ESA STATA TRAINING MANUAL

A 6-lesson guide to learning essential Stata for Econ 388 and beyond

Table of Contents

I. An Introduction to Stata

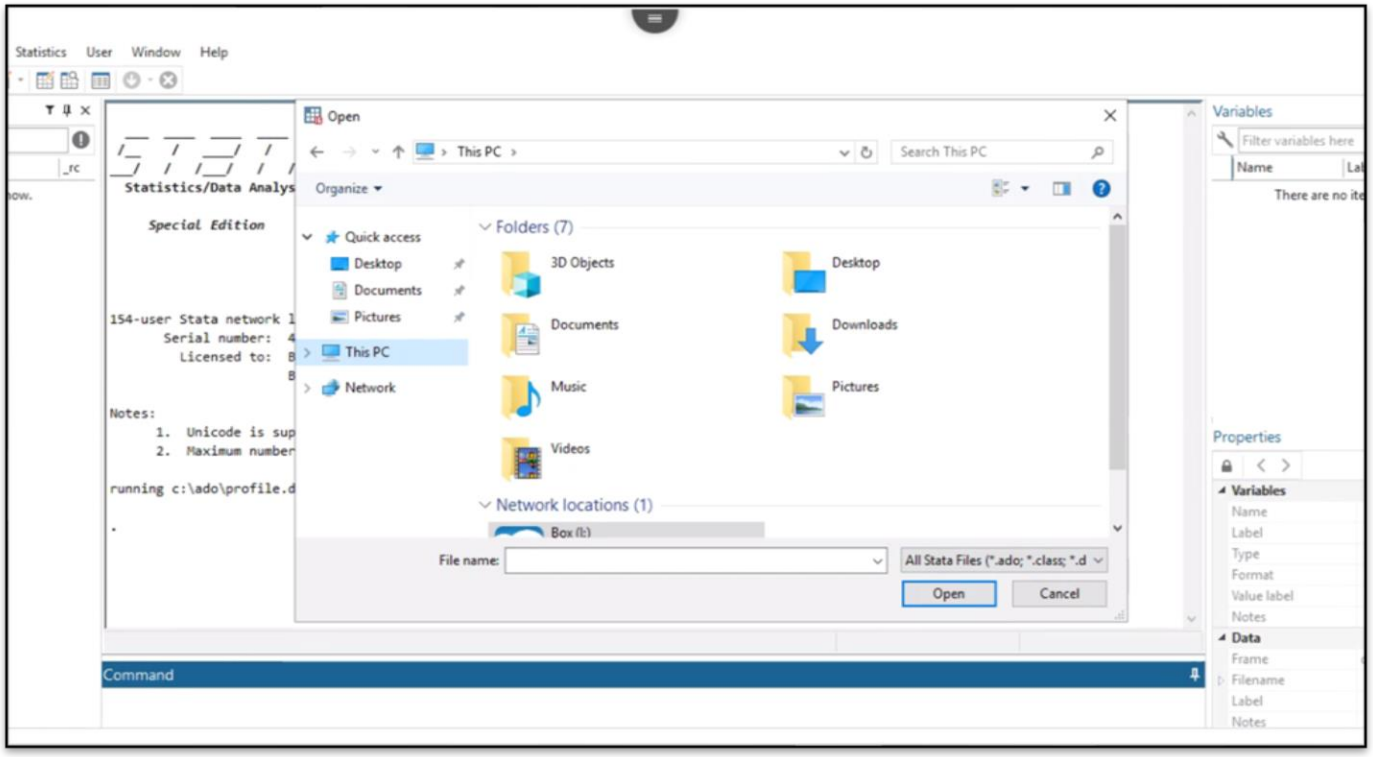
A. What is Stata

- Stata is a data processing software that is primarily used by economists to analyze and process data.
- Some advantages of Stata over other coding language are that it contains many popular and default econometrics packages, the syntax is simple and intuitive, and results are easily reproducible because Stata is not open source
- Some disadvantages of Stata include that it can only handle one dataset at a time, it cannot handle as large of datasets as other leading software, and since it is not open source, there are fewer commands available for more complex operations.

B. How do I access Stata?

- Citrix allows users to access Stata off campus and on their own personal computers. Most school computers have Stata downloaded onto them by default.
- Go to apps.byu.edu to access the BYU Citrix server. Use your BYU email address and password to log in.
- After logging in, select StataSE from the list of applications. After opening, Stata users must connect to Box (a file storing software) in order to access files. This is done by going to kumo.byu.edu, selecting Box, then providing a netid@byu.edu to authorize Box.
- After authorizing Box, users can open and save datasets and scripts by selecting Open>This PC then This PC>Box

Figure 1: View of file explorer after installing box



C. The console and setup

- Figure 2 illustrates the Stata console and describes its functionality
- For the purpose of this example, we will utilize the auto dataset. This can be uploaded by using the command sysuse auto.dta.

Figure 2: Illustration of the console

Note: 1: The history section shows all the commands that have been performed in the Stata session. Commands that failed to execute are shown in red. The user can click on an old command to rerun it. 2: The command window can be used to run one-line statements. 3: The results window shows the results, the error descriptions, and a list of all commands that have been run. 4: The variables section contains a list of all the variables contained in the data. 5: Shows the selected variable's name, label, type, and other important information. 6: This button creates a new do-file script.

D. Using scripts in Stata

- Scripts (called "do-files" in Stata) are documents containing chunks of code and are essential in writing reproducible and readable code. They also allow the user to run multiple lines of code at once.
- Users should also leave detailed comments in the script to clearly document assumptions and rationale behind decisions. Comments are not read as code by the computer but are notes for human users of the code. Comments are shown in green.
- Comments are created by using "*" for single line comments or "/*" to open the comment for multiple line comments and "*/" to close the comment

E. Important file types

- Before users can perform any processing or analysis, they must determine the file type that the data is stored in and select the appropriate function to read in the data.
- At BYU, most data will be available in.dta, .xlsx, or .csv format, so it is important to learn the following functions to read in the data.
- use "file path.dta", *clear*
  - Only for Stata datasets (or .dta files)
  - The clear option is mandatory if another dataset is currently in use. If the clear option is used, all data will be lost from the previous dataset

