features, the different STATA files, and the supporting tools which is important to know in order to run empirical analyses. This document refers to a number of different files: data-files (wage1.dta, occ_1987_1.dta, occ_1987_2.dta, occ_1987.dta, occ_1988.dta, Usquarter.dta, NYTdailysales.dta, occupati_Italia.xls, auto.prn, auto.raw, auto_c.raw, auto_cc.raw); data-describing-files (auto_c1.dct, auto_c2.dct, auto_cc.dct); programming-files (dsett_label.do, covaun.ado, covamu.ado, covamu1.ado, reg_matrix.ado).

Along with this note, other lectures are the following.

- Techniques for analysing your data, e.g., data checking, getting familiar with your data file, and examining the distribution of your variables: lecture_exploratoty_data_analysis.
- Topics on simple and multiple regression, demonstrating the importance of inspecting, checking and verifying your data before accepting the results of your analysis: lecture_OLS_bivariate, lecture_OLS_multivariate, lecture_nonlinearity_Chow.
- Topics on heteroskedasticty and errors correlated with the explanatory variables: lecture_GLS, lecture_IV.
- Topics on time-series analysis, and model specification and estimation: lezione_seriestoriche_in_stata.

[§] Very preliminary. Comments welcome. Thanks to the work of C. F. Baum and of UCLA (see section 4 for references).

- Topics on panel data: lecture_panel_theory, lecture_panel_static_application, lecture_panel_dynamic_application.

These notes want to cover a variety of topics about using Stata for regression: they are about "data analysis" and demonstrate how Stata can be used for regression analysis, as opposed to a book that covers the statistical basis of multiple regression.
It is assumed that you have had at least one statistics course covering regression analysis and that you have a regression book that you can use as a reference. In fact, the notes are designed to apply your knowledge of regression, to combine it with instruction on Stata, to perform, understand and interpret regression analyses.

The commands reported in the notes work on updated versions 7 or 8 of Stata. However, some commands, such as `xtabond2`, are written to take advantage of the Mata programming language first included in Stata 9. Hence, the updated version 9.2 of Stata is highly recommended. In what follows some of the important new additions of Stata 9.

- New matrix programming language Mata, that can be used by those who want to think in matrix terms and perform matrix calculations interactively, and it can be used by programmers who want to add features to Stata.
- New features in the longitudinal/panel data: new command `xtmixed` that fits linear mixed models, also known as hierarchical models or multilevel models; new features added to the maximum-likelihood estimators that do not have closed-form solutions and require numeric evaluation of the likelihood (`xtlogit, xtprobit, xtpoisson, xtcloglog, xtintreg, xttobit`); existing command `xtreg` now allowing options `robust` and `cluster()` when estimating fixed-effects (FE) and random-effects (RE) models; most [XT] commands now supporting time-series operators (`xtgls, xtreg, xtsum, xtcloglog, xtintreg, xtlogit, xtpoisson, xtprobit, xttobit, xtgee`); many commands requiring time-series data now working on a single panel from a panel dataset when that panel is selected using an `if` expression or an `in` qualifier (`ac, corrgram, cumsp, dfgls, dfuller, pac, pergram, pperron, wntestb, wntestq, xcorr` and new commands `estat archlm, estat bgodfrey, estat dwatson, estat durbinalt` - replacing commands[1] `archlm, bgodfrey, dwstat, durbina`).
- New features in the time-series statistics: existing command `dfuller` having new option `drift` for testing the null hypothesis of a random walk with drift, and having more accurate algorithm for calculating MacKinnon's approximate p-values; existing commands `corrgram` and `pac` having new option `yw` that computes partial autocorrelations using the Yule-Walker equations instead of the default regression-based method; existing command `arima` now estimating multiplicative seasonal ARIMA (SARIMA) models; new command `rolling` performing rolling-window or recursive estimations, including regressions, and collecting statistics from the estimation on each window; the [TS] manual having a glossary that defines commonly used terms in time-series analysis; new command `vec` fitting cointegrated vector error-correction models (VECMs) using Johansen's method; new command `vecrank` producing statistics used to determine the number of cointegrating vectors in a VECM, including Johansen's trace and maximum-eigenvalue tests for cointegration; new command `fcast` - which replaces old command `varfcast` - producing and graphing dynamic forecasts of the dependent variables after fitting a VAR, SVAR, or VECM; new command `irf` - which replaces the old command `varirf` – estimating the impulse-response functions, cumulative impulse-response functions, orthogonalized impulse-

[1] But old commands still work.

response functions, structural impulse-response functions, and forecast error-variance decompositions after fitting a VAR, SVAR, or VECM.

The synthax of the graph command has been completely revised since version 8. The (esay) syntax of version 7 is no more compatible with the syntax of Stata 8 or 9 and if you use it, you get an error message. Many tricks to use the stata 7 graph synthax are available, illustrated in the specific lectures. For example, replace `graph` command with `graph7` command. Alternatively, use the command `version 7: graph.` Alternatively, type the sequence:

```
version 7
graph
version 9
```

When possible, some examples of the stata 8/9 graph synthax are reported; this because stata 8/9 graphs are superior, maybe the time-series case excluded. An online tutorial about creating basic graphs in STATA 8/9 is available at the website http://www.stata.com/support/faqs/graphics/gph/statagraphs.html.

Stata 10 is now available, visit `http://www.stata.com`

2. Features of Stata

You can install 3 different version of Stata.

Stata/SE, Intercooled Stata, and Small Stata differ in the size of the dataset that each one can analyze and the maximum length of string variables. Here is a general guideline:

- Stata/SE — Stata for large datasets

Stata/SE allows datasets with up to 32,766 variables. The limit of observations is based on the amount of RAM in your computer. Stata/SE allows datasets to contain longer string variables — variables up to 244 characters long. Stata/SE also allows matrices up to 11,000 x 11,000 on computers with sufficient memory. An implication of this is that Stata/SE can estimate models with more independent variables (up to 10,998) and estimate certain panel-data models with larger time-series within panel.

- Intercooled Stata — the "standard" version of Stata

Intercooled Stata allows datasets with as many as 2,047 variables. The limit of observations is based on the amount of RAM in your computer. Intercooled Stata allows string variables to contain a maximum of 80 characters. In addition, Intercooled Stata can have at most 798 right-hand-side variables in a model.

- Small Stata — a smaller, "student" version of Stata (academic only)

Small Stata is limited to analyzing datasets with a maximum of 99 variables on approximately 1,000 observations. Small Stata allows string variables to contain a maximum of 80 characters. Small Stata can have at most 38 right-hand-side variables in a model.

Requirements

	Stata/SE	Intercooled Stata	Small Stata
Memory	128 MB	128 MB	128 MB
Disk space	60 MB	60 MB	60 MB

3. Facilitating applied economic research with Stata

Applied empirical research in economics and finance has traditionally involved the use of two types of high-level software. Each type has its strengths and weaknesses. In this essay, we describe the middle ground.

- Statistical/econometric packages, such as TSP, SAS, RATS or eViews

  Advantages

- ✓ They provide a variety of econometric procedures, with well-formatted results, batteries of diagnostics, and pre-designed graphical output.
- Data series may be referred to by name, and missing values and transformations such as lags and differences may be readily handled.

  Disadvantages

- ✓ Extending the context of these packages (adding estimation procedures, customising output, or reusing the results of computed quantities) may be difficult, and almost without exception the user-provided extensions to statistical packages will require special handling.
- ✓ User-written procedures will not be "first–class citizens" in most statistical package environments: they require modifications to make them available within the package; they do not always include comment statements providing on-line help.
- ✓ Hence, it may be a hard work to utilise a very recently developed econometric procedure that has not yet been implemented.

- Matrix languages, such as MATLAB, GAUSS, S-Plus or Ox.

  Advantages

- ✓ They circumvent the restrictions posed by the defined feature set of statistical packages. They have extensibility: you may translate any complex sequence of matrix operations into the language, without the explicit loops or extensive libraries that would be required in C, C++, or Fortran 90. Hence, the matrix languages support more rapid development and allow for interactive use.

  Disadvantages

- ✓ The execution speed: most (with the notable exception of Ox) are quite slow, even on powerful hardware.
- ✓ They impose on you a greater housekeeping burden, since the convenience features of the statistical packages' languages are generally absent. You must explicitly keep track of the correspondence between named variables and columns of matrices, take account of observations lost through lags, differences, and the handling of unbalanced panel data, and provide much of the logic for the formatted output of results.
- ✓ Hence, the matrix languages conceptually support reusable code, but it appears to be relatively scarce in practice[2]. For example, many researchers will provide the GAUSS code used in their work on request, but those routines, almost without exception, contain explicit references to the particular problem they have solved in terms of the dimensions of the problem, the specific files accessed, and the specific form of the output.

[2] A notable exception is James P. LeSage's Econometrics Toolbox for MATLAB (http://www.spatialeconometrics.com/), which provides an exhaustive set of econometric routines for MATLAB, with full documentation and an extensible development environment, at zero cost.

- ✓ They rely on numerous components, each sold separately (MATLAB "toolboxes" or GAUSS "applications"), implying that most users will have a different set of available functions in their copy of the language.[3] The matrix languages are generally more constrained in terms of cross-platform availability than are many statistical packages.

➢ The Stata software environment, a product of Stata Corporation (College Station, TX: `http://www.stata.com`)

- ✓ Stata provides a unique middle ground between "point and click" statistical packages and "open-ended" matrix languages. Stata has a defined feature set and it is easy to use. Stata has extensibility, and provides web-accessible features that enhance collaborative research and instruction.
- ✓ Like its major competitors (SAS, SPSS, RATS, TSP, Eviews), Stata offers a wide range of statistical and econometric capabilities for the researcher who merely wants to plug in the data and execute a routine statistical analysis.

