

Re-imagining a Stata/Python combination

James Fiedler
Stata Conference 2013

Abstract: At last year's Stata Conference, I presented some ideas for combining Stata and the Python programming language within a single interface. Two methods were presented: in one, Python was used to automate Stata; in the other, Python was used to send simulated keystrokes to the Stata GUI. The first method has the drawback of only working in Windows, and the second can be slow and subject to character input limits. In this presentation, I will demonstrate a method for achieving interaction between Stata and Python that does not suffer these drawbacks, and I will present some examples to show how this interaction can be useful.

Support

where I work

NASA Johnson Space Center

who I work for

Universities Space Research Association

What I have in mind

2

To start off, let me describe the kind of “Stata/Python combination” I have in mind for this talk.

The screenshot shows the Stata software interface. The main command window displays the following code and output:

```

. matrix list m

m[7,4]
      price      mpg      rep78      headroom
r1      4749         17         3         3
r2      3799         22         .         3
r3      4816         20         3         4.5
r4      7827         15         4         4
r5      5788         18         3         4
r6      4453         26         .         3
r7      5189         20         3         2

. di "$blah"
some global

```

The right-hand side of the interface features two panels:

- Variables Panel:** Lists variables and their labels:

Variable	Label
make	Make and Model
price	Price
mpg	Mileage (mpg)
rep78	Repair Record 1978
headroom	Headroom (in.)
trunk	Trunk space (cu. ft.)
weight	Weight (lbs.)
length	Length (in.)
turn	Turn Circle (ft.)
displacement	Displacement (cu. in.)
gear_ratio	Gear Ratio
foreign	Car type
- Properties Panel:** Shows details for the current variable 'make':

Variables	
Name	make
Label	Make and Model
Type	str 18
Format	%-18s
Value Label	
Notes	

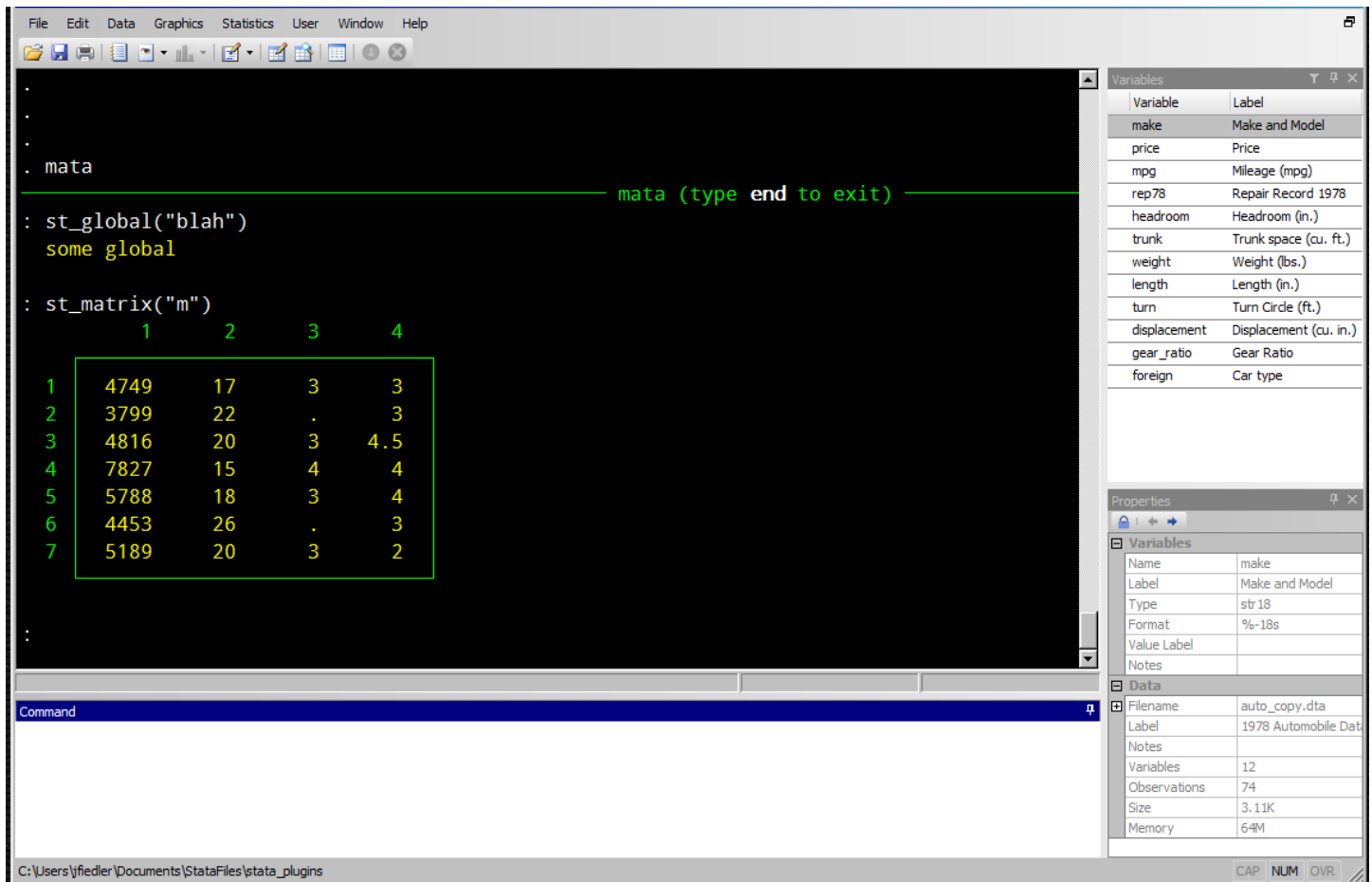
 Below this, it shows data properties:

Data	
Filename	auto_copy.dta
Label	1978 Automobile Data
Notes	
Variables	12
Observations	74
Size	3.11K
Memory	64M

The status bar at the bottom indicates the current location: C:\Users\jfedler\Documents\StataFiles\stata_plugins and shows the status CAP NUM OVR.

