Tracy Garnar

2/2/19

Importing data into Stata, verification of file format conversion

This community resource builds upon previous community resources prepared by Megan McKendry and Sabrina Nardin. This resource provides information on importing data into Stata as well as verifying the integrity of the converted data once it is imported.

Importing data into Stata

Stata can read several file formats natively, including Stata .dta files, ASCII (plain text) files and datasets in SAS transport format.

However, datasets exist in many other formats, including but not limited to CSV (comma-separated values), Excel, and SPSS. Stata cannot read these file formats without first being converted.

There are several options to convert these datasets into a format that Stata can handle:

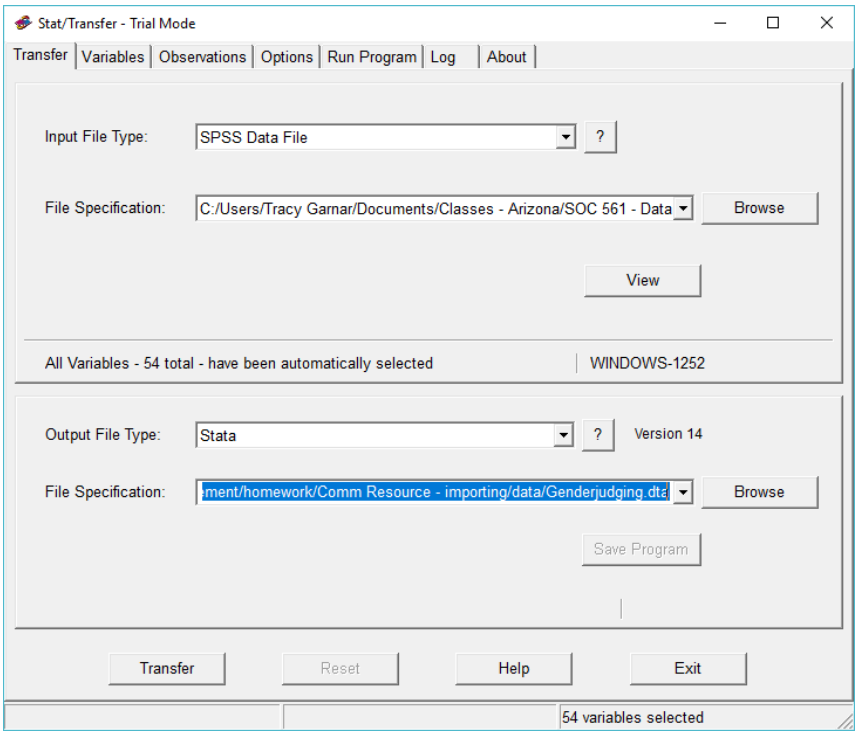
1. Have someone with the appropriate statistics program export the file to Stata format for you.
2. Using a program such as Stat/Transfer ($, demo version deletes every 16th observation, available at https://stattransfer.com/) A demonstration using the demo version will appear below.
3. Stata has the `infile` and `insheet` commands to import data using a do file. I will not cover this here as Sabrina has previously outlined this in detail (http://snardinstata.tumblr.com/).
4. If you want to import the data using Stata, it is recommended to use the dialog boxes so that you do not inadvertently mistype a command or miss a vital option.

Converting data using Stat/Transfer

I will now demonstrate using Stat/Transfer (the demo version) to convert an SPSS file into Stata format.

First, download and install Stat/Transfer. Leave all options checked (one is for Stata support).

Here is the window you will see upon startup:



Transfer tab

Stat/Transfer is able to convert 40 different file formats. Select the format of the file you're interested in importing into Stata under Input File Type. Note that depending on the file format, you may need to know what version of the program was used to create it. To get additional information on any file type in the list, select it in the dropdown menu, then click the "?" button next to it.

You'll point Stat/Transfer to the file you want to convert using the File Specification dialog. Browse to the file you wish to convert.

If you'd like to view the file within Stat/Transfer (to make sure you have the right one, for example), hit the View button.

A message will show up advising you how many variables will be imported. Unless you make any changes in the Variables tab, all variables will be automatically imported.

Similarly, under Output File Type, select the file format you wish the file to be converted to. Here, we've selected Stata. The current version of Stat/Transfer outputs Stata files into version 14 format by default; this can be changed under the Options tab (see below).

