

Fundamentals of Scientific Computing with Stata

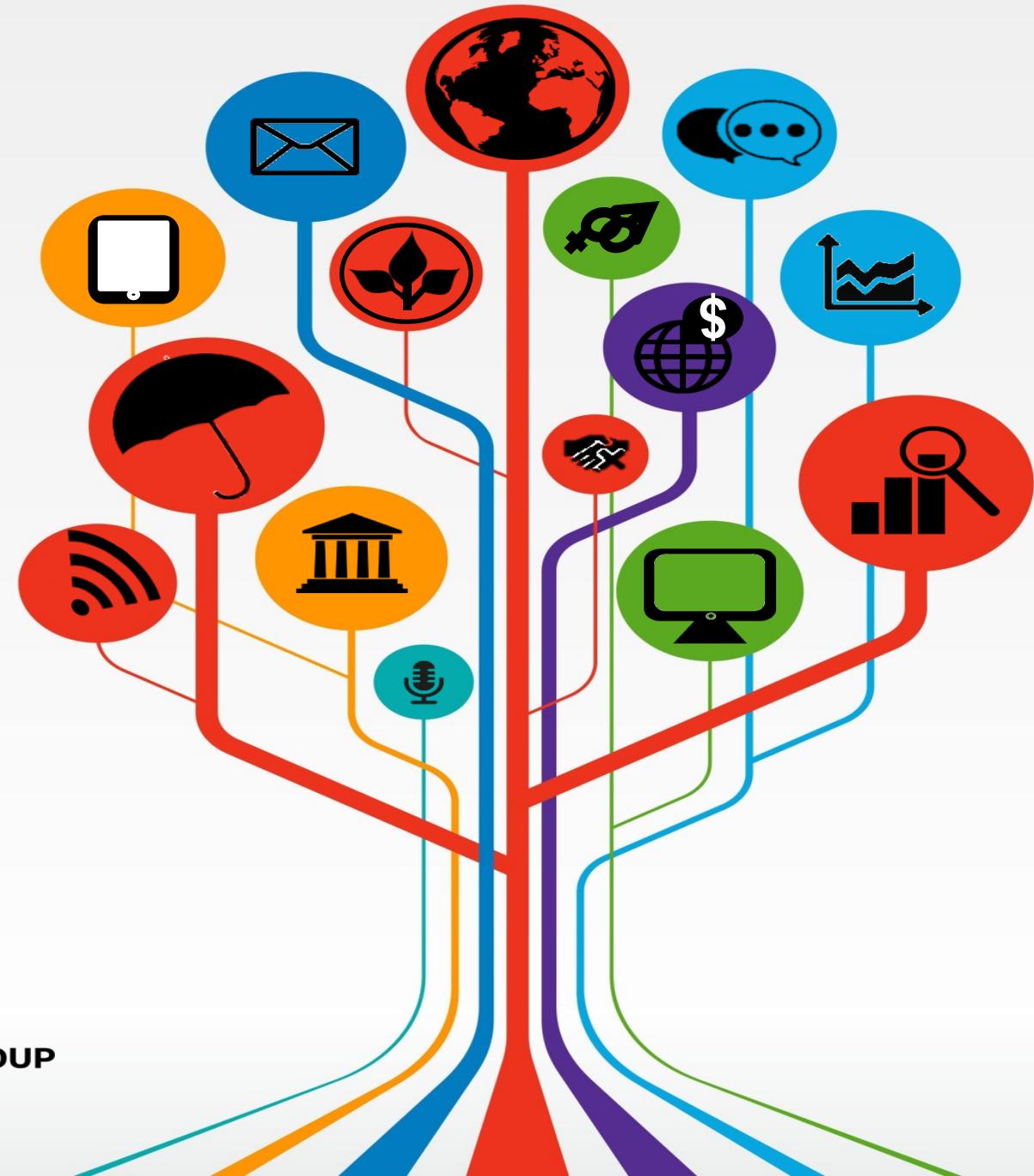
Research Assistant Onboarding

Prepared by DIME Analytics

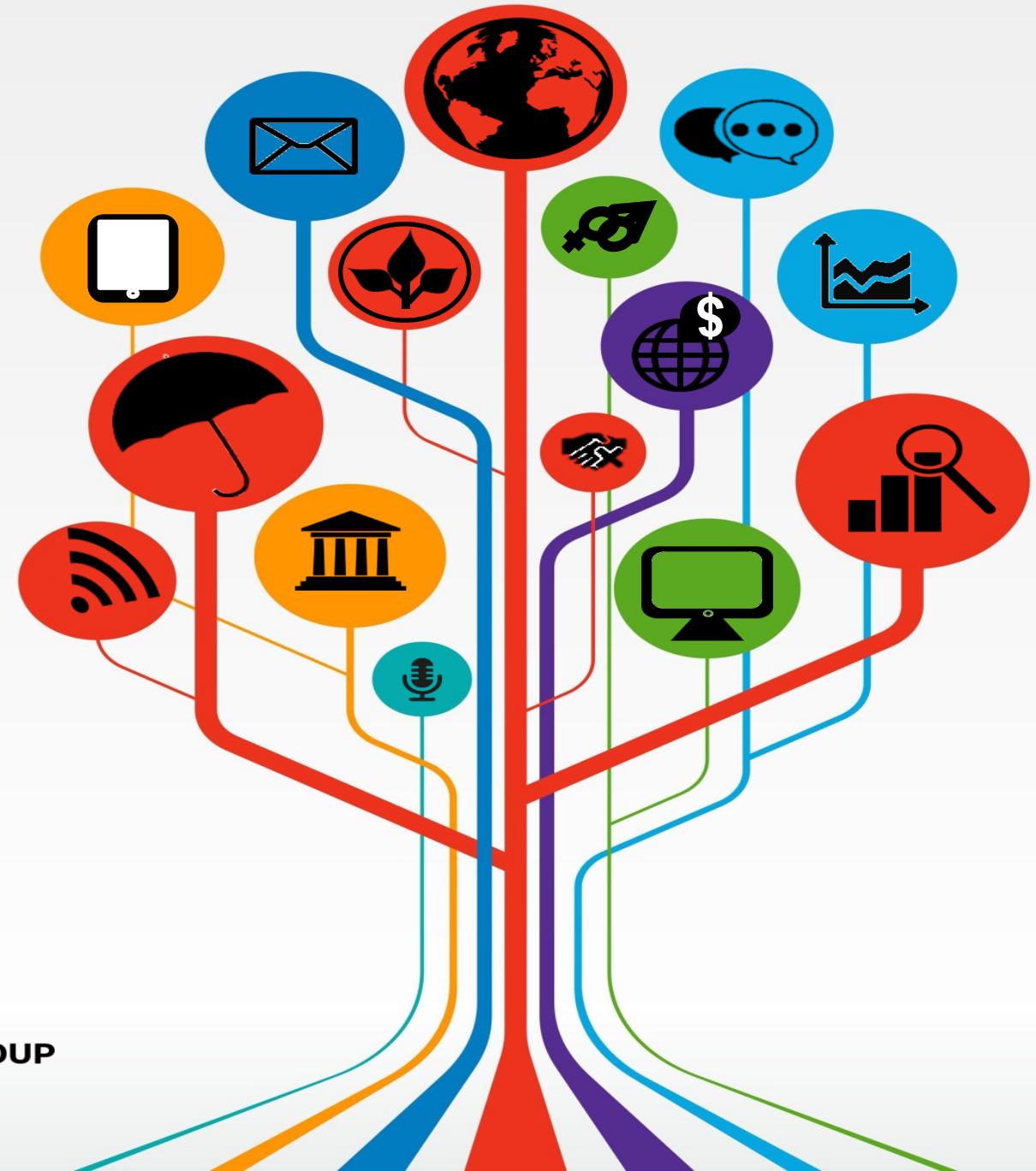
