Abstract:

At last year’s Stata conference, I presented projects that facilitate the combined use of Stata and Python. One project provides the ability to use Python within Stata via a C plugin. The other project provides a custom Python class that can be used to open, modify, and save Stata datasets. In this talk, I will begin by describing some modifications and extensions to these projects. I will then present a few new ideas for useful combinations of Stata with other tools. Some of these ideas can be realized using last year’s Python projects, some using JavaScript and a web browser.
Support

where I work
NASA Johnson Space Center

who I work for
Universities Space Research Association
The past couple years I’ve been playing with the idea of combining Stata with other software to extend its functionality. I’ve mostly been using the Python programming language and third-party Python modules.

Last year I demonstrated two Stata/Python projects.
The first project from last year is a plugin for using Python directly in Stata. As shown here, the python command puts the user in an interactive Python session.
The second project from last year is a Python package `stata_dta`, for opening, modifying, and saving dta files in Python.

Here I use the interactive Python session to demonstrate `stata_dta`. First I import the function `open_dta`, then use that function to create a Python variable `dta` containing the information in `auto_copy.dta`, and then finally call the `describe` method on `dta`. All of this is being done in Python. The variables panel shows that there is no dataset loaded in Stata.
Changes

The python command will now search for Python files in the ado path.

A new st_mirror() function which returns a dynamic view onto the Stata dataset. (The st_view() function returns a static view.)

I’ve made a few changes to last year’s projects. Most of these changes are in line with making these projects more convenient.

I will demonstrate the new st_mirror() function on the next page.
The Python plugin includes a function `st_view`, which, like Mata’s `st_view`, provides a static view of the Stata dataset. The plugin now includes a function `st_mirror` to provide a dynamic view. Here I will demonstrate the difference.

In the picture above, no dataset is loaded in Stata. I create Python variable `v` using `st_view` and Python variable `m` using `st_mirror`. I query the rows in each, and for both there are no rows.
Continuing from the last page, here I exit Python, load the auto dataset, and re-enter Python.

I again query the rows in $v$ and $m$. Again the view $v$ contains no rows, but $m$ contains all the rows in the loaded dataset. This is the main difference between `st_view` and `st_mirror`. The number of rows and columns in $v$ is fixed based on what was loaded when it was created. The number of rows and columns in $m$ always reflects what is current in Stata.
The last major addition to mention here is the *stata_math* module.

Documentation for the Python plugin can be found at
New ideas

Now for a few new ideas about extending Stata functionality.
Physical units

There's no dedicated place for recording physical units.

There's no simple way to convert from one unit to another.

First idea: Add functionality for recording and converting physical units.
To explore this idea, I added on to the `stata_dta` module described earlier to create the module `units_dta`.

Here I import `UDta` from `units_dta`, use `UDta` to open `auto_copy.dta`, and call the `describe` method. (So far, this demonstration matches the demonstration for `stata_dta`.)

Notice that in the auto dataset units have been recorded in the variable label.
I will move units out of the variable label and into a dedicated place. One part of the added functionality is the units_set method. Here (continuing from the describe on the previous page) I use units_set to say that headroom is measured in inches and mpg is measured in mpg.

Predicting that I might convert the mpg variable to other units, I change its name to efficiency.
I could continue setting units on the other variables, but instead I will load a dataset where that has already been done. Also, I have removed the units information from the variable labels.
To see what units have been defined, I use the `units_list` method.
If we go to the trouble of recording units, it might be helpful to be reminded of them when we use common Stata commands. The `units_dta` module has a modified version of `summarize` that displays units in the second column, as demonstrated above.

```stata
>>> dta.summ()

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Obs</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>make</td>
<td>USD1978</td>
<td>74</td>
<td>6155.26</td>
<td>2949.5</td>
<td>3291</td>
<td>15906</td>
</tr>
<tr>
<td>price</td>
<td>mpg</td>
<td>74</td>
<td>21.2973</td>
<td>5.7855</td>
<td>12</td>
<td>41</td>
</tr>
<tr>
<td>efficiency</td>
<td>mpg</td>
<td>69</td>
<td>3.4058</td>
<td>0.989932</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>rep78</td>
<td>inch</td>
<td>74</td>
<td>2.99324</td>
<td>0.845995</td>
<td>1.5</td>
<td>5</td>
</tr>
<tr>
<td>headroom</td>
<td>inch</td>
<td>74</td>
<td>5.9360</td>
<td>4.2774</td>
<td>5</td>
<td>23</td>
</tr>
<tr>
<td>trunk</td>
<td>ft</td>
<td>74</td>
<td>13.7560</td>
<td>4.2774</td>
<td>5</td>
<td>23</td>
</tr>
<tr>
<td>weight</td>
<td>lb</td>
<td>74</td>
<td>3019.46</td>
<td>777.194</td>
<td>1760</td>
<td>4840</td>
</tr>
<tr>
<td>length</td>
<td>inch</td>
<td>74</td>
<td>187.932</td>
<td>22.2663</td>
<td>142</td>
<td>233</td>
</tr>
<tr>
<td>turn</td>
<td>ft</td>
<td>74</td>
<td>39.6486</td>
<td>4.39935</td>
<td>31</td>
<td>51</td>
</tr>
<tr>
<td>displacement</td>
<td>inch**3</td>
<td>74</td>
<td>197.297</td>
<td>91.8372</td>
<td>79</td>
<td>425</td>
</tr>
</tbody>
</table>

| gear_ratio | 74  | 3.01486  | 0.456297 | 2.19 | 3.89 |
| foreign    | 74  | 0.297297 | 0.460188 | 0    | 1    |
```
Now suppose you give this dataset to someone outside of the US. They might prefer that headroom be measured in something other than inches, and they might prefer efficiency to be measured in something other than mpg.