  It allows the user to name variables, operate on variables with wildcard syntax, and rely on the program to keep track of missing values, lags, and the messy details of unbalanced panel data.

  Advanced features include the ability to transform "wide" data (such as those used in seemingly unrelated regression estimation) into "long" data (that which has been stacked by column with the "vec" operator), suitable for panel data estimators such as fixed effects and random effects.

  Stata provides a broad set of preformatted output routines, so that the user need not specify the format and perform housekeeping troubles in order to generate all of the items needed for presentation of the results at the desired precision.
- ✓ Unlike some statistical packages, Stata operates in a vector context, so that transformations (generating new variables and revising existing variables) are specified without explicit loops that carry a speed penalty. The entire dataset is held in memory, so that transformations and most estimation procedures are very fast, even when hundreds of thousands of observations are defined. This implies the need for sizeable memory resources, but the cost of relaxing that constraint today is quite minimal for most systems.
- ✓ Stata diverges from statistical packages and from matrix languages (MATLAB, GAUSS, Ox) by promoting its extensibility, and providing explicit support for the development and dissemination of procedures crafted by its user community.
- ✓ The core functionality of the package itself may be readily upgraded by its vendor between major releases: as we will see, the upgrades may be readily acquired and installed by the user base. Stata is a rapidly evolving software environment, in which researchers collaborate from diverse locations, rely on other Stata users for assistance, implement new capabilities, and interchanged them via the package's Internet-access features.

4. Updating and software development in Stata

The preponderance of Stata commands are provided by several hundred separate "`ado-files`": automatic `do-files`, or procedures, which are plain text files and whose names correspond to user commands. These files are in a "adopath": a Unix-like path containing a number of directories, each of them containing specific ado-file for specific command.

[3] Jurgen Doornik's language (http://www.nuff.ox.ac.uk/Users/Doornik/doornik.html#ox) is better placed in this regard, in that additional packages in that environment are generally freely downloadable.

This design promotes timely updates to the software. This can be done by copying the corrected files to the appropriate locations, or automatically. Suppose you have installed version 9 and you have an Internet access. It is very easy, at any time, to evaluate the status of your copy versus the most recent available, and to instruct Stata to query the vendor's site for updates.

```
. update query
(contacting http://www.stata.com)

Stata executable
    folder:               C:\Programmi\Stata9\
    name of file:         wsestata.exe
    currently installed:  17 May 2006
    latest available:     6 Jul 2006

Ado-file updates
    folder:               C:\Programmi\Stata9\ado\updates\
    names of files:       (various)
    currently installed:  17 May 2006
    latest available:     11 Jul 2006

Recommendation
    Type -update all-

Click to edit automatic update checking preferences
------------------------------------------------------------------------------------------------
```

You can also make extensively use of utility routines not provided in "Official Stata". An extensive library of user-written commands, documented in the over the last ten years, is freely accessible from Stata's web site. A continuously expanding archive of user-written additions to Stata is available, for example, at:

- the Boston College Statistical Software Components (SSC) archive (http://fmwww.bc.edu/EC/), which provides free access to several hundred Stata modules; this site is maintained by Christofer F. Baum;
- University of California Los Angeles (UCLA) Academic Technology Services (ATS) and UCLA Stat Computing Portal (http://www.ats.ucla.edu/stat/).

It is very easy incorporating user-authored components into your copy of Stata. The ado–files and associated help files for these new commands may be downloaded, installed in the appropriate directory ("`ado/stbplus`"). Hence, the commands defined therein will become first–class citizens in the Stata command language.
Finally, you can create by yourself your personal ado files in directories such as "`ado/personal`". The only warning is that a user-written command may not take the name of an "official" Stata command, in order to prevent confusion.

There is an innovative concept underlying this structure: a user-written command, whether downloaded from the Stata web site, acquired from a colleague, or written oneself is indistinguishable from any "official" Stata command once it is placed on the "adopath".
Stata's developers have promoted the development of this quality work in the user community by providing high–level tools for ado–file development: the same tools that they themselves employ.

4.1. Stata web site

At web site www.stata.com you may find.

- Commercial information
- FAQs section
- Net courses

- Statalist, a free email listserver where over 2,000 Stata users from experts to neophytes maintain a lively dialogue about all things statistical and Stata. You can subscribe by sending an email to *majordomo@hsphsun2.harvard.edu* and ask particular questions. (unsubscribe if you we be away of your email for few days). Since 1997, all items posted to Statalist (over 600) have been placed in the Boston College SSC Archive in RePEc (http://ideas.repec.org) and EconPapers (http://econpapers.repec.org).
- The *Stata Technical Bulletin* (STB) that, between 1991 and 2001, served as a means of distributing new commands and Stata updates, both user-written and "official". Since 2001, the STB evolved into The Stata Journal, a quarterly publication containing articles about statistics, data analysis, teaching methods, and effective use of Stata's language. The Journal publishes reviewed papers together with shorter notes and comments, regular columns, book reviews, and other material of interest to researchers applying statistics in a variety of disciplines.

5. How to use Stata: introduction

Stata has many strengths:

- cross-section, time series and panel data manipulation
- statistical analysis
- graphical analysis
- econometric analysis

Stata files are:
binary data-set (.dta)
-optional:
file in which you can save your output (text file .log, or for use with Viewer .smcl)
graphics output file (.gph)
user program file (.do)
automatic do-file, that defines a Stata command (.ado) with an accompanying help file (.hlp) with instructions
ASCII data file (.raw, .cvs, .txt) and data dictionary file for describing data-set to be read by Stata with infile (.dct) (see Section 6).

Stata has 4 windows:
Variables: you see the list of the variables in the data-set you are using.
Command: you type the commands.
Review: you have the list of the executed commands; you may re-use them by double click; you may modify them by single click.
Results: you see the output, i.e. the results obtained by running the commands.

Before starting your working session, It is preferable that you select a particular directory. Let's say you are using Windows and you want to store the data file in a folder called **c:\myanalysis**. First, you can make this folder within Stata by the command:

```
mkdir c:\myanalysis
```

You can then change to that directory:

```
cd c:\myanalysis
```

IMPORTANT: You have reproducibility and extendibility of what you have done (i.e. what you gradually see in the *review* and *results* windows) by opening a file.log before starting your working session:

```
log using mylog.log
```

This file permanently saves on your computer a report with all the commands you have run[4] and the corresponding output (only note that graph files have to saved separately). This file may be read and manipulated by every word process, becoming the relevant part of your applied research report. More useful, you may open and manipulate your log file in the "do-file editor": leaving only the commands you have entered, you create, save and execute a mydo.do file, i.e. a sequence of commands that you can apply to every data-set you want.

If you type:

```
log using mylog
```

by default Stata uses the .smcl extension, i.e. the stata markup and control language; this is nice to be read and printed, but not useful for word processor or the "do-file editor".

Notice: commands and variables' names are key sensitive: for example, *OCC* is a variable different from *occ* or *Occ*. Usually do not use capital letters for commands.

Stata works in memory. Thus, may be important to select how much memory is needed in order to use the data-set and to perform the analyses you are interested in. For example:

```
set mem 10m
```

The command for opening a Stata data-set is

```
use mydata, clear
```

where the option `clear` (see Section 5 on Stata syntax) will empty the current content of memory.

Once you have read the data file, you probably will modify it and want to store the changes:

```
save mydata, replace
```

where the option `replace` (see Section 5 on Stata syntax) will overwrite the current data-set.

You may also use the Stata menu. The menu is useful for interacting with the computer's file system, for managing multiple windows, for changing screen defaults, for printing results and graph, and the like. Innovative in Stata 8 or 9 is that you may also run commands by using menus; in this case you have a direct access to the help. You maintain reproducibility and extendibility of what you have done through both the *review* window and the file.log. However, we think it is preferable typing commands as a more effective and efficient research strategy, and as a more rigorous way of learning estimating techniques. In this case, if you have doubt about a command syntax, you may type:

```
help log
```

that accesses help on all installed commands.

When you are connected to the web, the command:

```
search log
```

[4] The commands appear with a leading dot, a convention to indicate that the command is executed. You will find many times this leading dot in the lectures on the web. Do not type the leading dot in the Stata *command* window.

will locate new commands that have been documented in the Stata Journals or in other web sites. With one click you may install them in your version of Stata. With `search` you can find the command you need by entering one or more keywords, even if you do not know the command's name.

There are on-line tutorial also available:

`tutorial contents`	Current list of available tutorials
`tutorial intro`	Introduction to Stata
`tutorial graphics`	How to make graphs
`tutorial tables`	How to make tables
`tutorial regress`	Estimating regression models, including 2SLS
`tutorial anova`	Estimating one-, two- and N-way ANOVA and ANCOVA models
`tutorial logit`	Estimating maximum-likelihood logit and probit models
`tutorial survival`	Estimating maximum-likelihood survival models
`tutorial factor`	Estimating factor and principal component models
`tutorial ourdata`	Description of the data provided with Stata
`tutorial yourdata`	How to input your own data into Stata

The Stata Base Documentation Set contains over 2,900 pages, including formulas and detailed examples for the most-commonly used Stata commands. The Base Documentation Set includes a four-volume Base Reference Manual, the Stata Graphics Reference Manual, the Stata User's Guide, the Getting Started with Stata manual for Windows, Macintosh, or Unix (the Stata Data Management Reference Manual, and a Quick Reference and Index in Stata 9).