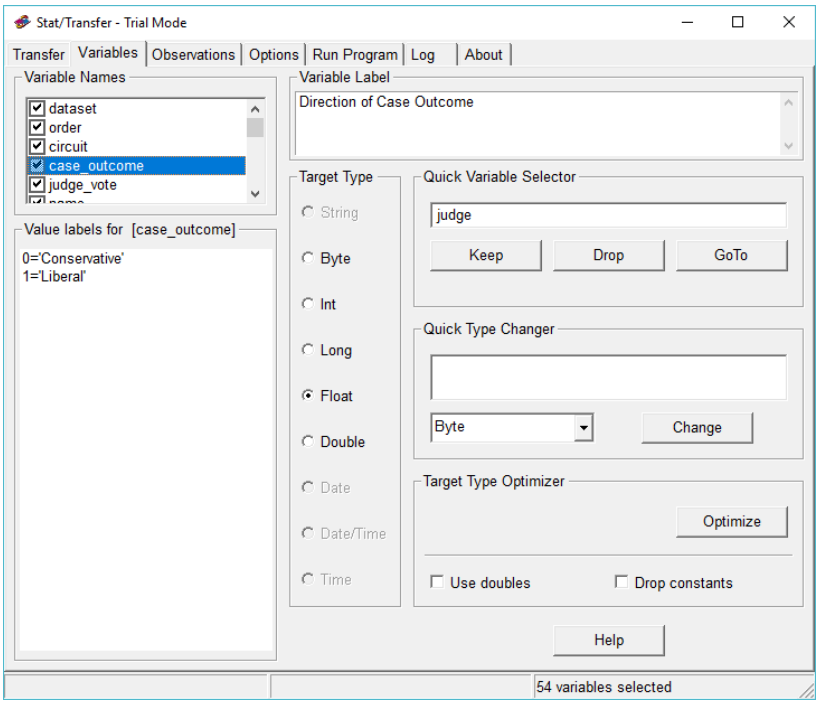
You can change the name of the output file under File Specification if you wish (under Output File Type).

Variables tab

The variables in the dataset are listed in the upper left-hand corner. Variables that will be imported are checked, and variables that will not be imported are not checked. By default, all variables are selected; to deselect a variable, simply click on the checkmark to unselect it.

Clicking on a variable in the list will bring up the variable label, variable values, and variable value labels for that variable. The variable type (as it is set in the original file) will also be selected in the Target Type section; you may change this by ticking one of the other radio buttons. Generally, you'll want to leave this alone.

Particularly useful for large datasets containing many variables is the Quick Variable Selector box. Provided you know the exact name of the variable you're looking for, you can type it in the box and either Keep it in the imported data set, Drop it from the imported dataset, or just Go To it in the list (to examine the labels and values). You must know the exact name, though; wild cards are not accepted.

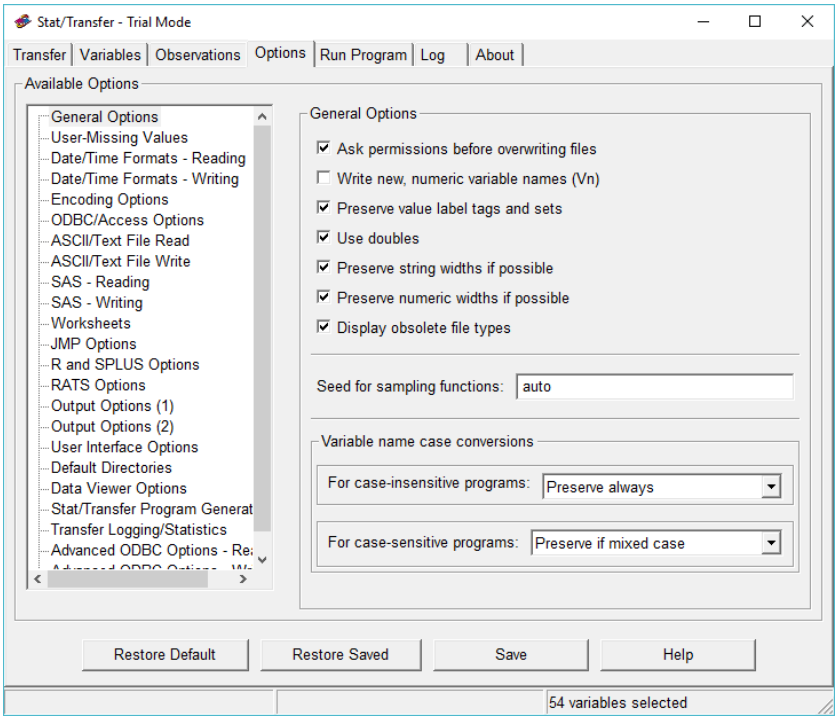


Observations tab

If you wish to import only a subset of observations from the original dataset, you may select the observations you wish to import using regular expressions. A list of variables is provided in this tab for your convenience. I do not recommend this, as you will have to go through the entire process again should you later discover you need observations that aren't in the imported dataset.

Options tab

The Options tab provides a ton of options to help you ensure that data is imported successfully. The most common sets of options you may need are located under the General Options, User-missing Values, and Output Options (1) and (2).

