Stata 16 — Under the Hood

Bill Rising

StataCorp LLC

2019 Italian Stata Users Group Meeting
26 September 2019
Firenze
Learn the basics of the frames feature in Stata 16
See what is new in report generation, aka dynamic documents
Methods

- For frames, it will be easy to demonstrate commands and capture their output.
- For the dynamic documents, demonstrating commands is fine, but the output are documents, so the presentation will become much less definite.
- We’ll be working in a series of folders which correspond to each of the topics.
  - If you copied the italy19_rising.zip folder and expanded the files.
  - Make the resulting folder your working directory.
  - The examples here will work relative to that directory.
Frames in Stata 16

- Frames were introduced in Stata 16
- At their simplest, they are a way to have multiple datasets open at once
- They are also something which acts like `merge`
  - But they can save space
- Lastly, there are some things which get sped up because of frames
Basics of Frames

- Think of a frame as a place to hold data
  - The data can be in a dataset or simply in the frame
- Each frame has an internal Stata name
  - The first frame, which exists when you start Stata, is called default, by default
Starting Simple: Frames for Multiple Datasets

- First, go to the frames folder
  - . cd frames
- Open a dataset
  - . use visit_info
- Create a second frame
  - . frame create patients
- Open another dataset in that other frame
  - . frame patients: use patient_info
Glancing at the Datasets

- Open the data editor, to see the dataset
  - `.edit`
- Switch back and forth between frames via `cwf`
  - `.cwf patients`
- Or switch back and forth using `frame change`
  - `.frame change default`
- Or switch back and forth using the `frames` dialog
  - `.db frames`
Changing Frame Names

- The default frame has a forgetable name in our case
  - it forces us to remember which dataset has this special status
- We can change the name of the default frame name to something more informative
  . frame rename default visits
- We can then look at what frames we have
  . frame dir
    - The numbers given are observations × variables
    - Or if you prefer rows × columns
It would make sense to combine the information in the `visit_info` and `patient_info` datasets.

- This is normally a task for the `merge` command.

Instead of using `merge`, you can link together datasets in frames:

- This can be good for very long datasets.
- It has some other advantages (and disadvantages).
How to Link

- The possible link types are 1:1 and m:1
  - There is fine; the 1:m really is not needed because all that need be done is to switch the active frame
- In this example there can be multiple visits per patient, so we need to have the visits frame active
  - . cwf visits
- Now we can link on patid
  - . frlink m:1 patid, frame(patients)
Upshot of Linking

- A new variable gets created in the dataset in the active frame
  - By default, this is named after the frame which was linked
- You can tell indirectly which observations matched up in the active frame
  - Those which matched have non-missing values for the linking variable
  - Those which did not match up with data in the linked dataset have missing variables for the linking variable
- You cannot tell which observations did not match in the linked frame
  - This is similar to having `_merge` values of 1 and 2 only
The `frval()` function allows you to use values from a variable in the linked frame without actually copying the variable into the current frame.

- Which saves space if the active frame is long.

We could list all the visits from the female patients:

```
. list patid-doctor if frval(patients, gender) eq "Female"
```

This function can be used in any `exp` anywhere:

```
. gen ins_diff = insurance != frval(patients, insurance)
```

- This shows where the insurance differs in the two datasets.

```
. list patid visitdt insurance if ins_diff
```
Adding Variables from a Linked Frame

- You can bring over variables from a linked dataset
  - `frget birthdate, from(patients)`
- `frget` copies the data as well as all metadata from the linked variable
- This is similar to
  - `merge m:1 patid using patient_info, keepusing(birthdate)`
    - As it turns out, linking has better behavior for value labels, as we will see
- This is good for computing age
  - `do genage`
- Here are the ages
  - `list patid visitdt birthdate age`
Adding a Variable Whose Name Exists

- If you want to bring over a variable whose name matches one of the variable names in the active frame:
  - You can generate a new variable with a different name
    . frget pat_insurance = insurance, from(patients)
  - You can use a prefix or a suffix
    . frget insurance, from(patients) prefix(another_)
  - If you don’t try to change the conflicting name, you will get an error
Good Value Label Behavior

- If the variable you bring over has a value label
  - If the value label does not exist in the active frame, the value label comes over
  - If the value label exists in the active frame and the definitions match, then nothing need be done
  - If the value label exists in the active frame and the definitions do not match, then the brought-over value label gets renamed
    - This is better behavior than with `merge`, which simply issues a warning
In this example, the value label `instype` exists in both datasets.

