DATA MANAGEMENT IN STATA*

Paul M. Collins, Jr.
Department of Political Science
University of North Texas
pmcollins@unt.edu
http://www.psci.unt.edu/~pmcollins

July 29, 2013

CONTENTS

* Prepared for the 2013 Applied Methods Summer Series, Department of Political Science, University of North Texas.

INTRODUCTION

The purpose of this guide is to provide graduate students with a more complete understanding of issues related to good data management in Stata. To do this, we will discuss a number of topics, ranging from the relatively simple (e.g., calling data into Stata) to the somewhat advanced (e.g., transforming the unit of analysis in a dataset). I have made these notes, as well as the dataset and do files we will be using, available on my webpage (http://www.psci.unt.edu/~pmcollins/courses.htm). Note that our purpose here is not to deal with statistical models; instead we will focus solely on data management issues.

Prior to working with a dataset in Stata, there are two rules you should always follow:

1) **Know your data.** This goes for working with any data, whether it is qualitative or quantitative, in Stata, SPSS, R, SAS, etc. Before you start running models, you should know where your dataset came from, if any changes have been made to it, whether it has missing values and why, and what the unit of analysis is. By and large, this can be accomplished by reading the codebook corresponding to your data and visually inspecting the data.

2) **Use do files.** Whenever you work in Stata and you are transforming data or running models on data, you should be working through a do file. While interactivity has its place, the profession demands replicability and one easy way to meet this standard is to catalog all of the changes you make to your data so your analysis can be replicated by others (and yourself after you forget what you did). I also recommend that you keep at least one copy of your data in its original format. Because the do file contains the commands for any changes you make, this shouldn't be a problem. Keeping at least one copy of the data in its original format is useful when you want to revisit an analysis on those data (or start a new project using the data).
    a. Note: Many researchers also use log files, which catalog all of the information that crosses the screen. While I am generally ambivalent about their use, a good case can be made that they are beneficial.

For the purposes of this guide, we will be working with Harold Spaeth's *Original United States Supreme Court Judicial Database*, which contains information on the U.S. Supreme Court for the 1953-2004 terms. However, it is important to note that the discussion here is applicable to virtually any dataset. All you have to do is retrofit the commands for use with your data and you are in business.

Finally, I also want to emphasize that it is useful to work with data conversion software, such as STAT/TRANSFER, for transferring data from one format to another (e.g., from SPSS to Stata).

CALLING DATA INTO STATA

The first thing we want to do is call the data into Stata. To do this, open the do file that has been provided. Before we open data, however, we should allocate adequate memory to Stata. While various datasets require different memory allocations, 100M will suffice for our discussion. We allocate memory using the following command (note all of the commands are in the do file on my webpage).

set mem 100m

use "…allcourt_stata.dta", clear

An alternative way to call data into Stata is to use the **open** icon (denoted by a folder) on the Stata toolbar. Even if you open your data this way, it is generally useful to include details on the location of the dataset you are using in a do file.

SIDENOTE: In the do file, you will notice a series of comments that are preceded by, and end with, asterisks (*). This is the syntax for making comments that are ignored by Stata when it is working with your data, a very useful tool when you are making a lot of changes to your data.

<u>Analyzing the Data and Transforming Variables</u>

Once we have called the data into Stata, the next step is to take a look at the variables contained in the database. To do this, we want Stata to describe the data for us:

describe

The **describe** command is very useful. It gives us the number of observations in the data, the number of variables, the size of the dataset, as well information pertaining to the variables themselves.

list is another useful command, although it can be a bit unwieldy with large datasets. It will review each observation in the data.

list

The **sum** command will provide you with summary statistics for numerical variables.

sum

You can also get to know your data with the **codebook** command, which will provide you with detailed information on the variables in the data, such as the number of unique and missing values.

codebook

Note the **describe**, **list**, **sum**, and **codebook** commands can be used for single variables (e.g., **codebook us**).

On occasion, a researcher might want to reorder the variables in the dataset. For example, let's say we want to move the **chief** and **natct** variables to the first and second positions, respectively.

order chief natct

If you want to switch a variable's position with another variable, you can use the following commands:

move natct chief

With that command, we have swapped the positions of the **natct** and **chief** variables.