dimeanalytics@worldbank.org

Presented by Benjamin Daniels

bdaniels@worldbank.org

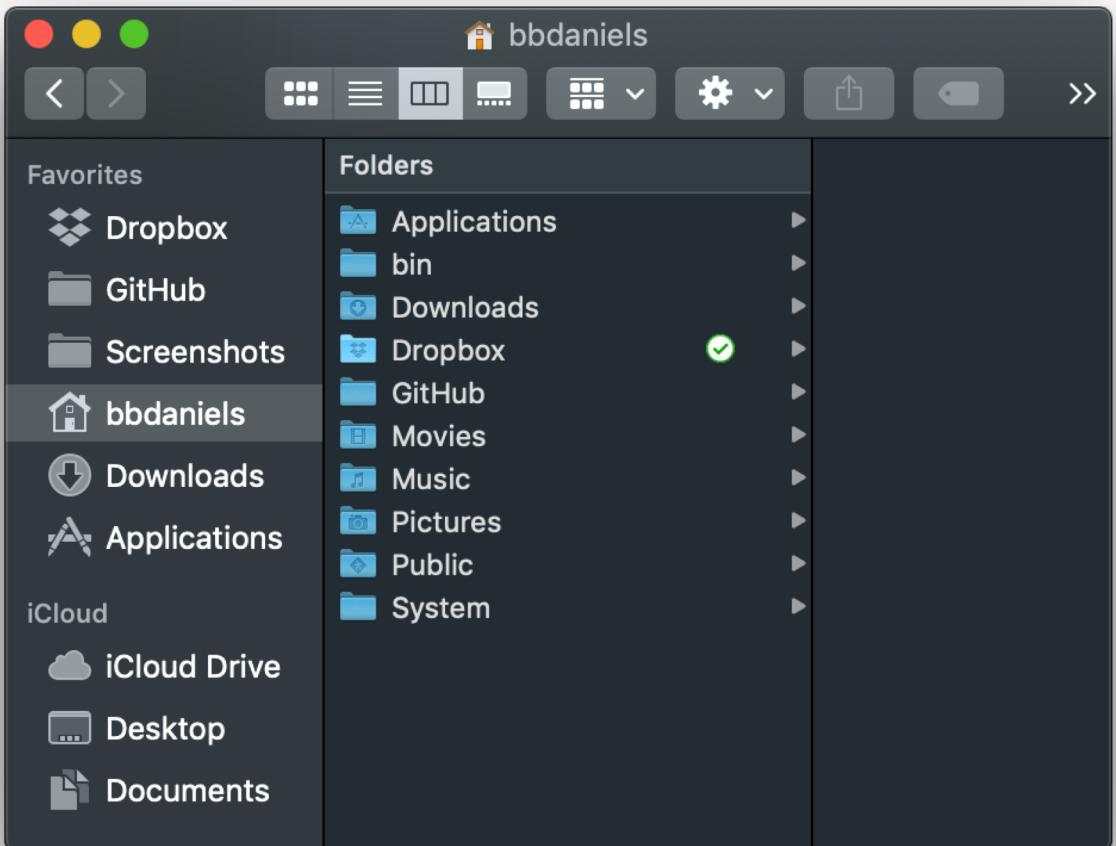


Keeping Stuff Organized



Folder Structure

