Stata Workshop – Walkthrough

FEB, University of Groningen

February 2024

This file contains a series of sections that introduce various aspects of Stata from basic program syntax to advanced data analysis and modeling. The topics are tailored towards practitioners of econometrics and assume basic knowledge of econometrics. Questions, corrections or comments can be sent to t.kantarci@rug.nl.

1. Program Window

The program window consists of five panels. The 'Command' window is where you type command syntax. Type `use "M:\SW - Dataset One.dta"` in the command prompt, which loads a dataset.

The 'Variables' window shows the variables loaded and held in memory during a Stata session. It lists the names and labels of the variables stored in the data file.

The 'Properties' window provides details on the type and format of the variables, and on the data file. You will learn about the data formats in a future section.

The 'Review' window keeps a history of the commands you type. If you click on a command in this window, it will appear in the Command window. In this way you can adjust a command without typing it in the Command window all over again.

```
use "M:\SW - Dataset One.dta"
```

2. Interacting with Stata

You can interact with Stata in three ways. In the first way, you type commands in the Command window. For this you need to know the command syntax which will be explained in the next section.

Instead of typing commands in the Command window, you could follow the point-and-click approach. E.g., on the toolbar of the program window, select Statistics > Summaries, tables, and tests > Summary and descriptive statistics > Summary statistics. A dialog box will appear with three tabs. The first tab shows, among other things, a pull-down menu named 'Variables'. Choose 'wage'. For 'Options', choose 'Display additional statistics'. At the 'by/in/if' tab, type female == 1 as the if condition, and at 'Use a range of observations' specify 'From' as 1, and 'to' as 50. Press OK. This is the point-and-click approach.

In the third way, you execute pre-written commands stored in a do file. You will learn about the do file in the next section. As you use Stata, you will learn and remember the standard command syntax. You can then use the program menu to call special command syntax when in need.

How do we execute command syntax contained in a do file? Consider the do file prepared for this workshop. Go to Section 2. Using your mouse, highlight the line containing the command `sum wage`. On the toolbar of the do file editor, click on the 'Do' button. This will execute the highlighted command. A more practical way of executing a command is to use the keyboard. Use the up- and down-arrow keys on the keyboard to place the cursor to the left of the command `sum wage`. Press and keep on pressing the left shift key, and use the right-arrow, or the down-arrow key to highlight the command. To execute the highlighted command, hit together the control key and the letter D while the command is highlighted on a Windows operating system. Hit the left shift key, the command key, and the letter D on an OSX operating system.

```
sum wage
```

3. Creating a Do File

A do file is a simple text file that contains command syntax exactly as you would type them in the command prompt. When you send it to a third party, it allows the correspondent to trace

your calculations, and replicate them in Stata if you provide along the data file too. The do file takes the extension 'do'. 'SW - Do file.do', prepared for this workshop, is an example do file. It contains all the commands executed during the workshop, and allows you to review and edit them on your own.

To create a do file from scratch, on the program toolbar, click 'Do-file Editor'. This will open the do file editor. You can also simply type `doedit` in the command prompt. You can then start writing command syntax in the editor. When you are ready with editing, simply go to the toolbar menu of the do file editor, and select 'save' to save this simple text file as a do file.

```
doedit
```

4. Program Syntax

Stata commands have the following syntax diagram: `command [varlist] [if] [in] [weight] [, options]` where `command` is a command such as `summarize`; `varlist` is the list of variables for which the command is executed; `if` is a condition imposed on the command; `in` specifies the range of observations for the command; `weight` is used, e.g., when some sample observations are to be weighted differently than others; and `, options` specify the command options. The syntax diagram is only about how you should type a certain command and use a command's options. The square brackets indicate that the concerning item is optional rather than required. When we are typing syntax, we do not use the brackets, but we learn it because Stata manuals always introduce a command using the syntax diagram. For more syntax options, type `search command syntax` in the command prompt, and in the window that appears click on 'language'. Note that Stata commands are case sensitive so that lower case and capital letters have different meanings.

As an example, consider the syntax `summarize wage if female == 1 in 1/50, detail`. It computes summary statistics for wage, among women, in the first 50 observations of the sample, and provides additional details. Note that this command syntax is exactly the same syntax produced by the point-and-click approach in the previous section and printed in the 'Results' window.

Commonly used commands can be shortened to the first three letters. E.g., instead of `summarize wage`, you could type `sum wage`.

```
search command syntax
summarize wage if female == 1 in 1/50, detail
sum wage
```

5. Commenting and Breaking Syntax in a Do File

There are several ways to include comments in a do file. First, you can begin a line with an asterisk (`*`). Everything after the asterisk to the end of the current line is considered a comment. Second, after a command, you can place two forward slashes (`//`), and everything after the slashes to the end of the current line is considered a comment. Third, you can place a comment within delimiters (`/* */`) which can then be placed within a command line. See below examples for each of the three types of syntax commenting.