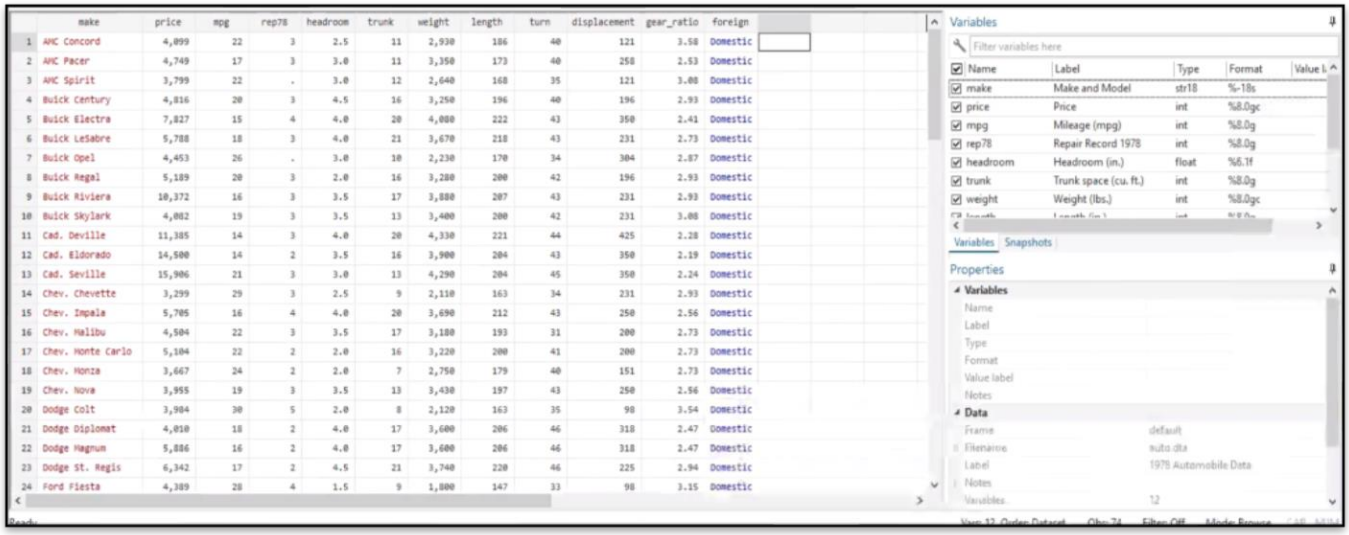
- The auto data that we are currently using is a .dta file

- import excel "file path.xlsx", *clear case(lower | upper | preserve) allstring firstrow sheet("sheet_name")*
  - Can read in xlsx, xls, and other excel workbooks
  - The clear option removes the dataset in memory
  - The sheet option allows the user to specify which sheet within the excel sheet to read in
  - The firstrow option instructs Stata that the first row of the data should be used as the variable names
  - The allstring option instructs Stata to read in all of the variables as strings (or characters) instead of numeric
  - The case option allows the user to specify if the variable names should be read in as lower case, upper case, or to preserve the case
- import delimited "file path.txt", *delimiters("delimiter") case(lower | upper | preserve) clear varnames(row of variable names or nonnames)*
  - Can be used for data structures that are separated by delimiters such as commas (csv), tab, semicolon, or many others
  - The clear option removes current dataset from memory
  - The delimiters option specifies how the data is separated. For example, a csv would use delimiters(",") to read the data
  - The case option allows the user to specify what case to use when reading in variable names
  - The varnames option allows the user to specify which row the variable names are located, or nonames to specify there are no variable names

F. Initial glance at the data

- After reading in the data, it is important to get a general idea of the structure and contents of the data. The following functions may be useful:
- browse
  - Brings up a grid view of the data. Allows the user to get a feel for the size and general contents of the data
  - Strings (or characters) are shown in red and numeric values are shown in black. Blue values are variables that have value labels or are encoded.

Figure 3: Results from the browse function



	make	price	mpg	rep78	headroom	trunk	weight	length	turn	displacement	gear_ratio	foreign
1	AMC Concord	4,099	22	3	2.5	11	2,930	186	40	121	3.58	Domestic
2	AMC Pacer	4,749	17	3	3.0	11	3,350	173	40	258	2.53	Domestic
3	AMC Spirit	3,799	22	.	3.0	12	2,640	168	35	121	3.08	Domestic
4	Buick Century	4,816	20	3	4.5	16	3,250	196	40	196	2.93	Domestic
5	Buick Electra	7,827	15	4	4.0	20	4,080	222	43	350	2.41	Domestic
6	Buick LeSabre	5,788	18	3	4.0	21	3,670	218	43	231	2.73	Domestic
7	Buick Opel	4,453	26	.	3.0	10	2,230	170	34	304	2.87	Domestic
8	Buick Regal	5,189	20	3	2.0	16	3,280	200	42	196	2.93	Domestic
9	Buick Riviera	10,372	16	3	3.5	17	3,880	207	43	231	2.93	Domestic
10	Buick Skylark	4,082	19	3	3.5	13	3,400	200	42	231	3.08	Domestic
11	Cad. Deville	11,385	14	3	4.0	20	4,330	221	44	425	2.28	Domestic
12	Cad. Eldorado	14,500	14	2	3.5	16	3,900	204	43	350	2.19	Domestic
13	Cad. Seville	15,906	21	3	3.0	13	4,290	204	45	350	2.24	Domestic
14	Chev. Chevette	3,299	29	3	2.5	9	2,110	163	34	231	2.93	Domestic
15	Chev. Impala	5,705	16	4	4.0	20	3,690	212	43	250	2.56	Domestic
16	Chev. Malibu	4,504	22	3	3.5	17	3,180	193	31	200	2.73	Domestic
17	Chev. Monte Carlo	5,104	22	2	2.0	16	3,220	200	41	200	2.73	Domestic
18	Chev. Monza	3,667	24	2	2.0	7	2,750	179	40	151	2.73	Domestic
19	Chev. Nova	3,955	19	3	3.5	13	3,430	197	43	250	2.56	Domestic
20	Dodge Colt	3,984	30	5	2.0	8	2,120	163	35	98	3.54	Domestic
21	Dodge Diplomat	4,010	18	2	4.0	17	3,600	206	46	318	2.47	Domestic
22	Dodge Magnum	5,886	16	2	4.0	17	3,600	206	46	318	2.47	Domestic
23	Dodge St. Regis	6,342	17	2	4.5	21	3,740	220	46	225	2.94	Domestic
24	Ford Fiesta	4,389	28	4	1.5	9	1,800	147	33	98	3.15	Domestic

- codebook variable list, *all mv*
  - ♦ Will give a brief description of all the variables in the dataset (or only those listed in the variable list)
  - ♦ The all option instructs Stata to provide a complete report
  - ♦ For all variables, if will provide the variable type, # of unique values, and the number of missing values
  - ♦ For numeric variables, it will also provide a range of values, mean, standard deviation, and several percentiles of the data
  - ♦ For strings, it will provide a list of all the unique data entries with their frequencies of occurrence in the data
  - ♦ If the mv option is specified, Stata will attempt to find patterns in the missing values in the data

