Center for Digital Scholarship
Hesburgh Libraries
University of Notre Dame

James Ng
1202B Hesburgh Library
james.ng@nd.edu

Stata Primer

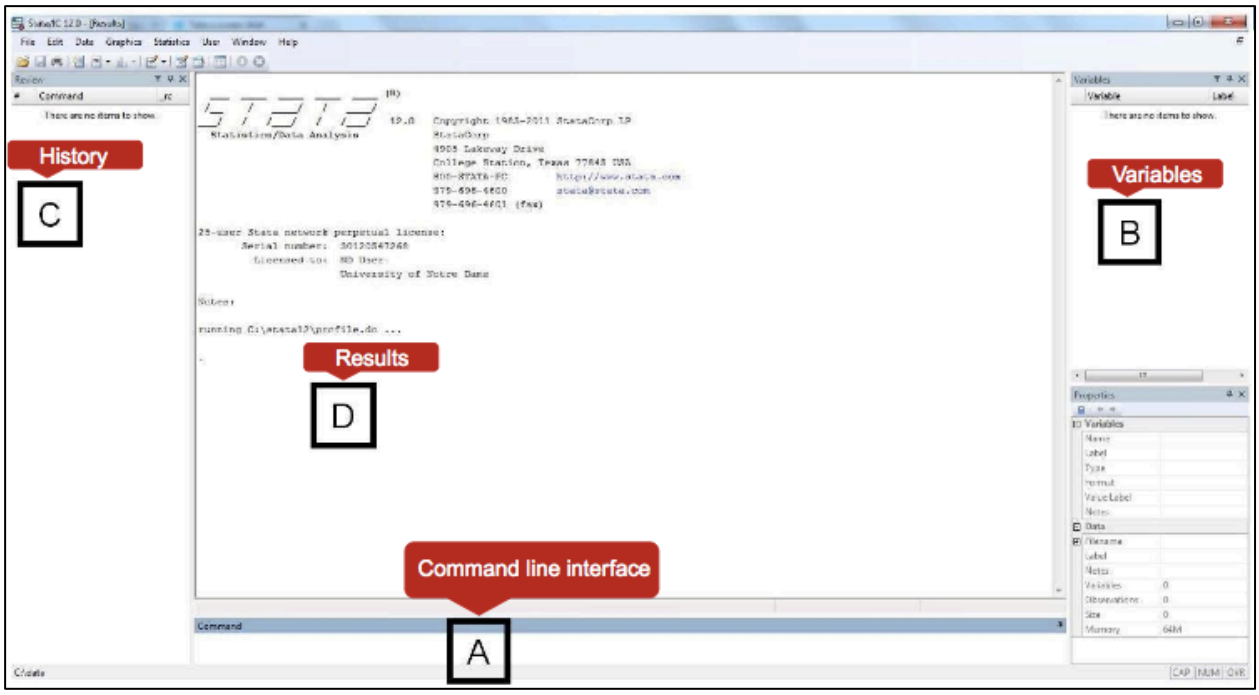
Example datasets and do-files are stored here: http://www3.nd.edu/~jng2/workshop/

Part 1: Getting Started

Stata is a statistical software package widely used by economists, other quantitative social scientists (e.g. sociologists, political scientists), and epidemiologists. One of the most attractive things about Stata is its extensive collection of commands that can be used to easily accomplish virtually any manipulation and analysis of data that an economist would need. This is no accident: Stata was written by a bunch of economists in 1985, so it is geared towards the needs of social scientists.

The Stata Environment

Box A: This is the command line interface. This is where you type in your commands, similar to what you would do on any Unix-type terminal.
Box B: This is where all the variables in the dataset that is currently in memory are displayed.
Box C: This is where a history of all commands you have entered is displayed. This history vanishes when you terminate Stata.
Box D: This is where the result of each command entered is displayed.



Layout of Data

You can think of data loaded into Stata as a spreadsheet, with each row containing an observation, and each column containing a variable.

Variable Names

Variables are case sensitive. For example, a variable named AGE is distinct from a variable named age. Variable names can contain only alphanumeric characters and cannot start with a number. Exception to the rule: the underscore is allowed in variable names.

Sending Commands

You instruct Stata to accomplish specific tasks by sending specific commands. There are three ways to send commands in Stata:

1. Point and click using the drop down menus.
2. Enter commands in the command line interface.
3. Create a "do-file" containing commands, which you will then execute.