- Base Reference Manual (4 volumes, about 2000 pages):

Volume 1, A – F

Volume 2, G – M

Volume 3, N –R

Volume 4, S-Z

- Graphics Reference Manual (580 pages)
- User's Guide (370 pages) a first quick guide to the basic syntax on preliminary data analysis, regressions, graphs, and an help in the use of the 4 Stata manuals
- One of the following:

Getting Started with Stata for Windows (183 pages)

Getting Started with Stata for Macintosh (177 pages)

Getting Started with Stata for Unix (190 pages)

for an introduction to Stata installation, lecture of data and print of output.

Additional subject-specific volumes may be purchased separately. These include:

- Programming Reference Manual (450 pages)
- Longitudinal/Panel Data Reference Manual (249 pages)
- Time-Series Reference Manual (440 pages)
- Survival Analysis and Epidemiological Tables Reference Manual (397 pages)
- Survey Data Reference Manual (98 pages)

- Cluster analysis (118 pages)

Stata 9 provides extensions of previous manuals, as well as

- Data Management Reference Manual (489 pages) brings together in one volume Stata's Data Management commands. There you will find such advanced features as reshaping data and using ODBC, as well as such fundamental features as merging and appending.
- Mata Reference Manual (620 pages) describes Mata, Stata's new matrix-programming language. Mata can be used to perform matrix calculations interactively and to add new features to Stata. In fact, many of Stata 9's new features were written in Mata.
- Multivariate Statistics Reference Manual (481 pages) documents all the multivariate analysis features added in Stata 9, including multidimensional scaling, correspondence analysis, biplots, and many others.
- Quick Reference and Index (97 pages)

The Full Documentation Set, containing over 6,000 pages, includes everything in the Base Documentation Set, plus all the Stata subject-specific volumes.

Cross-referencing

These manuals all cross-reference each other. All the manuals in the Stata documentation have a shorthand notation, such as [U] for the User's Guide and [R] for the Base Reference Manual.

5.1. How to use Stata: command syntax

Stata command syntax follows strict rules; thus, once you have learned the way a few commands work, you will be able to use many more without extensive study of the manual

The general syntax is:

```
[bysort varlist:] cmdname [varlist] [=exp] [if exp] [in range] [weight]
                                [using spec] [, options]
```

where elements in square brackets are optional for some commands. All the Stata commands may be abbreviated.

5.1.1. The output of a data-descriptive command

Before illustrating in detail the syntax of a Stata command, we prefer to explain the output produced by a very easy command used to describe the data-set in use. This command is important given that it embodies the fundamentals of data management in Stata.

As an example, we use one of the files, wage1.dta, presented in Jeffrey M. Wooldridge (2003), Introductory Econometrics: A Modern Approach, 2e, Thomson at pages 7, 34-35, 38, 76-77, 93, 123-124, 180, 190-192, 214, 222-223, 226-228, 232, 235, 254, 260-261, 311, 648. These data are from the 1976 Current Population Survey, collected by Henry Farber.

```
use wage1, clear
describe
```

```
Contains data from D:\LAVORI\books\basic\WAGE1.DTA
  obs:           526
 vars:            24                          16 Sep 1996 15:52
 size:        18,936 (99.8% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
wage            float  %8.2g                 average hourly earnings
educ            byte   %8.0g                 years of education
exper           byte   %8.0g                 years potential experience
tenure          byte   %8.0g                 years with current employer
nonwhite        byte   %8.0g                 =1 if nonwhite
female          byte   %8.0g                 =1 if female
married         byte   %8.0g                 =1 if married
numdep          byte   %8.0g                 number of dependents
smsa            byte   %8.0g                 =1 if live in SMSA
northcen        byte   %8.0g                 =1 if live in north central U.S
south           byte   %8.0g                 =1 if live in southern region
west            byte   %8.0g                 =1 if live in western region
construc        byte   %8.0g                 =1 if work in construc. indus.
ndurman         byte   %8.0g                 =1 if in nondur. manuf. indus.
trcommpu        byte   %8.0g                 =1 if in trans, commun, pub ut
trade           byte   %8.0g                 =1 if in wholesale or retail
services        byte   %8.0g                 =1 if in services indus.
profserv        byte   %8.0g                 =1 if in prof. serv. indus.
profocc         byte   %8.0g                 =1 if in profess. occupation
clerocc         byte   %8.0g                 =1 if in clerical occupation
servocc         byte   %8.0g                 =1 if in service occupation
lwage           float  %9.0g                 log(wage)
expersq         int    %9.0g                 exper^2
tenursq         int    %9.0g                 tenure^2
-------------------------------------------------------------------------------
Sorted by:
```

The command `descr` without arguments offers a description of the current contents of memory.

In the first column you have the name of the variable that can be changed by the command `rename.`

In the second and third columns there are the type and the display format that can be changed by the command `format`. Stata has the following formats.

- For real numbers:

```
Type           Bytes
-------------------
float           4
double          8
```

Floats (the default) have about 7 digits of accuracy; the magnitude of the number does not matter. Thus, 1234567 can be stored perfectly as a float, as can 1234567e+20. The number 123456789, however, would be rounded to 123456792. In general, this rounding does not matter.

If you are storing identification numbers, however, the rounding could matter. In this case, use doubles which have 16 digits of accuracy.

Stata stores numbers in binary and this has a second effect on numbers less than 1. 1/10 has no perfect binary representation just as 1/11 has no perfect decimal representation. In float, .1 is stored as .10000000149011612. Note that there are 7 digits of accuracy, just as with numbers larger than 1.

Stata, however, performs all calculations in double precision. If you were to store 0.1 in a float called x and then ask `list if x==.1` (the command is explained below), there would be nothing in the list. The .1 that you just typed was converted to double, with 16 digits of accuracy (.10000000000000014...) and that number is never equal to 0.1 stored with float accuracy.

One solution is to type (explained below) `list if x==float(.1)`. The float() function rounds its argument to float accuracy.

The other alternative would be store your data as double, but this is probably a waste of memory. Few people have data that is accurate to 1 part in 10 to the 7th. Among the exceptions are banks, who keep records accurate to the penny on amounts of billions of dollars. If you are dealing with such financial data, store your dollar amounts as doubles. See [U] 16.10 Precision and problems therein.

- For integers:

```
Type           Bytes
-------------------
byte            1
int             2
long            4
```

Long is the most accurate, with 9 digits.

- For non-numbers:

```
Type           Bytes         length
---------------------------------
str1            1              1
str2            2              2
...
str244        244            244
---------------------------------
```

In order to economise on the disk space and memory required to store variables the command is `compress.`

In the forth colum there are value labels that associate numeric values with character strings. They exist separately from variables and will be displayed in printed output instead of the numeric values. Thus, they are useful for improving the comprehension of a variable that we desire to maintain in a numeric format.

```
label define femalelb 0 male 1 female
label values female femalelb
label list

femalelb:
           0 male
           1 female

descr

Contains data from D:\LAVORI\books\basic\WAGE1.DTA
  obs:           526
 vars:            24                          16 Sep 1996 15:52
 size:        18,936 (99.8% of memory free)
------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
wage            float  %8.2g                  average hourly earnings
educ            byte   %8.0g                  years of education
exper           byte   %8.0g                  years potential experience
tenure          byte   %8.0g                  years with current employer
nonwhite        byte   %8.0g                  =1 if nonwhite
female          byte   %8.0g       femalelb   =1 if female
married         byte   %8.0g                  =1 if married
numdep          byte   %8.0g                  number of dependents
smsa            byte   %8.0g                  =1 if live in SMSA
northcen        byte   %8.0g                  =1 if live in north central U.S
south           byte   %8.0g                  =1 if live in southern region
west            byte   %8.0g                  =1 if live in western region
construc        byte   %8.0g                  =1 if work in construc. indus.
ndurman         byte   %8.0g                  =1 if in nondur. manuf. indus.
trcommpu        byte   %8.0g                  =1 if in trans, commun, pub ut
```

```
trade           byte   %8.0g                 =1 if in wholesale or retail
services        byte   %8.0g                 =1 if in services indus.
profserv        byte   %8.0g                 =1 if in prof. serv. indus.
profocc         byte   %8.0g                 =1 if in profess. occupation
clerocc         byte   %8.0g                 =1 if in clerical occupation
servocc         byte   %8.0g                 =1 if in service occupation
lwage           float  %9.0g                 log(wage)
expersq         int    %9.0g                 exper^2
tenursq         int    %9.0g                 tenure^2
------------------------------------------------------------------------------
Sorted by:
```

Label values may be saved in the data-set:

```
save wage1, replace
```

In the fifth column we have variable labels, which are character strings (maximun 80 characters) describing the variable. For example, the label of female has been created by the command (the same command is to be used to change it):

```
label var female "=1 if female"
```

5.2. Command syntax: the varlist

In varlist you put a list of one or more variables on which the command is to operate. The order of the variables is important in the following cases.

- In the regression commands the first variable is the dependent one, and the following variables are the explanatory ones.
- You may use interval between first and last variables:

```
. descr  services- clerocc

              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
services        byte   %8.0g                 =1 if in services indus.
profserv        byte   %8.0g                 =1 if in prof. serv. indus.
profocc         byte   %8.0g                 =1 if in profess. occupation
clerocc         byte   %8.0g                 =1 if in clerical occupation
```

- You may use "wildcards" to refer to all variables with a certain prefix:

```
. descr prof*

              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
profserv        byte   %8.0g                 =1 if in prof. serv. indus.
profocc         byte   %8.0g                 =1 if in profess. occupation

. descr ????occ

              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
profocc         byte   %8.0g                 =1 if in profess. occupation
clerocc         byte   %8.0g                 =1 if in clerical occupation
servocc         byte   %8.0g                 =1 if in service occupation
```

5.3. Command syntax: exp clause

The exp clause is used in command such as `generate` and `replace.`

For example, the variable lwage has been created by the command:

g lwage=log(wage)

where the log(.) belongs to the mathematical funcion of Stata:

Type of function	See help
Mathematical functions	mathfun
Probability distributions & density functions	probfun
Random numbers	random
String functions	strfun
Programming functions	progfun
Date functions	datefun
Time-series functions	tsfun
Matrix functions	matrix functions

Arithmetic operators are:
+ addition
- subtraction
* multiplication
/ division
^ power

Additionally, Stata has more functions available if you use the command `egen` (extended generate).

