Stata for Applied Econometric Policy Evaluation

Søren Leth-Petersen and Daniel le Maire

Department of Economics

University of Copenhagen

January 23, 2019

Contents

1 Introduction

Stata is a very useful statistical software package for conducting a wide range of econometric estimations. One major advantage of Stata is the possibility for users to write and share user-written canned procedures which make econometric estimations very easy from a technical point of view.

These notes introduce some basic Stata commands, which will be used in the course Applied Econometric Policy Evaluation. These notes far from cover all the commands that you need to know for the course. For example, these notes do not consider most of the routines performing the actual estimations. Instead, the aim of these notes is to give you some basic tips and tricks for data handling and simple data analysis using Stata.

In addition to this, these notes will not contain an exhaustive list of options which are possible to use in connection with each Stata command. There are two reasons for this. First of all, Stata's help is very user-friendly. Second of all, most of Stata's commands have the same following structure:

```
command [varlist] [if exp] [in range] [weight] [, options]
```

where the terms in brackets are not always required:

- `varlist` specifies the variables to be used by the specific command. If no variables are specified some Stata commands will use all variables, whereas others will not work.
- `if exp` (expressions) are used to select the appropriate observations to be used by the command (if any).
- `in range`: Specifies the range of observations to be used (if any).
- `weight`: Specifies the weighting scheme to be used (if any).
- `options` are specific to the command (if any).

Rather than listing all options etc. the reader is encouraged to check Stata's help. There are two ways of accessing Stata's help. From the menu one can select Help. Alternatively, one can use the command line to write `help` and enter. If one needs help for a specific command, say, `summarize` then write `help summarize` in the command window.

2 Initial options

It is possible to execute one command at a time by using the command line and this can sometimes be useful when developing the code. In Stata the program file is called a do file. When opening Stata you can find the do file editor by selecting the "Window" menu and subsequently "Do-file Editor" or just use the short-cut key CTRL+9.

It is always a good idea to clear the memory when we start a new do file.

```
clear all
```

The main window of Stata is the Results window. Stata allows us to get all this output automatically into a log file. Notice that Stata can only have one log file open at the same time, so we cannot create a new log without closing an existing log. Therefore, it is a good idea to begin the do-file by closing any open log file. We do this by writing

```
capture log close
```

Next, we create the log file. We need to specify in which folder we want the log file saved. You can name the log file whatever you prefer, but you should use the extension .log. Finally, we need to tell Stata that every time we run the program the log file should be replaced.

```
log using "Z:\Stata\CardKrueger.log", replace
```

However, if we are going to use a special folder for data and log files it is useful to specify such folder in order to avoid writing the whole path every time we want to read or write a file from this folder. We create a directory by writing `cd` and the particular folder.

```
cd "Z:\Stata\"
```

Then, we can write

```
log using CardKrueger.log, replace
```

When using a log file we should end the do file by closing the log, that is

```
log close
```

When running a Stata do-file or any Stata command the Results window will pause until you press a key in order to let you follow the stream of output as Stata performs the estimations. To avoid this, you can write

```
set more off
```

In some cases Stata allows us to use a shorter notation. Therefore, let us repeat all the commands but in the short form.

```
clear all

capture log c

cd "Z:\Stata\"

log using CardKrueger.log, replace

set more off
```

And then at the end of the do file:

```
log c
```

3 Reading a text data set

The example which we will consider in these notes is the effect on employment of raising the minimum wage. The study, which was carried out by Card and Krueger (1994), focuses on the fast-food industry in Pennsylvania and New Jersey. 410 fast-food restaurants were interviewed before and after New Jersey raised its minimum wage from \$4.25 to \$5.05. For Pennsylvania, the minimum wage stayed constant at

$4.25. Here, the data comes in two data sets; a Stata data set (.dta) for the first interview round and a text data set (.txt) for the second interview round.