Do-Files

Anytime you expect to work on a project in more than one sitting, you should use a do-file. Why should you work with do-files?

A do-file contains every command that you ever used for your project, from the very first step (loading data) to the very last (exporting your results). It documents every step you took in the process of manipulating and analyzing data. If you need to modify or repeat certain steps, you simply modify your do-file appropriately instead of redoing everything.

To create a do-file, in Stata click on the icon on the toolbar that looks like this:  (Note: If you hover over

the icon, a label saying "New Do-file Editor" appears.)

Alternatively, on the top menu bar, choose Window > Do-file Editor > New Do-file Editor.

Operators

Assignment operator: =

Relational operators:
- equals to: ==
- not equals to: != (alternatively ~=)
- less than: <
- less than or equal to: <=
- greater than: >
- greater than or equal to:>=

Logical operators:
- AND: &
- OR: |
- NOT: ! (alternatively ~)

Conditional statements:
The `if` conditional statement can be used two ways:

1. As a qualifier at the end of a command—`if` at the end of a command means the command is to use only the data specified. The `if` qualifier is allowed with most Stata commands. For example, the following command regresses gdp on happy using observations between the years 1975 and 1997.
   ```
   regress gdp happy if year>=1975 & year<=1997
   ```

2. As a programming command, like you would in any other language. The syntax for `if` is strict; for illustrative examples, see
   a. Example 4 of Part 5 of this tutorial.
   b. Stata's help file for the `if` command (`help ifcmd`).

Loading Data into Stata

Here are all the commands for loading data into Stata. Which command you use depends on the file type that is being read.

Command	File Type	File Extension
`use`	Stata format	.dta (always)
`infix`	Fixed format ASCII	.dat, .raw, .fix, or simply nothing
`infile` (version 1)	Free format ASCII	
`infile` (version 2)	Fixed format ASCII, with a "dictionary"	
`import delimited`	Text-delimited (comma, tab, space, etc.) ASCII	
`import excel`	Excel	.xls, .xlsx

Getting Help

Stata has a very extensive built-in help system. To pull up the help file for a particular command, say describe, simply enter `help describe` into the command line interface. Googling also usually yields useful results.

Part 2: Manipulating Data and Plotting Line Graphs

The goal of this exercise is to produce **Table 1** and **Figure 1** below.a

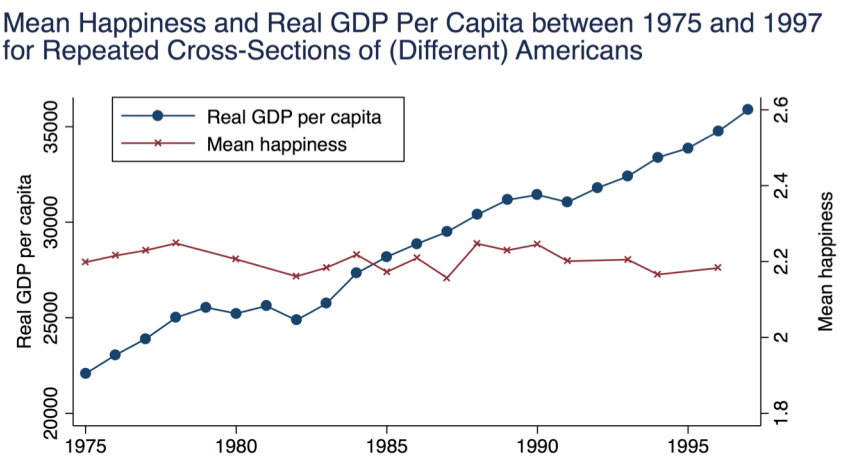
You will learn:

a. How to merge two datasets
b. How to generate and label new variables
c. How to "collapse" observations
d. How to plot a graph

Table 1: Annual real GDP per capita and mean happiness, 1975-1997

year	rgdppc	meanhappy
1975	22075.38	2.19798
1976	23055.56	2.215477
1977	23896.55	2.229208
1978	24999.11	2.247858
1979	25538.14	
1980	25220.93	2.205882
1981	25615.52	
1982	24869.67	2.160647
1983	25745.56	2.183725
1984	27333.99	2.217993
1985	28183.51	2.172549
1986	28864.97	2.20911
1987	29485.49	2.156742
1988	30390.34	2.24693
1989	31163.08	2.229358
1990	31430.6	2.244673
1991	31046.02	2.200798
1992	31796.23	
1993	32392.62	2.204872
1994	33384.2	2.165603
1995	33868.18	
1996	34749.89	2.183016
1997	35882.78	

Figure 1



Notes: Right-hand scale is the average of the answers to the question from the United Sates General Social Survey: Taken all together, how would you say things are these days--would you say that you are (3) very happy (2) pretty happy, or (1) not too happy? Real GDP per capita is measured in 2005 US dollars.