Any variable could be eliminated by the command:
`drop lwage`

5.4. Command syntax: if and in clauses

Unless you do not impose conditions, Stata commands will automatically apply to all observations currently defined.

In order to apply a command only to observations respecting some conditions you may proced as follows.

```
. list female if wage>20

     +--------+
     | female |
     |--------|
 15. |   male |
 59. | female |
112. |   male |
186. |   male |
229. |   male |
     +--------+

. list female if wage>20, noobs

  +--------+
  | female |
  |--------|
  |   male |
  | female |
  |   male |
  |   male |
  |   male |
  +--------+
```

where, as an example, the noobs option suppresses the listing of the observation numbers.

Relational operators (numeric and string) are:
> greater than
< less than
>= > or equal
<= < or equal
== equal
~= not equal
!= not equal

Note that:

- for string variables you need to put the value between “ ”;
- missing values appear as the dot (.) for numeric variables and as the null string (“”) for string variables; missing values take on the largest possible positive value, so in the presence of missing data you do not want to say

```
. list female if wage>20, noobs
```

but rather

```
. list female if wage>20&wage!=., noobs
```

Logical operators are:
~ not
! not
| or
& and

in at the end of a command means the command is to use only the observations specified:

```
. list wage in 10

     +------+
     | wage |
     |------|
 10. |   18 |
     +------+

. list wage in 1/10

     +------+
     | wage |
     |------|
  1. |  3.1 |
  2. |  3.2 |
  3. |    3 |
  4. |    6 |
  5. |  5.3 |
     |------|
  6. |  8.8 |
  7. |   11 |
  8. |    5 |
  9. |  3.6 |
 10. |   18 |
     +------+
```

Negative numbers may be used to specify distance from the end of the data; lowercase el indicates the last observation, as –1 (compare output of `list wage in -10/l` with output of `list wage in -10/-1`).

```
. list wage in -10/l

     +------+
     | wage |
     |------|
517. |  3.1 |
518. |  9.3 |
519. |  7.5 |
520. |  4.8 |
521. |  5.7 |
     |------|
522. |   15 |
523. |  2.3 |
524. |  4.7 |
525. |   12 |
526. |  3.5 |
     +------+
```

5.5. Command syntax: bysort and sort

Compare previous list output with the following:

```
. sort wage
```

```
. list wage in 10

     +------+
     | wage |
     |------|
 10. |    2 |
     +------+

. list wage in 1/10

     +------+
     | wage |
     |------|
  1. |  .53 |
  2. |  1.4 |
  3. |  1.5 |
  4. |  1.5 |
  5. |  1.6 |
     |------|
  6. |  1.7 |
  7. |  1.8 |
  8. |    2 |
  9. |    2 |
 10. |    2 |
     +------+

. list wage in -10/l

     +------+
     | wage |
     |------|
517. |   18 |
518. |   18 |
519. |   19 |
520. |   20 |
521. |   20 |
     |------|
522. |   22 |
523. |   22 |
524. |   22 |
525. |   23 |
526. |   25 |
     +------+
```

The `sort` command arranges wage in ascending-order; thus, the `in` clause list the ten lowest wages or the ten highest wages. To obtain descending-or-ascending-order see the command `gsort`.

Many Stata commands may be prefixed with a bylist, and thus performed repeatedly for each element of the (categorical) variables in that list.

```
. bysort female: summ wage

-------------------------------------------------------------------------------------------
> female = 0

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        wage |       274    7.099489    4.160858        1.5      24.98

-------------------------------------------------------------------------------------------
> female = 1

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        wage |       252    4.587659    2.529363        .53      21.63
```

Previous command provides descriptive statistics for both female and male wages. The `bysort` prefix instead of the `by` prefix is useful when the data are not already sorted by the bylist variables and we do not want to sort in that way. In

what follows you have another example of an option for a specific command: `detail` (abbreviated with d). The specific output will be explained in lecture 2.

```
. bysort female: summ wage, d

-------------------------------------------------------------------------------------------
> female = 0

                    average hourly earnings
-------------------------------------------------------------
      Percentiles      Smallest
 1%         1.75            1.5
 5%         2.92           1.67
10%            3           1.75       Obs                 274
25%         4.11              2       Sum of Wgt.         274

50%            6                      Mean           7.099489
                        Largest       Std. Dev.      4.160858
75%         8.77           21.86
90%         12.5            22.2       Variance       17.31274
95%        15.38           22.86       Skewness       1.575794
99%         22.2           24.98       Kurtosis       5.828708

-------------------------------------------------------------------------------------------
> female = 1

                    average hourly earnings
-------------------------------------------------------------
      Percentiles      Smallest
 1%          1.5             .53
 5%          2.3            1.43
10%          2.9             1.5       Obs                 252
25%            3            1.63       Sum of Wgt.         252

50%         3.75                      Mean           4.587659
                        Largest       Std. Dev.      2.529363
75%         5.54           14.58
90%          7.5              15       Variance       6.397678
95%            9              18       Skewness       2.818724
99%           15           21.63       Kurtosis       15.02116
```

The `by` prefix should not be confused with the `by` option available on some commands. For example:

```
. ttest wage, by(female)

Two-sample t test with equal variances

------------------------------------------------------------------------------
   Group |     Obs        Mean    Std. Err.   Std. Dev.   [95% Conf. Interval]
---------+--------------------------------------------------------------------
       0 |     274    7.099489    .2513666    4.160858    6.604626    7.594352
       1 |     252    4.587659    .1593349    2.529363    4.273855    4.901462
---------+--------------------------------------------------------------------
combined |     526    5.896103    .1610262    3.693086    5.579768    6.212437
---------+--------------------------------------------------------------------
    diff |             2.51183    .3034092                1.915782    3.107878
------------------------------------------------------------------------------
Degrees of freedom: 524

                     Ho: mean(0) - mean(1) = diff = 0

     Ha: diff < 0                 Ha: diff != 0                 Ha: diff > 0
       t =   8.2787                 t =   8.2787                 t =   8.2787
   P < t =   1.0000             P > |t| =   0.0000             P > t =   0.0000
```

Previous command performs a two-sample t test of the hypothesis that wage has the same mean within the two groups, male and female. Below, another example of an `option` for a specific command:

```
. ttest wage, by(female) unequal

Two-sample t test with unequal variances
```

```
------------------------------------------------------------------------------
   Group |     Obs        Mean    Std. Err.   Std. Dev.   [95% Conf. Interval]
---------+--------------------------------------------------------------------
       0 |     274    7.099489    .2513666    4.160858    6.604626    7.594352
       1 |     252    4.587659    .1593349    2.529363    4.273855    4.901462
---------+--------------------------------------------------------------------
combined |     526    5.896103    .1610262    3.693086    5.579768    6.212437
---------+--------------------------------------------------------------------
    diff |            2.51183    .2976118                1.926971     3.09669
------------------------------------------------------------------------------
Satterthwaite's degrees of freedom:  456.327

                     Ho: mean(0) - mean(1) = diff = 0

     Ha: diff < 0                 Ha: diff != 0                 Ha: diff > 0
       t =   8.4400               t =   8.4400                  t =   8.4400
   P < t =   1.0000           P > |t| =   0.0000          P > t =   0.0000
```

The `by` prefix also modifies the meanings of the observation number symbol. usually `_n` refers to the current observation number, which varies from 1 to `_N`, the maximum defined observation. Under a bylist, `_n` refers to the observation within the bylist, and `_N` to the total number of observations for that category.

```
. bysort female married nonwhite: g obs=_n

. list female married nonwhite obs

     +-----------------------------------+
     | female   married   nonwhite   obs |
     |-----------------------------------|
  1. |      0         0          0     1 |
……
 77. |      0         0          0    77 |
 78. |      0         0          1     1 |
……
 86. |      0         0          1     9 |
 87. |      0         1          0     1 |
……
100. |      0         1          0    14 |
……
```

5.6. Command syntax: the using clause

In many empirical research projects, the data to be utilised may be stored in a number of separate files. Stata only permits a single data-set to be accessed at one time; however, commands like `merge, append, joinby` are available.