It would be good to look at the definitions.

We would like to do this without having to switch back and forth between frames.

- In the `visits` frame, which is active.
  - `label list instype`

- In the `patients` dataset.
  - `frame patients: label list instype`

- Ignoring that the `visits` frame is active.
  - `frame visits: label list instype`

In any case, we can see that the value labels are all defined well.
Suppose our `patient_info` dataset were not quite so nice.

The `patient_ohno` dataset fits this bill.

- We will want to link to this.

Let’s look at it the frames way.

First create a frame.

- `. frame create ohno`

Now open up the dataset in that frame.

- `. frame ohno: use patient_ohno`

And look at it.

- `. frame ohno: codebook`
Things to Note

- The patid is now called just id
- The insurance variable is encoded differently, but still has the instype value label
  - This would be a big problem when using merge, update
We can still use `frlink` to link to a dataset where the key variables have different names

- Key: variable list which identifies individual variables in one dataset

To do this, we must specify the `keyvarlist` in the `frame()` option

```
    . frlink m:1 patid, frame(ohno id)
```
Avoiding A Dangerous Data Error

- Just to drive home the point, check that the instype value labels differ
  - First in the active frame
    . label list instype
  - Now in the linked dataset
    . frame ohno: label list instype

- Try to bring in the insurance variable from the ohno frame
  . frget insurance, from(ohno) prefix(ohno_)

- Look at the value labels
  . label list

- Stata renamed the value label from frget to avoid a data error!
  - This is better behavior than in merge
Notes about Linking

- You can use `frget` to grab many variables from the linked dataset
  ```
  frget varlist ...
  ```

- You could grab all but some variables by using the `exclude()` option
  ```
  frget _all, exclude(notthisvarlist)
  ```

- This is like using the `keepusing()` option in `merge` except that it allows excluding instead of just including variables
Changing the key in the active frame is dangerous!

Here is such a dangerous change
\[
\text{. replace patid = 9 if patid == 4 & visitdt==mdy(10,19,2015)}
\]

Now go and get the gender variable
\[
\text{. frget gender, from(patients)}
\]

Because the linking is static, you can get odd results
\[
\text{. tabulate patid gender}
\]
If you are unsure of the state of the links, you should rebuild them.
  . frlink rebuild patients

Now go and grab the `gender` variable again.
  . drop gender
  . frget gender, from(patients)

Now there are no problems.
  . tabulate patid gender
Clearing out

- The equivalent to `clear` for frames is `. clear frames`
  - This gets rid all data and frames and changes the active frame name to `default`:
    - `. frames dir`
  - `frames reset` is a synonym

- In case you wondered, `clear all` runs a `clear frames`
Frames as Holding Areas

- You can also use frames for holding data
  - In this case, they are something of a substitute for temporary files
  - They are also faster, especially in networked environments

- `frput` will copy data to another frame
  - The opposite of `frget`

- `frcopy` will copy an entire frame to another frame
  - It will also create the frame to use the copy, making it a nice manual preserve

- `frame post` can be used to post observations
  - Similar to `post`, but without tmp files
The `preserve` command now uses frames for preserving in Stata/MP

- This happens for files under 1GB by default
- The maximum size can be changed using `set max_preservemem`

This speeds up commands which use `preserve` heavily

- `grexample` for looking at graph examples

This is especially useful when on a network where temporary files end up being stored on a server, instead of locally
Linking Many Datasets

- You can have up to 100 frames at once
- This means you can link together 100 datasets if need be
- This could be useful in very wide datasets
The report generation (aka dynamic document) tools have been extended.

dyndoc now has a docx option which produces a docx document directly from markdown.

putdocx has many additions for headers and footers, as well as a way to make narrative easier to use.

html2docx converts web pages (html) to Microsoft Word compatible documents (docx).

docx2pdf converts docx files to pdf files.