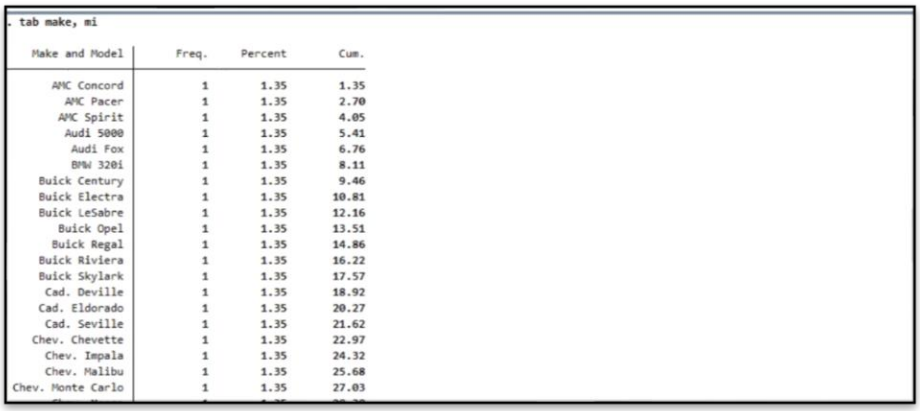
Figure 4: Results from the codebook function on a string variable

```
                Label:  1978 Automobile Data
   Number of variables:  12
Number of observations:  74
                 Size:  3,182 bytes ignoring labels, etc.

_dta:
  1.  from Consumer Reports with permission

-------------------------------------------------------------------------------
make                                                        Make and Model
-------------------------------------------------------------------------------

                type:  string (str18), but longest is str17

        unique values:  74                       missing "":  0/74

            examples:  "Cad. Deville"
                       "Dodge Magnum"
                       "Merc. XR-7"
                       "Pont. Catalina"

             warning:  variable has embedded blanks

.
```

- tabulate, *missing*

- ♦ The tabulate function is helpful for looking at the different values of string variables as well as their frequencies
- ♦ The missing option specifies to Stata that the user wants to also see how many missing values are in the dataset

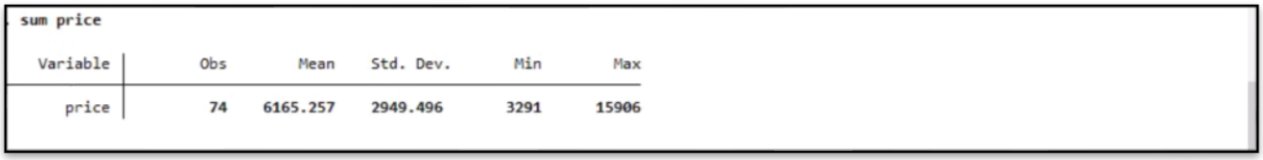
Figure 5: Results from the tabulate function



```
. tab make, mi

  Make and Model |      Freq.     Percent        Cum.
-----------------+-----------------------------------
       AMC Concord |          1        1.35        1.35
         AMC Pacer |          1        1.35        2.70
        AMC Spirit |          1        1.35        4.05
         Audi 5000 |          1        1.35        5.41
           Audi Fox |          1        1.35        6.76
           BMW 320i |          1        1.35        8.11
     Buick Century |          1        1.35        9.46
     Buick Electra |          1        1.35       10.81
     Buick LeSabre |          1        1.35       12.16
        Buick Opel |          1        1.35       13.51
       Buick Regal |          1        1.35       14.86
     Buick Riviera |          1        1.35       16.22
     Buick Skylark |          1        1.35       17.57
      Cad. Deville |          1        1.35       18.92
     Cad. Eldorado |          1        1.35       20.27
      Cad. Seville |          1        1.35       21.62
    Chev. Chevette |          1        1.35       22.97
      Chev. Impala |          1        1.35       24.32
      Chev. Malibu |          1        1.35       25.68
Chev. Monte Carlo |          1        1.35       27.03
```

- summarize, *detail*
  - ♦ The summarize function is only used for numeric variables to see basic summary statistics
  - ♦ The detail option includes more such as skewness, kurtosis, and more detailed percentiles.

Figure 6: Results from the summarize function



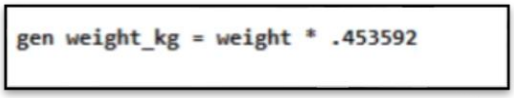
```
. sum price

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       price |         74    6165.257    2949.496       3291      15906
```

II. Data Processing

A. Generating, egen, and replace

- Variables can be created from information contained in other variables or from scratch
- gen new variable name = value *if*
  - ♦ The gen command creates a new variable with a value specified by the user.
  - ♦ The if option allows the user to specify a condition that Stata will follow when the new variable is created.

Figure 7: Code for generating a new variable "weight_kg" that is the weight multiplied by the conversion factor to KG



```
gen weight_kg = weight * .453592
```

Figure 8: Code for generating a new variable price_over_6000 which is 1 if the price is greater than 6000

```
gen price_over_6000 = 1 if price > 6000
51 missing values generated)
```

Note: This variable generates missing values for all observations that do not have a price greater than 6000

- Sometimes, the generate command is insufficient for creating a variable that is a combination of multiple observations within a variable. For example, the generate function could not calculate the total price for all cars in the dataset. However, Stata has the egen function to handle these problems
- egen new variable name = function(variable) *if*
  - The function section of the code allows the user to specify a function (such as total, std, ect. See all options by typing "help egen" into the command window) and apply that function to the include variables.

Figure 9: Code for egen command to create a variable that contains the total price for all observation in the dataset

```
egen total_price = total(price)
```

Note: If the user browsed the total_price variable, they would see the value is the same for all observations in the data because the egen command typically uses all observations in the price variable to generate one value.