Interview1.dta	
Storeid	Unique fast-food restaurant id
Chain	Character variable for the chain for which the individual restaurant belongs. “Burger King”, “KFC”, “Roy Rogers”, and “Wendys”
co_owned	Dummy equal to 1 if company-owned and dummy equal 0 if franchisee-owned
Nj	Dummy for New Jersey. 1 = New Jersey, 0 = Pennsylvania
Empft	Number of full-time employed in first interview
Emppt	Number of part-time employed in first interview
Nmgrs	Number of managers and assistant managers in the first interview
wage_st	Hourly starting wage in the first interview
Hrsopen	Number of hours open per day in first interview
Psoda	Price of medium soda including tax in first interview
Pfry	Price of small fries including tax in first interview
Pmeal	Price of a standard meal including tax in first interview

Interview2.txt	
Storeid	Unique fast-food restaurant id
status2	Character variable for the restaurant’s status for second interview. “refuse”: refused to be re-interviewed, “answer”: answered, “renova”: Closed for renovation, “perman”: permanently closed, “highwa”: closed for highway construction, “mallfi”: closed due to mall fire
empft2	Number of full-time employed in second interview
empppt2	Number of part-time employed in second interview
nmgrs2	Number of managers and assistant managers in the second interview
wage_st2	Hourly starting wage in the second interview
hrsopen2	Number of hours open per day
psoda2	Price of medium soda including tax
pfry2	Price of small fries including tax
pmeal2	Price of a standard meal including tax

The command used for reading text data sets is `infile`. Each command line in Stata ends when the line ends. Hence, you cannot let a command span over two or more lines unless you tell Stata that the line does not change. This can be done the by ending the first line by /// as done below.

```
infile storeid str12 status2 empft2 emppt2 nmgrs2 ///

    wage_st2 hrsopen2 psoda2 pfry2 pmeal2  ///

     using interview2.txt, clear
```

4 Sorting and browsing the data

It is useful to be able to `sort` data. Suppose we want to `sort` by the *storeid*, we write

```
sort storeid
```

We can sort more than one variable by writing a variable list after `sort` rather than just one variable. Stata's `sort` function only sorts data in ascending order. If we wanted to sort data in descending order after *storeid*, we should use `gsort` and write a minus in front of the variable, we want to sort in descending order

```
gsort -storeid
```

To have a look at observations in the data set, we can write `browse` in the command window. If we are only interested in looking at the variables *storeid*, *empft2*, *emppt2*, and *nmgrs2*, we write

```
browse storeid empft2 emppt2 nmgrs2
```

5 Getting the number of observations

We can obtain the total number of observations in various ways. If we are just interested in the number of observations, we can simply write

```
count
```

Alternatively, we can write describe or the short form

```
desc
```

This also gives us information about the type of variables in the data set.

As with essentially all Stata commands, we can use the `count` command with an `if` expression if we want to count the number of stores answering the second interview, i.e.

```
count if status2=="answer"
```

Notice that we need to use the double equality sign when we want to test for equality rather than when we define a variable. We will return to this in section 9, where we consider how to define new variables. Furthermore, notice that since *status2* is a string variable we need to use double quotes around the string value.

Instead, we could get the observation count for each value of *status2*, by using a one-way frequency tabulation function `tabulate`. In short form, we would write

```
ta status2, missing
```

We use the `missing` option to also tabulate missing values, but in this case it turns out that there are no missing values. If we want a variable to hold the counts for each value of the categorical variable, *status2*, we can first sort the data by *status2* and then `count` the number of observations. For this, we need to use Stata's extended generate function, `egen` in connection with `count`

```
bysort status2: egen obsstatus2=count(storeid)
```

We could use any numerical variable in place of *storeid*. However, we should notice that Stata will only count non-missing values.[1] Finally, we can also number each consecutive observation by 1, 2, 3,… To do this simply write

```
gen obsno=_n
```

Furthermore, we can use _n within each category of status2 such that for each category, we restart the counting

```
bysort status2: gen obsno2=_n
```

Finally, if we use `_N` instead of `_n`, we would get the total number of observations. For example,

```
bysort status2: gen obsno3=_N
```

would give us a variable *obsno3* which has as values the total number of observations for each value of *status2*.

6 Saving the data set

We can `save` the data set by writing

```
save interview2.dta, replace
```

As Stata data sets have the extension .dta, this will create a Stata data set. It is important to include the `replace` option if you expect to run the do file more than once since leaving this out, Stata will not replace an existing data set. In this case we did not write the path, where we wanted the data set saved and Stata will put it in the working directory. We can change the working directory using the `cd` command. Alternatively, we could specify the path when we save the data set.

```
save "Z:\Stata\interview2.dta", replace
```

7 Merging data sets

Since we will `merge` the data set by *storeid* we need to figure out which type of `merge` we will conduct. The following command tells us whether the variable we will merge by is unique or not.

```
duplicates report storeid
```

As we can only have one data set in memory, we will clear the memory before reading and sorting the data set interview1.dta

```
clear all
```

By using the Data Browser you can verify that there no longer is data in memory. Next, we want to read the interview1.dta, we write

[1] We can use several other functions than count in connection with egen. For example, we could calculate the mean of a variable within each status2 category. See the help for egen for the full list of functions.

```
use interview1.dta, clear
```

Notice that when we `use` a Stata data set we do not need to specify each of the variables as their names and order are saved when using the Stata .dta format. This is in contrast to using infile for a text file as Stata has no chance of guessing what variable a given column of numbers corresponds to and, therefore, we have to specify the variable names. Furthermore, in the .dta format it is stored whether a variable is numeric or character and, hence, we do not need to specify that chain is a string variable.

