Introduction to Stata for Economists

Tancredi Rapone UAB & BSE[*]

January 2025

Contents

1 Data Management **4**

1.1 Preliminaries . . . 4

1.2 Loading data . . . 5

1.3 New commands and packages . . . 5

1.4 Syntax . . . 5

1.5 Do files & Logs . . . 6

1.6 Working Directory . . . 6

1.7 Data . . . 7

1.8 Data efficiency . . . 7

1.9 Managing strings . . . 7

1.10 Missing data . . . 8

1.11 Explore the data . . . 8

1.12 Generating variables . . . 9

1.13 Labeling the data and the variables . . . 11

1.13.1 Value labels . . . 11

1.13.2 Variable labels . . . 11

1.14 Logical conditions, loops and lists . . . 11

1.14.1 Basic syntax using logical operators . . . 11

1.14.2 Loops and macros . . . 12

1.14.3 Lists of variables . . . 13

1.15 More advanced data manipulations . . . 14

1.15.1 Collapse, Preserve and Restore . . . 14

1.15.2 Merge, append & joinby . . . 14

1.15.3 Reshape . . . 16

1.16 Duplicate data and _n & _N . . . 16

[*]Thanks to Joseph Emmens for kindly sharing his teaching material.

2 Data Analysis **19**
2.1 Summarizing the Data . . . 19
2.2 Graphing . . . 20
2.2.1 Schemes . . . 20
2.2.2 Continuous variables: Histograms and kernel densities . . . 20
2.2.3 Discrete variables: Bar charts . . . 21
2.3 Two variables . . . 21
2.3.1 Discrete-Discrete . . . 21
2.3.2 Discrete-Continuous . . . 21
2.3.3 Continuous-Continuous . . . 21
2.3.4 Binscatter . . . 22
2.3.5 Fitted lines . . . 23
2.3.6 Regression results . . . 23

3 Modeling **26**
3.1 Standard regression model . . . 26
3.2 Factor variables . . . 27
3.3 Interactions . . . 28
3.4 Weights . . . 29
3.5 Presenting Tables . . . 29
3.6 Advanced Regression Methods . . . 30
3.6.1 Discrete Choice . . . 30
3.7 Model selection and goodness of fit . . . 30

4 Post-Estimation **34**
4.1 Standard regression model . . . 34
4.1.1 Predictions and residuals . . . 35
4.2 Marginal effects . . . 35
4.3 Accessing statistics . . . 36

References **37**

A Statistical Appendix **38**
A.1 Further Literature . . . 38
A.2 Random Variables and Their Distributions . . . 39
A.3 Expectation of a Random Variable . . . 40
A.3.1 Properties of Expectation . . . 41
A.4 Summary Statistics . . . 41
A.5 Kernel Density Estimation . . . 44

B Hypothesis Testing and Statistical Models **44**
B.1 Central Limit Theorem (CLT) . . . 44
B.2 Hypothesis Testing . . . 45
B.3 Estimators and Their Properties . . . 47
B.4 Ordinary Least Squares (OLS) . . . 49

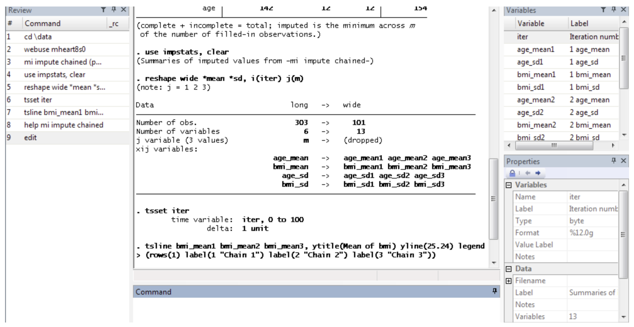
B.4.1 Panel Data Models 51
B.4.2 Binary Choice Models (Logit) 52

1 Data Management

1. Introduction & basic command syntax
2. Commands, do files & logs
3. Installing packages & finding help
4. Importing data, understanding data efficiency and types
5. Generating variables
6. Summarising the data (collapse)
7. Using bysort
8. Reshaping the data
9. Combining multiple datasets using merge, append & joinby
10. Identifying duplicate observations
11. Exporting data

1.1 Preliminaries

When you open STATA you will see four sections which show (from left to right), the review window where you can see past commands you executed, the results panel where you see the output from your commands and the variables panel where you can see the variable names and labels of the data you currently have loaded into memory. In the bottom half of the interface you have the command line prompt where you can write all of the commands we will go over in these notes. In the bottom right corner you have the properties window which shows key information about the dataset you currently have loaded in memory, such as how much storage it is taking, the number of variables etc.



#	Command	_rc
1	cd \data	
2	webuse mheart8s0	
3	mi impute chained (p...	
4	use impstats, clear	
5	reshape wide *mean *s...	
6	tsset iter	
7	tsline bmi_mean1 bmi...	
8	help mi impute chained	
9	edit	

```
           age |        142           12           12 |        154
(complete + incomplete = total; imputed is the minimum across m
 of the number of filled-in observations.)

. use impstats, clear
(Summaries of imputed values from -mi impute chained-)

. reshape wide *mean *sd, i(iter) j(m)
(note: j = 1 2 3)

Data                               long   ->   wide
-----------------------------------------------------------------------------
Number of obs.                      303   ->     101
Number of variables                   6   ->      13
j variable (3 values)                 m   ->   (dropped)
xij variables:
                               age_mean   ->   age_mean1 age_mean2 age_mean3
                               bmi_mean   ->   bmi_mean1 bmi_mean2 bmi_mean3
                                 age_sd   ->   age_sd1 age_sd2 age_sd3
                                 bmi_sd   ->   bmi_sd1 bmi_sd2 bmi_sd3
-----------------------------------------------------------------------------

. tsset iter
        time variable:  iter, 0 to 100
                delta:  1 unit

. tsline bmi_mean1 bmi_mean2 bmi_mean3, ytitle(Mean of bmi) yline(25.24) legend
> (rows(1) label(1 "Chain 1") label(2 "Chain 2") label(3 "Chain 3"))
```

Variable	Label
iter	Iteration numb
age_mean1	1 age_mean
age_sd1	1 age_sd
bmi_mean1	1 bmi_mean
bmi_sd1	1 bmi_sd
age_mean2	2 age_mean
age_sd2	2 age_sd
bmi_mean2	2 bmi_mean
bmi_sd2	2 bmi_sd

Variables	
Name	iter
Label	Iteration numb
Type	byte
Format	%12.0g
Value Label	
Notes	
Data	
Filename	
Label	Summaries of
Notes	
Variables	13

Figure 1: Stata interface

To execute the code in the following examples, you simply need to write it in the command window and press enter. You can (and should) run commands also from a dofile, which is STATA's version of a text-editor where you can write and store a sequence of commands.

1.2 Loading data

For this class we will use the `patent_data` set. To load it into stata simply type:

```
use $path/patent_data
```

Where you would replace "$path" with the folder path where the data is stored. STATA also comes with several pre-installed datasets you can use to test commands or run examples, the most common of these is the `auto` dataset, which contains prices and characteristics for 74 automobiles. To load this example dataset into stata type:

```
sysuse auto, clear
```

In the command window.

1.3 New commands and packages

The command `help` brings up information on commands and other STATA uses. E.g.

```
help regress
```

STATA comes with many packages installed, however you can install new versions or user-written packages. You can check which packages you have installed with

```
ado
```

New packages are installed using

```
ssc install estout
```

If you cannot find the package through SSC, then searching for it, you often can find the install documents

```
search packagename
```

1.4 Syntax

The generic command syntax uses the following structure

```
[prefix:]  command [varlist] [=exp] [if] [in] [weight] [using filename]
, [options]
```

for example

```
summarize
```

compared to

```
summarize patent_number year cts_wt, d
```

1.5 Do files & Logs

The command line is useful for quick and preliminary queries, however for replication, you should use a .do file, and save the output in a .log file. Initialise a log file as

```
log using "path/logfile_name.log"
```

⋆Remember to close the log file with `log close` at the end of use, else it will throw an error next time you run the script. A dofile is a script, essentially a sequence of commands written down and saved as a STATA redeable text file which allows you to keep track of what you did with your data without needing to remember all the commands you ran in the command window should you want to replicate your result at a future date. To open a dofile just type `doedit` in the command window and then save it to a folder of your choosing (ideally a different one dedicated only to dofiles).

1.6 Working Directory

There are two ways of structuring your code and path. To check your current working directory use

```
pwd
```

and to change it use

```
cd "path".
```

You can either organize all your data into one directory on your computer, declare that as your current working directory, and load/save files using simply the file name, e.g.

```
use filename
```

compared to

```
use path/filename
```

Or you can declare the path each time (not recommended). I personally recommend that you setup your paths at the top of your dofile in a series of global variables with appropriate names, e.g.:

```
global figures path/figures
global raw_data path/data
global tempfiles path/tempfolder
global tempkeep path/tempkeep
global code path/code
```

Once declared, you move between paths by calling the global variable using a dollar sign, e.g.: `cd $figures` will change the working directory to the path you declared after `global figures`. This also works when saving figures, merging datasets and calling files in any stata command, e.g.

```
graph export $figures/graph.png , as(png)
```

```
merge m:1 using $tempkeep/data_to_merge.dta , nogen keep(3)
```

Important: notice I am not using spaces in file or folder names. Using spaces is generally bad practice and can lead to errors in stata as well as other programming languages. It is recommended that you use an underscore "_" instead of a space. If you must include a space in a path then you will need to use quotes when referencing the global variable containing the path name, e.g.: `cd "$figures"` instead of `cd $figures`.

1.7 Data

Data can either be imported in STATA .dta format, or other non-proprietary formats, such as .csv. To import a .dta file

```
use filename, clear
```

whereas for text files, e.g. csv

```
import delim using path/filename.csv, clear
```

1.8 Data efficiency

When you have imported the data, you can see important information such as memory used and size in the properties window. As in all programming languages, you can store data in a range of types, and this can lead to a number of clashes. The principle data types are strings, integers, floats and bytes. Strings are flexible, for example if you want to match two data sets on an i.d variable that contains both numbers and characters, you will often use strings. However, in general they are memory inefficient. For numerical variables bytes require less memory than integers which in turn require less memory than floats. This has to do with the amount of information you can store in each data type: a byte variable can only contain numbers from -127 - 127. Likewise, strings become increasingly inefficient as the length of the longest string in the variable increases. To set each variable to the most memory efficient type, use

```
compress
```

1.9 Managing strings

To change a variable from numeric to string use

```
tostring varname, replace
```

and vice-versa

```
destring varname, replace
```

Sometimes, you will have non-numerical and numerical values in a string variables, this could happen if, for instance, the data codes missing values as "n.a." and is otherwise a numerical value. To make stata treat non-numerical values as missing observations you can use the option `force` together with `destring`.