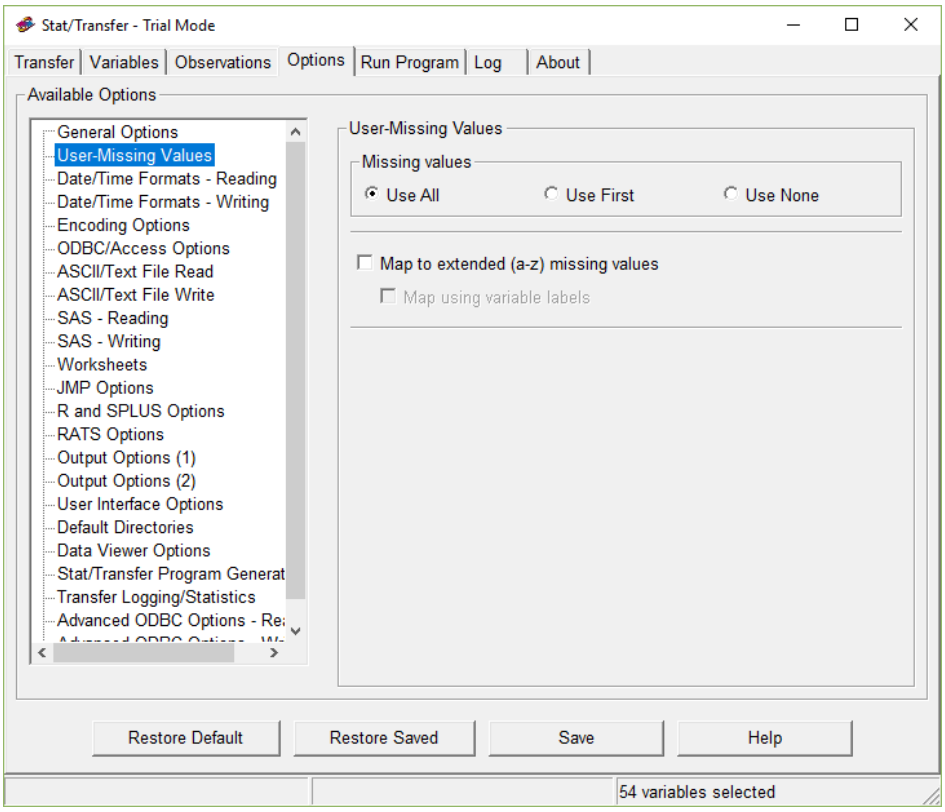


In General Options, I've checked "Preserve string widths if possible" and "preserve numeric widths if possible" since I'm importing a file from SPSS (which specifies string and numeric widths in its datasets), but left everything else on the default setting.

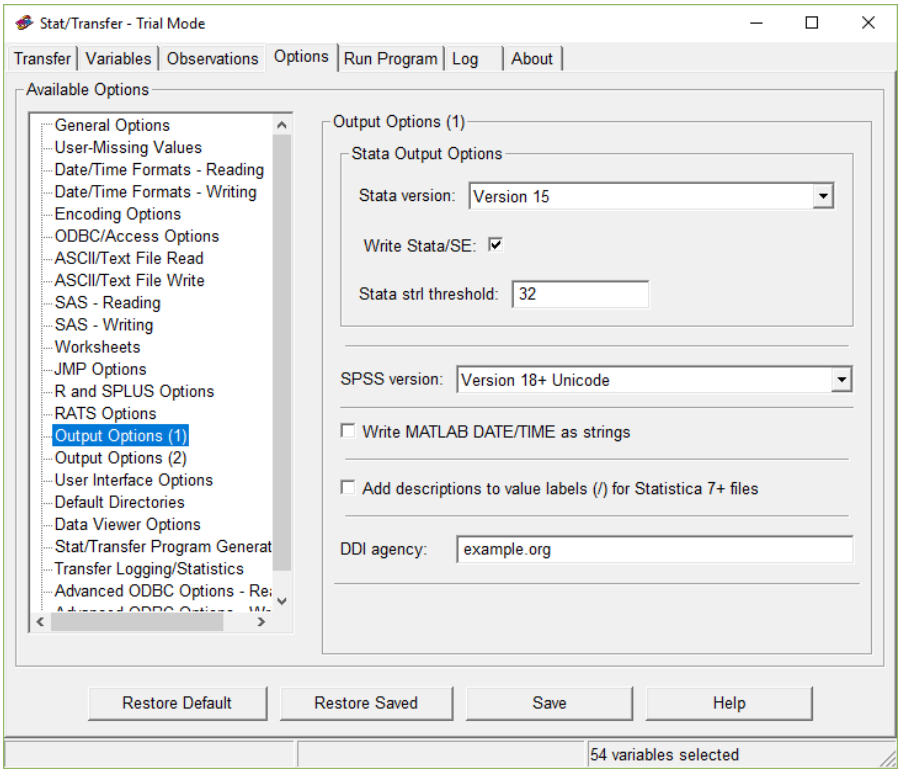
Under User-Missing Values, since SPSS does distinguish between system-missing (values that result from, for example, dividing by zero) and user-missing (missing because there was no valid data input for that observation), we would like to preserve that distinction in the imported dataset.

The default selection, Use All, collapses all missing values into a single missing value. This isn't particularly helpful if you want to distinguish between missing data due to inapplicability (i.e., pregnancy questions for male respondents) and missing data due to non-response. Use First maps the first user-missing value (from the original dataset) to system-missing in the new dataset, and then all other user-missing values are transferred as is to the new dataset. Use None transfers all user-missing values as is to the new dataset.

Since Stata allows for extended user-missing designations, you may want to make sure that "Map to extended (a-z) missing values" is checked. The first user-missing value will be mapped to ".a", the second to ".b", and so on. It is not recommended to check "Map using variable labels". Be *extremely* careful with this – many dataset conversions fail because missing values aren't converted properly.

