Basic Stata Commands

# Directory Commands

Stata uses the standard UNIX directory commands, which you can use to navigate through folders inside Stata. A quick rundown of these commands:

# pwd

This command prints the path of the current working directory.

# cd

Changes the working directory to whatever you specify. A special case is “cd ..” which changes the directory one level up from the current one.

# ls

Prints the contents of the current working directory.

# display

The **display** command is used to do simple calculations in the Stata window. Display can be shortened to

**di**.

Example: Calculate 400 divided by 6.

di 400/6

Example: Calculate the natural logarithm of 12.

di log(12)

# generate

Generate is used to create new variables, and can be abbreviated as *gen*. The command is always followed by the name you want to give to your new variable, and then by the values you want to assign to this variable.

Example: Create a variable called *newvariable* and assign a value of 1 to every observation

gen newvariable = 1

It is common to use *gen* to create a dummy variable with a 1 if a certain condition is met, and 0 otherwise.

Example: Create a variable called *dummy* and assign a value of 1 to observations with Age over 35 and a 0 otherwise.

gen dummy = Age > 35

Generate is quite powerful, and allows a variety of operators to create variables. Example: Create a variable called *RelTimewithFirm* that equals the tenure with the firm divided by total experience.

gen RelTimewithFirm = TenurewithFirm/Experience

# replace

Replace is a companion command to generate, which is used to change values of an already-existing variable. Generally, it uses the same format as generate. A useful feature is replacing values only for observations that fulfill certain criteria.

Example: We consider sick time that is less than 4 to be trivial and so want it replaced with a zero.

replace SickTime = 0 if SickTime < 4

Conditionals can also be based off other variables.

Example: We don’t want to count sick time for employees over 60.

replace SickTime = 0 if Age > 60

# egen

Generating more nuanced variables requires **egen**, a very powerful and versatile command.

Example: Create a variable *AgeMean* where all observations are the mean of *Age*

egen AgeMean = mean(Age)

Example: Create a variable ExperienceSum where all observations are the sum of *Experience*

egen ExperienceSum = sum(Experience)

The true power of **egen** requires us to use the **by** option. This requires a categorical variable to divide up the data.

Example: Suppose we want to know mean salary for each level of education.

egen SalaryEduc = mean(Salary), by(Education)

We could then use this to create a variable showing how much each person’s salary deviates from the mean for their education level.

gen SalaryDev = Salary – SalaryEduc

It is also possible to use multiple categories in the **by** option.

Example: Create a variable showing the mean salary for each person’s education level and age.

Egen SalaryEducAge = mean(Salary), by (Education Age)

# collapse

The collapse command works similarly to **egen**, but instead of creating a new variable, it compresses the data down to the specified categories. As such, the **by** option is required for a meaningful collapse.

Suppose we have a dataset of individual people’s income and the state in which they live.

Example: Transform the individual-level dataset into a state-level dataset.

collapse (mean) income, by(state)