In your do file editor, sometimes you might need to type a line of command that is wider

than the width of the do file editor. You can simply write the long line of command. However, if you want your code to stay within the limits of the width of your do file editor, you can split a long line of command across multiple lines by enclosing the command in `#delimit;` and `;#delimit cr`. See below an example.

```
* You can add a line of comment.
sum wage if female == 1, detail // You can annotate a command line.
sum wage /* You can annotate within a command.  */ if female == 1, detail
* You can split a line of command across multiple lines.  E.g.:
#delimit;
sum wage
if female == 1
, detail
;#delimit cr
```

6. Opening Data Files

Type `clear` in the command prompt to clear the system memory. Stata data files have the file extension 'dta'. There are two ways to load a data file to the memory. First, on the toolbar of the program window, we could click on the 'Open' icon, locate the dta file, select it, and click 'Open'.

In the second way we can use command syntax. In particular, we can use the `use` function. During this workshop, often we will be closing and opening different data files. Therefore, from the outset it is convenient to define a working directory that can be called with a proxy name at different occasions so that we would not need to type the directory name over and over again. First define a directory, or path, name. Type in the Command window `global path "M:"`. Now, in the command prompt you can type `use "$path\SW - Dataset One.dta"`.

To open other types of data files, other than the dta type native to Stata, on the program toolbar click File > Import, and see the file types Stata can handle. If it is a data type that Stata cannot handle, you can use 'StatTransfer' which is a program that converts different data file formats into the dta format.

```
clear
global path "M:"
use "$path\SW - Dataset One.dta"
```

7. Describing Data

There are several commands that help us get familiar with our data. `describe` produces a summary of the dataset. `codebook` presents information about variable names, labels, and values. `inspect` produces a heuristic histogram, and reports the number of cases where the value of a variable is negative, zero, or positive, and where the value is an integer, non-integer, or missing. `list` lists the observations.

You can restrict these commands to present the results for a restricted set of variables, by typing the variable names after the command. E.g., `codebook wage educ`, or `inspect educ` produce results for wage and education only.

You can restrict a command to present results for a subset of the data using `if` and `in` conditions. E.g., `inspect educ if educ >= 5` inspects the education level, but for five or

more years of education. Type `help operators` in the command prompt to see the complete list of the logical and relational operators that can be used in command syntax.

`list educ in 1/40` lists the observations of the education variable, but for the first 40 observations.

When you execute `list educ in 1/40`, you will realize that the output produced by the command will not fit to the Results window, and Stata will split the output, and present it in a sequence of screens if you keep on clicking on the `--more--` link that appears at the end of the Results window. This can be tedious if you want to see the results all at once. You can turn this feature off by typing `set more off, permanently`.

```
describe
codebook
inspect
list
codebook wage educ
inspect educ
sort educ wage
inspect educ if educ >= 5
list educ in 1/40
set more off, permanently
```

8. Browsing the Data

To examine the data in memory, click on the 'Data Browser' icon located on the program toolbar. To edit the data, click on the 'Data Editor' icon. The Data Browser ensures that you do not accidentally change a data value. You may also type `browse wage educ` or `edit wage educ` to browse or edit selected variables.

When you want to apply a built-in Stata function, such as the `browse` function, to multiple variables, it is tedious to type one by one all the variable names after typing the function name in the Command window. There is an easier way. Type `browse` in the command prompt. Now go to the Variables window, select the multiple variables of interest, and right click on your mouse while the mouse cursor is pointing at one of the selected variables. In the menu that will appear, select 'Send varlist to Command Window'. This will add the variable names in the Command window.

In large datasets with numerous variables, it is difficult to locate a certain variable in the data browser. In this case you can use the `order` command to bring certain variables to the very left hand side of the data sheet. E.g., type `order black white`, and observe in the data browser that these variables are brought to the left hand side of the data sheet.

You could also browse certain observations of a variable. E.g., `browse if female == 1` will open the data browser for females only.

It is possible to list observations contained in select rows of the dataset in the Results window. E.g., type `list in 1/15` to list the rows from 1 to 15 of all variables.

To sort the values of educ in an ascending order, and then to sort within the sorted values of educ the values of wage in a descending order, use `gsort educ -wage`.

```
browse wage educ
edit wage educ
order black white
```

```
browse if female == 1
list in 1/15
gsort educ -wage
```

9. Producing Descriptive Statistics

To produce descriptive statistics for wage, type `sum wage`. `sum wage, detail` provides additional statistics. For example, it shows various percentiles. We learn that 75 percent of the observations of wage take a value smaller than 12.81.

To produce a one-way table of frequency count of the observations of a variable, you can use the `tabulate` function. For example, try `tabulate educ`. Now make use of the `missing` option of the `tabulate` function to inspect if `educ` has missing observations in the data. Also, make use of the `sort` option of the `tabulate` function to sort the frequency counts in a descending order. That is, consider the command `tabulate educ, missing sort`. To produce a two-way table of frequency count of the observations of two variables, you can use the command `tabulate educ female`. You can restrict your tabulation using the relational operators. E.g., `tabulate educ if educ >= 4` produces a table of frequency counts for education for those with four or more years of education. Type in the command prompt `help table`, and `help tabstat` to learn more about the other types of summary representations.

