Intro to Stata 4

1. **The Objectives of Stata Tutorial 4**
   - To learn how to import data (in non-Stata format) into Stata
   - To learn a bit about graphic commands
   - To learn downloadable commands

2. **Preparing for Tutorial 4**
   Let's create a directory called "tutorial4" under "C:\temp\stata-tutorial\" (under which you should find the directories "tutorial1" to "tutorial3" unless you erased them). We use new data sets auto_excel.xls and auto_fixed.txt and a new file auto_fixed.dct to demonstrate Stata commands in this tutorial. Copy both two data sets and auto_fixed.dct into the new directory "tutorial4" and set the directory as the working directory (If you are confused, you must go back to Tutorial 1).
   The two data files, auto_excel.xls and auto_fixed.txt, contain the same data as auto.dta we used in Tutorials 1 and 2, but they are different from each other and from auto.dta in data format. auto_fixed.dct is a supporting file we will use in importing auto_fixed.txt into Stata.

3. **Importing Non-Stata Data into Stata**
   Stata historically read only text (ASCII) files (except data in Stata format), but recent versions of Stata have added commands to directly import data in non-Stata formats. For example, data in an Excel spreadsheet and data in SAS XPORT format can be directly read by Stata 12 (and newer in the future).[1] If the data you want to import to Stata is in either Excel or SAS XPORT format, there are handy commands to do this. Here, I demonstrate how to import data in Excel format using the command *import excel*. How to use the command *import sasxport* is similar and if necessary refer to the help

[1]Stata 12 uses the commands *import excel* for importing data in Excel format and *import sasxport* for importing data in SAS XPORT format. In Stata 8 through 11, Stata directly reads SAS XPORT data by the command *fdause*. Also, in Stata 9 through 11, Stata directly reads an Excel speadsheet saved as *xml* format which is available in Excel 2003 and newer, by the command *xmluse*. When using the command *xmluse* in one of the older versions of Stata, you may want to use as many options as applicable to help Stata correctly read data. In my experience, Stata is prone to make errors in reading data if you do not use options.

for the command *import sasxport* for details. Next, I briefly discuss data-conversion software. Finally, I demonstrate how to import text (ASCII) data into Stata.

(a) Importing Excel Data into Stata

Let's start importing data in Excel format by the command *import excel.* The syntax is very easy. Just type the following command in the command line.

```
import excel auto_excel, firstrow
```

The option, *firstrow* tells Stata that the first row in the spreadsheet contains variable names rather than the contents of data. An Excel file has the extension of *xls* or *xlsx.* Stata first searches for auto_excel.xls in the working directory and if it cannot find it, Stata next searches for auto_excel.xlsx.

You might have noticed that the Excel file auto_excel.xls contains 0 or 1 as the contents of the variable *foreign.* If the variable *foreign* contained either "domestic" or "foreign", the command *import excel* reads the variable *foreign* as *string.* In this case, you need to use the command *encode* to convert *string foreign* to *numeric foreign.* Use the *help* menu to look up *encode.* (One exercise problem at the end of this tutorial asks you to use the command *encode.*)

(b) Data-Conversion Software

If the data you need to import to Stata are in neither Excel nor SAS XPORT format, then the easiest way is to use a data-conversion software sold in the market, such as Stat/Transfer. If you have access to one of such software, I strongly recommend you use it. All public computer terminals in the MLIC computer labs have a trial version of Stat/Transfer installed in them. Unfortunately, a trial version has limitations in converting data files, so large-sized data files would not be completely converted into another data format. However, at least a few computer terminals in the smaller PC room (Room 124) have a licensed version of Stat/Transfer installed in them. Also, the public computer terminals in the student dormitory have a licensed version of Stat/Transfer. If a trial version of Stat/Transfer gives you a warning message that the conversion is not complete, you need to use a licensed version of Stat/Transfer. After converting data into Stata format (.dta file), you simply use the command *use* to read the data into Stata.[2] There are other ways to import non-Stata data into Stata, but, as you see below, some extra work is necessary on your part, and it is typically not automatic to keep *variable labels* and *value labels* in original data files in the process of transferring data into Stata format. If you have one of data-conversion software available, how to use it is self-explanatory, and I have nothing to say in this Tutorial.[3] Below is for the case where you do not have access to such data-conversion software. Also, some data are provided in so-called fixed format in which case you cannot use a data-conversion software.