- The difference between the generate and egen command can potentially be difficult to understand at first. In most cases, the generate command uses arithmetic operations, constants, and variables to create a new variable; inversely, egen always requires a function to manipulate an already existing variable to create a new one
- The replace command is useful for changing the contents of an already existing variable
  - replace variable name = new value *if*
  - This command is also often used with an if statement to only change a subset of the total observations

Figure 10: Replaces the missing values from the previous example with "0"s so that all observations have a value for the variable price_over_6000

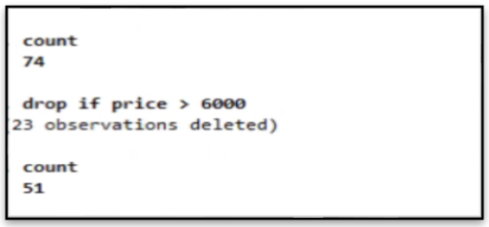
```
. replace price_over_6000 = 0 if price_over_6000 != 1
(51 real changes made)
```

B. Drop, keep, and rename

- The drop and keep command can be used for dropping and keeping variables or observations.
- To drop or keep variables, the syntax is: drop/keep variable name
  - If the drop command is used, only the listed variables will be removed from the dataset; however, if the keep command is used, all variables that are not in the list will be removed from the data

- To drop or keep observations within a variable, the syntax is: drop/keep if condition
  - Like variables, if the drop command is used, all observations that meet the condition will be removed. For the keep command, all observations that do not meet the condition will be removed
- Users should be cautious with these commands because it will remove data from the dataset. If these commands are used accidentally, the data will have to be reloaded

Figure 11: Results of using the drop command to remove 23 observations from the dataset that have a price that is greater than 6000

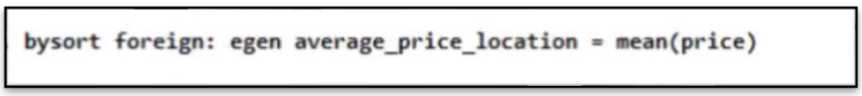


- The rename command can be used to change the name of a given variable or variables
  - rename old name new name
  - rename (old names) (new names)

C. Order, sort and bysort

- The order command allows the user to specify the order in which the variables appear in the data. This is helpful for readability in large datasets
  - order variable name
  - This command would put whatever variables contained in the list at the front of the dataset
- The sort command allows users to sort observations in ascending order based off the variable list included.
  - sort variable name, *stable*
  - The stable option should always be included for reproducibility purposes. See "help sort" for more details
  - If the user needs to sort in descending order, the gsort command may also be useful
- Occasionally, users may want to apply specific functions (like mean or standard deviation) across certain subsets of the data. The bysort command, in combination with the egen command, makes this possible
  - bysort variable to subset : egen new variable name = function(variable)

Figure 12: Code for calculating a separate average price for both foreign and domestic cars that will be stored in average_price_location



D. Dealing with duplicate observations

- Identifying and dealing with duplicate observations in data is an essential skill for any user processing data.

- Isid variable name
  - The first step in identifying duplicate observations. The Isid command will return an error if there are any observations that are duplicates

Figure 13: The results from using isid on mpg and mpg & weight

```
. isid mpg
variable mpg does not uniquely identify the observations
r(459);

. isid mpg weight

.
```

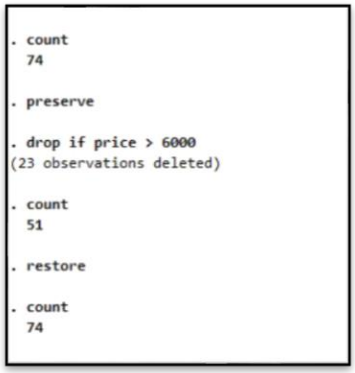
Note: The first command fails because many observations have the same mpg, but the second does not because no observations have both the same mpg and weight

- duplicates tag variable names, gen(name of marker variable)
  - This command generates a variable which marks how many duplicates a given observation has within the dataset. It is useful to combine this with isid > duplicates tag > tabulate to determine which observations are duplicates and how many there are.
- duplicates drop variable names, *force*
  - Useful for dropping the duplicate observations; however, it will return an error if a variable list is included unless the force option is used
  - It is dangerous to use this command unless the user is very familiar with the dataset. It is better to tag the duplicates and remove them from analysis.

E. Preserve and Restore

- The preserve function saves a copy of the current dataset. It may be helpful to think of preserve as a function that takes a snapshot in time of a dataset. The restore function restores the dataset to the previous snapshot
- Preserve and restore are inverse operations that allow users to get over the one dataset limitation of Stata. They are also useful for temporarily saving a dataset if it is going to be temporarily changed.
- Preserve cannot be used again until a restore is used. If the user does not want to restore to the previous dataset in memory, but does want to use a restore to keep a snapshot of the current dataset in memory, Restore, not will remove the previously used and allow a preserve to be used again.

Figure 14: Results of using a preserve and restore



```
. count
  74

. preserve

. drop if price > 6000
(23 observations deleted)

. count
  51

. restore

. count
  74
```

Note: Before the preserve statement, the dataset has 74 observations in it. After the preserve, 23 observations are removed because they do not meet selected criteria. If the user decides that he or she wants to return to the original dataset before the observations are removed, he or she can just use the restore command to return to the dataset containing all 74 observations.

III. Reorganizing, Combining, and Exporting data

A. Reorganizing

- When analyzing data, it is often important to aggregate or reorganize data in a way that makes it more readable or easier to interpret
- collapse (function) variable names 1, by(variable names 2) *fast*
  - The collapse function aggregates the value of variable name 1 based on variable name 2. This aggregation is done by using a function such as sum, mean, max, min etc.
  - For example, in Figure 15 the mean of weight is collapsed by foreign versus domestic cars. Instead of 71 observations and over 10 variables before the collapse, the dataset is reduced to 4 observations with 3 variables.

Figure 15: Results from collapsing weight by foreign and price over 6000

	foreign	price_o~6000	weight
1	Domestic	No	3,063.7
2	Domestic	Yes	4,005
3	Foreign	No	2,118.5
4	Foreign	Yes	2,601.1

  - Another important function of collapse is that it changes the dataset in memory. If the dataset is not preserved or saved before using a collapse, the previous contents of the dataset will not be accessible and will need to be reloaded. Therefore, it is common to use a preserve statement before collapsing the data.
  - The fast option allows the collapse command to compute the new dataset faster and should be used with large amounts of data to increase efficiency.
- contract variable names

- The contract command is like the collapse command, but it is used with strings. It allows users to see which combinations of strings appear in the dataset and at what frequency.
- Like the collapse command, the contract command changes the dataset in memory. Therefore, it is important to typically preserve the dataset before using the function.