Source: http://www.jstor.org/stable/30033632?origin=JSTOR-pdf

STEP 1

- Start a new do-file and save it in a folder of your choice.
- On the very first line of the do-file, type the following:

```
cd Your_Folder
clear all
set more off
```

  where `Your_Folder` is the full path to a folder of your choice. If this path contains whitespace, you will have to enclose the entire path in double quotes.
- The `cd` command stands for "change directory"; it instructs Stata to treat `Your_Folder` as the current working directory so that any files generated in the current session are saved there.
- To find out your current working directory, use the `pwd` command.
- To get Stata to execute a single line or successive lines of code from a do-file, highlight the line/s and click the "Do" button in the menu bar of the do-file.

STEP 2

- Load the following dataset into Stata using the `use` command.[1] So that you do not lose your work, enter the command in the do-file, and then execute it.

```
use "http://www3.nd.edu/~jng2/workshop/gss_happy.dta", clear
```

STEP 3

- To see a list of all the variables, variable types, and sample size, type `describe`.
- To obtain summary statistics of all the variables in the dataset, type `summarize`.
- To browse the raw data in spreadsheet form, type `browse`.
- To obtain frequency counts of the values of the variable **happy**, type `tab happy`. Compare this to

```
tab happy, nolabel
```

STEP 4

- Notice that the values for **happy** are the opposite of what's been coded in Figure 1. We need to recode this variable. We can do so by generating a new variable called **happynew** that takes on the values that match what we see in Figure 1.

```
gen happynew = happy
replace happynew = 3 if happy == 1
replace happynew = 1 if happy == 3
```

- To confirm that **happynew** was coded correctly, cross-tabulate **happy** and **happynew**:

```
tab happy happynew, nolabel
```

[1] Stata commands to load ASCII files are `infix` and `infile`. Stata can read Excel spreadsheets with `import excel`.

STEP 5

- Data for the happiness variable are at the person level, but Table 1 and Figure 1 show annual average happiness across all persons at the country level.
- To obtain average happiness across all persons for each year, use the `collapse` command, like so:

```
collapse (mean) meanhappy=happynew, by(year)
```

- This collapses the data and generates a variable, **meanhappy**, which contains annual average happiness at the country level.
- You can label the variable to make it more descriptive for yourself and others.

```
label var meanhappy "Mean happiness: 3-very, 2-pretty, 1-not too"
```

STEP 6

- We only need observations from 1975 through 1997, so we will keep only observations from that range of years.

```
keep if year>=1975 & year<=1997
```

- Finally, save the current data in a new file using the `save` command as below. This file will be saved in your current working directory. The file will be in Stata format and have the extension *.dta (you do not have to specify the extension).

```
save gsshappy, replace
```

STEP 7

- Stata only handles one dataset at a time. To use variables stored in two separate datasets, you must combine the datasets.
- Remember that our end goal in this exercise is to combine the happiness and GDP datasets.
- We merge two datasets across observations using the `merge` command. Think of it as adding new columns to an existing spreadsheet.[2]
- To merge two datasets, you need a variable (or set of variables) that is common to the two datasets. This variable is the "identifier"; it identifies each observation (row).
- Then, you will need to figure out whether each observation of the identifier variable appears only once, or whether it repeats. You do so using the `duplicates report` command.
- In the present context, our goal is to merge annual GDP data to annual happiness data. Therefore the identifier variable is **year**.
- In the happiness dataset that we have open, it is obvious that **year** uniquely identifies each observation when we browse the raw data.
- To confirm that the observations are uniquely identified by the **year** variable, type

```
duplicates report year
```

[2] If you need to add new observations to existing data, the command for that is `append`. Think of this as adding new rows to an existing spreadsheet.