(c) Importing Data in ASCII Format into Stata

Many statistical software (such as SPSS and SAS) as well as spread-sheet software (such as Excel and Lotus 1-2-3) can export data in text (ASCII) format. Typically, data in ASCII format exported by another software is either tab- or comma-delimited or in fixed format. For an example of a data set in fixed format, open auto_fixed.txt using a text editor such as Notepad. Each row represents one car in this example. The contents of the variable *make* are stored in columns 1 through 17, the contents of the variable *price* are stored in columns 19 through 23, the contents of the variable *mpg* are stored in columns 25 through 26, and so on, so the contents of each variable are stored in a fixed physical location (in terms of columns) in the data file.[4] We do not have an exam-

[2]The version of Stata we use as of April 25, 2012 on the IUJ campus is SE Version 12. When using Stat/Transfer, choose Stata/SE as Output File Type. You can change the version of a Stata data file you are creating by doing as follows: Click on the *Options* tab, point to *Output Options (1)* in the window inside of the sheet with the *Options* tab, and select the version of your choice under *Stata Version:* from the drop-down menu.

[3]Use Stat/Transfer to convert auto_excel.xls into the Stata format. Stat/Transfer is very easy to use!

[4]In a data set with many variables, one observation is typically stored in multiple rows. As long as data are stored in fixed locations of columns and rows, the data set is considered as in fixed format. For example, each observation may need three rows to store data. The content of the first variable may be stored in columns 1 through 10 in the first row of each observation, $\cdots$, and the content of the nth variable may be stored in columns 60 through

ple of a tab- or comma-delimited data file now, but we will create such a file later when exporting a data file in ASCII format using MS Excel. In a tab- or comma-delimited data file, the content of each variable is separated by tab or comma, respectively.

Stata offers several commands to import ASCII data, and different commands are used, depending on how ASCII data are structured and whether *variable labels* and *value labels* need to be handled in the process of importing data.

First, you need to look at the structure of an importing data set. If a data set is tab- or comma-delimited, then you need to use one of the two Stata commands *insheet* or *infile.* If a data set is in fixed format, then you need to use one of the two commands *infix* or *infile.* If a data set is neither tab- or comma-delimited nor in fixed format, you must use the command *infile.* As you may suspect, the command *infile* is the most flexible command to import data, but as is often the case, it is the least handy. In this Tutorial, I am going to demonstrate the commands *insheet* and *infix.* Unfortunately, the commands *insheet* and *infix* can handle neither *variable labels* nor *value labels* in importing data. If you want to restore *variable labels* and/or *value labels* in original data sets, you must attach *variable labels* and *value labels* manually after importing data into Stata. If you want to deal with *variable labels* and/or *value labels* in the process of importing data, you must use the command *infile.* Further, the command *insheet* cannot handle a data set where a single observation is stored in multiple rows even when data are tab- or comma-delimited. In this case with a single observations stored in multiple rows where each entry is separated by a tab or comma, you must use the command *infile.* Use the *help* menu for the details of the command *infile.*

i. Importing Tab- or Comma-Delimited Data into Stata

   As previously said, you should use the command *import excel* or Stat/Transfer to import data in Excel format into Stata. Here I demonstrate another way mainly for the pedagogical purpose. Importing tab- or comma-delimited data

65 in the third row of each observation.

into Stata is the most basic data-importing task and gives you an illuminating example. First, you need to export ASCII data using Excel. Open auto_excel.xls with Excel and save it as a new file with the extension txt (tab delimited) under the working directory. You can choose whatever file name you like, but for the sake of demonstrative convenience, let's call the new file auto_excel.txt. Open auto_excel.txt with a text editor (such as Notepad), and you see that a single observation is stored in one row and that each entry of numbers within a single observation is separated by tab, which satisfies the requirements for the command *insheet* to be used for importing data. After confirming that there is a data set called auto_excel.txt in the working directory, type the following command in the command line and press enter. Then, the auto data are in Stata memory.

```
insheet using auto_excel.txt
```

A few points deserve attention. First, *raw* is the default extension for a data file after *using* in the command *insheet*. Because your ASCII file has the extension *txt*, you must specify the file name including the extension *txt*. If you rename your data file into auto_excel.raw, then you can skip the extension *raw*.