You may obtain descriptive statistics for a variable with respect to the values of another variable using the `by` prefix. E.g., `by female: sum educ` summarizes the number of years spent in education for each gender group. Before executing the command, you need to sort the observations of the two variables using the command `sort female educ`. For more summary statistics options, from the pull-down menu on the program toolbar, choose Statistics > Summaries, tables, and tests.

In econometrics, we are hardly interested in the statistics of a single variable but in the statistical relationships between variables. For variables wage and educ, you can obtain a correlation coefficient matrix, and the p-value of a test of zero correlation for each correlation coefficient in the matrix using the command `pwcorr wage educ, sig`.

A variable may contain missing values, for example because the survey respondent did not answer a question in the survey. In Stata, missing values in numeric variables are represented by a period (.), and they are treated as larger than any other numeric value. Missing values often cause problems in analyses. Consider the following example. The maximum value `educ` takes is 18. If you type `count if educ >= 18` in the command prompt, you will observe that there are 86 cases where `educ` takes a value of 18 or greater. But this is not correct. If you type `count if educ >= 18 & educ < .` in the command prompt, you will observe 83 cases. This means that we have to be cautious, and adjust for missing values in our analyses.

Consider the command `count if educ >= 18 & educ < .` just used. We used the syntax `educ < .` to avoid the missing cases getting counted. A more formal way of excluding the missing cases from any calculation is to use the `!missing(variable)` syntax. `missing` is a function that returns the logical value 'True' if the observation of `variable` is a missing, and 'False' if it is not missing. `!` is the negation operator for a logical value. Hence, we could use `count if educ >= 18 & !missing(educ)` to obtain the correct count of the cases of interest.

Missing values can take different formats: a blank record, or a code such as 9999 are possible formats. If missing values take such formats, you could use the `mvencode` or the `mvdecode` commands to transform different types of missing values to a certain format of your choice.

```
sum wage
```

```
sum wage, detail
tabulate educ
tabulate educ, missing sort
tabulate educ female
tabulate educ if educ >= 4
sort female educ
by female:  sum educ
pwcorr wage educ, sig
count if educ >= 18
count if educ >= 18 & educ < .
count if educ >= 18 & !missing(educ)
```

10. Creating Variables

You can generate new variables by applying arithmetic, logical, relational operators, or math functions to existing variables. E.g., `generate wage1 = wage+1`, `gen negwage = -wage`, and `gen wage2 = wage^2` use arithmetic operators to generate new variables.

You can also use the logical and relational operators when generating variables. For example, the command `gen wage3 = wage^2 if wage <= 40` uses both the logical and relational operators, or `gen lwage = ln(wage)` uses a built-in math function. To see the full range of math functions supported by Stata, type `help functions` in the command prompt.

Suppose you want to generate a dummy variable that takes a value of 1 if the number of years of education is equal to or larger than 10, and 0 otherwise. You can use the command `gen higheduc = educ >= 10`. Alternatively, you could use the programming function `cond` to generate the dummy variable: `gen higheduc = cond(educ >= 10,1,0)`. In the syntax, the first input argument of the function specifies the if condition. The second input argument specifies the value to be returned if the condition is true, and the third argument specifies the value to be returned if the condition is false.

You can also generate variables using the program menu. Go to Data > Create or change data > Create new variable, and press 'Create' in the dialog box that appears to see the full list of options to generate a variable. Or, type `db generate` in the command prompt to launch the same dialog box.

```
help operators
generate wage1 = wage+1
gen negwage = -wage
gen wage2 = wage^2
gen wage3 = wage^2 if wage <= 40
gen lwage = log(wage)
help functions
gen higheduc = educ >= 10
drop higheduc
gen higheduc = cond(educ >= 10,1,0)
db generate
```

11. Manipulating Variables

To delete a variable, say wage, from the data file, type `drop wage` in the command prompt. Now clear the memory, and reload the data. To keep wage but delete all other variables, type `keep wage` in the command prompt. Clear the memory. Reload the data. To rename the variable `exper` as `experience`, use the command `rename exper experience`.

You may want to delete observations of a variable. E.g., `drop in 985/1000` deletes the last 16 observations of all variables, or `by educ: drop if _n == 1` deletes the first instance of each unique value of the education variable. Here the particular command syntax that proves useful in data manipulation is the 'underscore variable' `_n`. It indicates the row number of the observation in the data sheet. To see other variable management options, on the program toolbar see Data > Data utilities.