For the example considered above, we did not need to use clear in the use statement since we just cleared the memory just above, but usually it is convenient to clear the memory when reading a data set since this will avoid that Stata stops when we prior to the use statement have forgotten to clear the data in memory.

Again, we sort the data by writing

```
sort storeid
```

and examine whether the variable *storeid* is unique or not.

```
duplicates report storeid
```

Finally, we `merge` the two data sets by the variable *storeid*

```
merge 1:1 storeid using interview2.dta
```

Notice that the `merge` used is a 1:1 merge as *storeid* is a unique identifier in both data sets. If it was not unique in just one of the data sets we would conduct a so-called one-to-many merge, that is either `1:M` or `M:1`. This would, for example, be the case if we merged the Card and Krueger data set with a data set with average state characteristics and using the state variable nj as the variable we merge by. We know that there are several fast-food stores in both New Jersey and Pennsylvania and, hence, the state variable (nj) is not unique. Nevertheless, it makes perfectly sense to merge the data set with average state characteristics and, in general, it is often completely fine to perform one-to-many merging. In contrast, many-to-many merging `M:M` should probably always be avoided.[2]

The merge command generates the variable *_merge* which value you should check after a merge. You can either browse the data set or use a frequency table (see section 5). Observations which only are in interview1.dta are given `_merge=1`. For observations only in interview2.dta we have that `_merge=2`. Finally, observations in both data sets have _merge=3. All observations in our case have `_merge=3`.

8 Limiting the data set

We can throw observations or variables away by using `keep` or `drop`. For example, we can `drop` the variable *_merge*

```
drop _merge
```

In the article, Card and Krueger only considers restaurants being in either of the following two groups:

[2] If you want to make a many-to-many `merge`, you should instead use the Stata command `joinby`.

A) Restaurants that answered in the second round (*status2=="answer"*) and we have information on starting wages in both interviews (*wage_st!=. & wage_st2!=.*).

B) Restaurants which permanently closed in second round as this potentially could be a result of the increased minimum wage (*status2=="perman"*).

We can select the appropriate group using the `if` option in connection with the `keep` command. As mentioned in section 1 the `if` option can be used with almost all Stata commands. Notice that there are two requirements which need to be fulfilled for condition A) being true. In Stata we use `&` as 'and'. Since either condition A) or B) needs to be met we use `|` which means 'or' in Stata.

```
keep if (status2=="answer" & wage_st!=. & wage_st2!=.)|  ///
   status2=="perman"
```

Before in A) we required that the starting wages should not be missing (.). While on the subject, it is important to notice that Stata treats missing values as infinity when using relational operators. Hence, had we written that (*wage_st>0 & wage_st2>0*), we would also have included observations with missing starting wages.

9 Generating new variables

The analysis of Card and Krueger (1994) deals with the effect on employment of increasing the minimum wage. Therefore, we need to construct a variable measuring each store's total employment for the first interview using the number of fulltime workers (empft), the number of part-time workers (emppt), and the number of managers (nmgrs). We need to tell Stata that we want to construct a new variable and for this we use the `generate` command.

```
generate emptot=emppt*.5+empft+nmgrs
```

We also need to construct the total employment measure for the second interview, that is after the minimum wage has been raised in New Jersey. This time, we will use a shorter form of writing `generate`

```
gen emptot2=emppt2*.5+empft2+nmgrs2
```

Using an even shorter form of `generate` we can compute the change in total employment.

```
g demp=emptot2-emptot
```

In a similar way, we calculate the change in starting wages.

```
g dwage=wage_st2-wage_st
```

For the OLS regressions in sections 13 and 15 we also need to create dummy variables for the type of fast-food chain. Here, we will show different ways of creating such dummies. We can use generate in connection with an `if` statement in order to create a dummy variable for Burger King.

```
g bk=1 if chain=="Burger King"

replace bk=0 if chain !="Burger King"
```