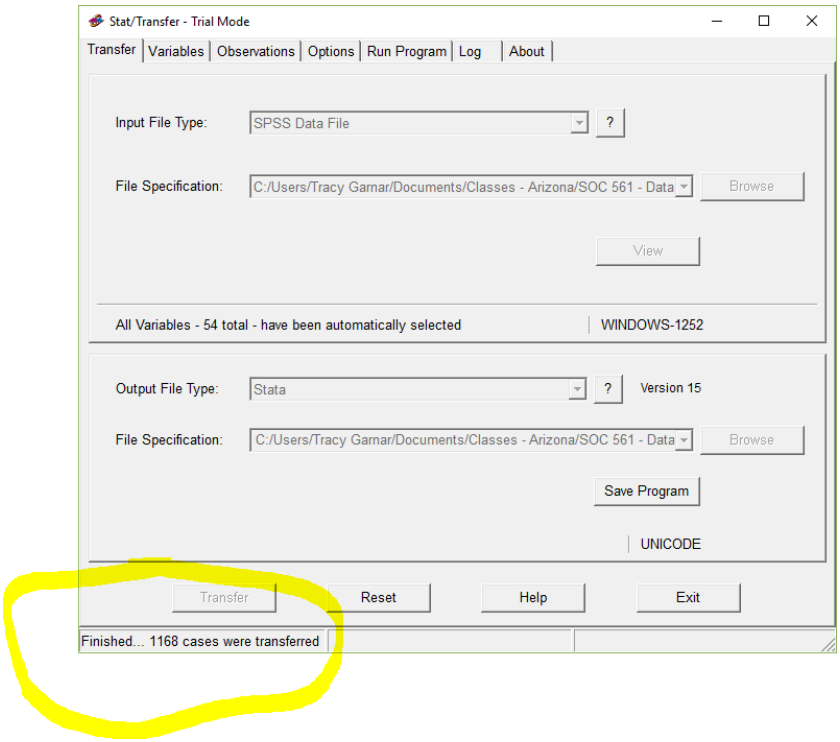


Under Output options, the options you'll most likely be concerned with are the Stata options. You can select the version of Stata you would like to use. Unchecking the "Write Stata/SE" option will enforce the limit on available variables imposed by the student version of Stata.

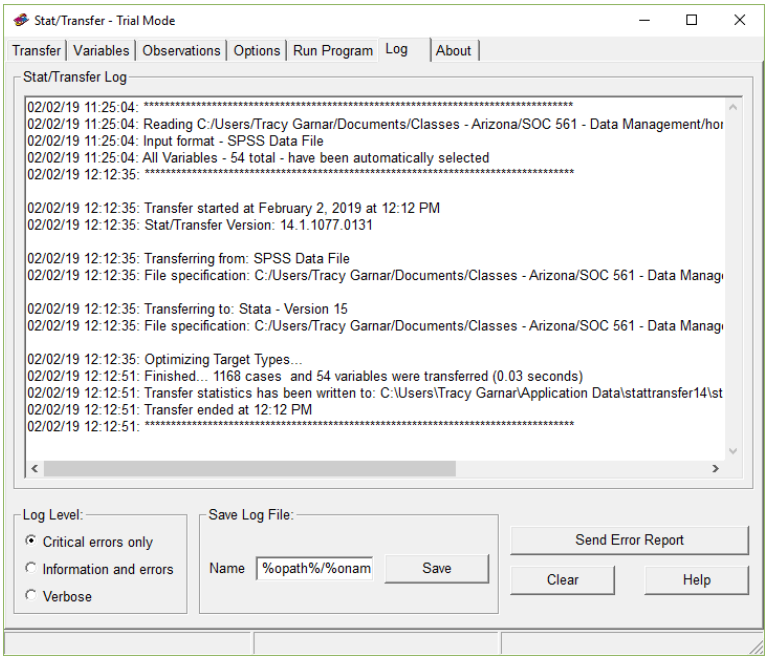


Once you have set up your options the way you like them, you can go back to the Transfer tab and hit Transfer to start the conversion.

When the conversion is complete, the status bar at the bottom of the screen will tell you how many cases were transferred. In this case, since I'm using the demo version, not all the cases were transferred.