There are a few other additions; these are the ones we’ll look at.
- We’ll start with the docx option for dyndocx
- Let’s move to the proper location
  . cd ../dyndoc
Looking at a *dyndoc* file

- Take a look at the *paper.md* file
  - `. doedit paper.md`

- This is an example markdown file using Stata’s dynamic tags
  - You can see that Stata 16 now has syntax highlighting for markdown
  - The *md* extension is what alerted the Do-file Editor to use this highlighting
  - You can change the language being highlighted

- Note that the *dyndoc* version has changed to 2
Making an html file

- As in Stata 15, this can be turned into a webpage
  - .dyndoc paper.md
  - The output is not shown, because it would include all the output needed to make the html file
- We can click on the link to open the page
Converting to docx

- We could then convert this to a docx file
  \texttt{html2docx paper.html, saving(paper_conv.docx)}
- Clicking the link will open the docx file in Microsoft Word
- The resulting file needs some fixing up, but we’ll do this later
Going Directly from Markdown to docx

- We could get the same result by using the new docx option for dyndoc
  . dyndoc paper.md, docx
- Again, the output is not shown

- This will look exactly like the preceding example, because in the background, Stata is running plain dyndoc then running html2docx

- Generally, this worked well
  - There is some wrapping of Stata output, however
  - This is not present here, but there are other html-only things, like special characters, which might need cleaning up
Tidying Up Wrapping

- Doing this conversion is nice, but it sometimes needs some tidying up due to wrapping
  - The font size of 10pt for the fixed-width font allows 77 characters per line for letter size paper with standard one-inch margins
  - If your Stata window is wide, commands like `describe` and `codebook` will draw dashed line the entire width of the your window

- There are a few things which can help
  - Use a `set linesize` command to set the linesize to 90 or less
  - Change the margins in the resulting `docx` document
  - Make a style sheet (css) for the document and `«dd_include»` the style sheet
    - See the first example in the `dyndoc PDF` documentation
Working With putdocx

- The files for putdocx are in the putdocx folder
  - cd ../putdocx
- First take a look at how putdocx looked in Stata 15
  - doedit putdocx15.do
- You can see here that there is no narrative mode
  - Everything is a Stata command
- You also cannot put Stata code into the document without repeating it
  - Once as simple text in a fixed-width font
  - Once as code that gets run
Making the docx Document

- Doing the do-file will make a docx document
  . do putdocx15.do

- On the Mac, you can open the resulting file from the Command window
  . ! open putdocx15.docx
Stata 16 allows headers and footers

- Headers and footers can change through the document with sections
- Headers and footers can work across appending files
- There is now something like a narrative mode
- Open up putdocx16.do to see these
  . doedit putdocx16.do
They get constructed in a couple of steps

Here are the steps for a footer

- Use `putdocx begin, footer(name)` to name the footer
- Use `putdocx paragraph, tofooter(name)`
- Then add to the paragraph
  - Using tables is good for multi-piece footers

For headers, simply use `header` in place of `footer` above
Headers and Footer Changes

- When sections change, you can change the header and/or footer
- Simply use `putdocx sectionbreak` in place of `putdocx begin` from above
Narrative Mode

- While `putdocx` is mostly all Stata command as before, there are now text blocks:
  - `putdocx textblock begin` starts a new paragraph which is simply text
  - `putdocx textblock append` appends to the current paragraph
  - `putdocx textblock end` ends a text block
  - `putdocx textfile` allows inserting a file as a text block

- These should make documents with a lot of plain narrative (i.e. most documents) much easier to work with
Making the .docx Document

- Doing the do-file will make a .docx document
  . do putdocx16.do

- Open the resulting file from the Command window
  . ! open putdocx16.docx
Other Changes

- While these are most of the changes, there have also been a few changes to:
  - markdown, which goes from markdown to html without processing Stata code
  - putexcel had 2 syntax changes
    - putexcel close has become putexcel save
    - putexcel has changed picture() to image()
  - Of course, version control will protect your Stata 15.1 and earlier do-files!
Frames are something brand new in Stata 16

The dynamic document (aka report) generation has had some nice additions