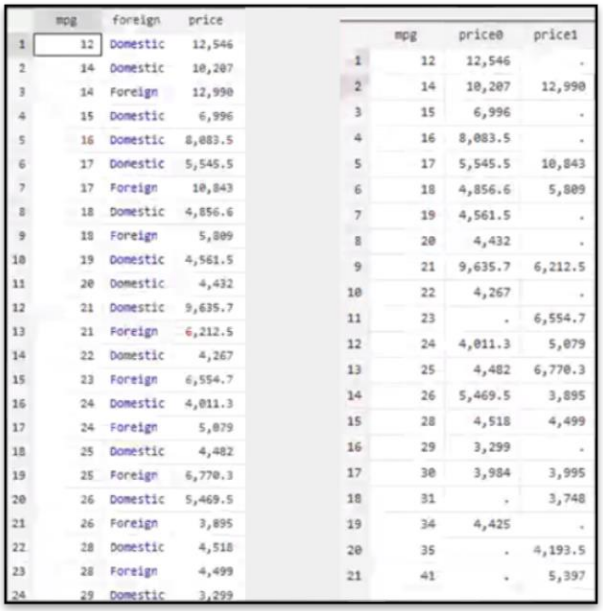
Figure 16: Results from a contract statement on make and foreign

	make	foreign	_freq
1	AMC Concord	Domestic	1
2	AMC Pacer	Domestic	1
3	AMC Spirit	Domestic	1
4	Buick Century	Domestic	1
5	Buick Electra	Domestic	1
6	Buick LeSabre	Domestic	1
7	Buick Opel	Domestic	1
8	Buick Regal	Domestic	1
9	Buick Riviera	Domestic	1
10	Buick Skylark	Domestic	1
11	Cad. Deville	Domestic	1
12	Cad. Eldorado	Domestic	1
13	Cad. Seville	Domestic	1
14	Chev. Chevette	Domestic	1
15	Chev. Impala	Domestic	1
16	Chev. Malibu	Domestic	1
17	Chev. Monte Carlo	Domestic	1
18	Chev. Monza	Domestic	1
19	Chev. Nova	Domestic	1
20	Dodge Colt	Domestic	1
21	Dodge Diplomat	Domestic	1
22	Dodge Magnum	Domestic	1
23	Dodge St. Regis	Domestic	1
24	Ford Fiesta	Domestic	1

Note: If there were certain makes that were made both foreign and domestically, there would be two entries for a given make. Furthermore, if there was more than one observation

- reshape wide/long variable name 1, i(variable name 2) j(variable name 3) *string*
  - The reshape command allows the user to change the structure of the data, but it does not aggregate the data in any way. Instead, it can change the data from long (few variables and many observations) to wide (many variables and few observations) or vice versa
  - This function is often used with a collapse to organize the data into a more readable format. Figure 17 shows an example of a reshape wide, but it may take some practice before it is clear how this command works.
  - Generally speaking, variable name 1 is the data that should be contained within the table (typically numeric), variable 2 is the variable that should be the row index, and variable 3 is the variable that should be made into several different variables based off its values.

Figure 17: Side by side comparison between long (left) data and wide (right) data



	mpg	foreign	price
1	12	Domestic	12,546
2	14	Domestic	10,207
3	14	Foreign	12,990
4	15	Domestic	6,996
5	16	Domestic	8,083.5
6	17	Domestic	5,545.5
7	17	Foreign	10,843
8	18	Domestic	4,856.6
9	18	Foreign	5,809
10	19	Domestic	4,561.5
11	20	Domestic	4,432
12	21	Domestic	9,635.7
13	21	Foreign	6,212.5
14	22	Domestic	4,267
15	23	Foreign	6,554.7
16	24	Domestic	4,011.3
17	24	Foreign	5,079
18	25	Domestic	4,482
19	25	Foreign	6,770.3
20	26	Domestic	5,469.5
21	26	Foreign	3,895
22	28	Domestic	4,518
23	28	Foreign	4,499
24	29	Domestic	3,299

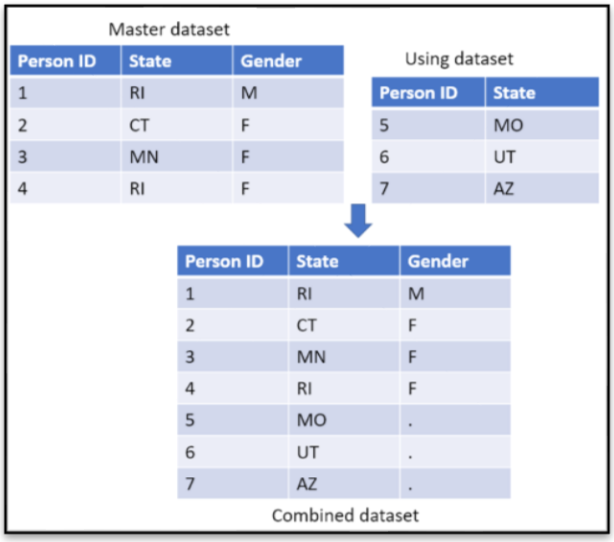
	mpg	price0	price1
1	12	12,546	.
2	14	10,207	12,990
3	15	6,996	.
4	16	8,083.5	.
5	17	5,545.5	10,843
6	18	4,856.6	5,809
7	19	4,561.5	.
8	20	4,432	.
9	21	9,635.7	6,212.5
10	22	4,267	.
11	23	.	6,554.7
12	24	4,011.3	5,079
13	25	4,482	6,770.3
14	26	5,469.5	3,895
15	28	4,518	4,499
16	29	3,299	.
17	30	3,984	3,995
18	31	.	3,748
19	34	4,425	.
20	35	.	4,193.5
21	41	.	5,397

Note: To move from long data to wide data, the code would be: reshape wide price, i(mpg) j(foreign). In this case, mpg retains its status as a variable, but foreign is split into its two levels (domestic = 0 and foreign = 1) with price.

B. Combining (Append and merge)

- Another critical skill for users to develop is combining multiple datasets appending and merging. Typically, the dataset in use is called the master dataset and the dataset that is being combined is called the using dataset.
- Appending, or a vertical merge, increases the length of the dataset (number of observations) by combining two datasets with identical variable names.

Figure 18: An example of a vertical merge, or an append.



Master dataset

Person ID	State	Gender
1	RI	M
2	CT	F
3	MN	F
4	RI	F

Using dataset

Person ID	State
5	MO
6	UT
7	AZ

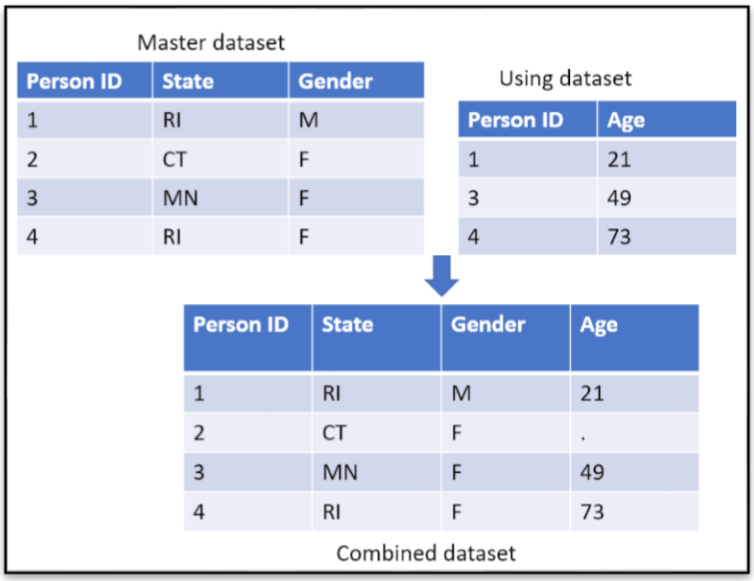
Person ID	State	Gender
1	RI	M
2	CT	F
3	MN	F
4	RI	F
5	MO	.
6	UT	.
7	AZ	.