Suppose you have the two following data-sets:

```
Contains data from occ_1987_1.dta
  obs:            21
 vars:             2                          19 Jul 2005 16:18
 size:           273 (99.9% of memory free)
------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
settori         str5   %9s                    from original data
dsett           float  %40.0g      dsettlb    group(settori)
------------------------------------------------------------------------------
Sorted by:  dsett

Contains data from occ_1987_2.dta
  obs:            21
 vars:             2                          19 Jul 2005 16:20
 size:           252 (99.9% of memory free)
------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
ult87           float  %9.0g                  labour units, annual averages
                                                in thousands
dsett           float  %40.0g      dsettlb    group(settori)
```

```
-------------------------------------------------------------------------------
Sorted by:  dsett
```

The merge command combines two Stata-format data-sets that posses variables in common, adding other variables to the existing ones. It works on a master data-set, currenlty in memory, and a using data-set, both sorted on one or more merge variables (in the example, dsett):[5]

```
. use occ_1987_1, clear
. merge dsett using occ_1987_2
(label dsettlb already defined)
. descr
Contains data from occ_1987_1.dta
  obs:            21
 vars:             4                          19 Jul 2005 16:18
 size:           378 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
settori         str5   %9s                    from original data
dsett           float  %40.0g      dsettlb    group(settori)
ult87           float  %9.0g                  labour units, annual averages
                                                in thousands
_merge          byte   %8.0g
-------------------------------------------------------------------------------
Sorted by:
     Note:  dataset has changed since last saved

. tab _m
     _merge |      Freq.     Percent        Cum.
------------+-----------------------------------
          3 |         21      100.00      100.00
------------+-----------------------------------
      Total |         21      100.00
```

In previous case we did a one-to-one merge, in which each record in the using data-set is combined with one record in the master data-set. This is appropriate when you acquired additional variables for the same observations. A new variable, `_merge`, is created. You can check it by using the `tabulate` command: if `_merge` takes the integer value 3 only, this indicates that all the observations appear in both the master and the using data-sets. When `_merge=1`, one or more observations appear in the master only (occ1987); when `_merge=2`, one or more observations appear in the using only (occ1988). You can remove those observations which remain unmatched with the command: `drop if _m!=3.`

The distinction between master and using data-set is important. When the same variable is present in each of the files, by default Stata holds the master data inviolate and discard the using data-set copy of that variable. This may be modified by the `update` option: the values from the using data are retained when the master data contains missing; the codes for `_merge` are:

```
_merge==1    obs. from master data
_merge==2    obs. from using data
_merge==3    obs. from both, master agrees with using
_merge==4    obs. from both, missing in master updated
_merge==5    obs. from both, master disagrees with using
```

The `update replace` option specifies that even when the master data contain nonmissing values, they are to be replaced with corresponding values from the using data when corresponding values are not equal. A nonmissing value, however, will never be replaced with a missing value.

[5] If one of the data-zet is not sorted, the command is: `. merge dsett using occ_1987_2, sort.`

Now look at the two following data-sets:

```
. use occ_1987, clear
. descr
Contains data from occ_1987.dta
  obs:            21
 vars:             4                          19 Jul 2005 10:35
 size:           441 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
settori         str5   %9s
ult             float  %9.0g                  labour units, annual averages
                                                in thousands
dsett           float  %40.0g      dsettlb    group(settori)
year            float  %9.0g
-------------------------------------------------------------------------------
Sorted by:  dsett
     Note:  dataset has changed since last saved

. tab year
       year |      Freq.     Percent        Cum.
------------+-----------------------------------
       1987 |         21      100.00      100.00
------------+-----------------------------------
      Total |         21      100.00

. use occ_1988, clear
. descr
Contains data from occ_1988.dta
  obs:            21
 vars:             4                          19 Jul 2005 10:34
 size:           441 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
settori         str5   %9s
ult             float  %9.0g                  labour units, annual averages
                                                in thousands
dsett           float  %40.0g      dsettlb    group(settori)
year            float  %9.0g
-------------------------------------------------------------------------------
Sorted by:  dsett
     Note:  dataset has changed since last saved

. tab year
       year |      Freq.     Percent        Cum.
------------+-----------------------------------
       1988 |         21      100.00      100.00
------------+-----------------------------------
      Total |         21      100.00
```

The `append` command stacks the two data-set, thus adding observations of the using data-set to the master data-set. This is useful when you need to create timeseries data extracted from different databases.

```
. use occ_1987, clear
. append using occ_1988
(label dsettlb already defined)
. descr
Contains data from occ_1987.dta
  obs:            42
 vars:             4                          19 Jul 2005 16:25
 size:           882 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
settori         str5   %9s
ult             float  %9.0g                  labour units, annual averages
                                                in thousands
dsett           float  %40.0g      dsettlb    group(settori)
year            float  %9.0g
-------------------------------------------------------------------------------
Sorted by:
     Note:  dataset has changed since last saved
```

```
. tab year

       year |      Freq.     Percent        Cum.
------------+-----------------------------------
       1987 |         21       50.00       50.00
       1988 |         21       50.00      100.00
------------+-----------------------------------
      Total |         42      100.00

. save occ_1987_1988
file occ_1987_1988.dta saved
```

6. Loading external data and creating a file.dta

6.1. Use of the editor for file.xls

Suppose your data (a balanced panel data-set) are in excel, as in the case of the file "occupati_Italia.xls". An important check is to verify that a dot is used to separate integers from decimals.

From the file "occupati_Italia.xls", sheet "dati", select all the columns from "Dates" to "ultsndv2" and all the rows from "Dates" to "97". From the menu select copy. Go to Stata, click on the Data Editor, from the menu select paste. You obtain:

```
. edit
(23 vars, 18 obs pasted into editor)
. descr
Contains data
  obs:            18
 vars:            23
 size:         1,620 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
dates           byte   %8.0g                  Dates
ultag           float  %9.0g
ulten           float  %9.0g
ultma1          float  %9.0g
ultma2          float  %9.0g
ultma3          float  %9.0g
ultma4x7        float  %9.0g
ultma8          float  %9.0g
ultma9          float  %9.0g
ultma10         float  %9.0g
ultma11         float  %9.0g
ultma12         float  %9.0g
ultma13         float  %9.0g
ultma14         float  %9.0g
ultcos          float  %9.0g
ultco1          float  %9.0g
ultco2          float  %9.0g
ulttr           float  %9.0g
ultcre          float  %9.0g
ultloc          byte   %8.0g
ultvar          float  %9.0g
ultsndv1        float  %9.0g
ultsndv2        float  %9.0g
-------------------------------------------------------------------------------
Sorted by:
     Note:  dataset has changed since last saved
```

This data format is called *wide*:

i	$X_{i,j}$		
Dates	ultag	ulten	ultsndv2
80	2993.8	189.6	452.6

where *i* indicates the variable identifying a record (in our case, a year, *80*); *j* indicates the variable whose unic values identify a sub-observation (in our case, the different industries, *ag*, *en*, …, *sndv2*); finally, $X_{i,j}$ indicates the variables that may be transformed on the basis of *i* and *j*.
This data orientation is the one required by Stata in order to estimate seemingly unrelated regressions (`sureg`), in which the data must have T observations, with separate variables for each cross-sectional unit.
On the contrary, panel estimation (`xtreg`) requires that the data are stacked, i.e. in modality *long*. With this format is also easier to generate data transformations, where a single variable is involved:

```
i                                  X_{i,j}
Dates         settori              ult
-------------------------------------
80            ag                   2993.8
80            en                    189.6
..            ..
..            ..
80            sndv2                 452.6
```

The data transformation from `wide` to `long` and vice versa is obtained by the command `reshape`, a very powerful (and complicated because of the many variations) tool:

```
. reshape long ult, i(dates) j(settori) string
(note: j = ag co1 co2 cos cre en loc ma1 ma10 ma11 ma12 ma13 ma14 ma2 ma3 ma4x7 ma8 ma9 sndv1 sndv2
tr var)

Data                               wide   ->   long
-----------------------------------------------------------------------------
Number of obs.                       18   ->     396
Number of variables                  23   ->       3
j variable (22 values)                    ->   settori
xij variables:
               ultag ultco1 ... ultvar   ->   ult
-----------------------------------------------------------------------------

. descr

Contains data
  obs:           396
 vars:             3
 size:         5,544 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
dates           byte   %8.0g                  Dates
settori         str5   %9s
ult             float  %9.0g
-------------------------------------------------------------------------------
Sorted by:  dates  settori
     Note:  dataset has changed since last saved
```

In simplier cases, to create a vector may be enough the command `stack`.

In what follow we show some examples of data manipulation in order to improve their readibility:

```
. rename dates year
. replace year=year+1900
year was byte now int
(396 real changes made)
. tab year

      Dates |      Freq.     Percent        Cum.
------------+-----------------------------------
       1980 |         22        5.56        5.56
       1981 |         22        5.56       11.11
       1982 |         22        5.56       16.67
       1983 |         22        5.56       22.22
       1984 |         22        5.56       27.78
       1985 |         22        5.56       33.33
       1986 |         22        5.56       38.89
       1987 |         22        5.56       44.44
       1988 |         22        5.56       50.00
       1989 |         22        5.56       55.56
```

```
       1990 |         22        5.56       61.11
       1991 |         22        5.56       66.67
       1992 |         22        5.56       72.22
       1993 |         22        5.56       77.78
       1994 |         22        5.56       83.33
       1995 |         22        5.56       88.89
       1996 |         22        5.56       94.44
       1997 |         22        5.56      100.00
------------+-----------------------------------
      Total |        396      100.00

. egen dsett=group(settori)
. list dsett settori in 1/22

     +-----------------+
     | dsett   settori |
     |-----------------|
  1. |     1        ag |
  2. |     2       co1 |
  3. |     3       co2 |
  4. |     4       cos |
  5. |     5       cre |
     |-----------------|
  6. |     6        en |
  7. |     7       loc |
  8. |     8       ma1 |
  9. |     9      ma10 |
 10. |    10      ma11 |
     |-----------------|
 11. |    11      ma12 |
 12. |    12      ma13 |
 13. |    13      ma14 |
 14. |    14       ma2 |
 15. |    15       ma3 |
     |-----------------|
 16. |    16     ma4x7 |
 17. |    17       ma8 |
 18. |    18       ma9 |
 19. |    19     sndv1 |
 20. |    20     sndv2 |
     |-----------------|
 21. |    21        tr |
 22. |    22       var |
     +-----------------+

. do dsett_label

. label define dsettlb 1 `"agricoltura"', modify
. label define dsettlb 2 `"commercio"', modify
. label define dsettlb 3 `"alberghi"', modify
. label define dsettlb 4 `"costruzioni"', modify
. label define dsettlb 5 `"credito"', modify
. label define dsettlb 6 `"prodotti energetici"', modify
. label define dsettlb 7 `"locazione fabbricati"', modify
. label define dsettlb 8 `"minerali/metalli ferrosi e non ferrosi"', modify
. label define dsettlb 9 `"tessile/abbigl/cuoio"', modify
. label define dsettlb 10 `"legno"', modify
. label define dsettlb 11 `"carta/stampa"', modify
. label define dsettlb 12 `"gomma/plastica"', modify
. label define dsettlb 13 `"altri prodotti industriali"', modify
. label define dsettlb 14 `"min/prodotti da min non metalliferi"', modify
. label define dsettlb 15 `"chimica/pharma"', modify
. label define dsettlb 16 `"prodotti metallo"', modify
. label define dsettlb 17 `"mezzi di trasporto"', modify
. label define dsettlb 18 `"alimentari/tabacco"', modify
. label define dsettlb 19 `"amm.ni pubbliche"', modify
. label define dsettlb 20 `"servizi domestici/istit. sociali private"', modify
. label define dsettlb 21 `"trasporti/comunicazioni"', modify
. label define dsettlb 22 `"servizi forniti alle imprese"', modify

. label values dsett dsettlb

end of do-file
```