- This returns the following result, which tells us that 18 observations (rows) in the data have unique **year** values, and that there isn't a year value that repeats over multiple observations. Conclusion: in **gsshappy.dta**, **year** uniquely identifies each observation.

```
--------------------------------------
   copies | observations       surplus
----------+---------------------------
        1 |           18             0
--------------------------------------
```

STEP 8

- Stata only handles one dataset at a time. Now that we have saved the data containing happiness, we will work with GDP data, and then merge the GDP data with the happiness data.

```
use "http://www3.nd.edu/~jng2/workshop/data/pwt_gdp.dta", clear

keep if year>=1975 & year<=1997

gen rgdppc = rgdpna/pop
label var rgdppc "Real GDP per capita"
```

STEP 9

- Verify that the observations are uniquely identified by **year** variable (see STEP 7).

```
duplicates report year
```

STEP 10

- We are now in a position to merge the GDP data to the happiness data stored in **gsshappy.dta**.

```
merge 1:1 year using gsshappy
```

- Stata refers to the dataset that you have open as the "master dataset". It refers to the dataset that you add to the master dataset in a merge operation as the "using dataset".
- Here, the master dataset is the GDP dataset that we currently have open, and the using dataset is **gsshappy.dta**.
- We just performed a "one-to-one" merge. We knew that a one-to-one merge was what was needed because the identifier variable used to merge the two datasets, **year**, uniquely identifies observations in both the master and using datasets.
- If the situation calls for it, you would perform a "one-to-many" (1:m), "many-to-one" (m:1), or "many-to-many" (m:m) merge.

STEP 11

- If you browse the resulting data, you will find that it contains the information in Table 1.
- To sort the data in ascending order of year, type

```
sort year
```

- To output the data to an Excel spreadsheet, use the `outsheet` command.

```
outsheet year rgdppc meanhappy using table1.xls, replace
```

STEP 12

- The command to plot a graph is `twoway` followed by a subcommand that specifies the type of graph.
- To plot a connected line graph of GDP over time, type

```
twoway connected rgdppc year
```

- To plot happiness over time, type

```
twoway connected meanhappy year
```

- We need these two plots in the same graph. To do so, the basic command is (in one line):

```
twoway ( connected rgdppc year, yaxis(1) ) ( connected meanhappy
year, yaxis(2) )
```

- Alternatively,

```
twoway connected rgdppc year, yaxis(1) || connected meanhappy
year, yaxis(2)
```

- To save the graph in Stata's *.gph file format, type

```
graph save happygdp, replace
```

- To export the graph to an external format such as PNG or PDF, type

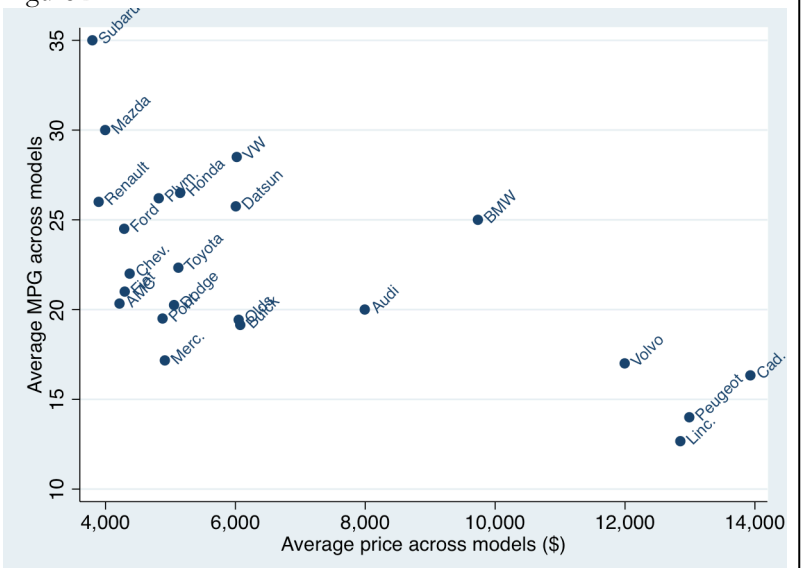
```
graph export happygdp.png, replace
```

- Complete commands are given in the do-file, **happygdp.do**, which you can download **here**.

Part 3: Plotting Scatter Graphs

The goal of this exercise is to produce **Figure 2** below.