Combined dataset

Note: The variables in common across both datasets, Person ID and State, are combined in the new combined dataset. Any variables that are only present in one are maintained, but there may be missing values in the combined dataset.

- Merging, or a horizontal merge, increases the width of the master dataset (number of variables) by using a variable that is common in both datasets.

Figure 19: An example of a horizontal merge, or a merge



Note: The variable that is used as a key variable is Person ID. Those observations of Person ID that are in common in both datasets are combined, whereas those that are unique in one of the datasets may contain missing values.

- append using "dta file name"
  - The append command combines the dataset in memory with the dataset that the user provides in the command.
  - The variable names must be identical in both the dataset in memory and the using dataset for the observations in the using dataset to be added to the dataset in memory.
- merge(1:1/ m:1 (1:m)/ m:m) key variable name using "dta file", *gen(new variable name)*
  - The merge command combines the dataset in memory with the dataset that the user provides in the command. The user must also provide a key variable on which the merge will be performed.
  - Additionally, the user must specify the type of merge: one to one, many to one, one to many, or many to many. A one to one merge indicates that there are all unique values in both datasets of the key variable. The example in Figure 19 is an example of a one to one merge. A many to one merge suggest that all values in the using dataset are unique, but they may correspond to more than one value in the master dataset. A many to many merges should rarely be used.
  - Finally, the gen option allows the user to name the _merge variable that is generate by default. This variable describes the result of the merge: 1 indicates the observation is only in the master dataset, 2 indicates the observation is only in the using dataset, and 3 indicates a successful merge. It is important to check the results of a merge and confirm that the merge worked as expected.

C. Exporting data

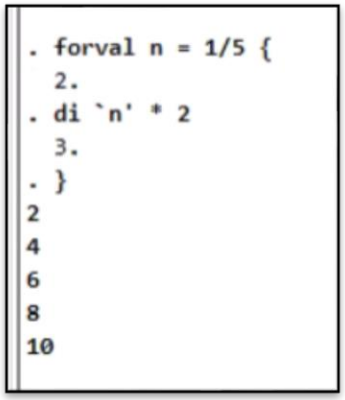
- Like importing data, there are several commands that may be useful to export data.
- save "name of file.dta", replace
  - The save command makes a copy of the Stata dataset in use and saves in the specified location as a dta file.
  - The replace option overwrites any file that is in the same location with the same name. It is useful if the user needs to make several edits to the dataset or if the script will be run multiple times.
  - It is also helpful to use the compress command before exporting a .dta file to store the variables most effectively.
- export excel using "name of file.xlsx", *sheets("name sheet") sheetreplace firstrow(variables/labels)*
  - The export excel command makes a copy of the Stata dataset in use and saves it as an excel workbook.
  - The sheets option allows the user to specify what the sheetname that will contain the data will be called.
  - The sheetreplace option allows the user to specify that if the script is run again, Stata should only replace the contents of the sheet listed in the sheets command and not the entire workbook.
  - The firstrow command allows the user to specify if the variable names or labels should be included in the excel workbook.
- There is also an export delimited command, but it is generally only useful for large datasets or if a specific file type is needed.

IV. Advanced functions

A. Loops

- Loops allow users to automate processes that are repetitive to save time and be more efficient
- Forval local macro name = starting index / ending index { process }
  - The Forval loop uses an index, or range of numeric values, to count the number of iterations the loop should perform in a given process.
  - For example, if the user provided "1/10" for the range of indexes, the process provided by the user would run 10 times. The index values are inclusive.
  - The process can be specified by the user and can use the local macro variable name to access the current value of the index.
  - The curly brackets must be included, and the final curly bracket should be on its own line.

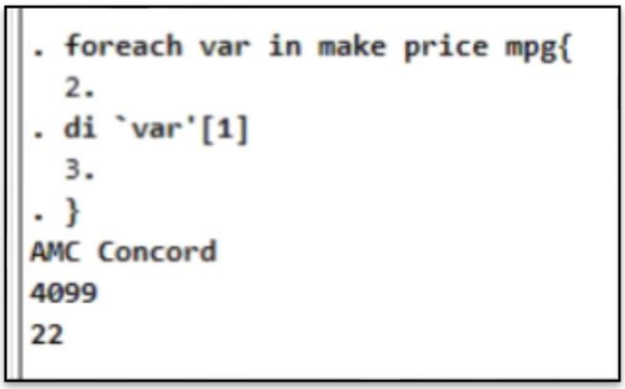
Figure 20: Example of a forval loop



Note: The results from this simple loop are “2,4,6,8,10” because it is the index multiplied by 2 (1*2 = 2, 2*2 = 4 and so on)

- Foreach local macro name in string values separated by spaces {process}
  - The Foreach uses a list of strings provided by the user and performs a process for each string value in the list.
  - The string values are often variable names so the process can be performed on variables.

Figure 21: Results from a foreach loop



Note: The foreach loop is commonly used to loop through variable names as shown here, but it can also be used to loop through strings. In this example, the first observation of the variables “make” “price” and “mpg” are displayed.

B. Strings

- There are many commands that can be used to manipulate string values. It is important to note that Stata is case sensitive when working with strings, so it is helpful to be consistent with case for observations and variables.
  - This can be done by using the upper(variable name) or lower(variable name) command for observations and reading in the variable names as lowercase in the import command.
- Tostring variable name, *replace gen(new variable name)*
  - The tostring command changes any numeric variable into a string variable and either replaces the existing variable or can generate a new variable.

- A command with similar syntax, destring, attempts to change a string variable into a numeric variable. This command may generate missing values if certain observations are unable to be changed into numeric.

- Substr(variable or macro name, starting character, number of characters to extract from starting character)
  - The substr command is useful for extracting a portion of a string variable and is often combined with a gen command to save the contents of the desired string
- Subinstr(variable name, "characters to be replaced", "characters to be used in replace")
  - The subinstr command can replace a phrase within a string with a user specified phrase. It is also typically combined with a gen command to save the contents.
- Split variable name, *parse(delimiter)*
  - Splits the specified variable into multiple new variables depending on the number of instances the delimiter. For example, if the string was "Hello world I'm John", and the delimiter was " ", then there would be 4 new variables created
  - The delimiter is by default a space, but can be changed to be a comma, dash, or anything.
- There are other more complicated functions such as the Regex command that can be used if more complex string manipulation is needed. Use online resources or the help command to find more detail.