There are several things to notice. First, we need double equality signs for conditions, but only one equality sign for definitions. Second, notice that the first line only assigns the value 1 if the chain is Burger King. However, for all other chains the value of bk is missing (.). Hence, we need to overwrite an existing variable, so we need to use `replace` command rather than `generate`. In other words, we cannot generate a variable which already exists. The `if` option in the second line uses 'not equal to' which in Stata is written as `!=`. Next, we will create a dummy for KFC. We will use a similar first line, but in the second we will use the function `recode`.

```
g kfc=1 if chain=="KFC"

recode kfc .=0
```

The dummy for Roy Rogers can be constructed using yet another way of constructing dummy variables.

```
g roy=(chain=="Roy Rogers")
```

Sometimes it is useful to "translate" a categorical string variable as chain into a categorical numerical variable. This can be achieved by writing

```
g chain_num=1*(chain=="Burger King")+2*(chain=="KFC")+ ///

   3*(chain=="Roy Rogers")+4*(chain=="Wendys")
```

In this case we only need to construct a few dummy variables for fast food chains. If we need to construct many dummy variables from a categorical level variable, it would be tedious and time-consuming to code each dummy separately. In section 10, we will use the `tabulate` command for an automated way of generating dummy variables. Alternatively, one can set up a loop. In section 12, we consider how to use loops in Stata. Another solution, which we will consider here, is to use the `xi` command.

```
xi i.chain
```

The `xi` command creates the dummies *_Ichain_1*, *_Ichain_2*, *_Ichain_3*, and *_Ichain_4*. We can call these by either writing *_Ichain_1-_Ichain_4* or *_Ichain_**, where * is used as a wildcard for the suffix. Hence, writing *_Ichain_** STATA will select all variables starting by *_Ichain_* no matter their suffix.

The `xi` command can also create interaction dummies. In the case of Card and Krueger's analysis, this is not needed, but we could create interaction dummies between fast food chain and state by writing

```
xi i.chain*i.nj, noomit
```

When using the `xi` command again, Stata drops the dummy variables previously created by the `xi` command. However, with the new `xi` command we create *chain* dummies, a *nj* dummy and interactions between *chain* dummies and the *nj* dummy. Using the *noomit* option implies that STATA construct all possible combination of *chain* dummies and *nj* such that we in regressions do not need to also include the *chain* dummies and the *nj* dummy.

10 Descriptive statistics

We can display simple descriptive statistics with the use of the `summarize` command. If we want to see descriptive statistics for all of our variables we should just write

```
summarize
```

and similar to other Stata commands we can use the short form `su` to instead of writing `summarize`. We can also specify a subset of variables for which the summary statistics should be calculated as well as we can get a more detailed output including percentiles, variance, skewness and kurtosis

```
su wage_st wage_st2, detail
```

The output shows that in the first interview at least 25 per cent of the stores' starting wages are equal to the minimum wage of $4.25, but at the second interview less than 10 per cent of the restaurants have a minimum wage of $4.25. However, in the second interview both the 25th and 90th percentiles (i.e. at least 65 per cent) of the stores' starting wages are equal to the new starting wage of New Jersey.

Sometimes we may need to create a variable which is the mean of another variable. In this case, we can use the extended `generate`, that is

```
egen wage_stM=mean(wage_st)
```

which we also considered in section 5. We can also calculate the mean for specific groups or individuals. For example, the average starting wage for each of the fast food chains can be computed as

```
bysort chain: egen wage_stchain=mean(wage_st)
```

In section 5, we showed that we can use the `tabulate` to display a frequency table. As mentioned in the previous section, we can also use the tabulate function to construct dummy variables. If we want to construct the variables *chaindum1-chaindum4* for the four possible fast-food chains we just need to write

```
tabulate chain, g(chaindum)
```

The ideal case for the analysis of Card and Krueger is if the fast food industry in New Jersey and Pennsylvania are similar. In order to examine this, Card and Krueger construct cross frequency tables. Below, we examine whether the different fast-food chains are distributed evenly among the two states using the short form `ta` to call the `tabulate` function.

```
ta chain nj, column
```

By specifying `column` we obtain percentages that for each column add to 100 percent. If we instead specified row we would get row percentages whereas we would get cell percentages by specifying cell. There does not seem to be much difference between the two states in terms of the fast-food chains.