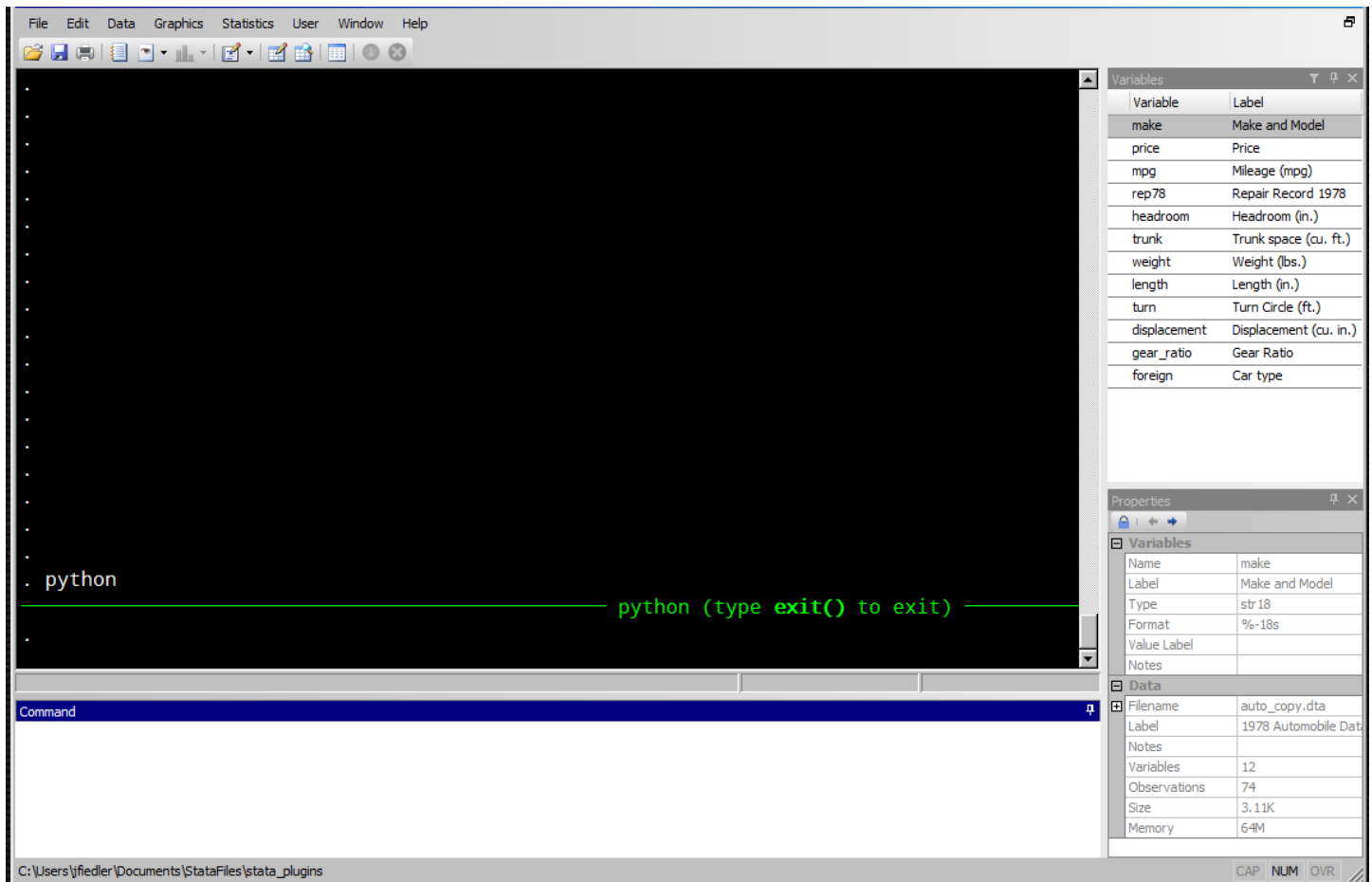
(In order to have some values to work with, I have loaded a copy of the auto data set, created a matrix m from a subset of the data values, and created a global macro blah.)

What I'd like to see is the ability to use Python interactively within Stata. Currently, there are two interactive modes. There's (what I call) "ado mode", which is what I used in this slide.



And with the mata command the user can enter a Mata interactive mode. Mata has different syntax, data structures, and functionality, but it's not completely disconnected from "ado mode" because there are functions for interacting with values defined in ado mode. For example, `st_global()` can be used to bring an ado-mode macro into Mata as a string and `st_matrix()` can be used to bring an ado-mode matrix in as a Mata matrix.

What I'd like to see is another interactive mode where Python can be used.



Just as there's a `mata` command in Stata, which puts the user into an interactive Mata session, there could be a `python` command which puts the user into an interactive Python session. And just as there are Mata functions for interacting with Stata values from Mata, there could be Python functions for interacting with Stata values.

Why add Python?

Functionality: Python has a lot of modules for data-related tasks.

Edge cases: There are things you *can* do in Stata that would be easier in Python.

Python is fun.

3

The first, obvious reason for adding Python is that doing so would add functionality. There is a large community of people using Python for data analysis, numerical computing, scientific computing, etc. As a consequence, there is a lot of Python code for doing the kinds of things that Stata users do. Having access to this code would expand what users can do in Stata, or at least expand what can be done practically.

Second, when there is overlap in what can be done in Stata and Python, sometimes it'll be easier to do it in Python.

Finally, one of the things Python programmers will consistently tell you is that Python is a fun language to use.

Problem:

Is anyone aware of a Stata user-written routine or other method of importing data stored in NetCDF files into Stata?

— Statalist, "Reading NetCDF files", February 2012

4

Here's an example from Statalist where adding Python would add functionality. This person had been given a NetCDF file and was wondering whether any commands had been written for opening such a file.

Response:

... it's common for researchers in the oceanographic/climate fields to use `python` to work with NetCDF

— Statalist, "Reading NetCDF files", February 2012

5

There weren't any Stata commands for opening the file, so one person suggested using Python.

The file was handled some other way, but the user wrote back with some conclusions, one of which was:

There are quite a number of programs that will extract from or use data in NetCDF files but all involve a minimum of one or two intermediate steps before the data can be imported into Stata. It would be nice to eliminate this, but I don't have the time or (probably) expertise to take it on because, at a minimum, it will involve linking C or Fortran [or, *ahem*, Python] programs to Stata.

With Python and a module for reading NetCDF files it would probably only take a few lines of code to get this file into Stata.

Problem:

I have a large number of large comma-separated text files that I am trying to import. "insheet" is not working

— Statalist, "importing quirky csv", November 2011

6

Here's an example of an edge case, again from Statalist. This person has a CSV file, which, because of some funky formatting, the user hasn't been able to import with the `insheet` command.