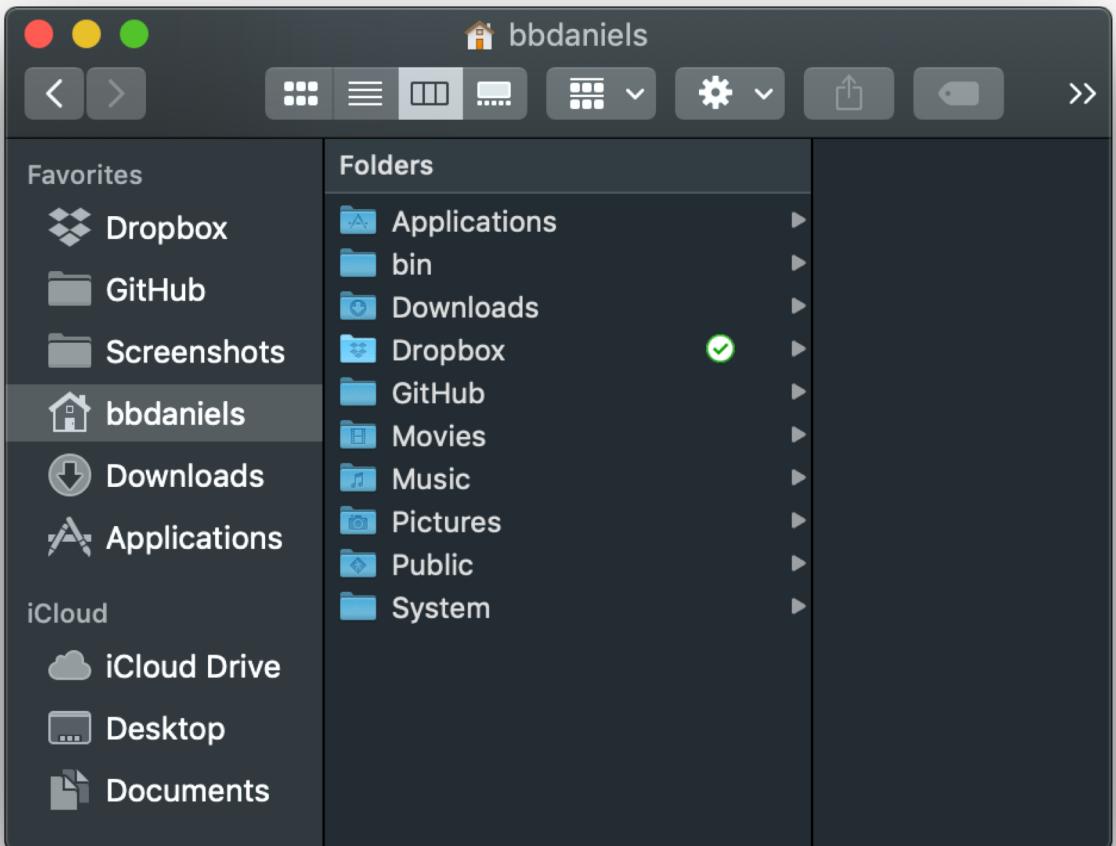
- Basic location: “Home” folder (~)
- Everything lives here
 - NOT your desktop
 - NOT a synced folder (including iCloud and Dropbox)
- Many people don’t have one!