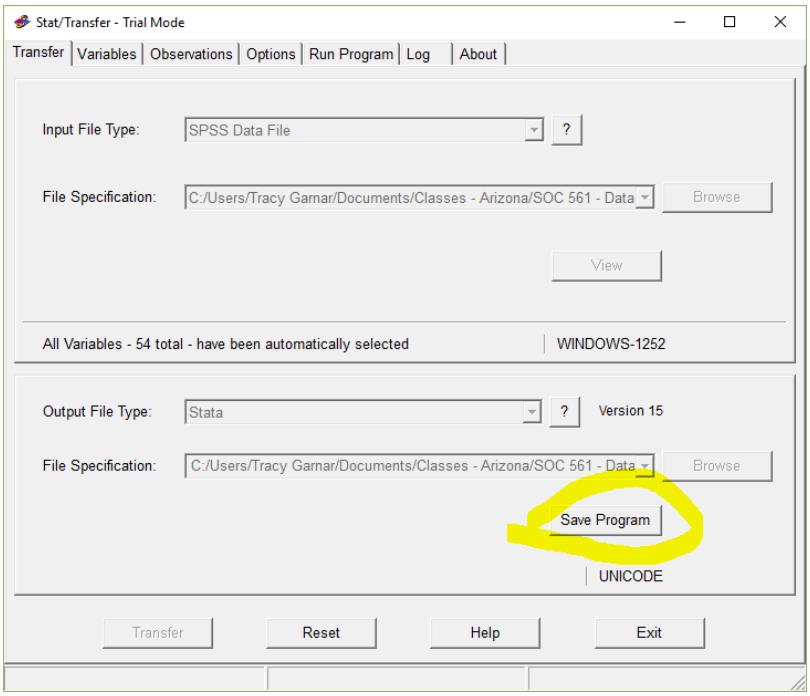


The Log tab will give you detailed information about the conversion process, which is especially helpful if there were errors during the conversion process that caused Stat/Transfer to choke.



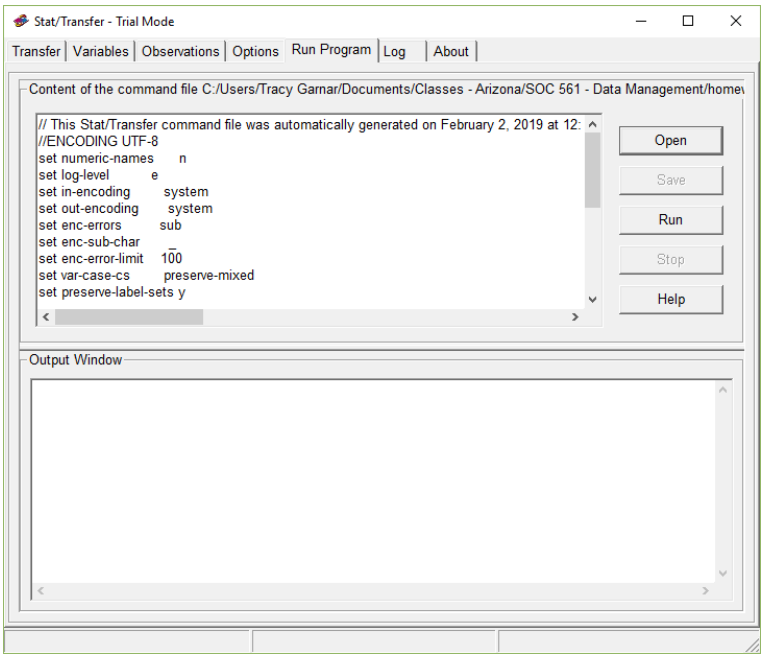
Once you have ensured that the programming you have done here worked correctly, you can save these settings as a Stat/Transfer command file so that the next time you need to convert another dataset of the same type, you can just point and click once instead of going through all these steps again.

Go back to the Transfer menu and hit Save Program.



Use the dialog box to specify a name for the file. I've named mine here "spss2stata".

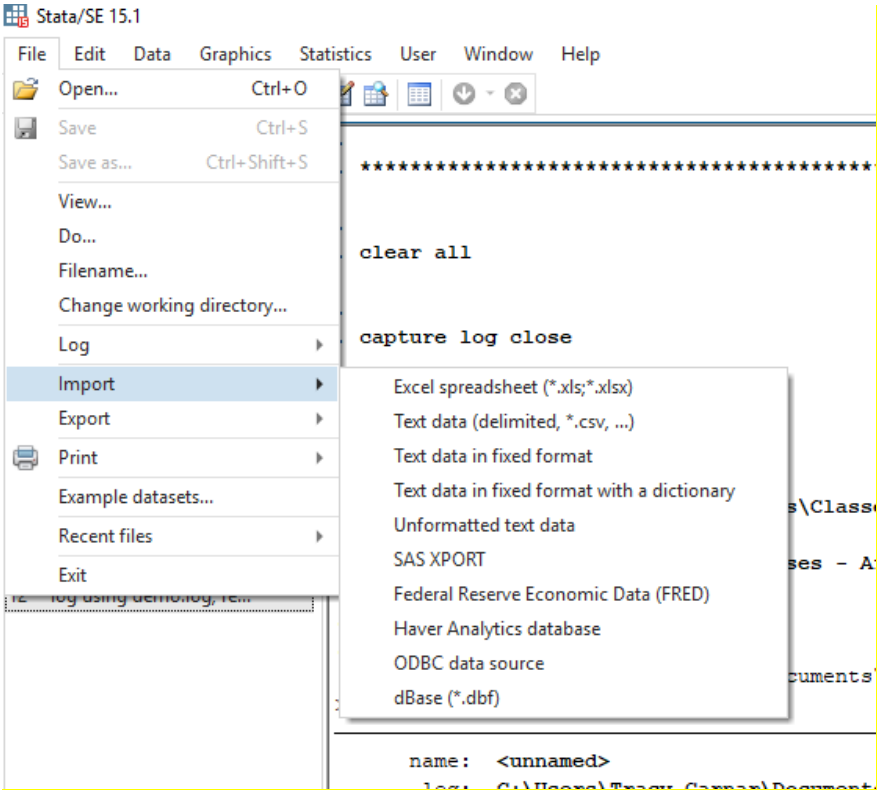
The next time you want to use it, go to the Run Program tab, click Open, and navigate to the command file.