Response:

I've found Python's csv parser to be quite robust and able to write out the csv files in such a way that Stata will happily read them.

— Statalist, "importing quirky csv", November 2011

7

Another user suggests that Python's csv module might help. Without more information, we can't be sure that the Python method would be easier in this case, but we can see that the respondent has found cases where it is.

How do we add Python?

Use a C plugin

Stata provides functions in C for interacting with data and matrices.

Python provides a C API for interacting with Python structures.

9

Stata, through its plugin system, provides C functions for interacting with Stata data, macros, matrices, etc. Python, through its C API (application programming interface) provides C functions for starting a Python interpreter and interacting with Python structures.

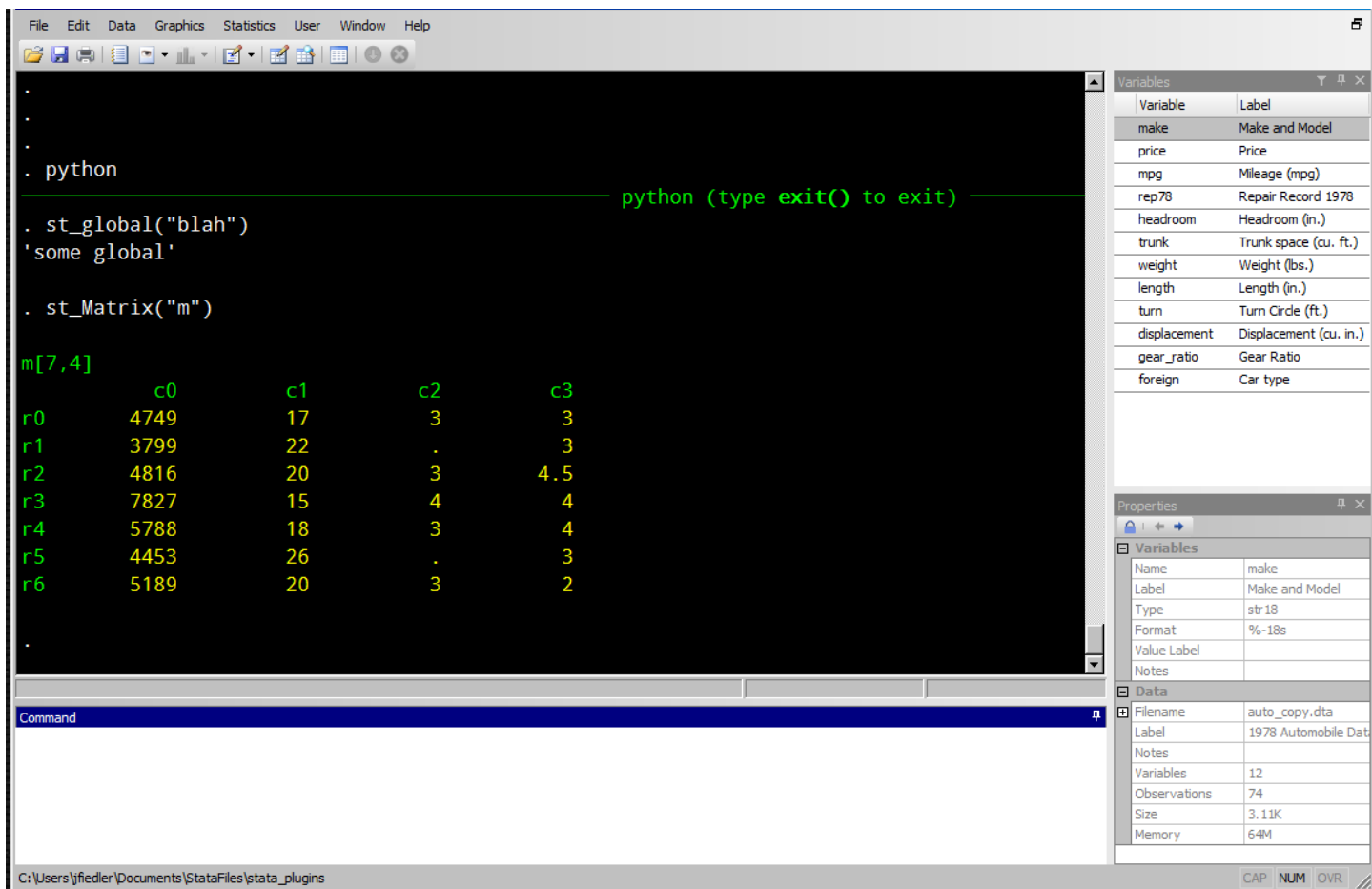
The idea is to match up Stata's C plugin functions with Python's C API functions. In a sense, the idea is to translate the C plugin functions into Python.

I think this is the way to go (i.e., much better than the methods I used at last year's Stata Conference), and I wish I could take credit for the idea. In fact, I'm pretty sure the idea came up in conversation with Stata Corp. employees last year, so credit for the idea should go to them.