Sometimes it is useful to display one-way tabulations for more than one variable. To achieve this, we can use `tab1` rather than multiple `tabulate` separately for each variable.

```
tab1 chain nj
```

Another way of checking whether the two states are similar in terms of their fast-food restaurants is to compare averages calculated for each state separately. Using the `tabstat` command we can calculate the means of number of employed, starting wages and the number of hours open per day. Again, it does not seem to be the case that the two states differ with respect to their fast-food industries - at least before the minimum wage was raised.

```
tabstat empft emppt nmgrs wage_st hrsopen, by(nj)
```

Two additional useful commands are `codebook` and `inspect`. The former counts the number of unique values of a variable and provides a frequency table for variables taking nine values or less and otherwise reports some deciles. The latter counts the number of negative and positive integer and non-integer values. `inspect` also provides a frequency table for variables taking nine values or less.

11 Graphs

The default option in Stata is to only display the latest graph created. Sometimes this is useful, but when using a do file together with `set more off` it may be the case that we want to display every graph created by the do file. In order to do so we assign a name to each graph. As we cannot assign the same name twice we should remember to delete the figures that Stata has in memory at the beginning of our do file. We do this by writing:

```
graph drop _all
```

In section 10, we used the detailed version of summarize to examine the distribution of the starting wages. An alternative is to simply depict the distribution of starting wages in fast food chains in New Jersey using a `histogram`. Using the short form, we simply write

```
hist wage_st if nj==1, name(fig1)
```

The `name` option stores the graph as *fig1* such that subsequent graphs do not replace this graph. For example, if we run the following code we will be able to see both the histogram with the starting wages from the first interview and the histogram with the starting wages from the second interview.

```
hist wage_st2 if nj==1, name(fig2)
```

As expected, it is evident that the increased minimum wage in New Jersey had a remarkable impact of the distribution of starting wages in fast food restaurants in New Jersey. We can combine the two histograms by referring to the names of each figures and using `graph combine`.

```
graph combine fig1 fig2, name(fig3)
```

However, even though we keep each graph in the work space, none of the graphs have yet been saved to a file. Suppose we only want to save *fig1* as figure1.gph, then we could write

```
graph save fig1 figure1.gph, replace
```

The `graph save` command saves the graph to Stata's native graph format .gph, which makes it possible for us to re-load the graph into Stata in a later Stata session. To export the graph to format which can be read by other programs such as Word or TeX, we can export the graph using the `graph export` command. Below, we export fig3 with the combined histograms and save it as a .png file.

```
    graph export figure3.png, name(fig3) replace
```

For the Card and Krueger diff-in-diff analysis, it is important that the increase in the minimum wage in New Jersey had a larger effect of starting wages in New Jersey compared to Pennsylvania. We can examine this by using the bar plot by using `graph bar`.

```
    graph bar (mean) dwage, over(nj) name(fig4)
```

We see that whereas the starting wage, on average, increase in New Jersey, it decreases in Pennsylvania.

To show how to construct two-way plots, we are going to change the structure of the data set. Instead of having the data set in a so-called wide format with the two interviews for the same restaurant in the same row, but in two columns, we want to have the data in the long format, where we have only one variable with the starting wage for each restaurant, but we have two observations (rows) for each restaurant. To achieve this, we will use the `reshape` command. However, since we want to return to the current wide version of the data set, we use the `preserve` command before using the `reshape` command and then afterwards use the command `restore`. This way we ask Stata to `preserve` the main data set, which we then return to at the point in the do file where we use the `restore` command. Since we want to go from a wide data set to a long data set we use `reshape long`. In principle, we could go back using `reshape wide`. However, to ease the understanding of what is going on, we begin by only keeping a small subset of the variables, whereby we need to use the `restore` command to return to the data set with all variables.

The main goal of the exercise below is to provide graphical evidence of the development in the starting wage in New Jersey and Pennsylvania around the time when the minimum wage was increased in New Jersey. To understand the following block of code, it is recommended to run the code line by line to understand the effect of each line even though that running the code line by line will result in Stata responding that there is nothing to `restore` when we get to the last line. Therefore, when you have understood what the code does, you should start all over running the code from the beginning of these notes such that the data set is restored.

The `reshape` command needs to know which variable corresponds to the first interview (at time 1) and the second interview (time 2). Therefore, we `rename` *wage_st* and *emptot* such that they both get the suffix *1*. Next, we use the `reshape long` command. We need to provide Stata with a variable that uniquely identifies each observation in the wide dataset, that is *storeid*. We also need to tell Stata what we will call the variable for each interview. Here, we call it *time*.