The dsett_label.do is a program we prepared in order to improve data readibility (dsett is a variable better than settori to indicate individuals of our panel). The program-file was written by looking at the list above and at the “legenda” sheet in the file “occupati_Italia.xls”.

```
. tab dsett

                        group(settori) |      Freq.     Percent        Cum.
---------------------------------------+-----------------------------------
                           agricoltura |         18        4.55        4.55
                             commercio |         18        4.55        9.09
                              alberghi |         18        4.55       13.64
                           costruzioni |         18        4.55       18.18
                               credito |         18        4.55       22.73
                    prodotti energetici |         18        4.55       27.27
                   locazione fabbricati |         18        4.55       31.82
 minerali/metalli ferrosi e non ferrosi |         18        4.55       36.36
                   tessile/abbigl/cuoio |         18        4.55       40.91
                                 legno |         18        4.55       45.45
                          carta/stampa |         18        4.55       50.00
                         gomma/plastica |         18        4.55       54.55
             altri prodotti industriali |         18        4.55       59.09
    min/prodotti da min mon metalliferi |         18        4.55       63.64
                         chimica/pharma |         18        4.55       68.18
                       prodotti metallo |         18        4.55       72.73
                     mezzi di trasporto |         18        4.55       77.27
                     alimentari/tabacco |         18        4.55       81.82
                       amm.ni pubbliche |         18        4.55       86.36
servizi domestici/istit. sociali privat |         18        4.55       90.91
                  trasporti/comunicazioni |         18        4.55       95.45
            servizi forniti alle imprese |         18        4.55      100.00
---------------------------------------+-----------------------------------
                                 Total |        396      100.00

. tab settori

    settori |      Freq.     Percent        Cum.
------------+-----------------------------------
         ag |         18        4.55        4.55
        co1 |         18        4.55        9.09
        co2 |         18        4.55       13.64
        cos |         18        4.55       18.18
        cre |         18        4.55       22.73
         en |         18        4.55       27.27
        loc |         18        4.55       31.82
        ma1 |         18        4.55       36.36
       ma10 |         18        4.55       40.91
       ma11 |         18        4.55       45.45
       ma12 |         18        4.55       50.00
       ma13 |         18        4.55       54.55
       ma14 |         18        4.55       59.09
        ma2 |         18        4.55       63.64
        ma3 |         18        4.55       68.18
      ma4x7 |         18        4.55       72.73
        ma8 |         18        4.55       77.27
        ma9 |         18        4.55       81.82
      sndv1 |         18        4.55       86.36
      sndv2 |         18        4.55       90.91
         tr |         18        4.55       95.45
        var |         18        4.55      100.00
------------+-----------------------------------
      Total |        396      100.00

. descr

Contains data
  obs:           396
 vars:             4
 size:         7,524 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
year            int    %8.0g                  Dates
settori         str5   %9s
ult             float  %9.0g
dsett           float  %40.0g      dsettlb    group(settori)
-------------------------------------------------------------------------------
Sorted by:
     Note:  dataset has changed since last saved
```

Try to type label list dsettlb.

6.2. Insheet command for file.prn, file.raw, file.csv (comma-separated)

The following command can be used for comma-separated or tab-delimited data files. Suppose, for example, the file auto.prn, in which you have an identifier for the company, a year, a company's name, sales and model of car:

```
1,1994,Fiat,350,500
2,1994,Ford,400,3000
3,1994,Chev. Monza, ,1500
```

You also may have a file as the following:

```
1      1994   Fiat   350    500
2      1994   Ford   400    3000
3      1994   Chev. Monza         1500
```

It is important that you verify the content of the file with the command:

```
. type auto.prn, showtabs
1,1994,Fiat,350,500
2,1994,Ford,400,3000
3,1994,Chev. Monza, ,1500
```

Even if you a have a blank in the name "Chev. Monza" and a missing for the sales of company 3, you will not encounter any problem:

```
. insheet codice anno nome fatt modello using auto.prn
(5 vars, 3 obs)

. descr

Contains data
  obs:             3
 vars:             5
 size:            69 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
codice          byte   %8.0g
anno            int    %8.0g
nome            str11  %11s
fatt            str3   %9s
modello         int    %8.0g
-------------------------------------------------------------------------------
Sorted by:
     Note:  dataset has changed since last saved

. list

     +-----------------------------------------+
     | codice   anno          nome   fatt   modello |
     |-----------------------------------------|
  1. |      1   1994          Fiat    350       500 |
  2. |      2   1994          Ford    400      3000 |
  3. |      3   1994   Chev. Monza             1500 |
     +-----------------------------------------+
```

Note that insheet cannot read space-delimited data or character strings with embedded spaces.

6.3. Infile command for file.asc, file.raw (ASCII format)

Suppose your data, contained in the file auto.raw, are in a free format, space-delimited:

```
. type auto.raw, showtabs
1 1994 "Fiat" 350 500
2 1994 "Ford" 400 3000
3 1994 "Chev. Monza" . 1500
```

You must use the following command:

```
. infile codice anno str10 nome fatt modello using auto.raw
(3 observations read)
```

```
. list

     +-----------------------------------------+
     | codice   anno         nome   fatt   modello |
     |-----------------------------------------|
  1. |      1   1994         Fiat    350       500 |
  2. |      2   1994         Ford    400      3000 |
  3. |      3   1994   Chev. Monz      .      1500 |
     +-----------------------------------------+
```

where the string variables without embedded spaces must be specified in the command. Note that single quotes can be used instead of double quotes. Single or double quotes delimiting strings can be omitted if the string contains no blanks or other special characters.

Now suppose the most complicated case, with fixed-format data including data containing undelimited string variables.

```
. type auto_c.raw, showtabs
11994Fiat350500
21994Ford4003000
31994Chev. Monza 1500
```

In order to read these kind of data fields, not delimited, you must create a dictionary file which describes the format of each variable and defines the variable' location. Moreover, you can separately read those records for which certain conditions are satisfied (the different formats are being properly specified on for different subsets of the file).

```
. infile using auto_c1.dct in 1/2

dictionary using auto_c.raw {

      long    codice   %1f  "Codice impresa"
      float   anno     %4f  "Anno di bilancio"
_column(6)
      str4    nome     %4s  "nome societa'"
_column(10)
      float   fatt     %3f  "fatturato"
      float   modello  %4f  "modello"
}

(2 observations read)

. list

     +--------------------------------------+
     | codice   anno   nome   fatt   modello |
     |--------------------------------------|
  1. |      1   1994   Fiat    350       500 |
  2. |      2   1994   Ford    400      3000 |
     +--------------------------------------+

. save pippo
file pippo.dta saved

. infile using auto_c2.dct in 3

dictionary using auto_c.raw {

      long    codice   %1f  "Codice impresa"
      float   anno     %4f  "Anno di bilancio"
_column(6)
      str11    nome    %11s  "nome societa'"
_column(17)
      float   fatt     %1f  "fatturato"
_column(18)
      float   modello  %4f  "modello"
}
(1 observations read)

. list

     +-----------------------------------------+
     | codice   anno          nome   fatt   modello |
     |-----------------------------------------|
  1. |      3   1994   Chev. Monza      .      1500 |
     +-----------------------------------------+
```

```
. append using pippo

. list

     +-----------------------------------------------+
     | codice   anno          nome   fatt   modello |
     |-----------------------------------------------|
  1. |      3   1994   Chev. Monza      .      1500 |
  2. |      1   1994          Fiat    350       500 |
  3. |      2   1994          Ford    400      3000 |
     +-----------------------------------------------+

. sort cod

. list

     +-----------------------------------------------+
     | codice   anno          nome   fatt   modello |
     |-----------------------------------------------|
  1. |      1   1994          Fiat    350       500 |
  2. |      2   1994          Ford    400      3000 |
  3. |      3   1994   Chev. Monza      .      1500 |
     +-----------------------------------------------+
.
```

Some important options for the dictionary are:

`_column()` says where the data begin for each variable. When you also write how many columns the data span (the %1f, %4s, etc.), some of this information may be redundant: after reading “anno”, Stata has finished with 5 columns read, and, unless instructed otherwise, it would proceed to the next column, column 6, to begin reading information about “nome”.

You may also specify that more than one record in the input file corresponds to a single observation in the data-set. Notice that:

`_line()` goes to the specified line of the observation

`_lines()` says how many lines each observation takes.