Demonstration

10

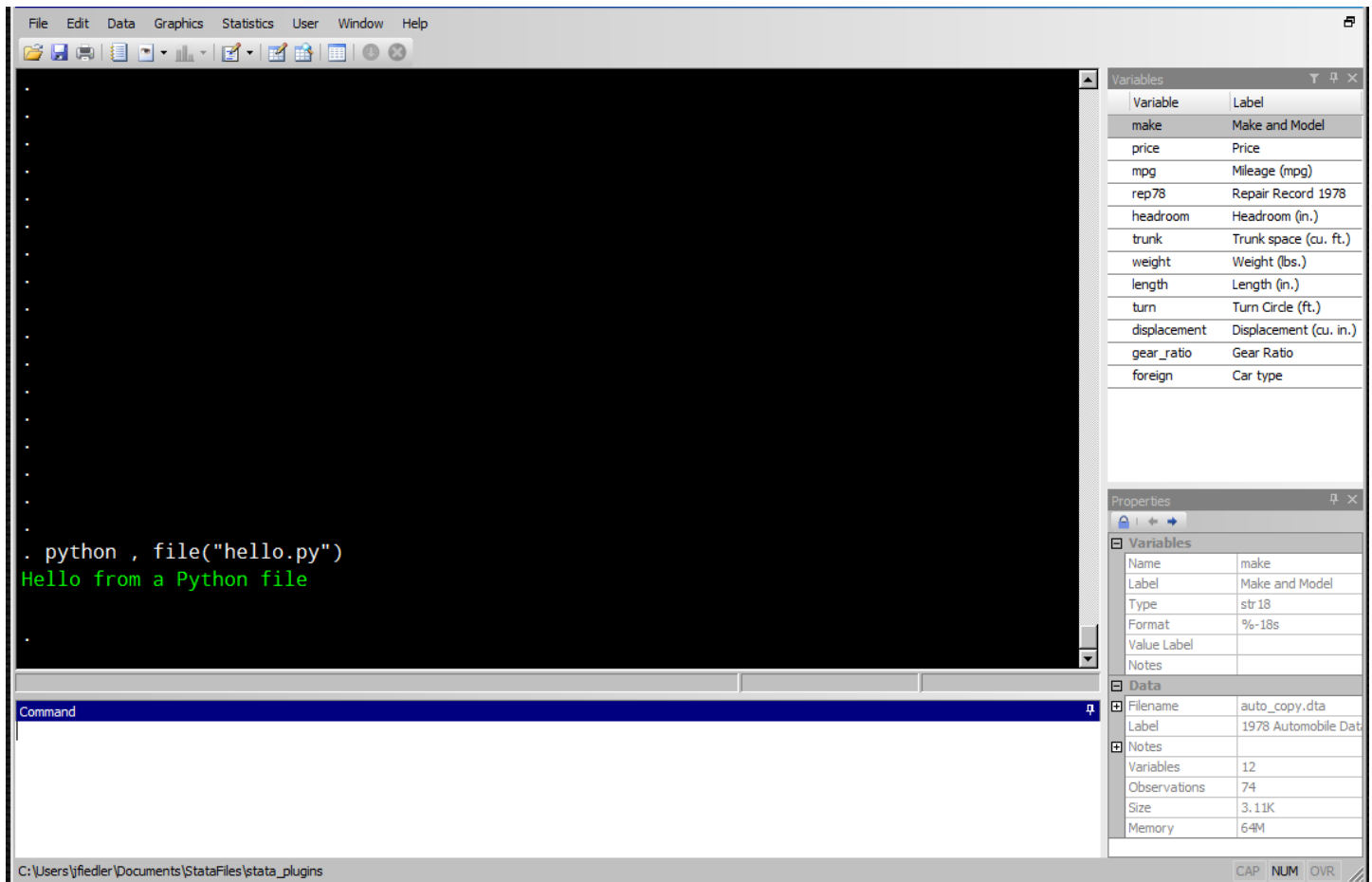
Before I begin a demonstration, I should probably make it clear that this isn't a simulation. Last year I used an imitation of the Stata GUI to demonstrate some ideas. This year there is no imitation Stata, and no imitation Python. This is a real instance of Stata, and I am really using Python within Stata.



The idea, as I said, is to have the ability to use Python interactively within Stata. With Stata's C plugin functions, the idea can be implemented. In the slide above the command `python` puts the user in an interactive Python session.

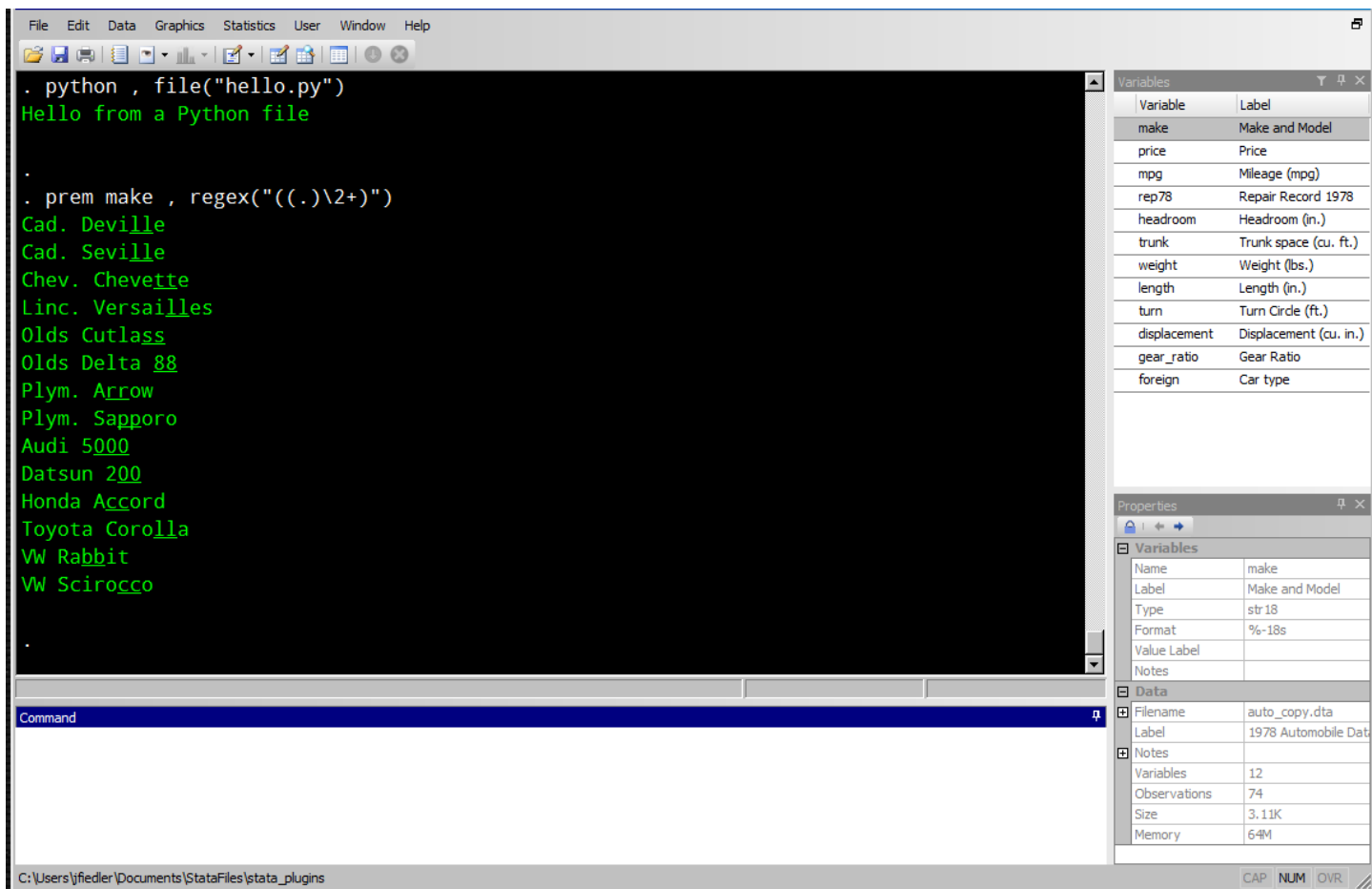
There's some leeway in how the C plugin functions are translated into Python functions. I've decided to make the Python functions mimic the Mata functions. The Python function for pulling in a global macro is `st_global()`, and the function for pulling in a Stata matrix is `st_Matrix()`.