In Stata, variables can be stored in two main formats: numeric and string. Numeric variables contain numbers and they are shown in the Variables window as 'float', 'long', or some other numeric storage type. String variables contain text, and they are shown as 'str' in the Variables window. To study an example, load a new data file using the commands `clear` and `use "$path\SW - Dataset Two.dta"`. In the Data Browser, inspect the variable `county`. It contains text data. Hence, it is a string variable. Type `sum county` in the command prompt. Stata will return 0 observations. This is because Stata cannot produce certain statistics on string variables. What is the mean value of a variable that takes string values? If we want to consider statistical analysis on a string variable, we can assign numerical values to the string values of the variable. For this we can use the `encode` function which creates a numeric variable from a string variable. As an example, consider the command `encode county, generate(countytwo)`. It generates a new variable, which we name as `countytwo`. It contains data of storage type 'long' where observations of the original variables are assigned numeric values. Type `browse county countytwo` in the command prompt to inspect the result. Now type `sum countytwo` in the command prompt to see, for example, the minimum and maximum values the variable takes.

You may encounter a variable that contains numerical values that are stored as strings, perhaps by mistake. In this case you can use the `destring` command.

Replace and save the data in memory with the last changes made by typing `save "$path\SW - Dataset Two.dta", replace` in the command prompt.

```
drop wage
clear
use "$path\SW - Dataset One.dta"
keep wage
clear
use "$path\SW - Dataset One.dta"
rename exper experience
drop in 985/1000
by educ:  drop if _n == 1
clear
use "$path\SW - Dataset Two.dta"
sum county
encode county, generate(countytwo)
browse county countytwo
sum countytwo
save "$path\SW - Dataset Two.dta", replace
```

12. Merging Two Datasets

We may need to combine two datasets because, e.g., observations of individuals from a household survey dataset needs to be linked to the observations of the same individuals from an administrative dataset.

We have two datasets at our disposal we can merge. Let us inspect their content first. Clear the data currently held in system memory by typing `clear` in the command prompt. Open the first dataset using the command `use "$path\SW - Datatobemergedone.dta"`. Note the variable named `dist_cod` in particular. Once again, clear the data currently held in system memory by typing `clear` in the command prompt. Open the second dataset using the command `use "$path\SW - Datatobemergedtwo.dta"`. Note in particular that this dataset also includes the variable named `dist_cod`.

We need to take the following steps to merge two datasets. First, choose one of the datasets to be merged as the 'master dataset'. The other dataset will be our 'using dataset'. Let us choose 'Datatobemergedone.dta' as the master dataset, and 'Datatobemergeedtwo.dta' as the using dataset.

Second, determine the so-called 'key' variable that is common to both datasets, which will be used to uniquely match the observations of different variables in the two datasets. It is the `dist_cod` variable in the example. Observe that the 'master' and the 'using' datasets both include this key variable.

Third, load the 'master dataset' to the memory using the command `use "$path\SW - Datatobemergedone.dta"`.

Fourth, run the command `merge 1:1 dist_cod using "$path\SW - Datatobemergedtwo .dta"`. `merge` instructs Stata that we are combining two datasets. `1:1` specifies that the merge is a one-to-one match merge. `dist_cod` is the 'key' variable. `using "$path\SW - Datatobemergedtwo.dta"` specifies the 'using dataset' to be merged with the 'master dataset' already held in the memory.

Finally, we can save our merged dataset using the command `save "$path\SW - Datamerged .dta"`.

To see the other types of merge, or the merge options, consider the command `help merge`. See also the commands `append` and `joinby` for other types of data combination options than `merge`.

```
clear
use "$path\SW - Datatobemergedone.dta"
clear
use "$path\SW - Datatobemergedtwo.dta"
clear
use "$path\SW - Datatobemergedone.dta"
merge 1:1 dist_cod using "$path\SW - Datatobemergedtwo.dta"
save "$path\SW - Datamerged.dta"
help merge
```

13. Regression Analysis Using Built-in Functions

In this section we will estimate the parameters of a simple linear regression equation, and interpret the standard regression output produced by Stata.

Clear the system memory from any preloaded information by typing `clear`. Load the data named 'SW - Dataset Two.dta' by typing `use "$path\SW - Dataset Two.dta"` in the

command prompt. An example syntax for OLS regression is `regress testscr str el_pct`. Do not execute this command yet. `testscr` is the dependent variable, and it represents test scores. The dependent variable is followed by two independent variables `str` and `el_pct`. `str` is the student teacher ratio, a measure of class size. `el_pct` is the fraction of English learners in the class. We would like to analyze the effect of class size on the performance of students in their tests, controlling for the effect of the fraction of English learners in the class.