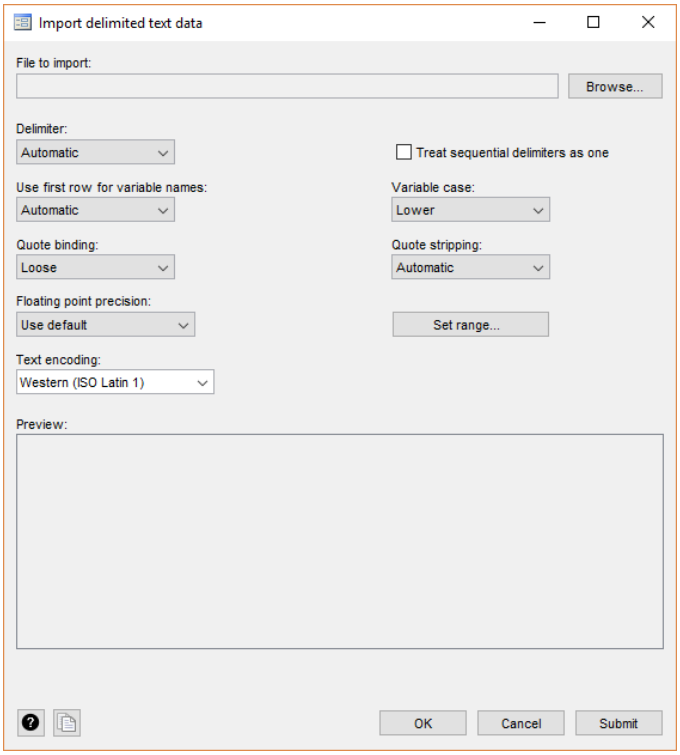
Converting data using Stata dialog boxes

You may also convert data using Stata's dialog boxes. As it is very easy to miss options that you need or mistype a command parameter, this is recommended over using `infile` or `insheet` if possible. However, Stata is not able to import all file formats using the dialog boxes. I will demo this, as the file we convert in the next section will be necessary in order to demonstrate the verification process below.

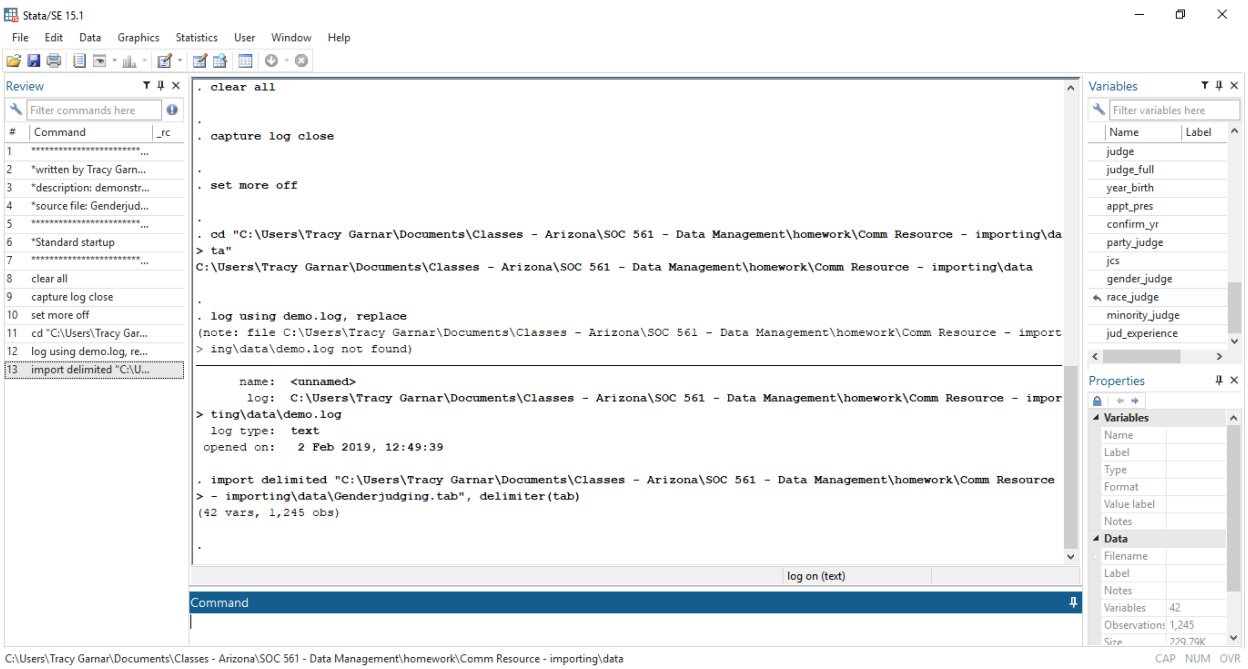
First, open Stata, and go to File -> Import. A list of the available options appear below. For this, I will use "Text data (delimited, *.csv, …)".