In the picture above I first called `summarize` on just headroom and efficiency (this will be useful in a moment). I then used the `units_convert` method to convert the units on headroom from inch to cm and the units on efficiency from mpg to lp100km (my abbreviation for *liters per 100 km*). Finally, I called `summarize` again to compare to the previous `summarize`. The labeled units have changed, but the values in the dataset have also been converted.

Side note: I predefined `lp100km` in `units_dta`. If I hadn’t I would first have to use the method `units_define` to define it.
As I said earlier, this example was built on top of my `stata_dta` module. Most of the work to convert units is done by the Sympy module.
Second idea: A multimedia spreadsheet that allows us to view embedded images as images, hear embedded audio, etc, in the same spreadsheet as the other values of our dataset.
In fact, all of us have a rich multimedia viewer on our computers: our web browsers.

So, to explore this idea I use an html page and a third-party spreadsheet library. The html page is pictured above.
I click on the “Browse” button, and choose `birds.dta`, a Stata dataset with embedded photos and audio.
When the dataset first opens it looks like this. Most of the columns contain plain text. Notice the third column, *photo*, which contains binary data. At the moment the spreadsheet only knows that it’s binary data (Blob = binary large object).
In the header, next to “photo”, I can click to open a menu. The menu includes the options to decode “as image” or “as audio”.
When I choose to decode as image, the decoded images appear. The images are small, so I put my mouse over one to show it larger (the cursor is invisible here).
Likewise, the third-from-last column, `audio_sample`, contains audio, and can be decoded using the same menu.
<table>
<thead>
<tr>
<th>common_names</th>
<th>wiki_links</th>
<th>photo_links</th>
<th>binomial</th>
<th>audio_a</th>
<th>audio_sample</th>
<th>other_links</th>
<th>audio_photo</th>
</tr>
</thead>
<tbody>
<tr>
<td>sedge wren</td>
<td>http://...</td>
<td>http://...</td>
<td>Larry Cislot</td>
<td>http://...</td>
<td>1.91 4(4)</td>
<td>2010-01-10</td>
<td></td>
</tr>
<tr>
<td>downy woo</td>
<td>http://...</td>
<td>http://...</td>
<td>Larry Picoid</td>
<td>http://...</td>
<td>0.06 4(4)</td>
<td>2010-01-10</td>
<td></td>
</tr>
<tr>
<td>red-tailed</td>
<td>http://...</td>
<td>http://...</td>
<td>Forest Buteo</td>
<td>http://...</td>
<td>0.02 4(4)</td>
<td>2010-01-10</td>
<td></td>
</tr>
<tr>
<td>common phoe</td>
<td>http://...</td>
<td>http://...</td>
<td>Dick D Phasia</td>
<td>http://...</td>
<td>0.02 4(4)</td>
<td>2010-01-10</td>
<td></td>
</tr>
<tr>
<td>common gr</td>
<td>http://...</td>
<td>http://...</td>
<td>Larry Quisca</td>
<td>http://...</td>
<td>0.17 4(4)</td>
<td>2010-01-10</td>
<td></td>
</tr>
<tr>
<td>red-winged</td>
<td>http://...</td>
<td>http://...</td>
<td>Larry Aagelai</td>
<td>http://...</td>
<td>0.93 4(4)</td>
<td>2010-01-10</td>
<td></td>
</tr>
<tr>
<td>common red</td>
<td>http://...</td>
<td>http://...</td>
<td>Larry Acanth</td>
<td>http://...</td>
<td>0.11 4(4)</td>
<td>2010-01-10</td>
<td></td>
</tr>
<tr>
<td>marsh wren</td>
<td>http://...</td>
<td>http://...</td>
<td>Larry Cislot</td>
<td>http://...</td>
<td>0.96 4(4)</td>
<td>2010-01-10</td>
<td></td>
</tr>
<tr>
<td>american w</td>
<td>http://...</td>
<td>http://...</td>
<td>Larry Turdu</td>
<td>http://...</td>
<td>0.72 4(4)</td>
<td>2010-01-10</td>
<td></td>
</tr>
<tr>
<td>house finch</td>
<td>http://...</td>
<td>http://...</td>
<td>Larry Haemor</td>
<td>http://...</td>
<td>0.59 4(4)</td>
<td>2010-01-10</td>
<td></td>
</tr>
<tr>
<td>indigo bun</td>
<td>http://...</td>
<td>http://...</td>
<td>Larry Passer</td>
<td>http://...</td>
<td>0.59 4(4)</td>
<td>2010-01-10</td>
<td></td>
</tr>
<tr>
<td>savannah</td>
<td>http://...</td>
<td>http://...</td>
<td>Larry Passer</td>
<td>http://...</td>
<td>3.16 4(4)</td>
<td>2010-01-10</td>
<td></td>
</tr>
<tr>
<td>american ...</td>
<td>http://...</td>
<td>http://...</td>
<td>Larry Spizel</td>
<td>http://...</td>
<td>0.93 4(4)</td>
<td>2010-01-10</td>
<td></td>
</tr>
<tr>
<td>dickcissel</td>
<td>http://...</td>
<td>http://...</td>
<td>Larry Spizel</td>
<td>http://...</td>
<td>1.00 4(4)</td>
<td>2010-01-10</td>
<td></td>
</tr>
<tr>
<td>harris's</td>
<td>http://...</td>
<td>http://...</td>
<td>Zonotrichia</td>
<td>http://...</td>
<td>0.25 4(4)</td>
<td>2010-01-10</td>
<td></td>
</tr>
</tbody>
</table>