(Notice the capital "M" in `st_Matrix()`. Python has no inherent notion of a matrix, so I've made a class `st_Matrix`, which is mostly just a view onto a Stata matrix.)



We can also use the Python plugin to run Python files. Here I've exited Python mode (not shown), so I'm back in ado mode, and I've used the python function with the file option. The hello.py file is just a single line:

```
print("Hello from a Python file")
```

(This example is in ado mode.)

The previous page shows a Python file used in isolation. Here a Python file will be used together with an ado file to find regular expression matches.

If you've never used regular expressions, these are basically a cryptic method of specifying patterns in text. If you have used regular expressions, Stata's regular expressions are probably sufficient for the vast majority of uses. Occasionally, though, some Stata users want to do something more complicated than what Stata's regular expressions were designed to handle.

Here I've created a Stata command called `prem` (for **p**rint **r**egular **e**xpression **m**atches). It takes a single Stata variable as its argument and takes a regular expression string as an option. In the example above, the regular expression uses a *back reference* to find entries repeated characters—for example, it finds the "bb" in "Rabbit" and the "000" in "5000"—then prints it to the screen with the match underlined.

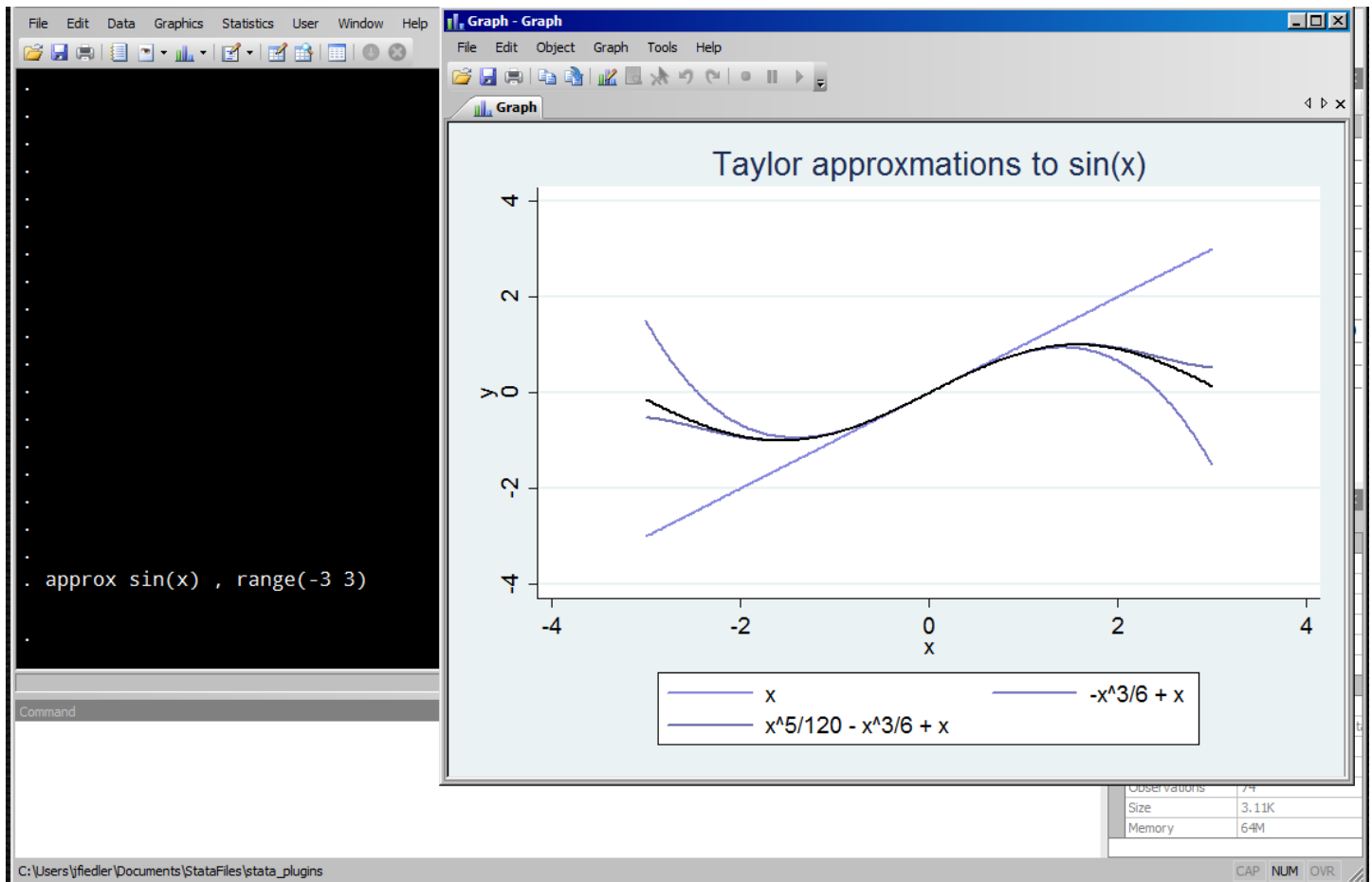
Sympy module

Sympy is a Python library for symbolic mathematics. It aims to become a full-featured computer algebra system.