When I select "text data (delimited, *.csv, …)", the following dialog box appears:



Browse to the file you wish to open. Once you select a file to open, you can examine the file in the Preview section. If you wish to change the way a variable stores information, select a column by left-clicking on it, then right-click and select the desired data type. You can select the delimiter used in the original data (or leave it on Automatic). You can designate whether the first row contains variable names or not (check this in the Preview window at the bottom of the window). Depending on how numeric data is recorded in the original dataset, you can choose to leave it on “Use default”, or force Stata to import all numeric data as floating- or double-precision. Since this data doesn’t require double precision, I’ll leave it on Use Default. You can also specify how you want the variable names to read in the imported dataset. If you need help understanding what these options mean, click on the “?” button in the lower left-hand corner of the dialog box. Once you have selected your options, click OK to issue the command.



You can copy and paste the code into your do file as well.

Verifying Dataset Conversion

It is vitally important to verify that your conversion process actually transferred all of the data you wanted, that the data transferred properly, and that the dataset formatting wasn't broken somewhere along the way.

There are several ways you can verify the integrity of your data in Stata. These include running summary statistics, checking missing values, and importing the data using two different methods. Examples of these follow.

Summary statistics

Running descriptive statistics are a very easy way to kill two birds with one stone – first, you're getting to know your data (which is always important to do anyway!), and you can also check to make sure everything converted properly. Ordinarily, you will want to do this for every variable in the dataset. For demonstration purposes, I'll spot check just a few variables.

First, I'll use the `ds` command to identify numeric and string variables.

```
. ds, has (type string)
circuit       name          cite          hostile_work  non_employ    judge         judge_full

.
. ds, has (type numeric)
dataset       num_fem       title_ix      denied_pro~n  first_a       emotional     jcs
order         judge_num     sec_1983      not_hired     damages       year_birth    gender_judge
case_outcome  lower_dir     constructi~e  fired         sec_1981      appt_pres     race_judge
judge_vote    retaliation   procedural    unequal_pay   age_disc      confirm_yr    minority_j~e
dec_year      female_pla~f  st_law        fourteenth_a  pregnancy     party_judge   jud_experi~e
```

I can also use the `ds` command to verify whether the value labels transferred over from the original dataset:

```
. ds, has(vall)

.
```

As we see here, none of the value labels transferred over from the original dataset! If we were going to use this dataset for actual work, we'd need to go back to the codebook and relabel everything before we did anything.

The `codebook, compact` command is also useful. This command lists all of the variables in the dataset and provides some quick summary information about each variable. A portion of the output from this command follows.

Variable	Obs	Unique	Mean	Min	Max	Label
dataset	1245	1	0	0	0	
order	1245	415	3209.429	2704	3710	
circuit	1245	12	.	.	.	
case_outcome	1245	2	.346988	0	1	
judge_vote	1245	2	.351004	0	1	
name	1245	411	.	.	.	
cite	1245	415	.	.	.	
dec_year	1245	8	1998.595	1995	2002	
num_fem	1245	3	.4096386	0	2	

From here, we can check to see if all of the possible variable values transferred over in categorical variables by looking at the "unique" column and checking the "min" and "max" columns. From this table, the Circuit column looks OK at first glance, as it is a categorical variable with values for each of the 12 federal circuits. In addition, case_outcome and judge_vote look OK at first glance, since they are dichotomous variables (conservative/liberal). Similarly, dec_year looks OK at first glance, since I know that this dataset only contains cases with a final decision issued between 1995 and 2002. If we were looking at continuous variables, we could get an idea of whether they transferred properly using the "mean" column as well.

We can use the `tab1` command to get an idea of how the variables are structured. `Tab1` allows us to specify multiple variables in one command, whereas for verification purposes, we would only be able to specify one variable at a time if we were to use `tabulate`.

```
-> tabulation of circuit
```

circuit	Freq.	Percent	Cum.
1	63	5.06	5.06
10	75	6.02	11.08
11	123	9.88	20.96
2	117	9.40	30.36
3	66	5.30	35.66
4	72	5.78	41.45
5	111	8.92	50.36
6	66	5.30	55.66
7	225	18.07	73.73
8	240	19.28	93.01
9	51	4.10	97.11
DC	36	2.89	100.00
Total	1,245	100.00	

```
-> tabulation of case_outcome
```

case_outcome	Freq.	Percent	Cum.
0	813	65.30	65.30
1	432	34.70	100.00
Total	1,245	100.00	

```
-> tabulation of judge_vote
```