To modify string variables the commands `subinstr` and `substr` are extremely useful. Suppose you want to replace spaces in a string with underscores, you can do this in the following way:

```
replace mystring = subinstr(mystring, " ", "_",.)
```

The `subinstr` command takes four arguments, the first is the original stirng variable name, the second is the substring you want to replace, the third is the string you want to replace it with and lastly you can specify which instance of that string you want to replace or a dot to replace all occurrences. If you want to extract the first n characters from a string you can do it with the `substr` command as follows:

```
gen first_two_letters = substr(mystring, 1, 2)
```

The first argument as before is the string variable you want to extract a substring from, the second is the first character you want to start at and the third argument is the length of the substring you want to extract. This command is particularly useful when dealing with dates, e.g. "18/02/2024" in a scenario where you want to extract the month you would use `gen month = substr(date,4,2)`.

1.10 Missing data

STATA stores missing data as a full stop symbol . , in numeric data. If the variable is a string then it is simply an empty quotes "". In practical terms, for numeric variables, STATA stores missing variables as very large numbers. Be aware of this when using logical conditions on variables with missing data! E.g. if you use

```
list varname if varname > 1000
```

and `varname` includes missing data, these will also be listed here.

1.11 Explore the data

You can get a snapshot of all, or individual variables using

```
codebook varname
```

This will give you examples, data type, number missing and number of unique variables. To get a list of unique variables, which can be used later in loops, use

```
levelsof varname, local(list_name)
```

This command takes all the different values of the variable `varname` and stores them in a local list which you can access later.

To count and store the number of unique observations, use the user-written command from ssc.

```
    unique varname
```

This comes in very handy later!

By, sort & bysort

Sort simply orders the data by the variable specified. If this is a numeric variable, this is done in increasing order, if a string then in alphabetical order. E.g.

```
    sort v1
```

`sort` is commonly used with `by` in the single command `bysort`. For example you can calculate the mean wage by age
`bysort age:  egen mean_wage = mean(wage)`
More examples on how to use `bysort` will be provided in the following section.

1.12 Generating variables

There are two commands for generating variables `gen` and `egen.` Generate may be abbreviated by gen or even g and can be used with standard mathematical operators like (+, -, $\times$, /, x^2). In addition to a large number of functions,

- `abs(x)` absolute value of x
- `exp(x)` antilog of x
- `int(x)` or `trunc(x)` truncation to integer value
- `ln(x)`, `log(x)` natural logarithm of x
- `round(x)` rounds to the nearest integer of x
- `round(x,y)` X rounded in units of y (i.e., round(x, 0.1) rounds to one decimal place)
- `sqrt(x)` square root of x
- `runiform()` returns uniformly distributed numbers between 0 and nearly 1
- `rnormal()` returns numbers that follow a standard normal distribution
- `rnormal(`x`,`y`)` returns numbers that follow a normal distribution with a mean of x and a s.d. of x

A number of more complex possibilities have been implemented in the egen command like in the following examples:

- `egen nkids = anycount(pers1 pers2 pers3 pers4 pers5), value(1)`

➩ counts the number of variables in the list `pers1-per5` whose value is 1

- `egen v323r = rank(v323)`

➩ returns the rank of each observation (row) according to the variable `v323`

- `egen myindex = rowmean(var15 var17 var18 var20 var23)`

➩ returns the mean variables `var15 var17 var18 var20 var23` for each observation (row), ignoring missing values

- `egen nmiss = rowmiss(x1-x10 var15-var23)`

➩ returns number of missing values across all variables between x1 and `x10` (inclusive) and var15 and `var23` (inclusive) for each observation (row).

- `egen vartot = rowtotal(x1-x10 var15-var23)`

➩ returns sum of all variables between x1 and `x10` (inclusive) and var15 and `var23` (inclusive) for each observation (row), ignoring missing values.

- `egen incomst = std(income)`

➩ returns the standard deviation of the variable income. Unless used together with `by` after `sort` or, more directly, `bysort`; this produces one value for the entire dataset which is the standard deviation of `income` in the set of all observations. If used with `bysort` this produces one value for each group identified by the variables following `bysort`, e.g.:

```
bysort age:  egen incomst = std(income)
```

will produce the variable `incomest` containing the standard deviation of income for each level of the variable `age`. Hence, an observation with age equal to 22, will have for the variable `incomest` the standard deviation of the set of all observations with age equal to 22.

- `egen mincome = mean(income)`

➩ Same as above but for the mean instead of the standard deviation.

1.13 Labeling the data and the variables

1.13.1 Value labels

When you have discrete numerical variable (e.g. gender = 0,1) you can assign value labels to the values in this variable, i.e. to assign the label "female" to the value 1 and the label "male" to the value 0. This is helpful in many cases, especially in increase the readability of the dataset and facilitating graphing. You can do this as follows:

```
sysuse auto, clear

label define my_labels 0 "domestic" 1 "foreign"
```

label values foreign my_labels

This snippet of code will load the auto dataset built into stata and label the values of the binary variable `foreign`, equal to 0 when a car is domestic and 1 when the car is foreign, with the relevant information.

1.13.2 Variable labels

Labeling your variables is extremely important, especially when dealing with a dataset with many variables where variable names are not especially informative (e.g. v11, v12, v13...). You can label a variable as follows:

```
label variable foreign "this variable tells me if the car is foreign"
```

This way in the variable window, in the main STATA interface you will be able to see the variable label next to the variable name.

1.14 Logical conditions, loops and lists

1.14.1 Basic syntax using logical operators

Like other programming languages, STATA understands basic logical syntax. This means you can use `if` statements `for` and `while` loops and the basic logical operators `>,<,!=,==,>=,<=,&,|`, which denote, respectively, larger than, smaller than, not equal, equal to, larger or equal and smaller or equal, and, or. For example if you wanted to create a variable in the `auto` dataset equal to 1 if the car is foreign and its price is larger than 2000 dollars, you can do this as follows:

```
sysuse auto

gen foreign_expensive = (price >2000) & (foreign==1)
```

Notice how, similarly to when plotting multiple graphs we use brackets to enclose each logical statement. The double equality in STATA, as in many other programming languages, refers to the evaluation of a condition, whereas a single equal sign refers to the assignment of a value. You could have achieved the same result by using an `if` statement in the following way:

```
gen foreign_expensive = 1

replace foreign_expensive = 0 if foreign==0 | price<2000
```

We generate the variable setting it to take value 1 for all observations, and then set all those which do not satisfy our conditions to 0. Although both ways of proceeding create the same result, the first method is more concise and easy to read.

1.14.2 Loops and macros

Before we move onto loops, it is necessary to introduce the concept of a macro. A local or global macro is a placeholder containing a string which STATA can store for you to access or manipulate at a later stage. We already encountered global macros when setting up our working directory in section 1.6, when we used short names to store our folder paths. A local macro is the same, except STATA will only store the information until the end of the execution of the dofile you are running, i.e. when the code stops running the macro will be deleted. This has several advantages, including that global macros can sometimes lead to bugs if named in ways which clash with existing functions in `Stata`'s source code. This is why I would recommend using globals only for path names, where being able to refer to the path without having to execute the entire script is particularly useful, and local macros for everything else. A popular way of using local macros is to create lists, these could be manually made:

```
local my_list "1 2 3"
```

Or contain estimation results such as, by far the most common way of using local macros, the unique values of each variable:

```
sysuse auto

levelsof foreign, local(foreign_list)

di foreign_list
```

The code above will display ”0 1” the different values contained in the variable foreign sorted by smallest to largest. If you would like to access the ith element of a list, akin to indexing in python, you can do this in stata as follows:

```
levelsof foreign, local(my_list)

local second_element:  word 2 of `my_list'

di `second_element'
```

This will print out ”1”, the second element of the list. Notice that contrary to other programming languages, STATA works with 1 indexing (as opposed ot 0 indexing), i.e. the first element of the list is the one indexed by the number 1. Also, notice how we use the backtick followed by an apostrophe to refer to a local macro within the loop, if the list were a global macro, then we would use the dollar sign.

The following is another example of code this time combining a local list with a `foreach` loop:

```
levelsof foreign, local(foreign_list)

foreach level of local foreign_levels {
summarize mpg if foreign == `level'
}
```

Loops are often not strictly necessary when coding, but can significantly clean up your code and increase readability facilitating debugging. Other common types of loops include the `forvalues` loop (akin to `for i in range(0,n)` in python) and the while loop. The basic syntax of these loops is:

```
forvalues i = 1(2)6{
di "`i'"
}
```

Which will print out "1, 3, 5", as the 2 in between the 1 and the six in the first line refers to the step size. The while loop is as follows:

```
scalar j = 1


while j<5{
scalar j = j +1
di j
}
```

Which will print out "2 3 4 5".

1.14.3 Lists of variables

Another type of local macro is a variable list or `varlist`. Sometimes you will want to rename a subset of the variables in your dataset, or execute another command for a set of variables. You can do this using a loop and a local macro as shown above, but it may be cumbersome to write down the variable names of all the variables you need to iterate over. Luckily STATA is smart enough to understand variable list syntax:

```
sysuse auto , clear

foreach var of varlist make-headroom{
sum `var' }
```

Will summarize all the variables between `make` and `price`, inclusive, in the order in which they currently appear in the variable window in the main STATA interface.

1.15 More advanced data manipulations

The following are very commonly used methods used to clean and transform data into the format required by your analysis.

1.15.1 Collapse, Preserve and Restore

Collapse converts the dataset in memory into a dataset of means, sums, medians, etc. The basic syntax of the `collapse` command is:

```
collapse (mean) var1_mean = var1 (sd) var1_sd = var1 [if] [in] [weight],
by(group1 group2)
```

In brackets you declare the type of summary statistic you want, in the options within the `by` you declare which varibles uniquely identify your samples. For example:

```
collapse (mean) income_mean = income (sd) income_sd = income , by(gender)
```

Will produce a dataset with as many observations as the different values contained in the variable gender and the variables `income_mean` and `income_sd` containing the mean and standard deviation of income for each gender group. This effectively generates a dataset of summary statistics. It is important to mention that STATA does not have a CNTRL + Z functionality that you would have in Excel. Hence, once you apply `collapse` it is impossible to return to the previous dataset (i.e. to "expand"). Instead, you can use the commands `preserve` and `restore`, these commands will allow you to move between instances of your dataset. For example when running:

```
preserve

collapse (mean) income_mean = income (sd) income_sd = income , by(gender)

save $data/summary_stats.dta , replace

restore
```

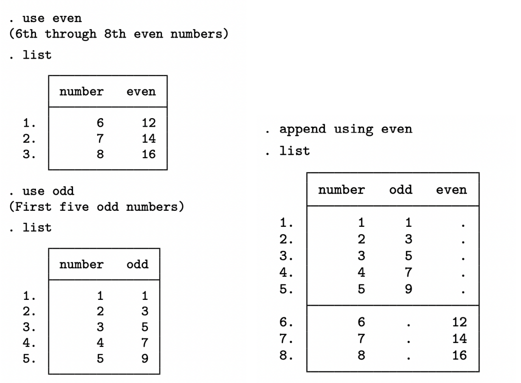
This will generate the summary statistics, save them to your `$data` folder and then revert back to your original dataset.