Execute the following commands: `regress testscr` and `regress testscr str el_pct`. Consider the estimation output of the second regression. The upper left panel presents the analysis of variance. The column titled SS contains the 'sum of squares'. Model sum of squares is given by $\sum(\hat{y}_i - \bar{y})^2$, and it is a measure of the spread of $\hat{y}_i$ from its mean $\bar{\hat{y}}$. Residual sum of squares is given by $\sum(y_i - \hat{y}_i)^2$, and it is a measure of the spread of $\hat{u}_i$ from its mean $\bar{\hat{u}}$ which is equal to 0. Total sum of squares is given by $\sum(y_i - \bar{y})^2$, and it is a measure of the spread of y_i from its mean $\bar{y}$. It can be shown that the total variation in $\hat{y}_i$, and the total variation in $\hat{u}_i$ add up to the total variation in y_i. The column titled MS contains the 'mean square'. In particular, it contains the ratio of the SS column divided by the df column. The model df (degrees of freedom) is 2 because the model contains two explanatory variables. The residual df is N – 3, which is the number of observations minus the number of model parameters, including the intercept. The Residual MS is the estimate of the variance of the errors. Square root of it, that is the Root MSE (root mean squared error) in the output, is the standard error of the regression. R-squared is the ratio of the sum of squares due to the regression to the total sum of squares, as defined above. Adjusted R-squared penalizes R-squared for addition of variables because addition of variables always increases the R-squared. We do not review its formula here.

The F-statistic is testing the significance of the model. That is, we would like to know if our model is statistically significant at a desired level of significance. This can be hypothesized as testing if the explanatory variables have no effect on the average value of the `testscr`. That is, we test for $H_0 : \beta_1 = 0, \beta_2 = 0$, against the alternative that at least one of the variables has an effect, $H_1 : \beta_1 \neq 0$ and/or $\beta_2 \neq 0$. Our test statistic is $F = (SSR_r - SSR_{ur})/q)/(SSR_{ur}/(n - k - 1)$. SSR_r and SSR_{ur} are the sum of squared residuals from the restricted and unrestricted models. Here the restricted model refers to `regress testscr`, and the unrestricted model refers to `regress testscr str el_pct`. q is equal to the number of restricted parameters which is 2. $n-k-1$ is equal to $420-2-1 = 417$. We obtain from the ANOVA tables of the two regressions `regress testscr` and `regress testscr str el_pct` that $SSR_r = 152109.6$ and $SSR_{ur} = 87245.3$. The resulting F value is 155.01. We would like to compare it to a critical value from the $F_{q,n-k-1}$ distribution. We can use the command `scalar F95 = invFtail(2,417,0.05)`. `F95` is a name we give to the scalar we want to compute, and `invFttail` asks for the critical value that leaves a probability area of 5% to the right of the F distribution. That is, the command `Ftail(q,n-k-1,f) = p` computes the p-value corresponding to some degrees of freedom, and an empirical F-statistic, and `invFtail(q,n-k-1,p) = f` computes the critical value from the inverse reverse cumulative (upper-tail) F distribution. The command `display F95` will return the computed value as 3.02. This number is well below our F-statistic, and we soundly reject the null hypothesis at 5% that `str` and `el_pct` have no effect in explaining the variation in testscr. Using the commands `scalar P155 = Ftail(2,417,155.01)` and `display P155`, we find that the corresponding p-value is virtually 0. The F value of 155.01, and the p-value of 0 appear in the regression output of the unrestricted model. A joint hypothesis test of the kind explained here is default in the Stata regression output.

The lower panel of the regression output presents the coefficient estimates and their standard errors. We defer the discussion of the calculation of the coefficient estimates to a future section. The column labelled `t` gives the t-statistic. E.g., the t-statistic for `str` tests the null hypothesis

that a change in the class size has no effect on test scores, when the percentage of English learners is controlled for. That is, it tests $H_0 : \beta_1 = 0$ against a two-sided alternative $H_1 : \beta_1 \neq 0$. We can compare its absolute value to the critical value from the t-distribution, say at the 5% significance level. The value of the t-statistic reported by Stata, -2.90, is the value of the ratio $(-1.1 - 0)/0.38$. We would like to compare its absolute value to a critical value from the t-distribution. We can use Stata to compute this critical value using the command `scalar tc975 = invttail(417,0.025)`. Here `scalar` instructs Stata that we want to compute a scalar, rather than a set of values, which, in the current case, is a percentile value. `tc975` is meant to stand for 't critical value 97.5 percentile', and it is just an arbitrary name we give to the scalar we want to compute. `invtttail` instructs Stata that we are interested in the inverse reverse cumulative (upper-tail) Student's t distribution: if `ttail(n,t) = p`, then `invttail(n,p) = t`. 417 is the degrees of freedom corresponding to the number of observations less the number of estimated parameters, and 0.025 is the area under each tail of the t distribution corresponding to a 5% significance level for this two-tailed test. To display the computed scalar, type `display tc975`. This will return the value 1.965, which is the percentile, or the critical value, of the distribution. Our empirical value of the t-statistic exceeds this critical value, and therefore we reject the hypothesis that class size has no effect on the test scores. Hence, class size is statistically significant at the 5% level.

The column label `P>|t|` means that the probability is greater than the positive value of t, and less than the negative value of t, referring to the two tail p-value for the null hypothesis that the coefficient is zero. The commands `scalar p29 = ttail(417,2.9)`, and `display p29` return the reverse cumulative (upper-tail) t distribution. If we multiply the resulting probability by 2, we will obtain the reported p-value. To learn more about the probability distributions and density functions, see `help density functions`.