In Stata, variables are stored in one of six types, falling into a numeric or string category:

Numeric Variables – byte, int, long, float, double. The exact storage type for each variable depends on its numerical value (not a terribly important point).

String Variables – the type of string is represented by the prefix "str" followed by the number of characters (i.e., letters and/or numbers) in the maximum value of the variable. So, if you have a variable that appears in the data as "jerrygarcia" it is a str11 variable.

Identifying the variables that are contained in string format is very important, particularly if they are variables of interest, because Stata cannot use these variables in their original form to perform statistical analyses.

SPLITTING UP STRING VARIABLES

Let's say, for whatever reason, we are interested in the number of cases published in each volume of the Lawyer's Edition of U.S. Reports. If we look at the variable in its raw form, we can see that it is a string.

describe led

Thus, if we try to obtain summary statistics for the variable, we can't.

sum led

```
   Variable |       Obs        Mean    Std. Dev.       Min        Max
--------------------------------------------------------------------
        led |         0
```

Because **led** is a variable that is in string format, and because it contains information on both the volume and page number of each case, we have to make two transformations to the variable in order to get Stata to run summary statistics on the variable.

First, let's split the variable up.

split led, p("/")

This command creates two new variables. **led1** is the volume number for each case (the numbers that precede the slash), while **led2** is the page number for each case (the numbers that follow the slash).

Note that, because these are string variables, we need to convert them to numeric to get Stata to show us summary statistics.

destring led1, generate(ledvolume)
destring led2, generate (ledpage)

These commands creates two new numeric variables, **ledvolume** and **ledpage**, thus allowing us to examine summary statistics from these variables (e.g., sum or tab).

These commands have a variety of implications, including splitting up variables that contain a person's first and last names into two new variables (i.e., first name, last name), as well as working with data that, for whatever reason, was inputted as a string variable, but is in numeric form.

DEALING WITH DATE VARIABLES

Stata stores date variables using its numeric, long format. For example, let's look at the **oral** variable, which contains information on the date a case was orally argued:

desc oral

list oral

sum oral

When we **sum** the **oral** variable, we get a mean of 6535.008. The reason for this has to do with how Stata stores date variables. Simply put, each date corresponds to a given number. For example, October 14, 1953 corresponds to -2270. This means that date variables have to be handled a bit differently than other types of variables.

For example, let's say we want to split the **oral** variable into three variables, corresponding to the day, month, and year the case was orally argued. The **desc** command above told me that the **oral** variable is stored in a D/M/Y (day/month/year) format. Given this, I can accomplish this with the following commands:

gen day=day(oral)
gen month=month(oral)
gen year=year(oral)

Now suppose that we wanted to take these three variables and combine them into a single date variable. We can do this as follows:

generate oraldate = mdy(month, day, year)

Now we have to tell Stata to format this as a date variable (month, day, year):

format oraldate %td

To see what we did, type:

list oral oraldate month day year

Creating and Recoding Variables

Next, let's create some new variables. At the onset, I want to mention that it is best if you recode and/or create variables not by changing a variable already in the dataset, but instead by creating a new variable. In fact, I never use the **recode** command. Let's start with creating a numeric variable based on another numeric variable.

Suppose we are interested in creating a variable that indicates that a case arrived at the Supreme Court through a certiorari grant.

First, we want to look at the variable where this information is contained, **jur**.

```
desc jur
label list jur
```

These commands tell us two things. First, **jur** is a numeric variable. Second, **jur** equals 1 if the case arrived through granting a writ of certiorari. Using this information, we can now create our new variable.

```
generate certiorari=0
replace certiorari=1 if jur==1
```

We've created our new variable. Next, we will want to attach variable labels to it.

```
label variable certiorari "Case arrived at Supreme Court through a writ of certiorari"
label define certiorari 1 "Yes" 0 "No"
label value certiorari certiorari
```

We have now successfully attached variable labels to our new variable.