— sympy.org

11

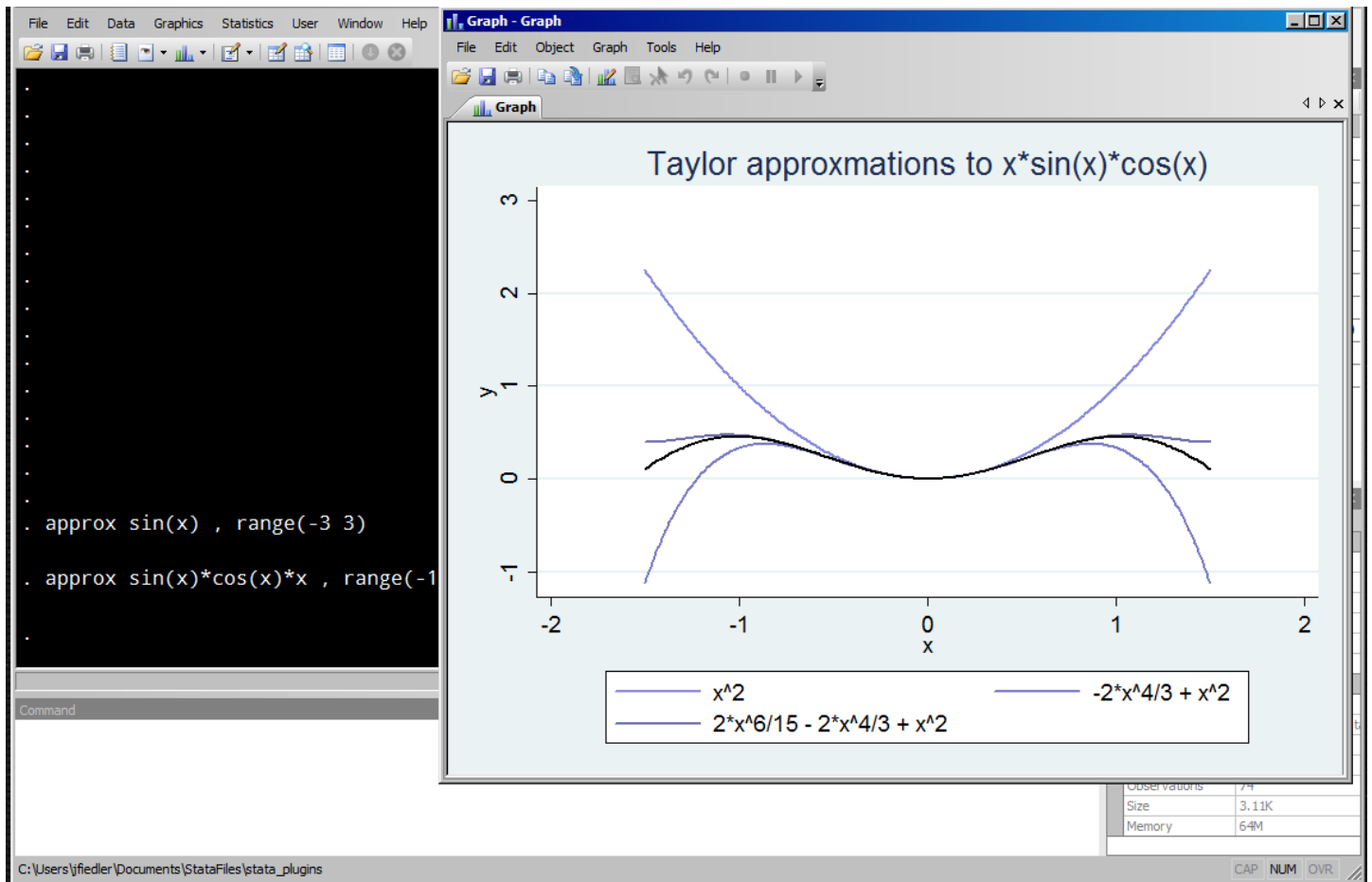
For the next couple examples I'll be using a symbolic math module called Sympy. With Sympy you can do things like take the limit of a function, factorize polynomials, and get the Taylor series expansion of a function.



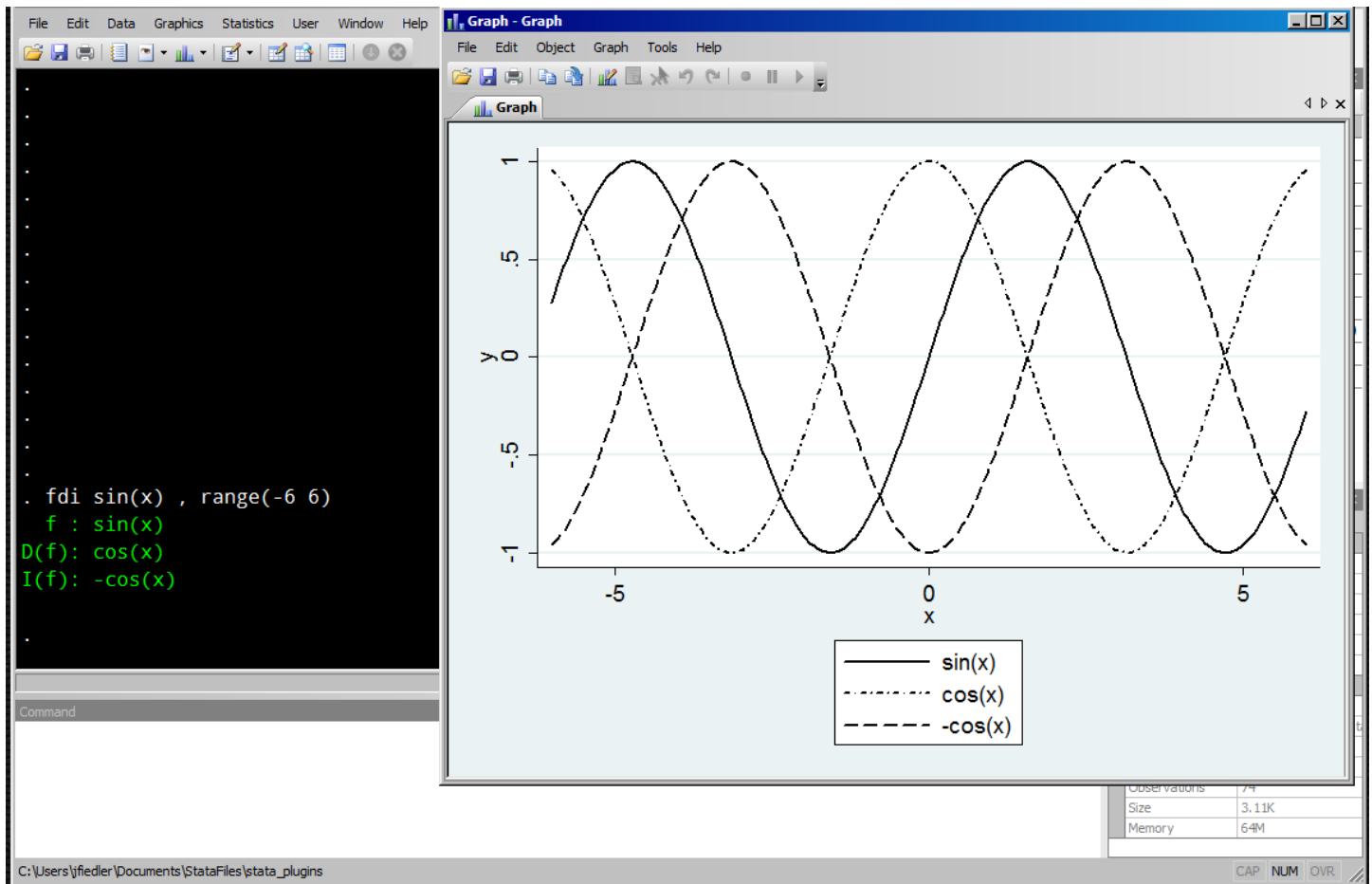
(This example is in ado mode.)