The last column in the lower panel of the estimation output is for the confidence interval. This interval can be stated as $P(\hat{\beta}_i - 1.96\sigma_{\hat{\beta}_i} < \beta_i < \hat{\beta}_i + 1.96\sigma_{\hat{\beta}_i}) = 0.95$. Given the single sample at hand, we can calculate this interval estimate for the population coefficient of the variable str as $[-1.10 \pm 1.96 * 0.38] = [-1.84, -0.35]$. The interpretation is that in repeated sampling, the probability that intervals like $[-1.84, -0.35]$ will contain the true β_i is 95%.

```
clear
use "$path\SW - Dataset Two.dta"
regress testscr
regress testscr str el_pct
scalar F95 = invFtail(2,417,0.05)
display F95
scalar P155 = Ftail(2,417,155.7)
display P155
scalar tc975 = invttail(417,0.025)
display tc975
scalar p29 = ttail(417,2.9)
display p29
help density functions
```

14. Prediction

Suppose that there is an out of sample student studying in a class with an average size of 19.6, and a percent of English learners of 35. What could be the test score of this student? In

Section 13 we have estimated a regression model using data representative of the underlying population. This model allows us to predict the test score of this out of sample student. That is, the predicted test score model is given by $\widehat{testscr} = 686 - 1.10str_i - 0.65el\_pct_i$, and we would like to evaluate it at $str_i = 19.6$ and $el\_pct_i = 35$. We can easily calculate this by hand, but we will ask Stata to do the work for us.

Before we make our prediction, we need to input the out of sample values of `str` and `el_pct` in our dataset. Values can be inputted through the Data Editor, but instead we will use the Command window to do this. Clear the system memory, and load the data file 'SW - Dataset Two.dta'. Keep in the data only the variables `testscr`, `str`, `el_pct` and `avginc` using the command `keep testscr str el_pct avginc`. Order the variables using the command `order testscr str el_pct avginc`. Type `input`, and in the next line type `. 19.6 35 .` where the order of the input arguments follows the order of the variables stored in the dataset, which explains why we have ordered the variables. Here a period (`.`) means that the data value is missing. We input a missing value for the particular case of `testscr`, which we in fact want to predict, but also for `avginc` because otherwise Stata will not allow us to complete the input procedure. In the next line type `end` to instruct Stata that our input procedure ends here. Once again, carry out the regression `reg testscr str el_pct`, and obtain the predicted values by typing `predict yhat, xb` in the Command window.

List the predicted value alongside the inputted values for the regressors using the command `list yhat str el_pct in 421`. Here `421` refers to the observation number in the dataset. The presented value of `yhat` is the predicted test score for a student studying in a class with an average size of 19.6, and a fraction of English learners of 35. It is the estimated expected value of the test scores given the particular values for the explanatory variables. That is, it is an estimate of $E(testscr|str = 19.6, el\_pct = 35)$.

You can obtain the standard deviation of your point prediction using the command `predict sdpointprediction, stdp`.

```
clear
use "$path\SW - Dataset Two.dta"
keep testscr str el_pct avginc
order testscr str el_pct avginc
input
.  19.6 35 .
end
reg testscr str el_pct
predict yhat, xb
list yhat testscr str el_pct in 421
predict sdpointprediction, stdp
list yhat sdpointprediction str el_pct in 421
```

15. Accessing Results

When you execute command syntax, Stata saves all the results produced by that command in the system memory. You may want to access certain results to use them in other calculations. The `r()` vector saves the results of general commands. E.g., after using the `summarize` command, this vector will contain `r(N)`, the number of observations. To see the full list of elements of the `r()` vector, type `return list` in the command prompt, after you execute a command producing results other than estimation results.

The e() vector saves the results from regression estimations. To see the elements of e() from the last regression we have carried out, type ereturn list in the command prompt. For example, it returns e(rmse), the Root MSE. To display e(rmse), execute the command display e(rmse).

In the output produced by the ereturn list command, note the matrix e(b). e(b) contains the coefficient estimates. To display the content of e(b), execute the command matrix list e(b). To access the content of e(b), you first need to assign e(b) to a matrix, say B, using the command matrix B = e(b). You can then assign a certain element of B to a scalar, say b, using the command scalar b = B[1,3]. b can now be used as input in other commands.

You can also access a coefficient estimate using the syntax _b[varname]. As an example, consider the last three lines of code at the end of the section. Suppose you allow average income to take a quadratic form in the regression, and you are interested in the partial effect of average income, evaluated at the mean value of income which is 15.3. You can consider the command lincom _b[avginc] + 2 * _b[avginc2] * 15.3 to obtain this estimate. lincom is a command for computing estimates, standard errors, and test statistics for linear combinations of coefficients, after any estimation command. Note how you use the syntax _b[avginc] to call the coefficient estimate of avginc in the lincom command. You can also call, e.g., the standard error of a variable with the syntax _se[varname].

```
sum str
return list
display r(N)
ereturn list
display e(rmse)
matrix list e(b)
matrix B = e(b)
scalar b = B[1,3]
display b
gen avginc2 = avginc^2
regress testscr avginc avginc2
lincom _b[avginc] + 2 * _b[avginc2] * 15.3
```