Recoding a variable works very much the same way. For example, say we want to recode a variable that indicates the types of cases the Court hears. First, we want to look at the variable from which this information can be obtained.

```
desc value
label list value
```

Let's say we want our new variable to reflect three types of cases: civil rights and liberties, economics, and "other."

```
generate newvalue=1 if value>0 & value<7
replace newvalue=2 if value>6 & value<9
replace newvalue=3 if value>8 & value<.
replace newvalue=3 if value==0
```

IMPORTANT: Notice the third command line. In it, I told Stata to score the variable **newvalue** a 3 if the existing variable **value** was greater than 8 and less than missing (.). I did this because Stata stores missing observations of numeric variables as extremely large positive values. If I just told Stata to score **newvalue** a 3 if **value** was greater than 8, it would recode the single missing observation of the variable **value** as a 3, which is not necessarily the issue area the case involved. Accordingly, you have to be very careful when using the greater than command. If the variable you are transforming has missing values, you should always remember to tell Stata not to recode missing values by using the less than missing (**<.**) command.

To find out and identify how many missing values a particular variable contains, enter the following commands:

```
egen missvalue=rowmiss(value)
```

This creates a new variable, **missvalue**, that is scored 1 for missing observations of the variable **value** and 0 for non-missing observations. If we use the **tab** command, we can see that there is 1 missing observation of the **value** variable:

```
tab missvalue
```

```
  missvalue  |     Freq.      Percent   Cum.
------------------------------------------------
      0      |     12,576      99.99     99.99
      1      |       1          0.01     100.00
------------------------------------------------
   Total     |    12,577     100.00
```

Having figured out how to find missing observations, we want to label our new variable.

```
label variable newvalue "Recoded Issue Area"
label define newvalue 1 "Civil Rights and Liberties" 2 "Economics" 3 "Other"
label value newvalue newvalue
```

Another way to create variables in Stata is to use the not command (~). That is, instead of specifying the values of an existing variable, we can create a new variable based on values that we do not want this variable to reflect. For example, let's create a variable that indicates the case involved a privacy issue.

```
generate privacy=.
replace privacy=0 if value~=5
replace privacy=1 if privacy==.
```

```
label variable privacy "Case Involved a Privacy Issue"
label define privacy 0 "No" 1 "Yes"
label value privacy privacy
```

Creating a new variable based on a string variable is just as easy, although it requires slightly different command language. Let's say we want a variable that indicates whether Justice Douglas authored the majority opinion; this information can be found in the **mow** variable.

generate douglasopinion=0
replace douglasopinion=1 if mow=="DOUG"

Creating a new variable from an existing string variable differs from creating a new variable from a numeric variable in two ways. First, you have to use quotes to identify the values of the string variable that will be represented by numbers in the new variable. Second, you have to be very attentive to spelling and capitalization with regard to the values of the string variable you are recoding.

Finally, we want to label our new variable.

label variable douglasopinion "Justice Douglas Authored Majority Opinion"
label define douglasopinion 1 "Yes" 0 "No"
label value douglasopinion douglasopinion

SIDENOTE: Many of the commands to create new variables used above rely on a conditional statement ("if commands"). Here is a useful list of commonly used conditional statements in Stata.

<	less than
<=	less than or equal to
= =	equal
>	greater than
>=	greater than or equal to
~=	not equal to
|	or (you can't spell out "or")
&	and (you can't spell out "and")
~	not
e(sample)	this returns only values from the estimated sample (the model most recently estimated)

CONCATENATING VARIABLES

Above, I covered how to split up and destring variables. It is often equally important to know how to create a new string variable based on two (or more) existing variables. I find this to be particularly useful for merging data.

Let's create a new variable that consists of each case's Lawyer's Edition to U.S. Reports citation and the case's docket number.

egen leddocket=concat (led docket), punct (" ")

This command created this new variable, which combines the **led** variable and the **docket** variable, separated by a space. If you do not want the new variable to separate the two variables by a space, you can leave out the **, punct (" ")** commands. Alternatively, if you want the original variable values to be separated by a comma, you can replace the **, punct (" ")** commands with these commands: **, punct (,)**

IDENTIFYING AND TAGGING DUPLICATE OBSERVATIONS

It is very common for data to contain multiple observations of the same phenomenon. Sometimes this is intentional. Other times, it is a result of oversights that occurred in the data collection process. Regardless of their cause, it is useful to know how to identify these duplicate observations.