In section 10, we used `bysort` and `egen` to calculate group means. This could be useful if you want to include a group average in a regression. If you instead want to make a graph with group means it is often easier to use the `collapse` command. Below, we use `collapse` to calculate the means of the starting wage and the employment for the two states before and after the raised minimum wage in New Jersey. It is important to stress that the `collapse` command completely overwrites the data currently stored in Stata's memory and only stores, for example, the means for the groups you request. Therefore, we would usually use the `preserve` and `restore` commands when using the `collapse` command.

The final bit of code draws plots with connected line plots. We begin by connecting the average starting wages in Pennsylvania (*nj==0*) at time 1 and time 2. Next, we overlay a similar connected line plot for New Jersey (*nj==1*). We use the `xlabel(1(1)2)` option to make sure that the horizontal axis run between 1 and 2, and that no ticks are made in between 1 and 2. Besides this, we add labels to the legend. We repeat this overlaid connected line plot[3] for the average total employment in the fast food restaurants and, finally, we combine the graphs for the starting wages and the average employment in fast food restaurants.

```
preserve

keep storeid chain nj wage_st wage_st2 emptot emptot2

rename wage_st wage_st1

rename emptot emptot1

reshape long wage_st emptot, i(storeid) j(time)

collapse (mean) wage_st emptot, by(nj time)

graph twoway (connected wage_st time if nj==0, xlabel(1(1)2)) ///

       (connected wage_st time if nj==1, xlabel(1(1)2)), ///

       legend(label(1 "Pennsylvania") label(2 "New Jersey")) ///

       name(fig5)

graph twoway (connected emptot time if nj==0, xlabel(1(1)2)) ///

       (connected emptot time if nj==1, xlabel(1(1)2)), ///

       legend(label(1 "Pennsylvania") label(2 "New Jersey")) ///

       name(fig6)

graph combine fig5 fig6, name(fig7)

restore
```

We find that both starting wages and employment in the fast food restaurants increase in New Jersey, whereas starting wages and employment in the fast food restaurants decrease in Pennsylvania. This is a surprising result, which contradicts standard economic theory, which would predict that a negative relationship between the minimum wage and employment.

12 Loops and locals

While-loops are useful to avoid repeating the same code over and over again. Let us return to the case of starting wages for the different fast food chains. In the previous section, we used `bysort` and that is a

[3] Stata's graph command offers a range of other possibilities including scatter and line plots.

simpler solution in this case than using `while` loops. Nevertheless, let us consider this example. Obtaining the means of starting wages could be achieved by writing

```
su wage_st if chain_num==1

su wage_st if chain_num==2

su wage_st if chain_num==3

su wage_st if chain_num==4
```

As we immediately can see, the only thing which changes is the chain number. Therefore, we can loop over just one of these lines and changing the chain number every time we run through the loop. Hence, with a while-loop we can write

```
local ii=1

while `ii'<=4 {

      display "Results for chain no. `ii'"

      su wage_st if chain_num==`ii'

      local ii= `ii'+1

}
```

In the while-loop we are looping over the lines in between the two curly brackets. The while-loop will keep on running until the local variable *ii* becomes larger than 4, i.e. it runs while the value of the local *ii* is less or equal to 4. Therefore, it is crucial that that we update this variable each time we go through the loop since otherwise Stata will not automatically exit the loop. We achieve this by the fifth line. It is important to notice that we just write *ii* when we define it, but whenever we want to use this local we need to write *`ii'* (in front of the local variable needs to be a grave accent and behind it needs to be a prime). The third line is not necessary, but it displays the current value of the local variable *ii* each time we go through the loop.

In this case, we knew that there were 4 different fast food chains, but we could let Stata loop through the different values of the chain variable without pre-specifying the number of fast food chains. Using the `levelsof` command we can store a list of fast food chains in a local *levchain*, which we subsequently loop over using a `foreach` loop rather than a `while` loop.

```
levelsof chain, local(levchain)

foreach ii of local levchain {

      display "Results for chain `ii'"

      su wage_st if chain=="`ii'"

}
```

13 Accessing estimation results

Almost all estimation routines temporarily save estimates etc. which correspond to the command's output to the results window. Suppose we want to use the mean after a `summarize`

```
su wage_st

return list
```

Then, writing `return list` after the `summarize` statement gives us a list of all the currently saved results and their names. In this case, where we have used `summarize` all the temporarily stored results are scalars. Since the current results will be lost when we use, for example, `summarize` next time, we can store the result in a new `scalar`

```
scalar avgwage=r(mean)
```

Writing `return list` we can always see the results from the latest r-class command. It turns out that many Stata commands are e-class commands. To access the estimation results, we write need to write `ereturn list`. An example of an e-class command is `regress`. We follow Card and Krueger (1994) and `regress` the change in employment from before to after the raise in the minimum wage in New Jersey on the dummy for New Jersey. We use the short form of `regress` and ask for `robust` standard errors

```
reg demp nj, robust
```

According to conventional economic theory assuming perfect competition, an increase in the minimum wage should unambiguously lead to lower employment. Consequently, we would have expected a negative coefficient to the store being in New Jersey, but surprisingly we get a positive coefficient.