C. Dates

- A date variable in Stata is a numeric that is recognizable by Stata but unrecognizable to a user unless it is formatted properly. There are two primary functions for creating a variable:
- Mdy(variable containing month, variable containing day, variable containing year)
  - This function can be used if there are several variables containing a numeric value that represent day, month, and year. A similar function my() can be used if there is no relevant day variable.
- Date(variable containing a string date variable "format")
  - This function can be used if there is one variable that contains a string value of the date.
  - There are many formats that a user can select to read in dates. Use "help date" to determine the necessary format.
- Format date variable %date format
  - After creating a date variable using one of the two methods listed above, the date must be formatted so that it is readable for the user. Use "help date" again to determine which format is needed for the situation. Generally, the format is %td for mdy variables.

Figure 22: Example using both date-creating methods

```
gen date_1 = date(date, "MDY")

format date_1 %td

gen date_2 = mdy(month, day, year)

format date_2 %td

tab date_1 date_2
```

	date_2									
date_1	01jan2010	02jan2010	03jan2010	04jan2010	05jan2010	06jan2010	07jan2010	08jan2010	09jan2010	10jan2010
01jan2010	1	0	0	0	0	0	0	0	0	0
02jan2010	0	1	0	0	0	0	0	0	0	0
03jan2010	0	0	1	0	0	0	0	0	0	0
04jan2010	0	0	0	1	0	0	0	0	0	0
05jan2010	0	0	0	0	1	0	0	0	0	0
06jan2010	0	0	0	0	0	1	0	0	0	0
07jan2010	0	0	0	0	0	0	1	0	0	0
08jan2010	0	0	0	0	0	0	0	1	0	0
09jan2010	0	0	0	0	0	0	0	0	1	0
10jan2010	0	0	0	0	0	0	0	0	0	1

Note: The dataset currently contains a date variable, which is string containing a day, month, and year value and three individual variables for day, month, and year. If a user wanted to make a date variable from the string variable, the mdy function would be used. Although this command would generate a functioning date, it would be impossible to read without the user also including a format statement.

V. Econometric Functions

A. Running a Regression

- Most users utilize Stata for the numerous regression options that are included by default
- Reg response variable explanatory variable(s), *options*
  - The reg command allows the user to specify a response variable and one or more explanatory variables and will output a table containing the coefficient estimates, standard errors, R^2, and other important statistics.
  - Robust is a common option that uses the White standard errors to correct for heteroskedasticity. There are also several other options for standard errors such as bootstrapping and clustering.
  - To identify other possible options, use “help reg”
- There are other regression commands such as areg, xtreg, or ivreg, but they will not be discussed in this manual.
- Similarly, there are functions for both probit and logit models that will not be discussed in detail.

Figure 23: Results from a reg function of price and weight

```
reg price weight

      Source |       SS       df       MS              Number of obs =      74
-------------+------------------------------           F(1, 72)      =   29.42
       Model |  184233937      1  184233937            Prob > F      =  0.0000
    Residual |  450831459     72  6261548.04           R-squared     =  0.2901
-------------+------------------------------           Adj R-squared =  0.2802
       Total |  635065396     73  8699525.97           Root MSE      =  2502.3

------------------------------------------------------------------------------
       price |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      weight |   2.044063   .3768341     5.42   0.000     1.292857    2.795268
       _cons |  -6.707353    1174.43    -0.01   0.995    -2347.89    2334.475
------------------------------------------------------------------------------
```

B. Fixed effects, interactions, and lags

- Fixed effect models are common in all fields of economics. Briefly, fixed effects are typically categorical variables (non-numeric variables) that are used to control for possible confounding variables in the model.
  - If the user needs to use a numeric variable (such as year) as a fixed effect, use the code i.variable name to generate a fixed effect.
  - If the desired fixed effect is in a string form, the user must create a new variable that represents all the values of the string with a numeric using encode variable name, gen(new variable name). After using the encode function, the same syntax can be used to generate a fixed effect.
- Interactions between variables are also common. An interaction attempts to measure the different effect of a given variable based on the value of another (for example, race may have a different effect on income for males and females).
  - To add an interaction term into the model, the syntax is: i.variable name#i.variable name
  - Recall that both variables must be discrete categorical variables, or the code will break.
- Lagged variables are important to include if panel data is being used and the user expects a delayed effect for one or all variables
  - First, users need to define the dataset as a panel dataset using the xtset or tsset command depending on the situation.
  - The user would use the syntax L.variable name to include a lagged version of the variable in a xtreg statement. The number of lags can be changed.

C. Outputting regression results

- After running a regression, users may wish to export the results of the regression to a word document or use the results of a regression in a future analysis.
- Outreg2 using name of new file, *replace type of file*
  - If outreg2 is not installed on the computer, use ssc install outreg2 to install the software.
  - If word is used as the type of file, Stata will output a word document with the Stata formatted tables.

- If the user wants to store part of the results in the active dataset, Stata provides a way that users can access the values of a regression using macros.
  - Users can use help reg to see the syntax for how to select various portions of the regression output.

VI. Tips and Tricks

A. Macros

- Macros in Stata are variables that can be declared by a user that contain a string or numeric value. They are also either defined as a global or local variable.
- global macro variable name macro variable value
  - Global macro variables are accessible if the Stata session they are declared in is open.
  - A global variable can be called by using the $macro variable name syntax.
- local macro variable name macro variable value
  - Unlike global macro variables, local macro variables are used for short term storage of values.
  - A local variable can be called by using the `macro variable name' syntax. Note that before the macro variable name, the " ` " is not an apostrophe, but at the end the " ' " is an apostrophe.
- Macros have many uses such as loops and organizing working directories.