Let's see if there are any duplicate observations of cases based on the Lawyer's Edition to U.S. Reports. To do this, we first have to sort the data based on the **led** variable, then we can tell Stata to identify duplicate observations.

sort led

duplicates report led

This gives us information on the number of duplicate observations in the data based on the **led** variable (i.e., all instances in which **led** is repeated).

Stata gives us the following table:

```
Duplicates in terms of led

--------------------------------------
  copies | observations    surplus
--------------------------------------
       1 |     6292            0
       2 |     3428         1714
       3 |      981          654
       4 |      780          585
       5 |      140          112
       6 |      324          270
       7 |       21           18
       8 |      136          119
       9 |       27           24
      10 |       80           72
      11 |       11           10
      12 |       72           66
      14 |       14           13
      15 |       60           56
      16 |       64           60
      17 |       17           16
      18 |       18           17
      28 |       28           27
      84 |       84           83
--------------------------------------
```

This tell us the following: there are 6,292 cases that appear in the data (based on the Lawyer's Edition citation) only once; there are 3,428 cases that appear in the data twice; there are 981 cases that appear in the data three times, etc.

If this was all Stata could do, it wouldn't be of much use. Instead, we want to tag the duplicate observations, so we know which cases are repeated in the data.

```
duplicates tag led, gen(ledduplicates)
```

Now we have created a new variable, **ledduplicates**, that tags each case on the basis of whether it is a duplicate observation. This variable is set up such that: 0 = not a duplicate; 1 = duplicate, appears twice; 2 = duplicate, appears three times, etc.

At this point, it is useful to introduce you to the **unique** command. This command is an add on to Stata, meaning that you need to download it. To download the command, type the following:

```
net install unique
```

This tells Stata to install the **unique** command from the internet. You can also locate the command by typing **findit unique**. We can use this command to identify how many unique values there are of a variable.

```
unique led
```

```
Number of unique values of led is 8661
Number of records is 12577
```

This tells us that there are 8,661 unique values of **led**. This command can also be combined with conditional statements. For example, to determine how many unique values of **led** there were during the 1995 term, we can type:

```
unique led if term==1995
```

DROPPING AND KEEPING VARIABLES AND OBSERVATIONS

Occasionally, a researcher might want to drop variables or observations from the analysis, usually to make the dataset more manageable or to purge it of duplicate observations. While this strategy might occasionally be desirable, it further highlights the need to keep at least one copy of the dataset in its original form; otherwise, you might never be able to get the data back into its original form.

That said, let's drop all of the duplicate observations, keeping the first occurrence of observations appearing in the data more than once.

```
sort led
duplicates drop led, force
```

We have now purged the data of the 3,916 duplicate observations.

If you want to drop variables from the dataset, the command is identical. For example, let's drop all of the variables we just created:

drop led1 ledvolume led2 ledpage day month year oraldate certiorari newvalue missvalue privacy douglasopinion leddocket ledduplicates

This command gets rid of these variables, bringing us back to the variables in the original database (although the duplicate observations are still purged).

Occasionally, you might want to drop a large number of variables from the dataset. One way to do this is through the **drop** command; you can list all of the variables you want to drop after the command (as I did above). Alternatively, you can tell Stata which variables you want to keep by using the **keep** command. For example, if I only wanted to keep the **term** and **led** variables, I could type: **keep led term**. That command would remove all of the other variables in the dataset.

TRANSFORMING THE UNIT OF ANALYSIS IN THE DATA

Often, data are set up such that the unit of analysis in the data does not facilitate answering your research question. For example, in the data we are using, the unit of analysis is the case. Often times, judicial scholars want to explain the decision making of the justices. Using the case as the unit of analysis is troublesome for these pursuits, as it assumes the Court is, in effect, a unitary actor. Accordingly, to answer justice-specific questions, you need to transfer the unit of analysis from the case to the justice-vote. Similarly, sometimes you want information on the aggregate behavior of actors (e.g., the percentage of liberal votes each justice cast during each term). This requires further manipulation of the data, changing the unit of analysis to the justice-term.