When `audio_sample` is decoded as audio, audio controls appear.
Let's take this a step further. Here I again clicked on the “Browse” button and will open another dta file, birds_formatted.dta. This is the exact same dataset, except that some formatting information has been added.
When `birds_formatted.dta` is opened, it looks like this. The big changes here are in the layout and in the fact that images and audio were decoded as such without the user having to inform the spreadsheet.

The formatting information has specified

1. that photo should be decoded as image and audio_sample as audio
2. layout information
3. text size and formatting
4. links be created from data URLs
5. which data appear in the display and which do not

This kind of functionality opens up the possibility of making automatic, multimedia slideshow presentations of the Stata dataset.
This example uses the SlickGrid library for its spreadsheet (the vast majority of the functionality comes from this library), and some custom JavaScript code for opening .dta files (available on my GitHub page, see last slide for URL).
Third and final idea: A new kind of interface for Stata. As a kind of preface, the benefit I see for this kind of interface is that notebook interfaces are a kind of editable history of a session. They are simultaneously a log file and a shareable presentation.
Here is the interface I will be using. Rather than try to define “notebook interface”, I will demonstrate some of the features of this particular notebook interface.

Input and output are organized in *cells*. Above you see the input half of a cell, waiting for the user’s input.
Here I’ve entered two inputs. The output appears immediately below. So far this is not much different from Stata’s default interface.
Here we see the first difference. Typically, with notebook interfaces all of the output appears in the same window. In Stata’s default interface, help files open in another window. Here the help file appears inline. In a sense, the notebook interfaces provides a more complete log of a session. In a Stata log you will see the command for `help scatter`, but the help file itself will not be there.
Here we see a few more differences. On the last page we opened a help file. If we plan on sharing this session later, we probably don’t want to advertise that we forgot how to use the `scatter` command. Since the interface is an *editable* history, we just edit that out.

Here we write over that input with our new input, in this case `scatter mpg weight` (notice that the help file began right after the `describe` output, which is where our new `scatter` command is). The new output replaces the old output. Just like the help file, the graph shows up inline rather than in a new window. And again, by including the output of the `scatter` command, this interface is, in a sense, providing a more complete log of the session.
There are many other features that help this notebook interface be an editable, shareable history of your session. Cells can be rearranged or removed (in addition to being written over). You can insert markdown cells for notes or explanations, and you can include LaTeX.
The majority of the functionality shown here comes from the IPython notebook. The modified version of `log2html` was used for converting the help file to html.
Resources

Python plugin  ssc describe python
Sympy  sympy.org/en/index.html
SlickGrid  github.com/mleibman/SlickGrid
IPython notebook  ipython.org/notebook.html
Example code  github.com/jrfiedler/StataCon2014

Contact  jrfiedler@gmail.com