1.15.2 Merge, append & joinby

Very often, you will have different datasets that you want to combine into one. This is done using one of either `merge` or `append`. The difference between the two is that `append` is to combine datasets with the same variables in them, such as different waves of the same sample where each observation a single respondent identified by a unique id. For instance:

```
append using file1 file2
```

Will combine file1 with file2. If file1 is already in memory simply type `append using file2`.



```
. use even
(6th through 8th even numbers)
. list

     +---------------+
     | number   even |
     |---------------|
  1. |      6     12 |
  2. |      7     14 |
  3. |      8     16 |
     +---------------+

. use odd
(First five odd numbers)
. list

     +--------------+
     | number   odd |
     |--------------|
  1. |      1     1 |
  2. |      2     3 |
  3. |      3     5 |
  4. |      4     7 |
  5. |      5     9 |
     +--------------+
```

```
. append using even
. list

     +---------------------+
     | number   odd   even |
     |---------------------|
  1. |      1     1      . |
  2. |      2     3      . |
  3. |      3     5      . |
  4. |      4     7      . |
  5. |      5     9      . |
     |---------------------|
  6. |      6     .     12 |
  7. |      7     .     14 |
  8. |      8     .     16 |
     +---------------------+
```

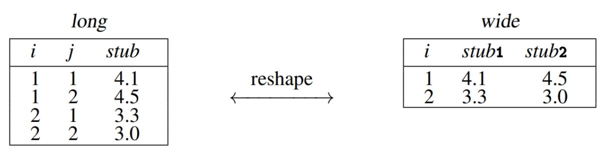
Merge on the other hand combines datasets which contain a common identifier, this is helpful when adding more variables to the dataset. There are three ways in which to merge data depending on whether the common identifier is unique or not in both the using and master data. The master data is that which is loaded into memory, and the using is the one to be merged in (these names apply to a wide range of other functions with the same meaning).

- `1:1` : one-to-one
- `m:1` : many - to - one
- `1:m` : one - to - many

An example of a `1:1` merge could be having a dataset of patient characteristics at a hospital (e.g. diagnosis, test results etc.) each identified by their social security number. If you had socio-economic data on these patients you could then merge that in by typing:

```
merge 1:1 social_security using $data/ses_data.dta ,
nogen keep(3) keeupusing(income)
```

Let's break this down. The `1:1` tells stata that for each row (observation) in the master dataset (the dataset you are currently using in memory) to match only one observation in the using dataset `ses_data` in the `$data` folder. The option `nogen` suppresses the automatic creation of a variable named `_merge` which stata automatically generates after a merge. This variable contains three values: 1 if the observation is in the master data but not in the using, 2 if the observation is in the using data but not the master and 3 if the observation is in both datasets (a successful match). This ensures that by default stata keeps all information in both datasets after a merge. For instance, if there are more people in the `ses_data` who are not patients at the hospital these will be in the new dataset after the merge with missing values for all the hospital related variables. If you only want to keep observtions which were successfully matched (their `social_security` id appears in both datasets) then specify the option `keep(3)`. The option `keepusing(varlist)` tells STATA which variables you want to keep from the using dataset (the one you are merging in).



i	*j*	*stub*
1	1	4.1
1	2	4.5
2	1	3.3
2	2	3.0

i	*stub*1	*stub*2
1	4.1	4.5
2	3.3	3.0

Figure 2: Reshape.

1.15.3 Reshape

Reshape is a powerful tool used to change the structure of your data. There are two types of reshapes which can be performed using the following commands:

- `rehsape long varlist, i(id) j(newvar)`
- `rehsape wide varlist, i(id) j(stubvar)`

What these commands do can be summarized in Figure 2: This converts a datasets between ”wide” format where variables are named in a sequence, e.g. gdp1960, gdp1961, gdp1962... to datasets in ”long” format where each of the variables in the sequence are stacked on top of each other. Data is typically easier to manipulate when it is in long format although this may significantly increase the number of observations which is why in most cases data will be in wide format when you access it (this is the case for all World Bank and IMF data). It is important that the variables in the `i()` suboption should uniquely identify observations when moving from long to wide and `i()` and `j()` should uniquely identify a cell when moving from wide to long. Typically errors using reshape are caused by having missing values in the identifier in the `i()` suboption.

1.16 Duplicate data and _n & _N

Duplicate data can be a large source of frustration in `Stata` since it can take a while for the program to identify this error when using large datasets. Be sure to check for duplicates using:

- `duplicates report`

➪ reports duplicaes in terms of all variables

- `duplicates tag, gen(duplicates)`

➪ same as above but generates a variable telling you which observations are duplicates

- `duplicates drop`

➪ drops all duplicates

- `duplicates drop var1 var2, force`

➪ drops observations which have the same values for var1 and var2, keeping only the first instance.

These commands are the most efficient way of dealing with duplicates, however to have more control over which observations are dropped it is sometimes helpful to combine other stata commands for this task. For example, suppose you have a dataset of regional statistics for multiple countries over many years, e.g. unemployment in counties in the US and the UK. Then the command:

```
bysort country county year:  gen dup = _n
```

Will create a variable `dup` which will have value 1 if the observation is the first instance in the dataset of a row having a particular value for the variables `country county` and `year`. If you wanted to create a variable containing the number of observations for each county you could do that using:

```
bysort country county year:  gen dup = _N
```

In stata language `_N` refers to the total number of observations (within a group when used together with a `bysort`) whereas `_n` refers to the observation number from 1 to `_N` in the current sort order (also within a group when used together with a `bysort`).

Practice

Please work through the following tasks, at your own pace. We will go over the answers throughout the class. Feel free to ask me for help, and remember that coding requires plenty of google and forum searching!

1. Load in the PatentMarketValue dataset provided.
   (a) How many unique patent numbers are there?
   (b) What steps can you take to improve data memory efficiency?
   (c) What is the most efficient data size you can achieve?
2. Are there any duplicates, if so, how many?
   (a) Remove the duplicates
3. What is the earliest year a patent in issued in?
   (a) Generate a binary variable which indicates whether that patent was published in the first year observed.
   (b) Tabulate this new binary variable, what percentage of patents were published in the earliest year in the dataset?
4. Merge in the PatentFirmNames and keep only the matched observations!
   (a) What is the maximum amount of data memory savings you can make?
5. Generate a variable which counts mean market value per firm

6. Plot these on a histogram with percent on the y axis - critique this graph
7. What is the mean and median number? Is there right or left skewness?
   (a) Save this dataset for later use.
8. Load the previously saved firm panel data set.
   (a) Build a firm x year panel with two additional variables - the yearly patent count and average market value of their patents (four variables in total)
   (b) Is the panel balanced? If it is unbalanced, what percentage of firms appear for the entire sample?
   (c) Generate a variable recording the Yearly difference patent count
   (d) Which five firms saw the largest change in their patent counts over the two years?
9. Load the dataset you saved previously in question 7. Drop the variable permno. Create a local list of of the following strings "Patent_number real_value nominal_value citations issuing_date filing_date", do not forget to use the underscores instead of spaces to separate words, as spaces separate *elements* here. For each variable between `patent_number` and `filing_date` apply the corresponding label in the local list you created above.

2 Data Analysis

In this section we will go over the main types of statistical analysis you can perform with STATA. We will cover:

1. Summarising the data
   (a) Calculating statistics
   (b) Collapse
   (c) Tabulate
   (d) T-test
2. One variable graphs
3. Two variable graphs
4. Summary, fitted & estimate result graphing
5. Designing and formatting graphs

We will be using the `ExecData.dta` data.

2.1 Summarizing the Data

There are a selection of basic commands which you can use to get a picture of the data you are working with. To get a technical overview we have already used `describe`, however

```
summarize varname, detail
```

is a common command which gives a range of moments and summary statistics for your dataset. The option `detail` is optional, and includes quartiles and median values alongside the mean, min and max. For categorical variables use

```
tabulate varname
```

to get the frequency counts or percentages of discrete variables type:

```
tabulate ,
```

to include the cross-variable counts and percentage values. When calling variables you can use the asterisk $*$ symbol to match all variables. However, STATA is even smarter and the asterisk can be used in combination with partial variable names to match a subset of variables which all start with the same prefix. E.g. the variables consumption_child, consumption_adult, consumption_retired, can all be called with consumption$*$. For example, to find the correlation between all variables in the data set use,

```
correlate *
```

As we discussed in section 1.15.1, to produce a set of summary statistics you can use the command `collapse`. The problem here is that this replaces the data in memory with the summary statistics, so be aware that you cannot go back unless you used the `preserve` and `restore` commands as discussed in section 1.15.1.

2.2 Graphing

When using STATA's graphing library there are a miriad of visualization options, which allow you to change everything from fonts, colors and overall plot styles to match your personal preferences. In this handout, we will go over some of the more popular ones, although the reader is referred to the STATA webpage or the stata help manual for a more extensive overview of these options.

2.2.1 Schemes

STATA's graphing library has just about every possible customization option available, allowing you to fully control the look of your figures. To make things easier for the user, there are several "schemes" which affect the overall style of plots which come preinstalled in stata. You can select the one which you prefer by typing at the top of your dofile:

```
set scheme [schemename]
```

A popular scheme is the "economist" which will make your figures consistent with the template from the popular magazine by the same name. I personally like the "cleanplots" scheme which does not come installed in STATA but can be downloaded by typing in the command prompt:

```
ssc install scheme-cleanplots.scheme
```

2.2.2 Continuous variables: Histograms and kernel densities

A histogram is a representation of the distribution function of a random variable. These types of graphs are extremely useful as they give a summary of the entire distribution of a variable, as opposed to its sample moments which we can obtain from a summary statistics table using `summarize varname , d`. It is a good idea when working with a new dataset to use one of `hist` or `kdensity` commands to see what the distribution of a variable looks like. This will inform us about its skewness, potential for outliers or unrecoded variables (e.g. some data sources code missing values as a large number like 999, this will show up in a histogram). To do this use:

```
hist varname, [options]
```

The y axis can be formatted as either a density, frequency or percentage in the options simply by typing one of these terms after the comma. Histograms are useful tools, but can be misleading when dealing with continuous variables. By grouping the variable into "bins" a histogram will mistakenly give the idea that within each group different realizations of a variable are equally likely. To avoid this we can plot a kernel density using the STATA command `kdensity`:

```
kdensity income, kernel(gaussian) lwidth(thick) lcolor(blue)
```

Will show an estimate of the density function of the variable "income". The theory behind the consistency of kernel density estimates of the density function is beyond the scope of this course, see the Statistical Appendix A if you are interested in how these functional estimates are calculated.