Second, the command shown above is of the simplest form. You can use the *help* menu for options available for *insheet*.

If the first row in your ASCII data file includes at least one alphabet, then Stata reads the first row as variable names instead of as the contents of variables (assuming that you do not use the option *nonames*).

Browsing the data set you have imported into Stata, you see that *make* is stored as *string*, and the rest of the variables is stored as *numeric*. You might wonder how Stata knows the data type of each variable. The command *insheet* uses a very simple rule to determine the data type of each variable. Stata looks at all entries for each variable (the entire column except the top cell), and if there is at least one entry including alphabets, then Stata regards the variable as *string*, and if all entries for a variable (the entire column except the top cell) are in numbers, then Stata regards the variable as *numeric*.

ii. Importing Data in Fixed Format into Stata

Next, we are going to import auto_fixed.txt into Stata. Please *clear* the data stored in Stata memory. auto_fixed.txt is an ASCII data set in fixed format, meaning that data are organized by physical location within the data file. We are going to use the *infix* command to import the auto data.[5] The command *infix* has some flexibility in command specification, so I show a typical example of importing data with the command *infix*.

Not surprisingly, *infix* needs information about from which file data come and where in the data file each entry of data is stored. A *dictionary* file can serve this role. The command *infix* can be used with a dictionary file. Open the dictionary file auto_fixed.dct with a text editor (such as Notepad). The first line of a dictionary file starts with

infix dictionary using *datafilename* {

where *datafilename* is the name of a data file.[6] The rest of a dictionary file is within curly brackets and tells Stata where in the data file each entry of data is located. A dictionary file contains information about which data file is used as the source of importing data as well as the physical location of each entry of data within the data file, so a *dictionary* file contains all necessary information to import data into Stata. The *dictionary* file auto_fixed.dct first tells Stata that auto_fixed.txt is the file where data are from and next tells that *make* is a *string* variable whose content is located in columns 1 through 17 in each row, *price* is a *numeric* variable whose content is located in columns 19 through 23 in each row, and the rest of the variables is similar. Note that a dictionary file must write explicitly *str* when importing a *string* variable. If you do not write *str* before *make* in the dictionary file, Stata regards *make* as a *numeric* variable and

[5]The command *infile* can also import ASCII data in fixed format.

[6]A dictionary file for the command *infile* needs different syntax.

cannot read alphabets, creating missing entries for all observations under *make*.

To import the auto data, type

```
infix using auto_fixed.dct
```

in the command line and press enter.

A few comments deserve mention. The default extension of a data set file is *raw* in a *dictionary* file, so if the extension of your ASCII data set is different from *raw*, you must explicitly write the extension in your *dictionary* file. In this particular example, you must explicitly type auto_fixed.txt when specifying a data file in a *dictionary* file. Similarly, the default extension of a *dictionary* file is *dct* in the command *infix*, so if the extension of your *dictionary* file is different from *dct*, you must explicitly write the extension in the command *infix*.

When importing a *string* variable, *infix* ignores blanks before the first non-blank and after the last non-blank. For example, the first entry of *make* is "AMC Concord" (without double quotes). Although *infix* reads all characters including blanks stored between the first column and the seventeenth column (inclusive) for the *string* variable *make*, blanks before the first non-blank (the character A in this example) and blanks after the last non-blank (the character d) are ignored, so "AMC Concord" (without double quotes) is recorded as a *string* entry of 11 characters long no matter how the entry is placed between the first column and the seventeenth column. For example, entries of "AMC Concord" (without double quotes) and " AMC Concord" (without double quotes; note a blank before A) in a fixed-format data set both result in "AMC Concord" (without double quotes) after Stata has imported data in its memory.