Here I'm using the command `approx` which takes a function of x as its argument, and graphs the function together with Taylor approximations. In the graph, the thick black line represents $\sin(x)$ and the blue lines show the first three Taylor approximations.

Again, the point of this is to show that ado files and Python files can work together seamlessly.



Here's another example with a more complicated function.



Sympy can also be used to find the functional form of derivatives and integrals. Here is a command for graphing a function with its derivative and integral.

Next: The dta module

12

Looking ahead and imagining people using the Python plugin, I would guess a common use would be to build or modify datasets. Users might want to build their datasets from scratch or by importing CSV files, for example, then save the dataset as a dta file for analysis in Stata. Or they might want to open and modify existing dta files before analyzing them in Stata.

I'm working on a module for doing these kinds of things.

The dta module provides

A Python version of the Stata data structure, the Dta class

Ability to create a Dta object from Python iterables (lists, tuples, etc.)

13

The module includes a complete Python version of the Stata data structure, or at least includes everything that a dta file would include.

The module provides the ability to create a Python dataset from Python array types (“iterable” types such as lists, tuples, generators, etc.).

The dta module provides

Methods for converting
Dta object ↔ .dta file

14

The module also provides the ability to create a Python dataset from a saved dta file, and to save to a dta file.

Demonstration

15

```
.  
. python  
python (type exit() to exit)  
. from stata_dta import Dta  
. v = [[10*i + j for j in range(5)] for i in range(6)]  
. v  
[[0, 1, 2, 3, 4], [10, 11, 12, 13, 14], [20, 21, 22, 23, 24], [30, 31, 32, 33, 34], [40, 41, 42, 43, 44], [50, 51, 52, 53, 54]]  
. for row in v: print(row)  
[0, 1, 2, 3, 4]  
[10, 11, 12, 13, 14]  
[20, 21, 22, 23, 24]  
[30, 31, 32, 33, 34]  
[40, 41, 42, 43, 44]  
[50, 51, 52, 53, 54]  
. .  
Command
```

Variable	Label
make	Make and Model
price	Price
mpg	Mileage (mpg)
rep78	Repair Record 1978
headroom	Headroom (in.)
trunk	Trunk space (cu. ft.)
weight	Weight (lbs.)
length	Length (in.)
turn	Turn Circle (ft.)
displacement	Displacement (cu. in.)
gear_ratio	Gear Ratio
foreign	Car type

Name	make
Label	Make and Model
Type	str 18
Format	%-18s
Value Label	
Notes	

Filename	auto_copy.dta
Label	1978 Automobile Data
Notes	
Variables	12
Observations	74
Size	3.11K
Memory	64M

C:\Users\jfedler\Documents\StataFiles\stata_plugins CAP NUM OVR

We need to do some preparatory work for the next example.

Here is the purpose of each of the above inputs (notice the labels 0-4 to the left of the slide):

0. Start the Python interactive mode. (We were in ado mode before this command.)
1. The “dta module” is actually called `stata_dta`. This line brings `Dta`, the dataset constructor, into the interactive session.
2. This line creates an array of arrays of values called `v`.
3. This is just to show the value of `v`. Think of the outer array of `v` as the dataset, and think of each inner array as a row within the dataset.
4. Another way of looking at what’s in `v`. This stacks the inner arrays to make it easier to visualize `v` as rows within a dataset.

The screenshot shows the Stata software interface. The main window contains the following code and output:

```

. data = Dta(v)

. data.list()

```

	var0	var1	var2	var3	var4
0.	0	1	2	3	4
1.	10	11	12	13	14
2.	20	21	22	23	24
3.	30	31	32	33	34
4.	40	41	42	43	44
5.	50	51	52	53	54

The right-hand side of the interface shows the Variables and Properties panels. The Variables panel lists the following variables and their labels:

Variable	Label
make	Make and Model
price	Price
mpg	Mileage (mpg)
rep78	Repair Record 1978
headroom	Headroom (in.)
trunk	Trunk space (cu. ft.)
weight	Weight (lbs.)
length	Length (in.)
turn	Turn Circle (ft.)
displacement	Displacement (cu. in.)
gear_ratio	Gear Ratio
foreign	Car type

The Properties panel shows the following information for the dataset:

Variables	
Name	make
Label	Make and Model
Type	str 18
Format	%-18s
Value Label	
Notes	

Data	
Filename	auto_copy.dta
Label	1978 Automobile Data
Notes	
Variables	12
Observations	74
Size	3.11K
Memory	64M

The Command window is empty. The status bar at the bottom shows the file path: C:\Users\jfedler\Documents\StataFiles\stata_plugins. The bottom right corner of the window displays the text: CAP NUM OVR.

A dataset is created from `v` just by calling `Dta` with `v` as argument. To see the values in a dataset, the `Dta` class has a `list()` method.