Figure 2



STEP 1

- Load the following dataset into Stata using the `sysuse` command.[3]

```
sysuse auto, clear
```

STEP 2

- Using the string function called `word`, extract automakers from the variable **make**, saving them in a new variable called **automaker**.[4]

```
gen automaker = word(make,1)
```

STEP 3

- Next, collapse the data to obtain the average **price** and **mpg** across models for each **automaker**.

```
collapse (mean) avg_price = price avg_mpg = mpg, by(automaker)
```

STEP 4

- Produce the graph using the `twoway scatter` command. You can break up a long command using three slashes, as the code below shows.

```
twoway scatter avg_mpg avg_price, ///
mlabel(automaker) mlabangle(45) ///
ytitle("Average MPG across models") ///
xtitle("Average price across models ($)")
```

[3] The `sysuse` command loads into memory an example dataset from Stata's default directory on your computer's hard drive. It is very useful for playing around with commands. To see a list of all available example datasets, type `sysuse dir`.

[4] For a list of string functions, type `help string`.

Part 4: Tabulating Summary Statistics and Analytical Results

The goal of this exercise is to produce publication-quality tables of summary statistics and regression results. We will first produce **Table 2** followed by **Table 3**. Refer to the do-file titled **nicetables.do** found here.

Table 2. Summary Statistics, NLSW88

Age	39.15
	(3.060)
Hourly wage	7.77
	(5.756)
Race:	
White	0.73
	(0.445)
Black	0.26
	(0.438)
Other	0.01
	(0.107)
College graduate	0.24
	(0.425)
N	2246

Standard deviations in parentheses.

Table 3. Regression Results

	(1)
	Dependent variable: Wage
Age	-0.0738
	(0.0382)
College graduate	3.525
	(0.276)
Race:	
Black	-0.991
	(0.268)
Other	0.155
	(1.094)
Constant	10.08
	(1.510)
N	2246
R^2	0.078

Standard errors in parentheses
The omitted race category is white.

These tables were produced using the `estout` suite of commands. It is an add-on; you can install it by simply typing `ssc install estout`.

Part 5: Coding and Using Loops

The goal of this exercise is to introduce you to the use of for-loops. Generally speaking, you can perform almost every manipulation imaginable using commands, without having to code up a loop.[5] That said, loops are still very useful for performing repetitive tasks. For a simple example, see Example 1 below.

Loops in Stata adhere to the same general principles as in other programming languages. The following are the three types of loops in Stata. The use of each is best demonstrated using very simple examples.

- `foreach`
- `forvalues`
- `while`

Example 1: `foreach`

[5] The `egen` suite of commands is particularly useful. Check it out.

```
*Objective: Attach the prefix _78 to the variables price and mpg
sysuse auto, clear
foreach v in price mpg {
     rename `v' `v'_78
}

*The following is equivalent:
foreach v of varlist price mpg {
     rename `v' `v'_78
}
```

Example 2: `forvalues`

```
*Objective: Count the number of observations in this dataset
sysuse auto,clear
local counter = 0
local N = _N

forvalues i = 1 / `N' {
   local counter = `counter'+1
}

display `counter'
```

Example 3: `while`

```
*Objective: Count the number of observations in this dataset
sysuse auto,clear
local counter = 1
local N = _N

while `counter' < `N' {
   local counter = `counter'+1
}

display `counter'
```

Loops can be nested. For example:

Example 4: Nested loops

```
*Objective: Count the number of foreign and domestic cars
sysuse auto, clear
local count1 = 0
local count0 = 0
local N = _N

forval orig = 0/1 {
     forval row = 1/`N'{
          if foreign[`row'] == `orig' {
               local count`orig' = `count`orig'' + 1
          }
     }
}

display "No. of foreign cars:`count1'"
display "No. of domestic cars:`count0'"
*You can obtain the same information using the following command:
bysort foreign: count.
```

Loops are always used with **macros**. A macro has a "macro name" and "macro contents". Everywhere a punctuated macro name appears in a command—punctuation is defined below—the macro contents are substituted for the macro name.

The contents of global macros are defined with the global command and those of local macros with the local command. See the above examples.

Global macros, once defined, are available everywhere until you quit Stata.

Local macros exist solely within the process in which they were first defined. A process may be an interactive Stata session, a do-file, a loop, or a program.

To see all the existing macros in your current Stata session, type `macro dir`.

Local or global macro -- which to use?

- Local macros are preferable in most situations.
- In loops, definitely use local macros, not global!
- Only create a global macro if you want to use it in different processes (say, across multiple do-files).
- A situation in which a global macro might be useful is if you want a list of variable names to be available to multiple do-files, to be used as control variables in various regression specifications, for example.