Figure 24: Benefits of using macros

```
. global numeric = 1

. di $numeric
1

. gen price_1 = price /weight

. tab price_$numeric
```

price_1	Freq.	Percent	Cum.
.0536	1	10.00	10.00
.0665385	1	10.00	20.00
.0735294	1	10.00	30.00
.0745455	1	10.00	40.00
.075	1	10.00	50.00
.0835294	1	10.00	60.00
.0885714	1	10.00	70.00
.095	1	10.00	80.00
.0994737	1	10.00	90.00
.1166667	1	10.00	100.00

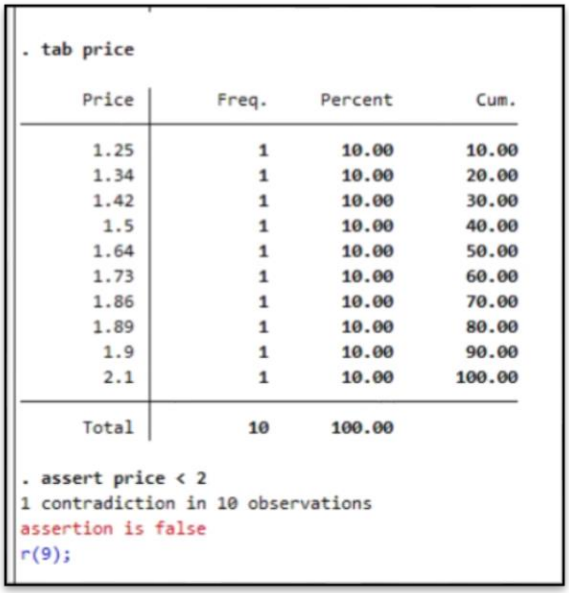
Note: A macro variable is defined as 1. A macro variable can be used to hold a value as shown with the di command or to combine in a variable name.

B. Other helpful commands

- Inrange & inlist
  - Often when subsetting data, users may wish to have a list of strings or a range of numeric values. The inlist and inrange make this possible for strings and numeric values respectively.

- For example, if the code “drop if inrange(price, 1.5,2) was used, all observations with prices between the range of 1.5 and 2 would be removed from the dataset
- Assert
  - The assert function is useful in scripts where the user wants to stop the script if certain conditions are not met. Many users use an assert command after a merge to confirm that a merge worked properly.

Figure 25: Proper use of the assert command



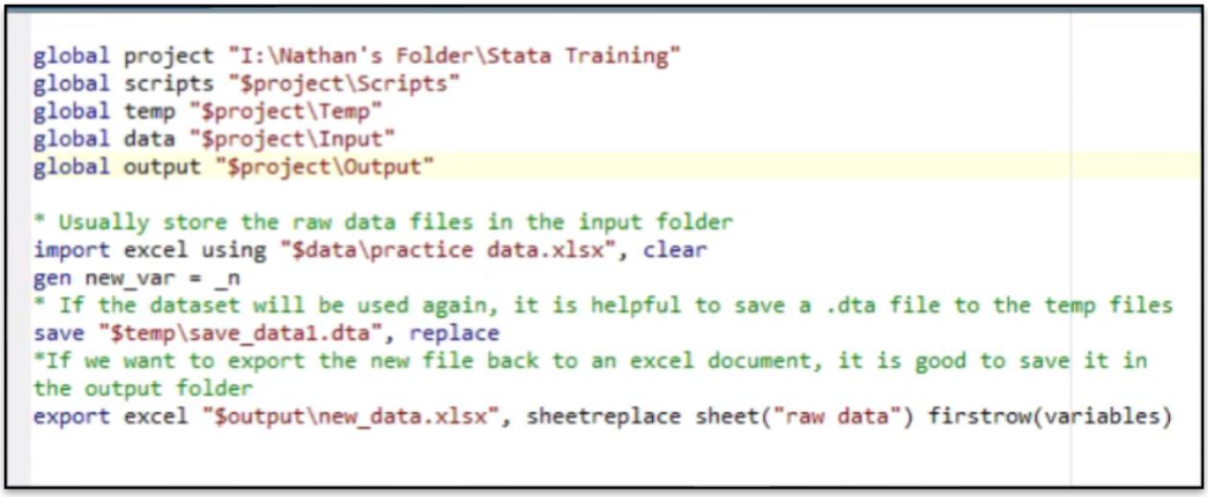
```
. tab price

      Price |      Freq.     Percent        Cum.
------------+-----------------------------------
       1.25 |          1       10.00       10.00
       1.34 |          1       10.00       20.00
       1.42 |          1       10.00       30.00
        1.5 |          1       10.00       40.00
       1.64 |          1       10.00       50.00
       1.73 |          1       10.00       60.00
       1.86 |          1       10.00       70.00
       1.89 |          1       10.00       80.00
        1.9 |          1       10.00       90.00
        2.1 |          1       10.00      100.00
------------+-----------------------------------
      Total |         10      100.00

. assert price < 2
1 contradiction in 10 observations
assertion is false
r(9);
```

Note: As shown from the tabulate function, there is one observation that is greater than 2. The assert command recognizes this and returns a contradiction.

- The *
  - The asterisk can be used to “fill in the blanks” of a variable name. For example, if a user wishes to drop all price variables (assume there are three variables for price: price, price_1, price_2), the user could use the syntax drop price*.
- Quietly
  - The quietly command can be used to suppress the default output to the results section of the consol. This command is useful whenever the user does not want any results to be outputted. The syntax is quietly function
- _n,_N, [ ]
  - The _n, _N, and [ ] notation are all related to the observation number within a dataset.
  - _n represents the index for a given observation in a dataset. For example, if the user used the code “drop if _n > 10” Stata would remove all observations in the dataset after the first 10 observations.
  - _N represents the total number of observations in the dataset. In any dataset, using the code “di _N” will reveal the total number of observations in the dataset.
  - Finally, the [ ] notation can be used to obtain the value of a specific observation within a variable. For example, if the user wants to find the value of the last observation in the price variable, he or she could use the syntax “di price[_N]”.

    - Be cautious in using these commands because they are very dependent on how the data is sorted.
- Set obs command
    - The set obs number of observations command is useful if the user wants to create a dataset from scratch. This command allows the user to specify how many observations will be in this dataset
- Organizing folder structure
    - If Stata is to be used regularly, it is helpful for every user to organize a folder structure so that files are properly mapped. It is also typical to use macros when working with multiple datasets or on multiple projects so that the paths to the data do not have to be written more than once.

Figure 26: Do file showing proper use of macros to organize folder structure



```
global project "I:\Nathan's Folder\Stata Training"
global scripts "$project\Scripts"
global temp "$project\Temp"
global data "$project\Input"
global output "$project\Output"

* Usually store the raw data files in the input folder
import excel using "$data\practice data.xlsx", clear
gen new_var = _n
* If the dataset will be used again, it is helpful to save a .dta file to the temp files
save "$temp\save_data1.dta", replace
*If we want to export the new file back to an excel document, it is good to save it in
the output folder
export excel "$output\new_data.xlsx", sheetreplace sheet("raw data") firstrow(variables)
```

Note: The first 5 lines of the code create global macros so that the user does not need to write down the file path whenever he or she is using the data. It is fairly standard to have 4 folders for every project: one for do files called “scripts”, another for temp files that save temporary dtas, one for the raw data that will not be altered, and another for any output.