CHANGING DATA FROM WIDE TO LONG FORMAT

Let's start by changing the unit of analysis from the case to the justice-vote; we'll cover altering it to the justice-term in a bit. In Stata's language, what we are doing is changing the data from wide to long format. Wide means that there is one observation for each case; long means there is one observation for each justice's vote in each case (hence, the unit of analysis will be the justice-vote in the transformed data). Graphically, we want to transform the data from this format:

case	rehndir	stevdir	ocondir	Scaldir	kendir	soutdir	thomdir	gindir	brydir
1	0	0	0	0	1	1	1	1	1
2	1	1	1	1	0	0	0	0	0

To this format:

case	justice	dir
1	rehn	0
1	stev	0
1	ocon	0
1	scal	0
1	ken	1
1	sout	1
1	thom	1
1	gin	1
1	bry	1
2	rehn	1
2	stev	1
2	ocon	1
2	scal	1
2	ken	0
2	sout	0
2	thom	0
2	gin	0
2	bry	0

The first step is to rename variables that describe the individual justice's voting behavior. It is necessary to rename these variables as the new code seeks out j suffixes in order to make the transformation (in the following example the j suffix is "b" for behavior).

```
rename har harb
rename blc blcb
rename doug dougb
…
rename bry bryb
```

Next, we have to rename variables that correspond to j suffixes. In particular, **dir** and **lctdir** are renamed as the code will identify the ideological direction of the individual justice's voting behavior based on the "dir" suffix attached to each justice's abbreviation. Similarly, **mult_mem** and **term** are renamed as the code will identify each justice's majority or minority voting based on the "m" suffix attached to each justice's abbreviation.

```
rename dir courtdirection
rename lctdir lctdirection
rename mult_mem multcase
rename term termofcase
```

Third, we have to create a unique identification number for each case in the database.

```
gen caseid = _n
```

Now we can reshape the data from wide format (unit of analysis = case) to long format (unit of analysis = justice-vote).

```
reshape long @b @a1 @a2 @v @o @dir @m, i (caseid) j (justid) string
```

Using the **@** character, Stata locates j suffixes and recognizes that those variables vary within cases. Note that there is no need to tell Stata which variables are constant within each case; Stata recognizes this and keeps those variables without a separate command. **i (caseid**) tells Stata that each individual case is uniquely identified by the **caseid** variable. **j (justid)** tells Stata to create a new variable, **justid**, that corresponds to each justice's abbreviation and **string** tells Stata that this variable is a string (alphabetic) variable.

The final step is to perform some data clean-up. The previous code transforms the data such that there are 29 justice-observations for each case, thus increasing the number of observations in the data from 8,661 to 251,169. This code purges the data of all missing observations resulting in a final data set of 77,027 observations.

```
drop if b=="" & o==. & a1=="" & a2==""
```

To be sure, these are pretty simple, yet powerful commands.

COLLAPSING DATA

Another way of transforming the unit of analysis in a dataset is through the **collapse** command. This command is particularly useful for obtaining summary statistics at the aggregate level.

Let's say we are interested in the proportion of liberal votes each justice cast in each term. To do this, we enter the following commands:

```
collapse (mean) dir, by (term justid)
```

We now have a dataset in which the unit of analysis is the justice-term. That is, the data now contain information on the proportion of liberal votes each justice cast for each term they served on the Court.

Note that the **collapse** command can also be used to return values other than the mean. For example, if we wanted a dataset that contained the standard deviations surrounding each justice's proportion of liberal votes, we can replace **(mean)** with (**sd)**. If we want minimum or maximum values, we can replace **(mean)** with **(min)** or **(max)**, respectively.

Merging Two Datasets

The final topic we will cover involves merging two datasets, one of the most relevant topics in any discussion of good data management as the need to combine two (or more) dataset will arise countless times throughout your careers.

For our purposes, we will merge the current dataset (in which the unit of analysis is the justice-term) with a dataset that contains the justice's ideology scores, as reported by Segal and Cover.