For *numeric* variables, all blanks are ignored no matter where they are located. For example, the first entry of *price* 4099 can be typed as 4 099, 40 99, or 409 9 as long as the entry is between the nineteenth column and the 23rd column (inclusive). If at least one non-numeric character (such as an alphabet) is included in an entry, Stata cannot read the entry and records it as missing represented by . (dot).

The command *infix* can handle a data file where a single observation is stored in multiple rows. Look up *infix* for details. (One exercise problem at the end of this tutorial asks you to import data in fixed format where each observation is stored in two lines.)

The data file auto_fixed.txt contains 0 or 1 as the contents of the variable *foreign*. If auto_fixed.txt contained "domestic" or "foreign" as the contents of the variable *foreign*, you would need to import *foreign* as a *string* variable and convert *string foreign* to *numeric foreign* by the command *encode*. Look up *encode* for details. (One exercise problem at the end of this tutorial asks you to use the command *encode*.)

4. **A bit about Graphic Commands**

Many graphic commands are available in Stata. Here, we briefly discuss menu-driven graphics rather than cover graphic commands one by one. In Stata 8 and newer, menu-driven graphics is available, and even quite complicated graphs can be created by using this facility.

I assume that the auto data are in Stata memory. To create a scatter plot of *weight* and *mpg*, choose *Twoway graph* under the *Graphics* menu. Press the *Create* button and choose *Basic plots*. Then select *Scatter* under *Basic plots: (select type)*, *weight* under X variable, and *mpg* under Y variable. Then press *Submit*. You have a nice scatter plot where you see a clear negative correlation between *weight* and *mpg*. You can copy a graph by right-clicking on the graph and paste it into MS Word and Excel.

Menu-driven graphics is not only intuitive but also gives opportunities to learn graphic commands. After creating a scatter plot of *weight* and *mpg* by the menu-driven graphics, you see the command

```
twoway (scatter mpg weight)
```

recorded in the *Review* and *Results* windows. You can re-create the same graph by typing the command in the command line or pointing and clicking on a particular command in the *Review* window.

Let's create a more complicated graph with some embellishments. Suppose you want scatter plots of *weight* and *mpg* separately for domestic and foreign cars. Suppose also you are not interested in cars with price $10,000 or higher. You can easily create the scatter plots you want with appropriate titles. Choose *Twoway graph* under the *Graphics* menu. Press the *Create* button and choose *Basic plots*. Then select *Scatter* under *Basic plots: (select type)*, *weight* under X variable, and *mpg* under Y variable. Click the *if/in* tab and type "*price*<10000" (without double quotes) under *If: (expression)*. Then, press the *Accept* button. Make sure that the plot you are making (some name like Plot 2) is highlighted under *Plot definitions:*. Click on the *By* tab to move the *By* page forward. Check the box on *Draw subgraphs for unique values of variables*. Choose *foreign* under *Variables*. Click on the *Titles* tab, and type "Heavier Cars Need More Gas" (without double quotes) under *Title*, and "Domestic (0) & Foreign (1) Cars Separately" (without double quotes) under *Subtitle*, and "prices less than $10,000" (WITH double quotes) under *Caption*. Then press *Submit*. You can press *Submit* when you want the menu-driven-graphic window to stay there even after a graph is created, so you can create another graph immediately. Otherwise, press *OK*.

5. **Downloadable Stata Commands**

   Finally, I want to discuss one excellent Stata feature you should know. Besides official commands that are already built in within Stata when you install a Stata package into your computer, many user-written commands are available to download through internet. They are easy to install, and you can use them as if they are official Stata commands once you install them. In the future, you might want to submit your own commands to the depository, so Stata users across the world can access them. The Stata Corporation maintains a depository of user-written commands, which is a collection of user-written commands published in a journal called *Stata Journal* formally known as *Stata Technical Bulletin*.[7] Besides the depository administered by the Stata Corporation, many users around the world make their user-written commands available to other Stata users through internet.