2.2.3 Discrete variables: Bar charts

For discrete variables we will produce histograms, except instead of discretizing a continuous variable by grouping it into bins we simply count how many observations fall into the different categories of the discrete variable. We can do this with:

```
graph bar (count) , over(catvar)
```

Which will produce the count of observations in each category of `catvar`.

2.3 Two variables

2.3.1 Discrete-Discrete

You can create a bi-variate count graph by specifying two categorical variables as in

- `graph bar (percent), over(`*catvar*$_1$`) over(catvar`$_2$`)`

This is a good moment to discuss how the order in which you run STATA commands matters! STATA runs from left to right, as written in the code. For example the command

```
graph bar (percent), over(catvar2) over((catvar1)
```

will produce a different graph. It will first split over *`catvar`*$_2$, then split each category across *`catvar`*$_1$.

2.3.2 Discrete-Continuous

The structure changes very little when comparing discrete to continuous variables. For example,

```
gr bar (mean) y, over(x)
```

This will show us the mean of the variable y for each category of the variable x. We could have chosen another statistic instead of the mean such as the standard deviation (sd) or the median. There are also a large number of other potential graphs other than bar charts, to choose from, depending on what you want to convey. For instance:

```
gr hbox y, over(catvar1) by((catvar2))
```

Will produce n different plots of the ineterquartile ranges of the variable y for each category of *catvar*$_1$, one plot for each level of the variable *catvar*$_2$.

2.3.3 Continuous-Continuous

When relating one continous variable to another, we use the stata `twoway` command and all of its sub commands to produce different types of graphics. For a full list of the plots you can produce with `twoway` check out the stata manual. A common plot to visualize a relationship between variables is the scatter plot, which we can produce in STATA using:

```
scatter y x
```

to layer numerous graphs on top of each other you can use || or brackets to separate each plot, e.g.:

```
twoway scatter y x || lfit y x

twoway (scatter y x) (lfit y x)
```

Will both produce scatter plots of y against x and overlay a fitted regression line. Some plot types are specific to specific types of data, for instance if you had time-series data such as the stock price of a company over time you would type:

```
tsset timevar

tsline stock_price
```

To plot the stock price over time. Alternatively, you could use:

```
line stock_price timevar
```

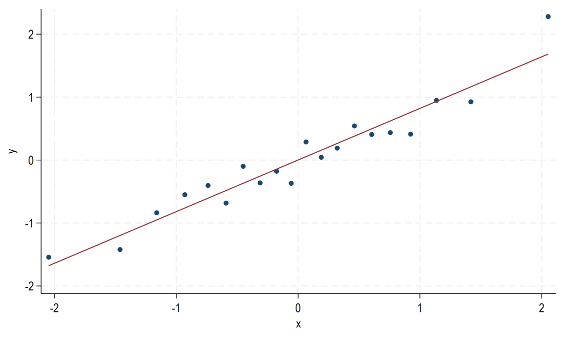
Although be sure that the data is first sorted by `timevar`, (check what happens if this is not the case).

2.3.4 Binscatter

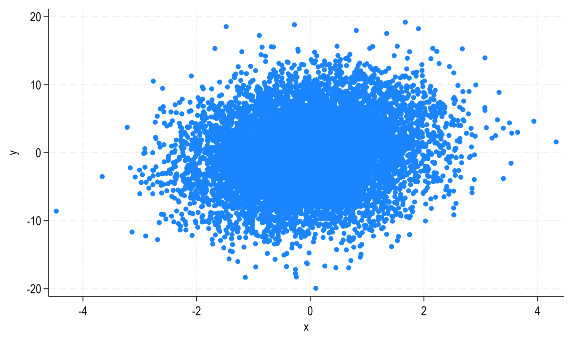
We often deal with datasets with many observations. This can lead to very messy scatterplots which look like a cloud of points. Because the points are all on top of each other when visualizing the data in a regular scatter plot, it becomes very difficult to see if there is a clear correlation between the variables. For instance, consider the following example:

```
clear all

set obs 10000

gen x = rnormal(0,1)

gen eps = rnormal(0,5)

gen y = 0.8 * x + eps

scatter y x

binscatter y x
```

The regular scatter plot is too crowded to interpret. This is notwithstanding we know there is an actual relationship between the variables as we created y to be a function of x. Binscatter reduces the dimensionality of the plot by grouping the x axis into a set number of bins (optimally chosen by the algorithm) and then taking the average value of y and the average value of x within that group. This drastically reduces the number of points on the graph enabling a clear visualization of the relationship between the two variables.



(a) Binscatter



(b) Regular scatter plot

Figure 3: The importance of binscatter.

2.3.5 Fitted lines

`binscatter` plots a regression line through the data, this can be done manually using the command

```
lfit y x
```

you can also plot quadratic relationships using

```
sysuse auto, clear

tw (qfit price mpg) (qfitci price mpg)
```

For more complex nonlinear relationships you should consider using non-parametric methods such as kernel weighted polynomials or splines. These methods are beyond the scope of this course, however for more information readers are referred to chapter 11 of Hansen (2022).

2.3.6 Regression results

Instead of reporting regression estimates in a table, it is becoming increasingly common to plot the coefficients. This is a far more intuitive data presentation method, and allows for easy comparison across dependent variables. To do so, use the user-written command `coefplot`. For instance, consider the following example:

```
sysuse auto, clear

set scheme cleanplots

reg price mpg headroom trunk

coefplot, drop(_cons) vertical yline(0) coeflabels(trunk= "trunk" headroom=
"headroom" mpg ="Miles per gallon") ciopts(recast(rcap))
```

This will output the following figure. Try changing the options to see how it affects the look of the plot. Notice this graph shows us not only the point estimates from our regression table which you can see in the results window, but also the 95% confidence intervals.

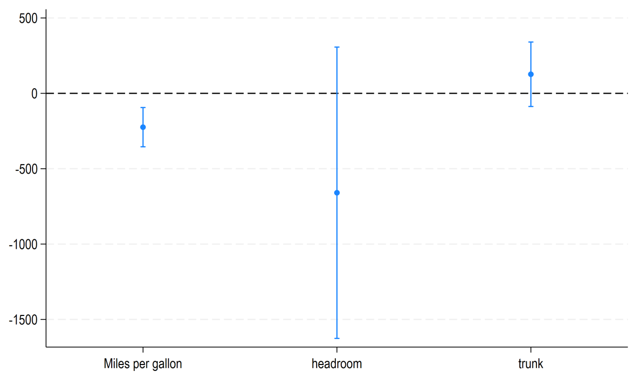
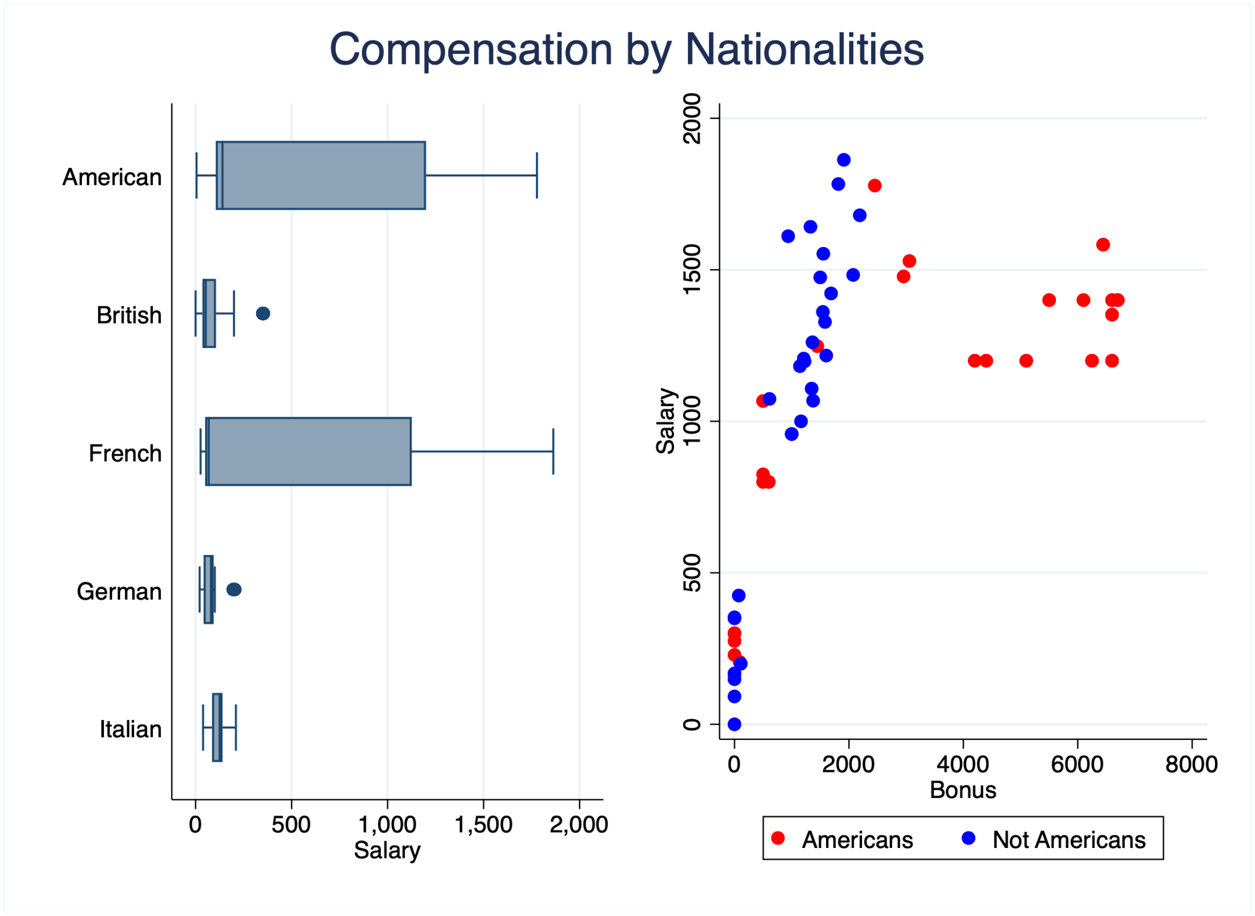


Figure 4: Coefplot

Practice

1. Load the `ExecutiveDatabase` data
2. Merge in the DirectorID data only keeping the matched observations using the `DirectorIDs` data. *Hint: Use the option keep(match)*
3. Efficiently store your data
4. Create a variable of Director Age in each year. Age = year - birth year. First create a year variable, for each observation using the variable `AnnualReportDate`. Hint: This is already stored as a Stata date type, so use the function year(). Plot this as a histogram.
5. Save your data for later use.
6. In a figure, summarize the distribution of the salary of men and women. *Hint: for this you can use a kdensity or a histogram, but make sure to have both distributions in one plot region!*
7. Replicate the following figure. Hint: drop observations from other countries before creating the graph. For the scatter, combine two seperate scatter graphs, one condition if `Nationality` == American, and one if `Nationality` != American.
   (a) Export the graph into a format you can easily open on your pc such as pdf or png and comment the results.