For example:

```
. type auto_cc.raw, showtabs
11994
Fiat
350
500
21994
Ford
400
3000
31994
Chev. Monza

1500

. infile using auto_cc.dct

dictionary using auto_cc.raw {

_lines(4)
_line(1)
      long    codice   %1f  "Codice impresa"
      float   anno     %4f  "Anno di bilancio"
_line(2)
      str11    nome    %11s  "nome societa'"
_line(3)
      float   fatt     %3f   "fatturato"
_line(4)
      float   modello  %4f   "modello"
}

(3 observations read)

. list
     +-----------------------------------------------+
     | codice   anno          nome   fatt   modello |
     |-----------------------------------------------|
  1. |      1   1994          Fiat    350       500 |
  2. |      2   1994          Ford    400      3000 |
  3. |      3   1994   Chev. Monza      .      1500 |
     +-----------------------------------------------+
```

Also see the command `infix`, an alternative to `infile` with a dictionary, and that has a syntax similar to that used by SAS.

6.4. The Stat/Transfer software

Stat/Transfer is a very useful tool when you need to read data already in the format of SAS, SPSS, Excel, GAUSS, MATLAB, DBF and many other packages. It converts file formats into Stata format, without loss of variable labels, value labels, and the like. See the stata.com web site.

7. Working with panel and time-series: basics

Stata has the `d., l. , f.` operators that may be used in time-series and panel data to specify first differences, lags, and leads, respectively. These operators understand missing data and numlists: `l(1/4).x` stands for x_{t-1}, x_{t-2}, x_{t-3}, x_{t-4}. Before using these operators, it is important to declare the data to be a panel or a time series and to designate that timevar represents time. For panel:

```
tsset indvar timevar
```

The observation indicator and the timeseries indicator may also be defined separately:

```
iis indvar
tis timevar
```

If you have a panel, or longitudinal data, you can obtain pure cross-section or pure time-series in different ways.

For example, if you need to summarise a variable to each time-period T, you can use:

```
. use occ_1987_1988, clear

. tsset dsett year
       panel variable:  dsett, 1 to 21
        time variable:  year, 1987 to 1988

. bysort year: summ ult
-------------------------------------------------------------------------------------------------------
-> year = 1987

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
         ult |        21    969.9857    1136.864          0     3972.2

-------------------------------------------------------------------------------------------------------
-> year = 1988

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
         ult |        21    985.3095    1155.506          0     3993.6
```

If you need to create average values for each time period averaged over individuals N inside your panel data-set, you can use:

```
. egen ultm=mean(ult), by(year)
. bysort year: summ ultm

-------------------------------------------------------------------------------------------------------
-> year = 1987

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        ultm |        21    969.9857           0   969.9857   969.9857

-------------------------------------------------------------------------------------------------------
-> year = 1988

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        ultm |        21    985.3095           0   985.3095   985.3095
```

If you need to create average values for each time period averaged over individuals N and, at the same time, to obtain a new datas-set, you can use:

```
. collapse (mean) ult, by(year)
. list

     +-----------------+
     | year        ult |
     |-----------------|
  1. | 1987   969.9857 |
  2. | 1988   985.3095 |
     +-----------------+
```

Dealing with time series requires the following steps.

- A timevar is created by using the functions for creating dates from year and week, year and quarter, etc:

`yw(year_exp, week_exp)`

`ym(year_exp, month_exp)`

`yq(year_exp, quarter_exp)`

`yh(year_exp, halfyear_exp)`

`mdy(month_exp, day_exp, year_exp)`

see help on `tsfun` and, in particular, on `ywfcns` and `mdyfcn`.

- How timevar will be displayed is selected by using the `format(%fmt), daily, weekly, monthly, quarterly, halfyearly, yearly, and generic`. Stata understands the following time scales `%t`:

```
Format   Description           Coding
-------------------------------------------------------------------------------------------
%td      daily (same as %d)    0 = 01jan1960, 1 = 02jan1960
%tw      weekly                0 = 1960w1,    1 = 1960w2
%tm      monthly               0 = 1960m1,    1 = 1960m2
%tq      quarterly             0 = 1960q1,    1 = 1960q2
%th      halfyearly            0 = 1960h1,    1 = 1960h2
%ty      yearly                1960 = 1960,   1961 = 1961
%tg      generic               0 = ?
-------------------------------------------------------------------------------------------
Note: times before 1960 are allowed. For instance, -1 means 31dec1959 in %td format and
1959q4 in %tq format.
```

As a first example, suppose that your data are quarterly, starting in 1959q1; thus, the first observation has timevar = -4. We call the timevar “time”.

```
. use USquarter, clear
. list anno trimestre

     +-----------------+
     | anno   trimes~e |
     |-----------------|
  1. | 1959          1 |
  2. | 1959          2 |
  3. | 1959          3 |
  4. | 1959          4 |
  5. | 1960          1 |
     |-----------------|
……
. g time=yq(anno, trimestre)
. list anno trimestre time

     +------------------------+
     | anno   trimes~e   time |
     |------------------------|
  1. | 1959          1     -4 |
  2. | 1959          2     -3 |
  3. | 1959          3     -2 |
  4. | 1959          4     -1 |
  5. | 1960          1      0 |
     |------------------------|
……
```

You could tsset and simultaneously format your data.

```
. tsset time, quarterly
        time variable:  time, 1959q1 to 2000q4

. list anno trimestre time

     +-------------------------+
     | anno   trimes~e    time |
     |-------------------------|
  1. | 1959          1   1959q1 |
  2. | 1959          2   1959q2 |
  3. | 1959          3   1959q3 |
  4. | 1959          4   1959q4 |
  5. | 1960          1   1960q1 |
     |-------------------------|
…..
```

You may obtain the same result by `tsset time, format(%tq).` See help `tdates` and `tfmt` for more information on the %t format (use whichever appeals to you) and the advantages of setting it.

The format makes the `tin()` and `twithin()` selection functions work so that later, after tsseting the data, you can type things like:

`regress ... if tin(1959q1, 1959q4)`

to run a regression on the subsample {1959q1 ≤ timevar ≤ 1959q4}, or

`regress ... if twithin(1959q1, 1959q4)`

to run a regression on the subsample {1959q1 < timevar < 1959q4}.

As another example:

```
. use NYTdailysales, clear
. g time=mdy(month, day, year)
. list year month day time

     +---------------------------+
     | year   month   day    time |
     |---------------------------|
  1. | 2000       7     3   14794 |
  2. | 2000       7     4   14795 |
  3. | 2000       7     5   14796 |
  4. | 2000       7     6   14797 |
  5. | 2000       7     7   14798 |
     |---------------------------|
…..

. tsset time, daily
        time variable:  time, 03jul2000 to 31oct2001, but with gaps
. list year month day time

     +--------------------------------+
     | year   month   day        time |
     |--------------------------------|
  1. | 2000       7     3   03jul2000 |
  2. | 2000       7     4   04jul2000 |
  3. | 2000       7     5   05jul2000 |
  4. | 2000       7     6   06jul2000 |
  5. | 2000       7     7   07jul2000 |
     |--------------------------------|
…….

. tsset time, format(%tdD/N/Y)
        time variable:  time, 03/07/00 to 31/10/01, but with gaps
. list year month day time

     +-------------------------------+
     | year   month   day       time |
     |-------------------------------|
  1. | 2000       7     3   03/07/00 |
  2. | 2000       7     4   04/07/00 |
  3. | 2000       7     5   05/07/00 |
  4. | 2000       7     6   06/07/00 |
  5. | 2000       7     7   07/07/00 |
     |-------------------------------|
```

……

In this example Stata is warning you that the timeseries are not complete. See commands: `tsfill`, `ipolate`. Alternatively, a trick may be:

```
. egen obs=group(day month year), label
. tsset obs
        time variable:  obs, 1 to 175
. sort year month day
. list year month day obs

     +---------------------------------+
     | year   month   day          obs |
     |---------------------------------|
  1. | 2000       7     3     3 7 2000 |
  2. | 2000       7     4     4 7 2000 |
  3. | 2000       7     5     5 7 2000 |
  4. | 2000       7     6     6 7 2000 |
  5. | 2000       7     7     7 7 2000 |
     |---------------------------------|
```

……

8. Working with commands' results and some preliminaries on programming

All Stata statistical commands leave results in the program's data structures, so that customized output is readily generated by extracting items from these data structures.

Commands are either "r-class" routines, which return results in the `r()` data structures, or 'e-class" (estimation) routines, which return results in the `e()` data structures. Each data structure may contain scalars, local macros, matrices, and functions: for instance, `r(mean)` contains the mean of a series after application of r-class command `summarize`, while `e(b)` is a matrix (row vector) of estimated coefficients returned by e-class command `regress`.

Results are available until another r-class or e-class command is executed, and may be examined with commands `return list` (for the r-class) and `ereturn list` (for the e-class). Also see [U] 18.9 Accessing results calculated by estimation commands.

Another useful capability is provided by the `postfile` and `post` commands, which permits a separate Stata data-set to be created in the course of a program; after the program is finished, this new data-set with statistics or fitted values or estimated standard errors can be opened and analysed. In the `post` command the parenteses () surrounding each exp are required; the new data-set can be named with `tempname` command and used by the `postfile` command as `` `new-data-set_name' ``

In what follows you will find some examples of Stata programs.

```
. use WAGE1.dta, clear
. covaun wage
Coefficient of Variation = .62636053
```

`Covaun.ado` is a program that displays the coefficient of variation of a variable. It uses macros, i.e. named entities that can contain both strings and numeric values, or numeric values only (at maximum precision). The definition of macros is really much more confusing than the concept. Look, for example, at the program `covaun.ado:`

```
capture program drop covaun
program define covaun
  version 8.0
  quietly summarize `1'
  local v1 = r(sd)/r(mean)  /* coefficient of variation */
  display "Coefficient of Variation = " `v1'
end
```

The macro `1`, is used to indicate the first argument passed to a program: when you type `covaun wage`, the contents of `1`, `` `1' ``, is wage. Using the same logic, you can add macro `2, 3, etc.` containing a second, third, etc. variable. The second macro contained in the program is `v1`, defined as the scalar `r(sd)/r(mean)`, i.e. by using post-results of the summarize command.