   Let me guide you to download a very useful user-written command, which is called *outreg2*. This command saves you a lot and a lot of tedious work when you format regression outputs into a nice table. There are many ways to access user-written commands. One way is to use the command *findit*. I assume your computer is connected to

[7] *Stata Journal* and *Stata Technical Bulletin* are sometimes abbreviated as *SJ* and *STB*, respectively.

internet. The command *findit* searches all available Stata resources including user-written commands not only published in *Stata Journal* and *Stata Technical Bulletin* but also made available through web resources by Stata users all over the world. You can type any keywords after *findit*. Suppose you want to check whether there is a nice command to format a regression table. Then, type

findit regression table

in the command line. Stata returns a *Viewer* window, in which you see a list of commands that is relevant to our key words "regression table." Looking at entries one by one, you find the entry titled (in blue fonts)

outreg2 from http://fmwww.bc.edu/RePEc/bocode/o

in the middle or in the two-thirds toward the end of the list. Click the blue fonts and in the next window, click *click here to install* (blue fonts). Now the download and installation are done. You can use the command *outreg2* as if it is an "official" command. Use *help* for the details of *outreg2*.

6. **Use *help* or *findit* for Searching Stata Commands**

   Besides the commands we have covered in this series of Tutorials, there are many other commands available in Stata for both data management and analysis. Use *help* or *findit* to find Stata commands of your needs. For example, you can easily find using either *help* or *findit* that *regress* is the command for running an OLS regression.

7. **Tutorial 4 Homework**

   (a) cancer_ascii.txt (which is provided with this Tutorial in the course folder) is an ASCII data set that is collected to test the effectiveness of a new drug. (I think data are real.) The data set includes observations for four variables: *studytime* (months to death or the end of the experiment), *died* (1 if patient died, and 0 otherwise), *drug* (drug type: placebo, drug A, or drug B), and *age* (patient's age at start of the experiment) for 48 patients. The data file cancer_ascii.txt contains data for these variables in the order of *studytime*, *died*, *drug*, and *age* for each individual. Data for each individual are stored in two lines. *studytime* and *died* are stored in the first line in the first through second columns (inclusive) and in the ninth column, respectively. *drug* and *age* are stored in the second line in the first through seventh columns (inclusive) and in the ninth through tenth columns (inclusive), respectively. Create and run a *do* file that implements the following tasks. You must use asterisk (*) to indicate the question numbers i. through iii. in your *do* file. You will be penalized for not using asterisk (*) to indicate the question numbers. Please also put your name and ID in the top of your do file using asterisk (*).

  i. In cancer_ascii.txt, a single observation is stored in two rows. For each patient, the first entry is *studytime*, the second entry is *died*, the third entry is *drug*, and the fourth entry is *age*. Import the ASCII data (cancer_ascii.txt) into Stata. You need to use the *help* function to complete this task.
  ii. After importing the data, use the command *encode* to convert *string drug* to *numeric drug.* This process needs a few steps. First, create the *numeric* version of *drug* with the variable name *drug1.* Then, *drop* the original *string drug.* Finally, *rename drug1* to *drug.*
  iii. Create a histogram of *age* separately for alive and dead patients, so you can have a sense of potential differences in the age distribution between alive and dead patients. Hint: Select *Histogram* under the *Graphics* menu.

(b) If you have questions, stop by one of TA's. Stata Exercise 4 is due at the beginning of the class on May 2 (Wednesday). No late assignment will be accepted. You need to turn in a hard copy of your result file (convert your *smcl* file into a *txt* file). A hard copy of a histogram is not necessary. You MUST create your result file from a *do* file. (To create an output to be submitted as homework, you must complete a *do* file first and then use the command *do* to run the *do* file. Please do not work interactively with Stata to create an output to be submitted as homework.) If your result file is created interactively with Stata, you get zero credits from this assignment. Please do not include error commands in your *do* file.

End of Tutorial 4