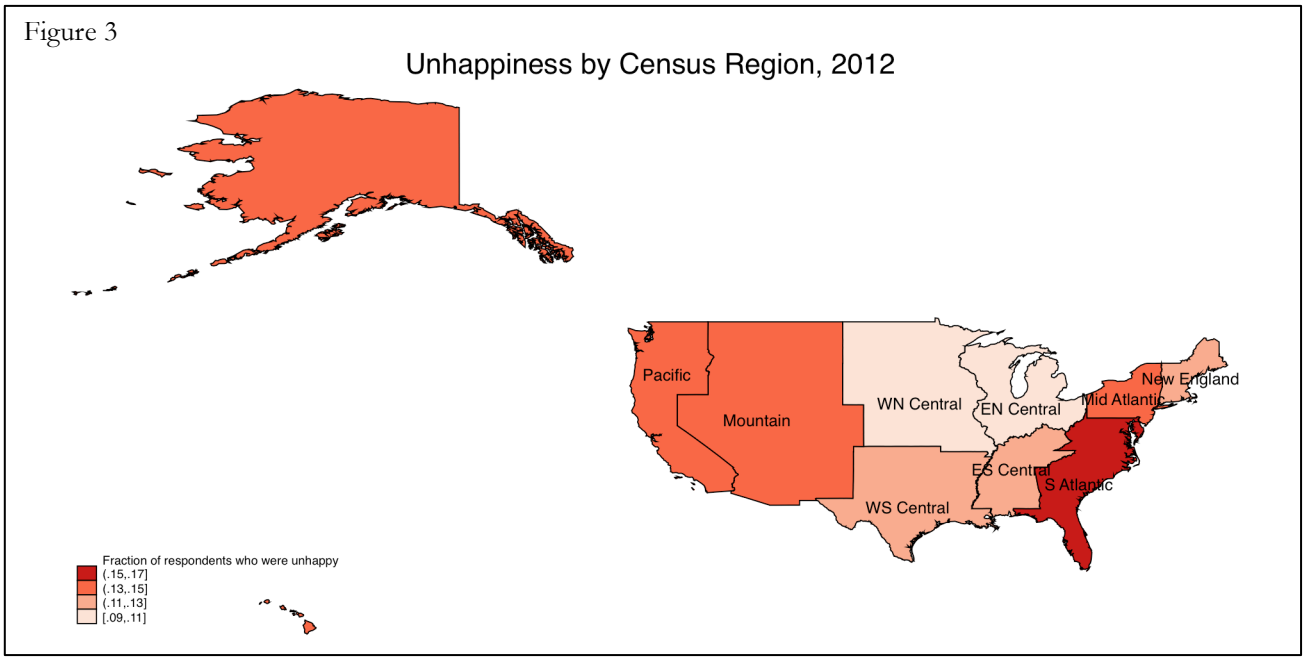
Punctuation for local and global macros

- To retrieve the contents of a global macro, the macro name is punctuated with a dollar sign ($) to the left.
- To retrieve the contents of a local macro, the macro name is punctuated with a back tick (`) to the left and apostrophe (') to the right.
- The examples above demonstrate how to define a macro and retrieve its contents.

Part 6: Producing Maps

The goal of this exercise is to demonstrate Stata's mapping capabilities. For a serious GIS project, you should NOT be using Stata, but nevertheless Stata is capable of producing basic thematic maps. We will produce the map shown in **Figure 3**.

Figure 3



To produce a thematic map, you need two things:

1. Data on the theme of interest in Stata format. This dataset must contain a variable that identifies the region you wish to map. Here, the dataset we use is **GSS2012_region.dta**. It contains data on happiness at the Census Region-level. Each Census Region is identified by the variable **subreg_id**.
2. A shape boundary file, or shapefile (a "shapefile" is actually a collection of 6 files that collectively form a GIS map). This is basically the map on which you will overlay the theme of interest. The shapefile for this exercise is found in the shapefile folder here.

STEP 1

- Refer to the do-file titled **censusregions.do**.
- Convert the shapefile to
    a. A Stata-format dataset containing information from the original dBase file that is associated with the shapefile.
    b. A Stata-format dataset containing geographical coordinates.
- The command that accomplishes this is `shp2dta`.

STEP 2

- Refer to the do-file titled **unhappinessmap.do**.
- Merge **GSS2012_region.dta** and the dBase data generated in Step 1a.
- Finally, use the `spmap` command to tie together the existing data and the coordinate data generated in Step 1b to create the desired map.