The screenshot shows the Stata software interface. The main window displays the following commands and their output:

```

. data.summ()

Variable | Obs   Mean   Std. Dev.   Min   Max
-----+-----+-----+-----+-----+-----
var0     |    6    25    18.7083     0    50
var1     |    6    26    18.7083     1    51
var2     |    6    27    18.7083     2    52
var3     |    6    28    18.7083     3    53
var4     |    6    29    18.7083     4    54

. data.save("temp.dta", replace=True)

. exit()

```

The Properties window on the right shows the following information:

Variables	
Variable	Label
make	Make and Model
price	Price
mpg	Mileage (mpg)
rep78	Repair Record 1978
headroom	Headroom (in.)
trunk	Trunk space (cu. ft.)
weight	Weight (lbs.)
length	Length (in.)
turn	Turn Circle (ft.)
displacement	Displacement (cu. in.)
gear_ratio	Gear Ratio
foreign	Car type

Properties	
Variables	
Name	make
Label	Make and Model
Type	str 18
Format	%-18s
Value Label	
Notes	

Data	
Filename	auto_copy.dta
Label	1978 Automobile Data
Notes	
Variables	12
Observations	74
Size	3.11K
Memory	64M

The Command window at the bottom is empty. The status bar at the bottom shows the file path: C:\Users\jfedler\Documents\StataFiles\stata_plugins and the status indicators: CAP NUM OVR.

We can also summarize the data with the Dta class's `summ` method, and once we're satisfied with the dataset, we can save it as a `dta` file. Above, the optional `replace` argument is used because there's already a `temp.dta` file in this directory. In the last line above, `exit()` is used to return to Stata's default mode.

The screenshot shows the Stata software interface. The main window displays the following commands and their output:

```
. clear  
. use temp  
. list
```

	var0	var1	var2	var3	var4
1.	0	1	2	3	4
2.	10	11	12	13	14
3.	20	21	22	23	24
4.	30	31	32	33	34
5.	40	41	42	43	44
6.	50	51	52	53	54

The Command window at the bottom is empty. The Variables window on the right shows the following information:

Variable	Label
var0	
var1	
var2	
var3	
var4	

The Properties window on the right shows the following information:

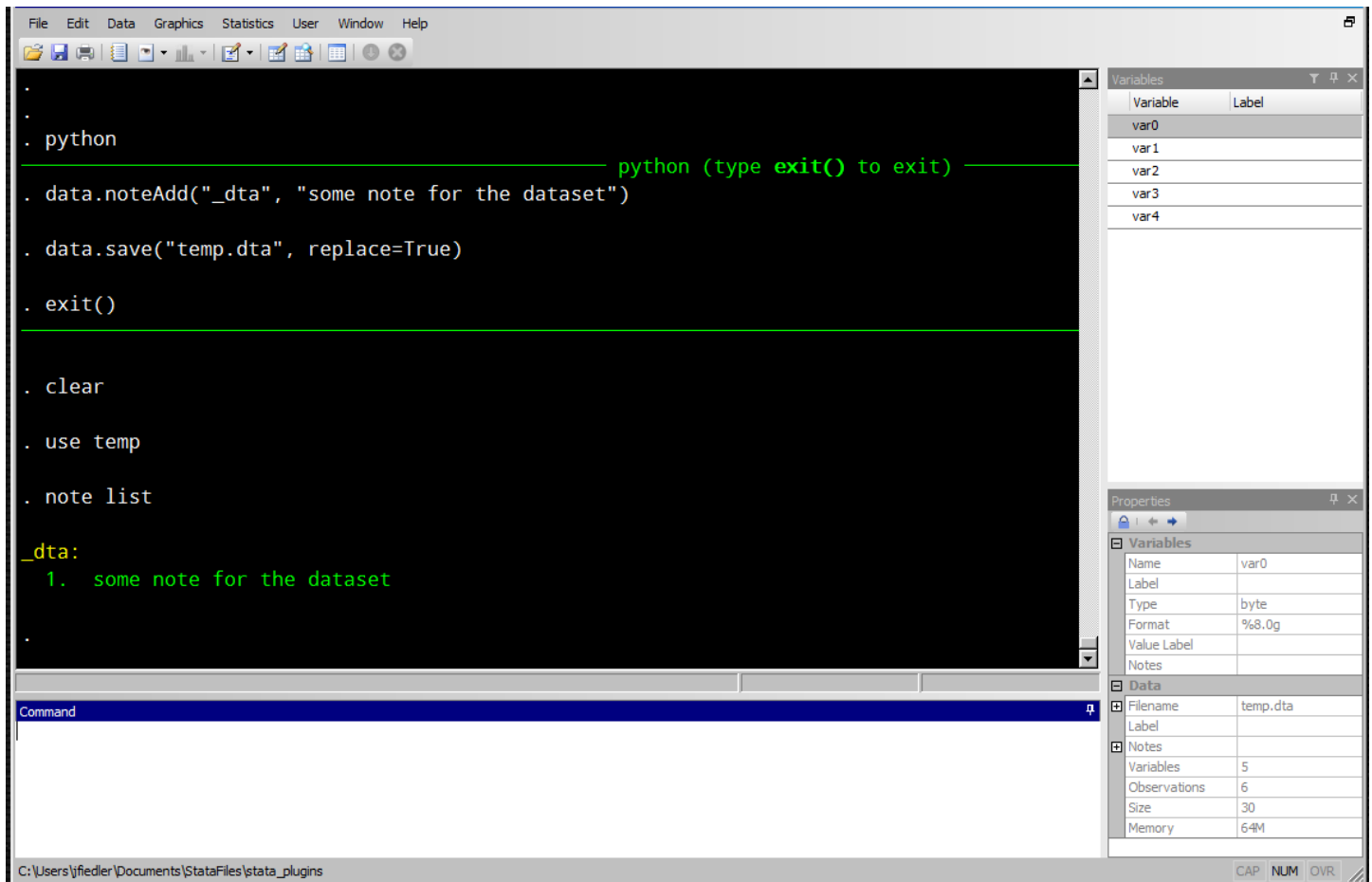
Variables	
Name	var0
Label	
Type	byte
Format	%8.0g
Value Label	
Notes	

The Data window on the right shows the following information:

Data	
Filename	temp.dta
Label	
Notes	
Variables	5
Observations	6
Size	30
Memory	64M

The status bar at the bottom shows the file path: C:\Users\jfedler\Documents\StataFiles\stata_plugins and the current view: CAP NUM OVR.

Now in ado mode, the inputs above clear the existing dataset (a copy of the auto dataset was loaded), loads temp.dta, and lists the values.



Python Dta objects can contain anything that a dta file can. The commands above show that notes can be added in Python. We see that when the dataset is saved (in Python) and opened in Stata, the note is there.

Resources

Stata plugins www.stata.com/plugins/

Python www.python.org

Python C API docs.python.org/3/extending/
docs.python.org/3/c-api/

Sympy sympy.org/en/index.html

Contact jrfiedler@gmail.com

16

The end.