Writing `ereturn list` gives us a list of saved scalars, vectors and matrices. We can see that the OLS coefficients are saved in the vector *e(b)*. If we want to extract the whole vector, we need to make use of matrices. In the first line below, we define a new `matrix` *beta* and in the second line we print it in the results window.

```
ereturn list

matrix beta=e(b)

matrix list beta
```

The vector beta is a row vector and has dimension (1 x 2). Suppose we want define a `scalar`, which takes the value of the constant term. Then, we can write

```
scalar constant=beta[1,2]

disp constant
```

However, we do not have to use matrices to extract the OLS coefficients or the standard errors. We can simply write

```
scalar beta1=_b[nj]

scalar se1=_se[nj]
```

Then, we can compute the t-statistics of the New Jersey dummy as follows

```
scalar t1=beta1/se1
```

To check that the stored values are as you expect you can use the `display` command

```
disp "The beta estimate for New Jersey is " beta1

disp "The corresponding standard errors are " se1

disp "The corresponding t-statistic is " t1
```

14 Globals

In section 12, we saw how to use locals. Globals are also very useful if we intend to re-use the same variable list several times. To show this, we follow Card and Krueger (1994) and control for the type of fast food restaurant and a dummy for whether the individual restaurant is company-owned or franchisee-owned. First, we `regress` the change in employment on the New Jersey dummy while including the control variables in the regression.

```
reg demp nj bk kfc roy co_owned, robust
```

Alternatively, we could first define the `global` *x* with the list of control variables

```
global x nj bk kfc roy co_owned
```

Then, we estimating the regression we can use the global macro by writing *$x* , that is

```
reg demp $x, robust
```

15 Producing tables for Excel and TeX

There are several ways of getting regression results into Excel and TeX. To begin with, we will show how to use the command `estout`. This command is not pre-installed in Stata so we have to install it ourselves. For this installation you need to be connected to the internet. In order to install the program you just need to write

```
ssc install estout, replace
```

You only need to install `estout` once. Next time you open Stata you can use `estout` in the same way as you would use any standard Stata commands.

We want to create a table where the first column shows the estimation results when only regressing the change in employment on the dummy for New Jersey and the second column contains the results from regressing the change in employment on the New Jersey dummy, the chain dummies and the ownership dummy. As Stata only stores the latest estimates we need to store the estimates every time we run a regression. We store the estimates for `estout` using the command `eststo`. Before storing any estimates let us erase any sets of estimates stored by `eststo`.

```
    eststo clear

    reg demp nj, robust

    eststo model1

    reg demp $x, robust

    eststo model2
```

Next, we will create the table with our results. To begin with we will not export the results to either Excel or Tex. Instead, we will display the table in Stata and see how we can use different options to export exactly what we prefer. Let us consider the simplest table by just writing `estout`.

```
    estout
```

The table has the overall structure as we intended, but several things are missing. For example, standard errors of the beta estimates are not displayed in the table. We use the `cells()` option to tell Stata what content we want in each cell. In this case we want the beta estimates (`b`) and the standard errors (`se`). Furthermore, we want that the former has 3 decimals and that the latter has 2 decimals. Additionally, we want to display stars to indicate significant parameter estimates. Finally, we want a `legend` to explain which significance level the number of stars means.

```
    estout, cells(b(star fmt(3)) se(par fmt(2))) legend
```

We also want to display R-squared (`r2`), the number of observations (`N`) and how the variance is computed (`vce`). In order to do so we use the `stats()` option. In this case, we assign labels to R-squared and the number of observations, but we did not have to. Finally, we assign `labels` to all variables such that they are easier to understand. To avoid that the labels are truncated we use varwidth(14) to tell Stata that the column width of the first column should correspond to 14 letters.

```
    estout, cells(b(star fmt(3)) se(par fmt(2))) legend ///

        stats(r2 N vce, labels("R-squared" "No. of obs.")) ///

        varlabels(nj "New Jersey" bk "Burger King" kfc "KFC" ///

        roy "Roy Rogers" co_owned "Company-owned" ///

        _cons "Constant") varwidth(14)
```