16. Restoring Estimation Results

It is possible to store estimation results in the system memory, and restore them in the Results window. Carry out the regression regress testscr str, and store the estimation results using the command estimates store modelone. Likewise, for a second model, execute the commands regress testscr str avginc and estimates store modeltwo.

estimates table is a built-in command to compare estimation results. The command estimates table modelone modeltwo, star stats(N r2 rmse) creates a table of the coefficient estimates from the last two regressions. It also adds additional statistics to the table as we utilise the options of the command. That is, star adds asterisks to indicate statistical significance levels. N adds the number of observations used in the analysis. r2 adds the R-squared. rmse adds the root mean square, which is the estimated standard deviation of the error term. stats accepts as input all the output produced by the command ereturn list. For further information see help estimates table and help postestimation commands.

```
regress testscr str
```

```
estimates store modelone
regress testscr str avginc
estimates store modeltwo
estimates table modelone modeltwo, star stats(N r2 rmse)
help estimates table
help postestimation commands
```

17. Creating Graphs

Clear the system memory and reload the dataset `SW - Dataset Two.dta`. Generate `avginc2` using the command `gen avginc2 = avginc*avginc`. Consider the regression `regress testscr avginc avginc2`. Obtain the predictions using the command `predict yhat, xb`.

Our aim is to visualise the income profile of test scores. Consider the command `twoway (scatter testscr avginc) (connected yhat avginc, sort)`. `scatter` and `connected` define the graph types, and `sort` is an option to make the fitted line smooth in the latter plot type.

The command `histogram testscr, percent title(Histogram of testscr data)` produces a histogram of the testscr data with the indicated title where the data frequency is considered in percentages. The graph can be copied by clicking on the 'Copy Graph' icon, or it can be edited by clicking on the 'Start Graph Editor' icon on the graph's toolbar. For more histogram options, click on 'Graphics' on the program toolbar, and then on 'Histogram'.

```
clear
use "$path\SW - Dataset Two.dta"
gen avginc2 = avginc*avginc
regress testscr avginc avginc2
predict yhat, xb
twoway (scatter testscr avginc) (connected yhat avginc, sort)
histogram testscr, percent title(Histogram of testscr data)
```

18. Regression Analysis using Matrix Algebra

In section 13, we have used the `regress` command to carry out an OLS estimation. `regress` is a built-in command. In this section we will make use of matrix algebra, and calculate the OLS coefficient estimates by ourselves. Stata is not in common use for matrix algebra, however.

The first two lines of the code listed at the end of the section clears the system memory, and loads the data named 'SW - Dataset Two.dta'. In the third line, `set matsize` is a command that controls the size of matrices in Stata. Our dataset contains 420 observations, and therefore we set a matrix size large enough to handle our matrix operations. The fourth line generates a constant term. In line five, `mkmat` stores the variables in the data in column vectors. That is, it instructs Stata that each variable stored as a column vector in the dataset with N observations is to be used as a N×1 matrix. Hence, we form a vector for each of the following variables: `cons`, `str` and `el_pct`. Alternatively, in the program menu, go to 'Data > Matrices, ado language > Convert variables to matrix' to carry out the same operation. The option `, mat(X)` requests that these vectors be combined in a matrix, where `X` is just a name we give to the new matrix. Likewise, in line six, we define the dependent variable `testscr` to be a column vector. In line seven, we calculate the coefficient estimates using the F.O.C. of the least squares problem given

by $(X'X)^{-1}X'y$, and store them in a vector named `b`. The next line shows the elements of vector `b`. Note that these are the same coefficients we would have obtained if we had run the regression using the command `regress testscr str el_pct`.

```
clear
use "$path\SW - Dataset Two.dta"
set matsize 500
gen cons = 1
mkmat cons str el_pct, mat(X)
mkmat testscr, mat(y)
matrix b = inv(X'*X)*(X'*y)
matrix list b
regress testscr str el_pct
```

19. Writing a Loop

In this section we will practice with a simple loop program. The foreach loop is one of the loop types in Stata. It is used to repeat a command over a specified set of variables. In the example it adds at once the suffix `this_is_it` at the end of the names of the specified variables.