8. Make a scatter plot of total salary income against the value of equity held in the firm with different colors for each nationality (one scatterplot with the color of the dots corresponding to each nationality). Critique this graph? Is there anything you can do to the variables to make this relationship easier to interpret?

9. Apply a transformation to the data and repeat Q8.

3 Modeling

In this section we will go over the main statistical models you can implement in Stata. We will not delve into the theory which is required to interpret the results from these models, should you want to refresh your memory consider reading the Statistical Appendix A. Specifically, this section will cover:

1. Estimating regression models
   (a) Options
   (b) Weights
   (c) Factor variables
2. Instrumental variable regressions
3. Non-linear regression models
   (a) Logit & Probit models
4. Fixed effects
5. Exporting results (estout)
6. Model selection and goodness of fit

3.1 Standard regression model

Remember the standard STATA command structure is

```
[by varlist1:]  command [varlist2] [=exp] [if exp] [in range] [weight] [using
filename] [,options]
```

To run a simple linear regression model of the form

$$y_i = \beta_0 + \sum_{j=1}^{K} \beta_j x_x + \epsilon_i \tag{1}$$

use

```
reg y x_1 x_2 ...  x_K
```

STATA provides a range of options and post estimation commands. We will base these for now on the standard linear regression model, before moving onto more advanced models, such as IV regression and discrete choice. The most important options you should be aware of when running regressions in Stata concern the standard errors. By default, Stata computes standard errors which are **not** robust to clustering or heteroskedasticity. Formally, this means that in 1 we are assuming that $\epsilon_i \sim i.i.d(0, \sigma)$ for all i. Any violation of this assumption will lead to biased standard errors which may lead us to incorrectly reject our hypothesis. It is generally best practice to always make standard errors robust to heteroskedasticity as it is rarely the case that the errors are homoskedastic and White standard errors - standard errors which are robust to heteroskedasticity - are consistent even when the data generating process is homoskedastic, hence we have little to lose and it is good practice to specify this option by typing `r` after the comma following the `regress` command. Should you have further reason to believe that the errors are correlated across groups of observations you can relax the independence assumption by allowing for group level correlation by typing `vce(cluster id)` where id is your group identifier. There are several statistical tests for heteroskedasticity and group correlation of the standard errors which you can use to determine whether it is necessary to apply these options. However, as a general rule, relaxing assumptions leads to efficiency losses (standard errors which may be slightly larger) but consistent results (as the number of observations grows, the standard error will shrink and the point estimate will converge on the true value). Hence, when in doubt it is always best to be conservative by applying the method which requires the least amount of assumptions, e.g. allow the standard errors to be dependent over a group id and exhibit heteroskedasticity.

3.2 Factor variables

Variables can either be discrete or continuous. For example, someone's height is continuous while their sex is discrete. This leads to different mathematical properties and how the β coefficient should be interpreted. STATA handles this by using a range of prefixes. As we saw in the previous section, to estimate a regression model with continous variables simply type `regress y x`. However, if x is discrete, use the following

```
regress y i.x
```

Where the `i.` prefix tells STATA that the variable is discrete. If `x` is binary (it has only two distinct values), the coefficient reported in the regression table will estimate the following. Suppose y is income and x is a dummy variable equal to 1 if the respondent is female or 0 otherwise, then when we estimate the model:

$$y_i = \beta_0 + \beta_f x_i + \epsilon_i \tag{2}$$

Then our estimate of β_f, which we will call $\hat{\beta}_f$, will be an estimate of:

$$\mathbb{E}[y|x = 1] - \mathbb{E}[y|x = 0]$$

The average difference in earnings between females and non-females. To see why this is the case consider 2 and take the expectation on both sides. As the expectation of ϵ_i is equal to 0 by assumption, it follows that when $x = 0$ - the respondent is non-female - then the expectation of income is β_0. However, when the respondent is female $x = 1$ meaning that the expectation of income is $\beta_0 + \beta_f$ so the expected difference between earnings of females and non-females is β_f, which we will consistently estimate by OLS minimizing the distance between the data and our model.[1]

[1]See the Statistical Appendix A, this will hold when the classical regression model assumptions are valid.

If `x` is categorical with j options, then the regression will estimate $j-1$ coefficients, each as the difference from the base category. Consider the following example where I generate some fake earnings at the daily level data for one year.

```
clear all

set obs 365

gen day_of_year = _n

gen earnings = 100 + rnormal(0, 10)

gen date = mdy(1, 1, 2023) + (day_of_year - 1)

format date %td

gen month = month(date)

reg earnings ib(1).month, r
```

In this model the base category will be January, assuming our variable `month` contains the number of each month. Then the coefficients will be interpreted as the average deviation of earnings in each month compared to January. As the data is randomly generated, we should expect all coefficients to be close to zero.

3.3 Interactions

Interactions are common in regression models to study discontinuities or compound effects. To interact two variables x_1 and x_2 use the #. Remember to declare the type of variable correctly, as either continuous or discrete. STATA defaults to discrete when using the #.

- *Discrete-Discrete Interaction:*
  `regress y i.x_1#i.x_2`
- *Discrete-Continuous Interaction:*
  `regress y i.x_1#c.x_2`
- *Continuous-Continuous Interaction:*
  `regress y c.x_1#c.x_2`

If you are unsure what an interaction effect is, this is the same as regressing `y` on a variable equal to the product of the two interacted variables. This could be interesting if for instance we wanted to see how experience affects earnings of women compared to men, then we would run a regression of the following form:

```
regress y i.age i.age#i.female
```

The first set of dummy variables will show the age profile of the variable `y`, showing how earnings vary over age with respect to a base level of age. The second set of dummies will show how this age profile of the variable `y` is different for females compared to non-females, i.e. observations where the value of `female` is 1.

3.4 Weights

Weighted Least Squares controls for the different proportion of certain groups within your data. When using samples which are taken as representative, conditional on the weights, then including them is essential for unbiased estimates. Otherwise, they are a form of cluster analysis which helps you to correctly estimate the standard errors. Include weights using

```
reg y x [pweight=weights], r
```

With large samples population weights are rarely needed as the sample itself is likely to be representative (i.e. the fraction of people with certain characteristics in the sample resembles the fraction in the population). However, with small samples in surveys where non-response is an issue, weights can affect the results substantially.

3.5 Presenting Tables

There are two leading packages for producing publication ready tables, `estout` and `out2reg2` (An update on `out2reg`). We will discuss `estout` here, however check out `out2reg2` in your own time, since the concept is the same and the choice is often down to personal preference. `estout` separates estimating models and reporting estimates by using `eststo` to first estimate and store results. This uses the following structure

```
eststo column : command
```

for example:

```
sysuse auto, clear

eststo c1 : reg price mpg

eststo c2 : reg price displacement

esttab c1 c2
```

This will print out a table with the regression results similar to those you see in published papers. This package has a variety of options which you can use to format the table, specifying for instance which statistics you would like to display or any additional information such as whether you included fixed effects, clustered the standard errors or the like. By default, `esttab` uses the variable name, which due to STATA formatting is often not appropriate for reporting results. To change the names to something easier to read use

```
label variable varname "Name to print"
```

then use the option `label`, to print these names instead. `esttab` supports exporting tables to many formats including but not limited to text, LaTex, HTML, word, excel etc. This is done by specifying the target file name under `using`, the type of file is specified by choosing the appropriate file extension e.g.
`eststo column using "path/filename.extension": command`
As with the graphs, there are now a multitude of options which you can set to format the design of the table. Refer to the full documentation as you go and see the code for this session to get an example of how to produce a regression table with the most popular options.

3.6 Advanced Regression Methods

So far we have only looked at basic linear regressions. Clearly STATA is capable of running far more complex models, however the syntax is structured the same as for `regress`, therefore once you have grasped how to manipulate that command, the others follow on easily.

3.6.1 Discrete Choice

Logit and probit models are very common methods of modelling discrete data, for example cases in which the independent variable is binary. E.g. it records whether an agent is employed, or not.

```
logit y x1 x2

probit y x1 x2
```

Since these are non-linear models, the interpretation of the coefficients requires care and there are various transformations which can be made. For example, a common choice is to report them as odds ratios. You will cover this in more detail in your econometrics courses. While these models are similar, their key differences rest in the distributional assumptions concerning the error term. For applied researchers, an important consideration is also that `logit` is significantly faster to compute when using large datasets.

3.7 Model selection and goodness of fit

A useful statistic when evaluating the fit of our model is the R-squared, which is the proportion of variation of the dependent variable (y) which is explained by our independent variables:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y}_i)^2}$$

Where $\hat{y}_i$ is our model prediction for the observation i:

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{1,i} + ... + \hat{\beta}_K x_{K,i}$$

The parameters $\hat{\beta}$ denote our least square estimates for the true population parameters β, i.e. those that minimize the residual variation (or maximize the R-squared) for our particular sample. For more information on OLS parameter estimation, readers are invited to consult the Statistical Appendix. Models with a higher R-squared, will therefore explain a larger fraction of the variation in the dependent variable and are, generally, to be preferred. However, due to the nature of OLS model estimation, you will have noticed that adding explanatory variables will *always* weakly increase the R-squared as the β parameters on the additional variables are chosen explicitly to maximize this quantity and setting these coefficients to zero is a valid option, hence the R-squared when adding an additional set of variables will have to be at least as large as the R-squared of the model without these variables. This is why when choosing among different models, we generally prefer the Adjusted R-squared which adds a penalty for additional variables to the formula:

$$\bar{R}^2 = 1 - \left(1 - R^2\right) \frac{n-1}{n-k-1}$$

Another reason to be cautious about choosing models which maximize the R-squared concerns the potential for capturing spurious correlations. When the R-squared is particularly high (above 80%) this should be a warning sign and not a reason to choose that particular model. The reason is that given the complexity of true data generating processes it is unreasonable to claim that our covariates (explanatory variables) explain such a high fraction of the variation in the dependent variable. If, for instance, we were regressing GDP per capita on the fraction of individuals receiving primary schooling in a panel of countries, and found an R-squared above 80%, we would mistaken in claiming that education alone explains 80% of cross country differences in income per capita. The reason for such a high R-squared is that education is likely correlated with many other factors which explain GDP per capita such as the quality of infrastructure, institutions, trade barriers, etc. Hence, while a high R-squared is generally a good sign, when it becomes *too high* this is a warning sign that our model could be misspecified and claiming that our explanatory variables alone can explain such a high fraction of the variation in the dependent variable is something we should have very good reason for. In most cases, this will depend on the particular model, type of data and other issues affecting the empirical analysis.