Now, we are satisfied with the table and we will export it to Excel and Tex. With respect to Excel we just need write which file we are using. For Tex we also need to specify the style.

```
    estout using Results1.xls, cells(b(star fmt(3)) se(par fmt(2))) ///
```

```
        legend stats(r2 N vce, labels("R-squared" "No. of obs.")) ///
        varlabels(nj "New Jersey" bk "Burger King" kfc "KFC" ///
        roy "Roy Rogers" co_owned "Company-owned" ///
        _cons "Constant") varwidth(14) replace


estout using Results2.txt, cells(b(star fmt(3)) se(par fmt(2))) ///
        legend stats(r2 N vce, labels("R-squared" "No. of obs.")) ///
        varlabels(nj "New Jersey" bk "Burger King" kfc "KFC" ///
        roy "Roy Rogers" co_owned "Company-owned" ///
        _cons "Constant") varwidth(14) style(tex) replace
```

Another option than using `estout` for saving estimation results is `outreg2`. Again, we need to install this command by writing

```
ssc install outreg2, replace
```

With `outreg2`, we need to say after each equation which Excel workbook we will add the estimation results (if any). The first time we use it, we should use the `replace` option to overwrite any existing Excel workbooks. Subsequently, we will append the Excel workbook such that each regression creates an additional column of results in the Excel workbook.

```
reg demp nj, robust

outreg2 using Results3.xls, replace ctitle(Model1)

reg demp $x, robust

outreg2 using Results3.xls, append ctitle(Model2)
```

Similar to `estout`, we can use variable labels in connection with `outreg2`. We specify the labels before using `outreg2` and then use the `label` option.

```
label variable nj "New Jersey"

label variable bk "Burger King"

label variable kfc "KFC"

label variable roy "Roy Rogers"

label variable co_owned "Company-owned"
```

`estout` and `outreg2` do not create a native Excel workbook, but rather a file, which can be opened by Excel. If we have many estimation regressions, it may be useful to collect them in different sheets in an Excel workbook. To do this, we will use `outreg2` to create a Stata data set with estimation results and then subsequently use Stata's `export excel` command to save our estimation results in separate sheets in an Excel workbook. Since we need to clear the memory when opening the data sets created by `outreg2`, we need to use `preserve` and `restore` to return to our estimation sample.

```
reg demp nj, robust

outreg2 using Results4a, dta replace ctitle(Model1) label

reg demp $x, robust

outreg2 using Results4b, dta replace ctitle(Model1) label

preserve

use Results4a_dta.dta, clear

export excel using Results4.xls, replace sheet("Model1")

use Results4b_dta.dta, clear

export excel using Results4.xls, sheetreplace sheet("Model2")

restore
```

16 Installation of additional packages for the course

For this course we need some additional routines, which you are expected to download. Below is a list of additional routines needed for the course:

- `binscatter`: In the Stata command window write `ssc install binscatter`. This should install the package. To check that this is installed properly write `help binscatter` in the Stata command window. If the help window opens, it has been installed. Alternatively, we can search for the package by writing `findit binscatter` in the Stata command window. Then, click on the link "binscatter from http://fmwww.bc.edu/RePEc/bocode/b". Next, click on the link "(click here to install)".
- `DCdensity`: First, you need to locate your "plus" folder in Stata's "ado" folder (alternatively you can use the "personal" folder. To locate these folders, write `sysdir` in the Stata command window. This is easy on a Windows computer, where the path is likely to be "C:\ado\plus\", but more difficult to locate the folder on a Mac. When you have located your "plus" folder, create a new folder called "d" if no such folder exists, i.e. on a Windows computer it is likely that you will create the folder "C:\ado\plus\d\". Second, go to http://eml.berkeley.edu/~jmccrary/DCdensity/ and download DCdensity.ado. It is important that the file you save is an ado file and not a txt file. The ado file should be saved to the folder "..\plus\d\". To check that this works you need to close Stata and restart it. There is no Stata help file for `DCdensity`. Therefore, we just type

`DCdensity` in the Stata command window (remember that Stata is case sensitive). If `DCdensity` is correctly installed, Stata will tell you that *"varlist is required"*.

- `rd`: In the Stata command window write `ssc install rd`. This should install the package. To check that this is installed properly write `help rd` in the Stata command window. If the help window opens, it has been installed.

17 Index