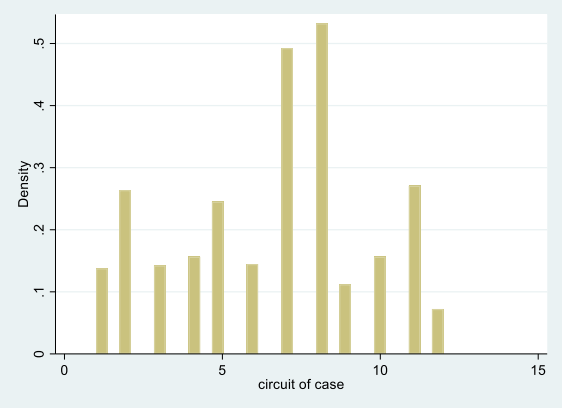
judge_vote	Freq.	Percent	Cum.
0	808	64.90	64.90
1	437	35.10	100.00
Total	1,245	100.00	

```
.
```

The `inspect` command is also helpful for identifying possible problems with the data import process. Long (2009) likes the `inspect` command because it outputs a histogram that allows you to identify if some of the observations did not import properly.

```
num_fem:                                        Number of Observations
                                                ------------------------------------------
                                                      Total       Integers   Nonintegers
  #                                 Negative            -              -             -
  #                                 Zero              742            742             -
  #                                 Positive          426            426             -
  #   #                                           ---------      ---------     ---------
  #   #                             Total           1,168          1,168             -
  #   #   .                         Missing             -
------------------------------------------      ---------
0                                 2             1,168
    (3 unique values)
```

However, the histogram output produced using the `inspect` command isn't terribly helpful if you have more than one or two categories in your variable. A regular histogram might be more helpful to us:



This tells us that the 7th and 8th circuits had the most cases, which is consistent with the tabulation above. Similarly, DC has the fewest cases (it appears here in the histogram at the end, since it is the only circuit with letters in the label).

If you're checking a variable that takes on many possible values, `dotplot` and `stem` may be helpful to you. I will not reproduce this here, as Megan and Sabrina have already explained how to do so in detail.

`Scatter` generates scatterplots that are useful to quickly examine links between two variables. Megan and Sabrina have already explained how to do this in detail as well, so I will not reproduce their efforts here either.

You can use the `assert` command (with the `rc0` option) to also check for various things that you know should be true about the dataset. For instance, we want to check to make sure that case_outcome is coded as either 0 or 1, since we know from above this should be the case:

```
. assert case_outcome == 0 | case_outcome == 1
```

Since this is in fact the case, Stata returns nothing.

We can ask Stata to assert that case_outcome is coded as either 1 or 2 (which we know from above should not be true):

```
. assert case_outcome == 1 | case_outcome == 2
762 contradictions in 1,168 observations
assertion is false
r(9);
```

Finally, we can ask Stata to produce a codebook and identify any common problems with data using the command codebook, problems:

```
   Potential problems in dataset   *\homework\Comm Resource - importing\data\Genderjudging.dta

              potential problem   variables
-------------------------------------------------------------------------------
  constant (or all missing) vars   dataset hostile_work non_employ
  string vars with embedded blanks   name cite judge judge_full
-------------------------------------------------------------------------------
```

We would want to follow up on anything listed here.

The `isid <varlist>` command checks whether a specified id variable uniquely identifies each observation. This may not be the case with panel data.

The `notes` command is very useful here to document any fixes we apply to variables with problems.

Inspection of missing data

One of the most common reasons datasets don't convert properly is because of differences in how different statistical programs handle missing data. Therefore, it is crucial to make sure that missing data is handled properly in your newly converted dataset.

The best way to check for missing values is via the `tab` command, using the `missing` option. Since this dataset was constructed from secondary data (case decisions), there are no variables with missing data to demonstrate here. However, Megan and Sabrina's resource files both demonstrate this in detail.

One thing to keep in mind, especially if you convert from SPSS or another file format that distinguishes between system-missing and user-missing, is that Stata considers "." as system-missing, and treats it as positive infinity in analyses. This has tripped up many researchers in the past, so be mindful of this.

Using the `compare` command

Long (2009) recommends as part of the verification process that we import the data using two different methods and use the `cf` command to compare them. Long (2009) provides examples where dataset conversions do verify properly, so I will demonstrate what happens when dataset conversions don't verify properly.

The `cf` command takes the following format:

```
cf <varlist> using <filename>
```

You need to have one of the dataset versions open within Stata. You tell Stata to compare the currently open dataset against the other converted version that you specify following `using`. Again, normally you would check all variables, but for the sake of demonstration, I will only check a couple.

```
. cf circuit case_outcome judge_vote using Genderjudging.dta
master has 1245 obs., using 1168
r(9);
```

Clicking on "r(9);" in blue in the Output screen shows the following:

```
[P]     error . . . . . . . . . . . . . . . . . . . . . . . .  Return code 9
        assertion is false
        no action taken
        Return code and message from assert when the assertion is false;
        see help assert.
        Or, you were using mvencode and requested Stata change `.' to #
        in the specified varlist, but # already existed in the varlist,
        so Stata refused; see help mvencode.
```

Ordinarily we would be very concerned about this because this is evidence that there was an error in the conversion process. However, in this demonstration example we are aware that Stat/Transfer dropped some of the observations in the output file, so I expected this result here. Obviously, for real analyses I would check the version I imported through Stata's dialog boxes and use that version once cleaned and verified.

Works cited

Long, J. Scott. 2009. *The Workflow of Data Analysis Using Stata*. College Station, TX: Stata Press.