- Everyone here must – it will standardize most of your \${directory} globals in Stata



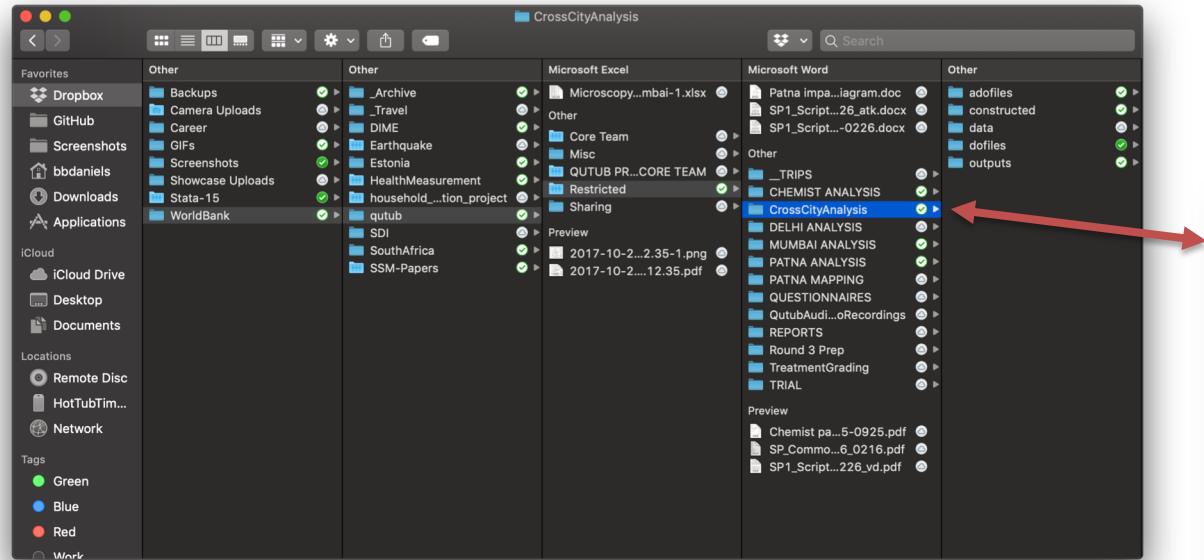
Folder Structure

Your work and tools live in:

- GitHub and Dropbox
- Program files
- Downloads
- Screenshots (in Dropbox)



Make file structure part of your workflow



Use a file browser and layout that helps you remember and interact with the file structure...

...and a code editor that zooms in on the portion of the file structure you need to work with.

A screenshot of a code editor showing a Stata do-file named '_Master_Qutub.do'. The code includes global statements, loops, and various Stata commands. A red double-headed arrow highlights the 'dofiles' section of the code, which corresponds to the 'dofiles' folder in the file browser above. The code editor has syntax highlighting and a scroll bar.

File and folder names must be structured

Three principles for names:

- 1. Machine Readable**
- 2. Human Readable**
- 3. Orders Well**

NO

- **myabstract.docx**
- **Joe's Filenames Use Spaces and Punctuation.xlsx**
- **figure 1.png**
- **fig 2.png**

YES

- 2014-06-08_abstract-for-sla.docx
- joes-filenames-are-getting-better.xlsx
- fig01_scatterplot-talk-length-vs-interest.png
- fig02_histogram-talk-attendance.png
- 1986-01-28_raw-data-from-challenger-o-rings.txt

(["naming things" by Jenny Bryan](#))

Follow a few simple rules in naming schemes

Rules for files & folders:

- **Underscore _ separates levels of information (order, purpose, date)**
- **En-dash - separates words**
- **Numbers are left-padded (01,02,03...)**
- **NO SPACES**
- **NO SPECIAL CHARACTERS**
- **NO CAPITAL LETTERS**

(“naming things” by Jenny Bryan)

This list orders correctly and you can tell what everything is:

- 01_marshall-data.r
- 02_pre-dea-filtering.r
- 03_dea-with-limma-voom.r
- 04_explore-dea-results.r
- 90_limma-model-term-name-fiasco.r
- helper01_load-counts.r
- helper02_load-exp-des.r
- helper03_load-focus-statinf.r
- helper04_extract-and-tidy.r

Dates: YYYY-MM-DD

Global ISO 8601 standard:

YYYY-MM-DD

If you do not do this, your files *will* be out of order.

PUBLIC SERVICE ANNOUNCEMENT:

OUR DIFFERENT WAYS OF WRITING DATES AS NUMBERS CAN LEAD TO ONLINE CONFUSION. THAT'S WHY IN 1988 ISO SET A GLOBAL STANDARD NUMERIC DATE FORMAT. THIS IS *THE* CORRECT WAY TO WRITE NUMERIC DATES:

2013-02-27

THE FOLLOWING FORMATS ARE THEREFORE DISCOURAGED:

02/27/2013 02/27/13 27/02/2013 27/02/13
20130227 2013.02.27 27.02.13 27-02-13
27.2.13 2013. II. 27. 27½-13 2013.158904109
MMXIII-II-XXVII MMXIII ^{LVII}/_{CCCLXV} 1330300800
 $((3+3)\times(111+1)-1)\times3/3-1/3^3$ 2013 
10/11011/1101 02/27/2013 

<https://xkcd.com/1179/>

Keeping Code Organized



Top 8 Best Practices for Scientific Computing

1. Write programs for people, not computers.
2. Let the computer do the work.
3. Make incremental changes.
4. Don't repeat yourself (or others).
5. Plan for mistakes.
6. Optimize software only after it works correctly.
7. Document design and purpose, not mechanics.
8. Collaborate.