Practice

Your objective is to first replicate this table

1. Load the Compustat Data
2. Generate a numeric gvkey and destring the SIC industry identifier
3. Drop all variables where `drop if indfmt == "FS"`
4. Keep all observations between 1996 and 2006
5. Generate the following firm × year level variables using the definitions. *Hint: in brackets you have the variable names as they appear in Stata*
   (a) R&D = xrd/at
   (b) Firm Size = ln(at)
   (c) Cash = che / at
   (d) Profitability = oibdp / seq
   (e) Capital Expenditure = capx / at
   (f) Market Capitalisation = prcc_f * csho
6. Label the new variables to match the table
7. Regress R&D on the independent variables listed in the table and export this table.

Table 1: Regression table

	(1)
	R&D
Firm Size	-0.158***
	(-4.43)
Cash	0.369
	(1.90)
Profitability	0.000217
	(0.41)
CapEx	7.312*
	(2.34)
Market Cap	0.00000581**
	(2.90)
Constant	0.428
	(1.54)
Observations	45496

t statistics in parentheses

* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

8. Save your data as a back-up
9. Merge in the firm $\times$ year measure of common ownership
10. Regress the common ownership measure on sales, controlling for the relevant firm characteristics. What is the sign of that coefficient, is it significant?

4 Post-Estimation

Sometimes, you will need to store results after running a command. For instance, using the `auto` dataset if you wanted to create a variable taking value 1 if a car has a higher than median price and 0 otherwise, so would do this as follows:

```
sysuse auto

sum price, d

scalar median_price = r(p50)

gen high_price = price > median_price
```

The scalar command creates a placeholder, like a local or global macro but specifically for a number. We then assign this placeholder to be equal to the median of price, which we obtain using the estimation result `r(p50)`. There are many other post-estimation tools which you can use for data analysis. In this section we will cover the following:

1. Post-regression analysis
   (a) Predict & residuals
   (b) Hypothesis testing
2. Estimating and plotting marginal effects
3. Accessing and storing estimation results
   (a) e-class
   (b) r-class

4.1 Standard regression model

Directly after estimating a model, there are a number of hypothesis tests which you can run to test how well you model performs or whether it meets the underlying assumptions. To be clear, these are run directly after estimating a command. If you estimate a model then generate a variable, you will not be able to run these commands. More generally, all post-estimation commands need to be run directly after the command, i.e. when referring to a result STATA will look for results produced by the most recent command. A large number of pos-estimation commands targeted specifically at regression models are part of the `estat` package. For example, you can test for the presence of heteroskedasticity[2] with

```
reg y x

estat hetest
```

[2] Check out the documentation for the mathematical foundations behind these tests.

Or for model miss-specification using[3]:

```
reg y x

estat ovtest
```

4.1.1 Predictions and residuals

When estimating the model

$$y = \beta x * \epsilon$$

the predicted value of y is given by

$$\hat{y} = y - \hat{\epsilon}$$

to calculate the predicted values after estimating a regression model use

```
reg y x

predict varname
```

Whereas to calculate the residuals use

```
reg y x

predict varname, res
```

These can then be used to further check model performance and underlying assumptions or for plotting purposes.

4.2 Marginal effects

One of STATAs leading strengths is the support for analysing marginal effects after estimation. This is most useful when analysing interaction effects, and you want to understand whether the marginal effect of increasing x_1 varies across observations. Consider the following three cases. The first will find the predicted value of `y` for each of the categories of `x`. `marginsplot` then plots this as a line graph.

```
 regress y catvar
margins catvar
marginsplot
```

As previously said, this is most interesting for interactions. The following example splits the previous graph over a binary variable to show the difference in marginal change.

```
reg y catvar#binaryvar
margins catvar#binaryvar
marginsplot
```

This can be extended in many ways, for example to three way interactions.

[3]Notice that the `ovtest` tests for miss-specification by checking whether additional combinations and transformations of the regressors such as squared and cubic terms can achieve a better fit than the existing model. This is not a test for omitted variable bias, i.e. if there is a confounder which is correlated with your `x` and has an effect on `y` but is unobservable this will *not* be picked up by the test.

4.3 Accessing statistics

After running a Stata command the results are stored in either r() or e(). Stata commands that perform estimation (regressions, factor analysis, anova, etc.) are e-class commands and the results are stored in e(). Other commands (summarize, correlate, etc.) are r-class commands and the results are stored in r(). After running a command you can run return list (r() objects) or ereturn list (e() objects) to see the values that were stored by the command. After running a given command use

`return list` or `ereturn list`

to see the values stored. You can access these at a later stage by saving them as scalars, locals or globals:

```
sysuse auto, clear

sum price, d

return list

scalar median_price = r(p50)

di median_price
```

This will store the median price of the automobile dataset as a scalar variable.

Practice

In this short task, we will look at whether we can identify pay discrimination amongst executives. We will use the same data from `ExecuComp` used previously.

1. Set up your working directory and load the `ExecutiveDatabase`.
2. Merge in the `DirectorIds` data and keep only matches.
3. Generate a discrete numerical variable which has as many different values as there are different roles in the variable `RoleName` and for each numerical value a corresponding value label equal to a role. *For this maybe the command `encode` would work well.*
4. Do the same for the variable `Gender` where the value labels should be ”Male” when `Gender` is equal to 0 and ”Female” when `Gender` is equal to 1.
5. Apply a log transformation to the variable Salary.
6. Recreate the table from Class4 ad add a column containing the estimation results from a model which includes a role fixed effect. Interpret the results.
7. Plot the marginal effects of Gender at different levels of tenure to see the wage growth profile of male and female executives over their careers. Do this with and without a role fixed effect and combine the two figures in one graph? Interpret the differences.

References

Brooks, C. (2019). *Introductory econometrics for finance.* Cambridge university press.
Cameron, A. (2005). Microeconometrics: Methods and applications. *Cambridge University.*
Creel, M. (2002). Graduate econometrics lecture notes.
Greene, W. H. (2003). Econometric analysis. *Pretence Hall.*
Hansen, B. (2022). *Econometrics.* Princeton University Press.
Hayashi, F. (2011). *Econometrics.* Princeton University Press.
Stock, J. H., & Watson, M. W. (2020). *Introduction to econometrics.* Pearson.

A Statistical Appendix

A.1 Further Literature

The following is intended as a concise summary of the main theoretical knowledge you will need to possess to use the statistical tools covered in these notes. For those seeking a deeper understanding of statistics and econometrics, there are several good textbooks that you should consider looking at:

- Undergraduate textbooks:
  - Brooks (2019): very good introductory textbook for financial econometrics with especially good coverage of time-series and many practical examples.
  - Stock and Watson (2020): the go-to textbook for undergrad econometrics, it is so popular for a reason. This textbook is sometimes recommended in grad school as a soft introduction to concepts such as panel data and instrumental variable regression for the ease and simplicity of its exposition.
- Graduate textbooks:
  - Greene (2003): One of the best econometrics textbooks out there. It has a wide coverage of all areas of econometrics and combines a rigourous treatment with very insightful and intuitive exposition.
  - Hansen (2022): Has more coverage of topics in which researchers have only recently taken an interest such as machine learning and non-parametric estimation.
  - Hayashi (2011): I have not used this textbook but it has a great reputation.
  - Cameron (2005): Excellent book, especially for those interested in structural microeconometrics and discrete choice models.
  - Creel (2002): The best lecture notes on graduate econometrics you can find. Very complete and concise, these lecture notes are not a substitute for a full textbook but will suffice for most researchers who need to look-up a topic and get an understanding of the fundamental concepts. You can also find these notes here.

A.2 Random Variables and Their Distributions

A **random variable** is a numerical value determined by the outcome of a random experiment. Random variables are essential in probability theory and statistics, as they provide a mathematical framework for modeling uncertainty. There are two types of random variables:

- **Discrete random variables**: These take on a finite or countably infinite set of distinct values.
- **Continuous random variables**: These take on an uncountable set of values, typically within an interval of the real number line.

Discrete Random Variables

A discrete random variable, X, is characterized by a **probability mass function (PMF)**, $P(X = x)$, which satisfies the following properties:

1. $P(X = x) \geq 0$ for all x.
2. $\sum_x P(X = x) = 1$, where the sum is over all possible values of x.

The PMF gives the probability that the random variable equals a specific value.
For example, if X represents the outcome of rolling a fair six-sided die:

$$P(X = x) = \frac{1}{6}, \quad x = 1, 2, 3, 4, 5, 6$$

The cumulative distribution function (CDF) of a discrete random variable, $F(x)$, is defined as:

$$F(x) = P(X \leq x) = \sum_{t \leq x} P(X = t)$$

where the sum is over all possible values t less than or equal to x.

Continuous Random Variables

A continuous random variable, Y, is characterized by a **probability density function (PDF)**, $f_Y(y)$, which satisfies the following properties:

1. $f_Y(y) \geq 0$ for all y.
2. $\int_{-\infty}^{\infty} f_Y(y)\, dy = 1$.

The PDF does not give probabilities directly; instead, the probability that Y falls within an interval $[a, b]$ is given by:

$$P(a \leq Y \leq b) = \int_a^b f_Y(y)\, dy$$

The cumulative distribution function (CDF) of a continuous random variable, $F(y)$, is defined as:

$$F(y) = P(Y \leq y) = \int_{-\infty}^{y} f_Y(t)\, dt$$

For example, if Y follows a uniform distribution on the interval $[0, 1]$:

$$f_Y(y) = \begin{cases} 1, & 0 \leq y \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

Key Differences Between Discrete and Continuous Random Variables

- **Discrete random variables** have a probability mass function (PMF) and probabilities are assigned to specific values. For example, $P(X = x)$ is meaningful for a discrete random variable. Examples of distcrete random variables are coin tosses, throwing a dice, the outcome of a football game, etc...
- **Continuous random variables** have a probability density function (PDF) and probabilities are assigned to intervals, not specific values. For example, $P(Y = y) = 0$ for any specific y, but $P(a \leq Y \leq b)$ can be non-zero. Examples of continuous random variables are a person's height or income, the sales of a firm, its stock price, etc...

Understanding these distinctions is crucial for applying appropriate statistical methods and interpreting results correctly.

A.3 Expectation of a Random Variable

The **expectation** (or expected value) of a random variable is a measure of its central tendency, representing the long-run average value that the variable takes when an experiment is repeated many times. It is denoted by $\mathbb{E}[X]$ or $E[X]$.