In Stata you have the following macros:

- local macros, which exist and are accessible only within the procedure where they were created; after definition, to use them, you must type `` `local_name' ``;
- global macros, which exist and are accessible across all procedures; after definition, to use them, you must type `$global_name`;
- scalar, that, differently from local and global macros, only can contain a single number (at maximum precision) but not a string.

Now look at a different program, that displays the coefficient of variation of more than one variable:

```
. covamu wage educ
Coefficient of variation for wage = .62636053
Coefficient of variation for educ = .22041552
```

where the program content is:

```
capture program drop covamu
program define covamu
  version 8.0
  local i = 1
  while "``i''" ~= "" {
    quietly summarize ``i''
    local v1 = r(sd)/r(mean)  /* coefficient of variation */
    display "Coefficient of variation for ``i'' = " `v1'
    local i = `i' + 1
  }
end
```

In this case, you see an example of local macro used as counters within a loop structure explicitly defined by the `while` command. The new local macro `i` is used to index through the list of variables that were typed after the program name. The first time through the loop the contents of `i` is `` `1' ``, the second time it is `` `2' `` and so on. Note that you need to use the compound double quotes (`“`something'”`) in order to make `something` becoming the content of the macro, where `something` itself contains quotation marks.

The same can be done avoiding the numbered macros altogether and istead looping the variables in `` `varlist' `` directly. In this way, the program is executed even more quickly.

```
. covamu1 wage educ

Coefficient of variation for wage = .62636053
Coefficient of variation for educ = .22041552

. covamu1 wage educ, title("all the sample")

all the sample:
Coefficient of variation for wage = .62636053
Coefficient of variation for educ = .22041552

. covamu1 wage educ if female==1, title("for females only")

for females only:
Coefficient of variation for wage = .55134073
Coefficient of variation for educ = .20074282
```

The program content is:

```
capture program drop covamu1
program define covamu1
  version 8.0
  syntax varlist [if] [in] [, title(string)]
  display
  if "`title'"!="" {
           display "`title':"
 }
  foreach var of local varlist {
    quietly summarize `var' `if' `in'
    local v1 = r(sd)/r(mean)  /* coefficient of variation */
    display "Coefficient of variation for `var' = " `v1'

  }
end
```

where the command `syntax` examines what the user typed and attempts to match it to the syntax diagram: if it does match, the individual components are stored in particular local macros (`` `var', `if', `in', `title' ``) where you can subsequently access them. Option in squared brackets indicate that they are optional (without squared brackets they are required: if you do not type them, an error message is issued). The command `foreach var of local varlist` repeatedly sets local macro `` `var' `` to each element of the list obtained from `syntax` and executes on it the commands enclosed in braces.

Stata also contains a full-featured matrix language that allows any estimation results to be stored in matrices and manipulated, thus supporting the programming of many estimators. In what follows we make a comparison between the output of Stata regression commands (`regress; vce; vce, corr`) and the output obtained by a program (`reg_matrix.ado`) that uses some matrix and post-estimation commands.

```
. use "D:\LAVORI\bookbogo\intro_to_stata\WAGE1.DTA", clear
. reg wage female educ

      Source |       SS       df       MS              Number of obs =     526
-------------+------------------------------           F(  2,   523) =   91.32
       Model |  1853.25304     2  926.626518            Prob > F      =  0.0000
    Residual |  5307.16125   523  10.1475359           R-squared     =  0.2588
-------------+------------------------------           Adj R-squared =  0.2560
       Total |  7160.41429   525  13.6388844           Root MSE      =  3.1855

------------------------------------------------------------------------------
        wage |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      female |  -2.273362   .2790444    -8.15   0.000    -2.821547   -1.725176
        educ |   .5064521   .0503906    10.05   0.000     .4074592     .605445
       _cons |   .6228168   .6725334     0.93   0.355     -.698382    1.944016
------------------------------------------------------------------------------

. vce

             |   female     educ    _cons
-------------+---------------------------
      female |   .077866
        educ |   .001196  .002539
       _cons | -.052325 -.032472  .452301

. vce, corr

             |   female     educ    _cons
-------------+---------------------------
      female |   1.0000
        educ |   0.0850   1.0000
       _cons |  -0.2788  -0.9582   1.0000

. reg_matrix wage female educ
```

```
------------------------------------------------------------------------------
        wage |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      female |  -2.273362   .2790444    -8.15   0.000    -2.821547   -1.725176
        educ |   .5064521   .0503906    10.05   0.000     .4074592     .605445
       _cons |   .6228168   .6725334     0.93   0.355     -.698382    1.944016
------------------------------------------------------------------------------

Covariance of the regression coefficients

symmetric e(V)[3,3]
            female        educ       _cons
female   .07786577
  educ   .00119562   .00253922
 _cons  -.05232474  -.03247231   .45230119

Correlation of the regression coefficients

symmetric C[3,3]
            female        educ       _cons
female           1
  educ   .08502941           1
 _cons  -.27881745  -.95818535           1
```

The `reg_matrix.ado` program content is the following:

```
capture program drop reg_matrix
program define reg_matrix, eclass

  version 8.0

    syntax varlist(min=2 numeric) [if] [in] [, Level(integer $S_level)]
    marksample touse                        /* mark cases in the sample */
    tokenize "`varlist'"

    quietly matrix accum sscp = `varlist' if `touse'
    local nobs = r(N)
    local df = `nobs' - (rowsof(sscp) - 1) /* df residual */

    matrix XX = sscp[2...,2...]             /* X'X */
    matrix Xy = sscp[1,2...]                /* X'y */

    matrix b = Xy * syminv(XX)              /* (X'X)-1X'y */
    local k = colsof(b)                     /* number of coefs */
    matrix hat = Xy * b'
    matrix V = syminv(XX) * (sscp[1,1] - hat[1,1])/`df'
    matrix C = corr(V)
    matrix seb = vecdiag(V)
    matrix seb = seb[1, 1...]
    matrix t = J(1,`k',0)
    matrix p = t

    local i = 1
    while `i' <= `k' {
      matrix seb[1,`i'] = sqrt(seb[1,`i'])
      matrix t[1,`i'] = b[1,`i']/seb[1,`i']
      matrix p[1,`i'] = tprob(`df',t[1,`i'])
      local i = `i' + 1
    }

 ereturn post b V, dof(`df') obs(`nobs') depname(`1') /*
      */ esample(`touse')
    ereturn local depvar "`1'"
    ereturn local cmd "matreg"

    display
    ereturn display, level(`level')

  display
  display "Covariance of the regression coefficients"
  matrix list e(V)
  display
  display "Correlation of the regression coefficients"
  matrix list C

  matrix drop sscp XX Xy hat seb t p C
  ereturn clear
end
```

Use the help and the new mata manual to better understand……..

9. An overview of useful commands

```
help            on line help on a specific command
search          on line references on a keyword or topic
log             logging to an external file
version         shifting from one Stata release to another
generate        create a new variable
replace         modify an existing variable
sort            change the sort order of the dataset
compress        economise on space used by variables
append          combine datasets by stacking
merge           merge datasets
encode          generate numeric variable from categorical variable
recode          recode categorical variable
destring        convert string variables to numeric
tab             tabulate 1- and 2-ways tables
table           tables of summary statistics
for             repeat a Stata command over a varlist, numlist or anylist
while           loop with a counter over a block of code
local           define or modify a local macro, the name to be typed as `varname'
global          define or modify a global macro, the name to be typed as $varname
scalar          define or modify a scalar variable
use             load a Stata data-set
save            write the contents of memory to a Stata data-set
insheet         load a text file in tab- or comma-delimited format
infile          load a text file in space-delimited format or as defined in a dictionary
outfile         write a text file in space- or comma-delimited format
outsheet        write a text file in tab- or comma-delimited format
clear           clear memory
drop            drop certain variables and/or observations
keep            keep only certain variables and/or observations
rename          rename variables
collapse        make a dataset of summary statistics
tsset           define the time indicator for panel data
quietly         do not show the results of a command
update query    see if Stata is up to date
exit            exit the program (, clear if dataset is not saved)

summarise       descriptive statistics
correlate       correlation matrices
corrgram        correlogram estimation
anova           1-, 2-, n-ways analysis of variance
arima           Box-Jenkins models
dfuller         unit roots tests
graph           produce a variety of graphs
regress         least squares regression
predict         generate fitted values, residuals, etc.
ttest           perform 1-, 2-samples and paired t tests
test            test linear hypotheses on parameters
```

```
testparm       usefgul alternative to test
lincom         linear combinations of parameters
cnsreg         regression with linear constraints
nlcom          non-linear combinations of parameters
testnl         test nonlinear hypotheses on parameters
nl             non linear least squares
ml             maximum likelihood estimation with user-specific LLF
mfx            marginal effects after nonlinear estimation
bstrap         bootstrap sampling
ivreg          instrumental variables regression
prais          regression with AR(1) errors
, robust       regression with robust standard errors
arch           models of autoregressive conditional heteroskedasticity
sureg          seemingly unrelated regressions
reg3           three-stages least squares
probit         binomial probit estimation
oprobit        ordered probit estimation
logit          binomial logit estimation
ologit         ordered logit estimation
tobit          Tobit regression
cnreg          censored normal regression
glm            generalised linear models
heckman        Heckman’s selection model
qreg           quantile regression, including median regression
xtreg          panel data estimation
```

In the following lectures you will see practical examples of these commands.