The first step in merging data is to make sure both datasets are sorted in accordance with the variable(s) that the data will be merged on. (Note: I have already sorted the Segal and Cover data). In our case, we will merge the data on basis of the **justid** variable, which is a unique identifier of each justice in the data.

sort justid

merge m:1 justid using "…segalcoverscores.dta"

Note we are using the **m:1** command since we are merging a dataset with duplicate (many) observations of justices with a dataset in which each justice appears only once. In other words, the unit of analysis in our master dataset is the justice-term, while the unit of analysis in the using dataset is the justice.

For other situations, you might need to use a different merge command:

One-to-one merge on specified key variables:

merge 1:1 varlist using filename

One-to-many merge on specified key variables:

merge 1:m varlist using filename

Many-to-many merge on specified key variables:

merge m:m varlist using filename

After we run this command, Stata spits out the following:

```
Result                    # of obs.
-----------------------------------------
not matched                       0
matched                         475  (_merge==3)
-----------------------------------------
```

This means that Stata successfully merged all 475 observations in the data. We can also check this with the following command:

tab _merge

This command tells you how many observations in both the master (the data we have been working with all this time) and the using (the Segal and Cover) data matched perfectly.

The **_merge** command is set up such that:

1	master	observation appeared in master only
2	using	observation appeared in using only
3	match	observation appeared in both
4	match_update	observation appeared in both, missing values updated
5	match_conflict	observation appeared in both, conflicting nonmissing values

In plain English, if **_merge** returns a 1, that means that these observations only contain information from the master dataset. In other words, the using data did not merge to the master data.

If **_merge** returns a 2, that means that these observations only contain information from the using dataset. In other words, the master data did not merge with the using data. In practice, this means that the number of observations in the data have increased in accordance with the number of 2s returned and none of these observations contain information from the master data.

If **_merge** returns a 3, that means that these observations contain information from both the master and using datasets (i.e., they merged correctly).

Note that codes 4 and 5 only arise if the **update** option was used. **update** means that you are telling Stata to update values in the master dataset with data in the using dataset.

If **_merge** returns a 4, this means that these observations contain information from both the master and using datasets (i.e., they merged correctly), and any missing values in the master dataset were updated with data in the using dataset.

If **_merge** returns a 5, this means that these observations contain information from both the master and using datasets (i.e., they merged correctly), and Stata is alerting you that that there are conflicting nonmissing values in the master and using datasets.

As only 3s were returned, we know our data merged correctly.

SIDENOTE: You can also use Stata's "old" merge command, **merge**. This command allows you to merge various types of datasets without specifying the type of merge that is to be conducted. For example, if you wanted to merge the two datasets we combined above, you could do so with the following commands:

merge justid, using "…segalcoverscores.dta"

After we run this command, Stata returns the following:

variable justid does not uniquely identify observations in the master data

Anytime Stata comments on a previous command, you should take this very seriously and figure out what the program is trying to tell you. In this example, Stata is telling us that the variable **justid** does not uniquely identify the observations in the master data. We know this, because the observations in the master data are based on justice-terms, not the justices themselves. In other words, the variable **justid** repeats itself for each term a justice served (which is why we used the **merge m:1** command above).

For Further Assistance

One of the great things about working with Stata is that the people who run the program do an excellent job at providing help when you need it.

To obtain help while working in Stata, type the following command, followed by help on the topic you are searching for (in this example I am looking for help with merging data).

help merge

If you can't find what you are looking for, try the reference manual, which is sorted by topic. You can download the most recent manual here: http://www.stata.com/manuals13/r.pdf

If you are still stuck, Stata's website has an excellent frequently asked questions section (http://www.stata.com/support/faqs/) and Stata has even launched a YouTube channel (http://www.youtube.com/user/statacorp).

If you still can't find what you are looking for, you can contact Stata directly or post your question to the Stata Listserv (http://www.stata.com/statalist/). However, be aware that you probably do not want to join the listserve indefinitely as it gets a lot of postings that will quickly fill up your mailbox. In addition, I do not recommend posting a question to the Listserv until you have exhausted all other resources.

Finally, there are a lot of great Stata resources/tutorials on the web:

http://projects.iq.harvard.edu/undergradscholars/book/export/html/6506

http://data.princeton.edu/stata/default.html

http://www.princeton.edu/~otorres/Stata/

Of course, you should always feel free to drop by my office or send me an email and I will do the best I can to help you out.