Expectation for Discrete Random Variables

For a discrete random variable X with possible values $x_1, x_2, \ldots$ and corresponding probabilities $P(X = x_i)$, the expectation is defined as:

$$\mathbb{E}[X] = \sum_i x_i \cdot P(X = x_i)$$

Where:

- x_i are the possible values of X.
- $P(X = x_i)$ is the probability of $X = x_i$.

Example: Suppose X represents the outcome of rolling a fair six-sided die. The possible values of X are $1, 2, 3, 4, 5, 6$, and each value has an equal probability of $\frac{1}{6}$. The expectation is:

$$\mathbb{E}[X] = \sum_{i=1}^{6} x_i \cdot P(X = x_i) = \frac{1}{6}(1 + 2 + 3 + 4 + 5 + 6) = 3.5$$

Expectation for Continuous Random Variables

For a continuous random variable Y with probability density function (PDF) $f_Y(y)$, the expectation is defined as:

$$\mathbb{E}[Y] = \int_{-\infty}^{\infty} y \cdot f_Y(y)\, dy$$

Where:

- y is the value of the random variable.
- $f_Y(y)$ is the PDF of Y.

Example: Suppose Y is uniformly distributed on the interval $[0, 1]$, with PDF:

$$f_Y(y) = \begin{cases} 1, & 0 \leq y \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

The expectation is:

$$\mathbb{E}[Y] = \int_0^1 y \cdot 1\, dy = \left[\frac{y^2}{2}\right]_0^1 = \frac{1}{2}$$

A.3.1 Properties of Expectation

- **Linearity:** For any random variables X and Y, and constants a and b:

$$\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y]$$

- **Expectation of a constant:** If c is a constant:

$$\mathbb{E}[c] = c$$

- **Non-Negativity:** If $X \geq 0$ for all outcomes, then $\mathbb{E}[X] \geq 0$.

Interpretation of Expectation

The expectation represents the **weighted average** of all possible values of the random variable, with weights given by their probabilities (for discrete variables) or their density (for continuous variables). It serves as a theoretical measure of the "center" of the distribution, analogous to the mean in descriptive statistics.

A.4 Summary Statistics

Random variables are characterized by distributions that convey information concerning the probability that the variable takes a specific value (if it is discrete) or falls in an interval (if it is continuous). In practice, when we collect data through an experiment (such as running a survey or in a laboratory setting), we are not able to determine the distribution functions of the random variables whose data we collect. For instance, even if we know that a variable is normally distributed, we may lack information on the specific parameters of the normal distribution such as the population mean or variance of the variable.[4] Summary statistics provide *estimates* of those parameters and other essential information concerning the dataset. Below, we present the main summary statistics used in empirical analysis.

Greek vs Latin letters: Notice that summary statistics (also referred to as sample moments) do not exactly match the population parameters they estimate: i.e. if we draw a sample of ten people from the city of Barcelona and measure their income, the average income of that sample will provide an unbiased estimate of the true mean income across all residents (the population moment) but will not equal that parameter exactly. This is why we use Greek letters to refer to population moments (or parameters of the distribution) and Latin letters to refer to sample moments (which can be computed using data we draw randomly from the population).

Mean ($\bar{x}$)

The mean, or average, is a measure of central tendency that represents the sum of all observations divided by the number of observations:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Where:

- x_i is the i-th observation.
- n is the total number of observations.

Median

The median is the middle value of a dataset when it is ordered. For an odd number of observations, the median is the central value. For an even number, it is the average of the two central values:

$$\text{Median} = \begin{cases} x_{\frac{n+1}{2}}, & \text{if } n \text{ is odd} \\ \frac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2}, & \text{if } n \text{ is even} \end{cases}$$

Mode

The mode is the most frequently occurring value in the dataset. Unlike the mean and median, the mode can have multiple values or none at all.

[4]This is often the case, as due to the central limit theorem we can generate variables which follow specific distributions such as the t, F or the χ^2.

Variance (s^2)

Variance measures the average squared deviation from the mean, representing the dispersion of the data:

$$s^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Where:

- x_i is the i-th observation.
- $\bar{x}$ is the mean.
- n is the total number of observations.

Standard Deviation (s)

The standard deviation is the square root of variance, providing a measure of spread in the same units as the data:

$$s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Skewness

Skewness quantifies the asymmetry of the data distribution. Positive skewness indicates a longer right tail, while negative skewness indicates a longer left tail:

$$\text{Skewness} = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^3}{s^3}$$

As we can see from the formula, if the skewness is positive this means that values greater than the mean are (on average) further away from the mean than values below the mean, hence the longer right tail. If the skewness is equal to 0 this means that the distribution is symmetric, there average distance of values above the mean is the same as that of values below the mean.

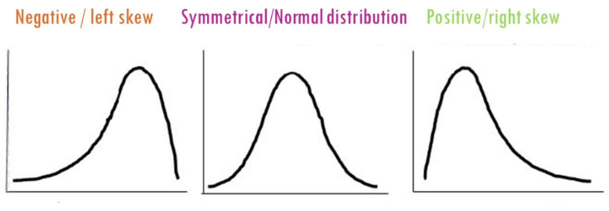


Figure 5: Skewness

Kurtosis

Kurtosis measures the ”tailedness” of the distribution, or the propensity for outliers:

$$\text{Kurtosis} = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^4}{s^4}$$

A kurtosis value greater than 3 indicates a distribution with heavier tails than a standard normal distribution.

A.5 Kernel Density Estimation

Kernel Density Estimation (KDE) is a non-parametric method for estimating the probability density function (PDF) of a *continuous* random variable. Unlike histograms, KDE produces a smooth curve, offering better insights into the distribution.

The KDE formula is:

$$\hat{f}(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

Where:

- $K(\cdot)$: Kernel function (e.g., Gaussian).
- h: Bandwidth parameter controlling the smoothness of the estimate.

This is very similar to plotting a histogram, except instead of dividing the data into bins and computing the fraction of the data in each bin, we repeat the counting process at each sample point, counting the observations within a distance (called the bandwidth) from that point and re-weight (using the kernel) to give less importance to observations far from the point and more to observations closer to the point. In practice, you can think of the value of the density at a point as the height of a histogram bar when the bin is chosen such that that point is the mid-point of the bin. Repeating this for all points and connecting, we get a smooth function, which avoids discretizing the variable which we would be forced to do if we wanted to estimate its distribution using a histogram. In Stata, KDE can be plotted using the command `kdensity varname`.

B Hypothesis Testing and Statistical Models

B.1 Central Limit Theorem (CLT)

The Central Limit Theorem (CLT) is a fundamental result in probability theory and statistics. It states that, under certain conditions, the sampling distribution of the sample mean approaches a normal distribution as the sample size increases, regardless of the shape of the population distribution.

Formal Statement of the CLT

Let $X_1, X_2, \ldots, X_n$ be a random sample of size n from a population with mean μ and finite variance σ^2. The sample mean is given by:

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i.$$

As the sample size n becomes large, the sampling distribution of $\bar{X}$ approaches a normal distribution:

$$\bar{X} \sim N\left(\mu, \frac{\sigma^2}{n}\right).$$

This result holds regardless of the population's original distribution, provided the population has finite variance. The CLT is particularly useful for making inferences about population parameters when the sample size is sufficiently large.

Applications of the CLT

- **Confidence Intervals:** The CLT justifies the use of normal-based confidence intervals for estimating population means.
- **Hypothesis Testing:** Many hypothesis tests assume normality of the test statistic, which the CLT helps ensure for large sample sizes.
- **Approximation:** The CLT allows approximations of probabilities and percentiles for sample means.

B.2 Hypothesis Testing

Hypothesis testing is a formal procedure used to evaluate claims or hypotheses about population parameters based on sample data. The goal is to determine whether there is enough evidence to reject a null hypothesis (H_0) in favor of an alternative hypothesis (H_1).

Steps in Hypothesis Testing

1. **State the Hypotheses:**
    - Null Hypothesis (H_0): A statement concerning a particular unknown population parameter. For example, $H_0 : \mu = \mu_0$.
    - Alternative Hypothesis (H_1): A statement that contradicts H_0. For example, $H_1 : \mu \neq \mu_0$.
2. **Choose the Significance Level (α):** The probability of rejecting H_0 when it is true. Common choices are $\alpha = 0.05$ or $\alpha = 0.01$.
3. **Select the Test Statistic:** Choose a statistic that summarizes the data and follows a known distribution under H_0. Examples include the t-statistic and z-statistic.

4. **Compute the Test Statistic:** Use the sample data to calculate the test statistic.
5. **Determine the p-value or Critical Value:**
   - **p-value:** The probability of observing a test statistic as extreme as, or more extreme than, the one computed, assuming H_0 is true.
   - **Critical Value:** A threshold that defines the rejection region for H_0. If the test statistic falls beyond this value, H_0 is rejected.
6. **Make a Decision:** Compare the p-value to α or the test statistic to the critical value:
   - If the p-value $\leq \alpha$ or the test statistic falls in the rejection region, reject H_0.
   - Otherwise, fail to reject H_0.

Types of Hypothesis Tests

- **One-Sample Tests:** Test whether a sample mean is equal to a hypothesized value.
- **Two-Sample Tests:** Compare means or proportions between two groups.
- **Paired Tests:** Compare means from two related samples (e.g., before and after measurements).
- **ANOVA:** Compare means across multiple groups.
- **Chi-Square Tests:** Test relationships between categorical variables.

Example: One-Sample t-Test in Stata

Suppose we have a dataset of test scores and want to test if the mean score differs from 75. The steps in Stata are:

- Load the dataset:

  ```
  use test_scores.dta, clear
  ```

- Conduct the one-sample t-test:

  ```
  ttest score = 75
  ```

- Interpret the output:
  - Check the p-value to determine whether to reject H_0. The interpretation of the p-value is that under the null hypothesis we would observe the test statistic (or an even larger one) only with a probability of p, hence a low p-value casts doubt on our null hypothesis.

Errors in Hypothesis Testing

- **Type I Error:** Rejecting H_0 when it is true. This happens with probability α, the significance level of the test.
- **Type II Error:** Failing to reject H_0 when it is false. Probability is β. We call $1-\beta$ the power of the test.

Ideally we want both β and α to be as close as possible to zero, which would be a perfect test: we always reject when the null is false and we never reject when it is true. In practice however, there is a tradeoff between the power and the significance level of the test. If the significance level is very low, meaning that we only reject for values of the test statistic which would occur (if the null were true) with a very low probability, it is generally also the case that we fail to reject when the null is false but the true value is close to the hypothesized value.

B.3 Estimators and Their Properties

An **estimator** is a rule or formula used to estimate an unknown population parameter based on sample data. For example, the sample mean is an estimator of the population mean. The quality of an estimator is assessed using several properties, which we discuss below.