In the first line of the code, we define a local macro with the command `local`, give it an arbitrary name `variables`, and specify the local macro content in double quotes as `testscr str el_pct`. We define this macro so that we do not have to rewrite the variable names in the code later. In second line, `foreach` is the type of the loop. `var` is an arbitrary reference name we give to refer to the elements of the local macro content. After this we call the local macro we have just defined. The open brace must be on the same line where the foreach command rests, and nothing should follow it. In the third line, `rename` is a built-in command used to rename variables. We then instruct Stata to add the suffix `this_is_it` at the end of each element of the local macro content where we refer the elements with the name `var`. This name is enclosed in single quotes. In the last line, the end brace must be on one line by itself.

```
local variables "testscr str el_pct"
foreach var of local variables {
rename `var' `var'_this_is_it
}
```

20. The Optimisation Problem of an Econometrician

There are two types of econometric software. The first type lets you click pre-programmed econometric routines listed on the program menu. The second type asks you to program the econometric routine by yourself. Stata is of the first type: *monkey see, monkey do*. That is, with a rough idea of the econometric theory, one can learn what to click, and have the results in seconds. This works fine for the casual analyst. MATLAB or R, among others, are of the second type: as *you* program the econometric routine, you have a better understanding of the mechanics, upon which your results rest. However, it is also possible to program in Stata. One advantage of programming in Stata is that you can call the built-in Stata commands within your custom code, for example the commands introduced in this workshop.

Here you will learn writing a simple program. The program specifies a log-likelihood func-

tion, and maximizes it for the Probit estimation. Here basic knowledge of the Probit model, and how it is estimated using the maximum likelihood principle is assumed. In the list of commands at the end of the section, the third line starts the program and names it. In line four we declare the arguments that are referred later in the program. In line five and six we define the log-likelihood function. `replace` indicates that we are substituting a new expression. `lnf` and `Xb` are the arguments we declared, and because we are referring them in this line, we enclose them in apostrophes. `$ML_y1` is the label for the dependent variable. Stata will substitute it with the suitable variable. In line seven we end the program. In line eight, `ml model` estimates the specified model. `lf` specifies the maximization method. The equation we would like to estimate is enclosed in parentheses. We then `maximize` the function. The next line displays the result. The last line is the built-in Probit regression syntax. Note that our program and the built-in probit command produces the same result.

```
clear
use "$path\SW - Dataset Three.dta"
program define probit_livingonmyown
args lnf Xb
replace `lnf'= log(normal(`Xb'))*ML_y1 if $ML_y1 == 1
replace `lnf'= log(1-normal(`Xb'))*(1-ML_y1) if 1-$ML_y1 == 1
end
ml model lf probit_livingonmyown (deny = piratio), maximize
ml display
probit deny piratio
```

21. Saving Your Work

Suppose that you have executed a series of commands which have produced output shown in the Results window. You want to save this output. The output in the Results window can be saved to an output file, which Stata calls a 'log file'. The log file keeps a full record of what you type and the output produced. Walk through the following steps. First, clear the memory by typing `clear`.

Second, execute the command `cd "M:\"` which sets the working directory to `"M:\"`. If this command is not executed Stata will save the log file to be created under the default working which is difficult to find through the file browser. Therefore we set the working directory to one of our own choice.

Third, type `log using swlogfile, replace` which instructs Stata to start recording what you type, and to record the output in a log file named `swlogfile`. The `, replace` option overwrites if an earlier, say, more preliminary version of the `swlogfile` was created. If it is the first time you are creating `swlogfile`, you can get rid of the `, replace` option. The log file is a formatted simple text document, and takes the extension smcl. smcl stands for Stata Markup and Control Language.

Fourth, load your data to the memory using the command `use "$path\SW - Dataset One.dta"`, and produce output by typing `sum wage, detail`. When you are ready with your work, you need to instruct Stata to close the log file. For this, execute the command `log close`.

The log file is created. One problem is that the .smcl file format can only be viewed with Stata. To view the .smcl file with a standard text editor, you need to convert the smcl file to a txt file. For this, execute the command `translate swlogfile.smcl swlogfile.txt, replace`.

```
clear
cd "M:\"
log using swlogfile, replace
use "$path\SW - Dataset One.dta"
sum wage, detail
log close
translate swlogfile.smcl swlogfile.txt, replace
```

22. Help System

Stata's help system consists of two parts. The interactive help files and the PDF documentation.

Help files

- On the program menu, click on 'Help' and select 'Contents' to explore available help topics.
- On the program menu, click on 'Help', select 'Search', and in the dialog box that appears type, e.g., 'summary statistics' to get help on this particular syntax that allows you to produce summary statistics. Or, execute the command `search summary statistics`, that is, in Stata's 'Command' window simply type `search summary statistics`.
- On the program menu, click on 'Help', select 'Stata Command', and type a command name, such as 'summarize', to get help on this command. Or, execute the command `help summarize`. In the window that will appear, click on the 'Dialog' tab at the upper right if the window to obtain the dialog box concerning the summarize command.

PDF documentation

- On the program menu, click on 'Help' and select 'PDF Documentation'. This will bring the Stata manuals. The PDF documentation is linked into the interactive help files described above. E.g., in the Command window type `help summarize`. This will bring a window where you can click on the link `[R] summarize` to open the PDF manual on the `summarize` command. This means that you can use the help files to review certain syntax, its options, and short examples involving that syntax, but use the PDF manuals to access the complete documentation of the corresponding syntax.

Statalist

For your specific questions, you can get help at the Statalist, the discussion forum of Stata users. Before posting your question, you need to register yourself to the forum at http://www.stata.com/statalist

```
search summary statistics
help summarize
```