1. Unbiasedness An estimator $\hat{\theta}$ is said to be **unbiased** if its expected value equals the true parameter value:

$$\mathbb{E}[\hat{\theta}] = \theta$$

This means that, on average, the estimator produces the correct value across repeated samples.
Example: The sample mean $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$ is an unbiased estimator of the population mean μ:

$$\mathbb{E}[\bar{X}] = \mu$$

2. Consistency An estimator $\hat{\theta}$ is **consistent** if it converges in probability to the true parameter value as the sample size increases:

$$\hat{\theta} \xrightarrow{p} \theta \quad \text{as } n \to \infty$$

Consistency ensures that the estimator becomes arbitrarily close to the true value as the sample size grows.
Example: The sample mean $\bar{X}$ is consistent for the population mean μ.

3. Efficiency An estimator $\hat{\theta}$ is **efficient** if it has the smallest variance among all unbiased estimators of θ. The variance of an efficient estimator achieves the Cramér-Rao lower bound:

$$\text{Var}(\hat{\theta}) \geq \frac{1}{n\mathcal{I}(\theta)}$$

Where $\mathcal{I}(\theta)$ is the Fisher information.
Example: Under the assumptions of the classical linear regression model, the ordinary least squares (OLS) estimator is the most efficient linear unbiased estimator of the coefficients.

4. Sufficiency An estimator $\hat{\theta}$ is **sufficient** if it captures all the information about the parameter θ contained in the sample. Formally, $\hat{\theta}$ is sufficient if the conditional distribution of the data given $\hat{\theta}$ does not depend on θ.
Example: The sample mean $\bar{X}$ is a sufficient statistic for the population mean μ in a normal distribution.

5. Robustness An estimator is **robust** if it is not unduly affected by small deviations from the assumptions, such as the presence of outliers or slight departures from normality.
Example: The sample median is a robust estimator of the central tendency compared to the sample mean, which can be influenced by extreme outliers.

Trade-offs Among Properties

In practice, estimators may not simultaneously satisfy all desirable properties. For example:

- A biased estimator might have lower variance than an unbiased one (e.g., shrinkage estimators like Ridge regression).
- Robustness can come at the cost of efficiency.

Thus, the choice of an estimator depends on the context, the sample size, and the underlying assumptions of the model.

Examples of Common Estimators

- **Sample Mean ($\bar{X}$):** Estimates the population mean μ.

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

- **Sample Variance (S^2):** Estimates the population variance σ^2.

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$$

- **Maximum Likelihood Estimators (MLE):** Estimate parameters by maximizing the likelihood function. For parameter θ, the MLE is:

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta} \mathcal{L}(\theta; X)$$

- **Ordinary Least Squares (OLS):** Estimates the coefficients of a linear regression model by minimizing the sum of squared residuals:

$$\hat{\beta} = (X^\top X)^{-1} X^\top y$$

B.4 Ordinary Least Squares (OLS)

The Model

The OLS model is a linear regression framework used to estimate the relationship between a dependent variable (y) and one or more independent variables (X):

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_k x_{ik} + \epsilon_i$$

Where:

- $\beta_0, \beta_1, \ldots, \beta_k$: Regression coefficients.
- ϵ_i: Error term.

Deriving the OLS Estimator $\hat{\beta}$

The goal of Ordinary Least Squares (OLS) is to estimate the coefficients $\beta_0, \beta_1, \ldots, \beta_k$ by minimizing the sum of squared deviations between the observed values of the dependent variable (y_i) and the predicted values ($\hat{y}_i$).
The predicted values are given by:

$$\hat{y}_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_k x_{ik}$$

The residuals, or deviations, are:

$$e_i = y_i - \hat{y}_i$$

The sum of squared residuals (SSR) is:

$$\text{SSR} = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \cdots - \beta_k x_{ik})^2$$

Step 1: Minimizing the SSR To minimize the SSR, we take partial derivatives of SSR with respect to each coefficient ($\beta_0, \beta_1, \ldots, \beta_k$) and set them equal to zero:

$$\frac{\partial \text{SSR}}{\partial \beta_j} = 0 \quad \text{for } j = 0, 1, \ldots, k$$

Expanding and simplifying, this leads to the following system of normal equations:

$$\sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \cdots - \beta_k x_{ik}) = 0 \quad \text{for } j = 0$$

$$\sum_{i=1}^{n} x_{ij}(y_i - \beta_0 - \beta_1 x_{i1} - \cdots - \beta_k x_{ik}) = 0 \quad \text{for } j = 1, \ldots, k$$

Step 2: Matrix Representation The normal equations can be written in matrix form for simplicity. Let:

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad X = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1k} \\ 1 & x_{21} & x_{22} & \cdots & x_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{nk} \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix}$$

The normal equations become:

$$X^\top (y - X\beta) = 0$$

Rearranging, we get:

$$X^\top X\beta = X^\top y$$

Step 3: Solving for $\hat{\beta}$ The solution to the normal equations is:

$$\hat{\beta} = (X^\top X)^{-1} X^\top y$$

Interpretation of $\hat{\beta}$: The OLS estimator $\hat{\beta}$ minimizes the sum of squared residuals and provides the best linear unbiased estimator (BLUE) under the Gauss-Markov assumptions. Each element of $\hat{\beta}$ represents the estimated change in the dependent variable y for a one-unit change in the corresponding independent variable x_j, holding other variables constant.

Gauss-Markov Assumptions

Under the Gauss-Markov assumptions, OLS estimates are unbiased and efficient:

1. Linearity: The model is linear in parameters.
2. Full rank: The independent variables are not perfectly collinear.
3. Exogeneity: The expected value of errors is zero ($\mathbb{E}[\epsilon_i] = 0$).
4. Homoskedasticity: Constant variance of errors ($\text{Var}(\epsilon_i) = \sigma^2$).
5. No autocorrelation: Errors are uncorrelated ($\text{Cov}(\epsilon_i, \epsilon_j) = 0$ for $i \neq j$).

Unbiasedness means that the expectation of our estimate is equal to the true value of the coefficient. Efficiency means that the precision of this estimate is higher (lower variance) than all other unbiased estimators. A low variance of the estimate means that drawing different samples of x and y and running the minimization again a large number of times will yield coefficients that are similar to each other.

Proof of Unbiasedness

This proof requires some knowledge of matrix algebra to be understood. If you are unfamiliar with this, do not worry. Most results in econometrics can be shown without matrix algebra.[5]

The OLS estimator is given by:

$$\hat{\beta} = (X^\top X)^{-1} X^\top y$$

Taking the expectation:

$$\mathbb{E}[\hat{\beta}] = (X^\top X)^{-1} X^\top \mathbb{E}[y]$$

Substituting $y = X\beta + \epsilon$ and using assumption 3:

$$\mathbb{E}[\hat{\beta}] = (X^\top X)^{-1} X^\top (X\beta) = \beta$$

[5]However, this substantially increases the burden of notation as matrix products will need to be expressed in terms of sums and products of their elements.

B.4.1 Panel Data Models

Panel data models analyze datasets that have both a cross-sectional and time-series dimension. These models are particularly useful for capturing individual-specific effects and dynamic behaviors over time. Let y_{it} denote the dependent variable for individual i at time t, and let x_{it} represent the independent variables.
The general form of a panel data model is:

$$y_{it} = \alpha + x_{it}^{\top}\beta + u_{it}$$

Where:

- α: Common intercept.
- x_{it}: Vector of independent variables for individual i at time t.
- β: Vector of coefficients to be estimated.
- u_{it}: Composite error term.

The composite error term is further decomposed as:

$$u_{it} = \mu_i + \nu_{it}$$

Where:

- μ_i: Individual-specific effect (time-invariant).
- ν_{it}: Idiosyncratic error (time-varying).

Fixed Effects (FE) Model The fixed effects model controls for μ_i by assuming it is correlated with the regressors x_{it}. To eliminate μ_i, the FE model uses **within transformation** (demeaning):

$$\tilde{y}_{it} = \tilde{x}_{it}^{\top}\beta + \tilde{\nu}_{it}$$

Where:

$$\tilde{y}_{it} = y_{it} - \bar{y}_i, \quad \tilde{x}_{it} = x_{it} - \bar{x}_i$$

Here, $\bar{y}_i$ and $\bar{x}_i$ are the individual means over time.

Random Effects (RE) Model The random effects model assumes that μ_i is uncorrelated with the regressors x_{it}. The RE model uses generalized least squares (GLS) to account for the structure of the error term:

$$y_{it} = \alpha + x_{it}^{\top}\beta + \mu_i + \nu_{it}$$

Choosing Between FE and RE The Hausman test is commonly used to decide between FE and RE models. It tests whether the individual-specific effects μ_i are correlated with the regressors. If they are, the FE model is preferred. However, in general it is safer to assume fixed effects as specifying a fixed effects model when the true d.g.p. is random will lead to an efficiency loss but consistent results (in a large sample this will not matter much), whereas specifying the random effects model when the d.g.p. is fixed will lead to biased parameter estimates and inconsistency of the results.
—

B.4.2 Binary Choice Models (Logit)

Binary choice models are used when the dependent variable takes only two possible values, typically $y = 1$ (success) or $y = 0$ (failure). The logit model is one of the most common approaches for such cases.
The logistic regression model specifies the probability of success as:

$$P(y = 1|X) = \frac{\exp(X\beta)}{1 + \exp(X\beta)}$$

Where:

- X: Vector of independent variables (including an intercept term).
- β: Vector of coefficients to be estimated.

Log-Likelihood Function The coefficients are estimated by maximizing the log-likelihood function:

$$\ell(\beta) = \sum_{i=1}^{n} [y_i \ln(P(y_i = 1|X_i)) + (1 - y_i) \ln(1 - P(y_i = 1|X_i))]$$

Odds and Odds Ratio The odds of success are defined as:

$$\text{Odds} = \frac{P(y = 1|X)}{1 - P(y = 1|X)}$$

The odds ratio for a one-unit increase in x_j is:

$$\text{Odds Ratio} = \exp(\beta_j)$$

Example: Predicting Loan Default Suppose we want to model whether a loan applicant defaults ($y = 1$) or not ($y = 0$), based on income (x_1), credit score (x_2), and loan amount (x_3). The logit model is:

$$P(y = 1|X) = \frac{\exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3)}{1 + \exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3)}$$

Marginal Effects To interpret logit coefficients, marginal effects are often calculated:

$$\frac{\partial P(y=1|X)}{\partial x_j} = P(y=1|X) \cdot (1 - P(y=1|X)) \cdot \beta_j$$

Extension to Multinomial and Ordered Logit The logit model can be extended to:

- **Multinomial Logit:** For dependent variables with more than two unordered categories.
- **Ordered Logit:** For dependent variables with ordered categories.