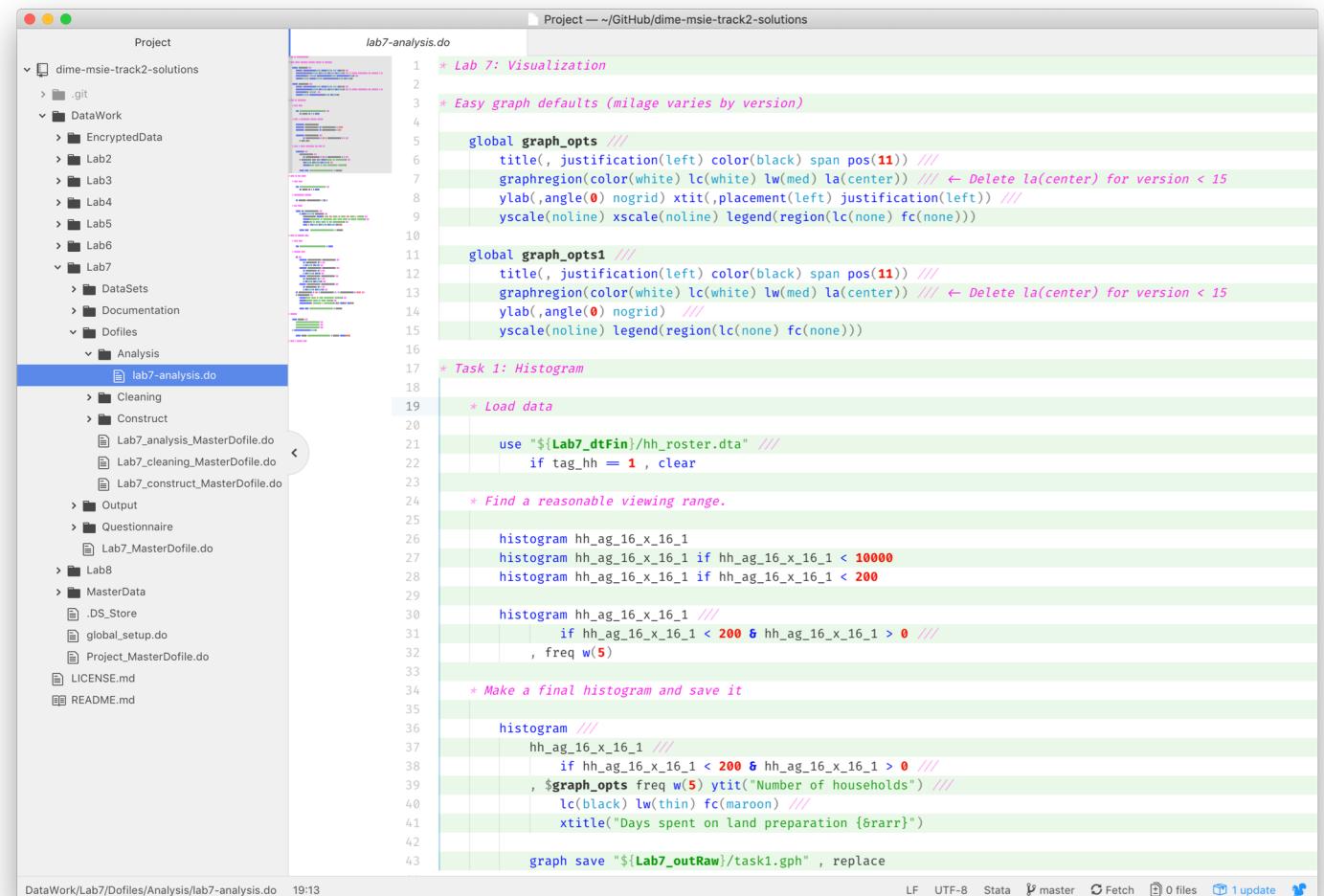
Wilson et al., “[Best Practices for Scientific Computing](#)”.



<https://xkcd.com/1597/>

Write programs for people, not computers.

- Very short code “chunks”
- Clear purpose for each
- Many comments (easier if your editor allows pretty colors for comments)
- Logical indentation – shows “belonging” of every line of code

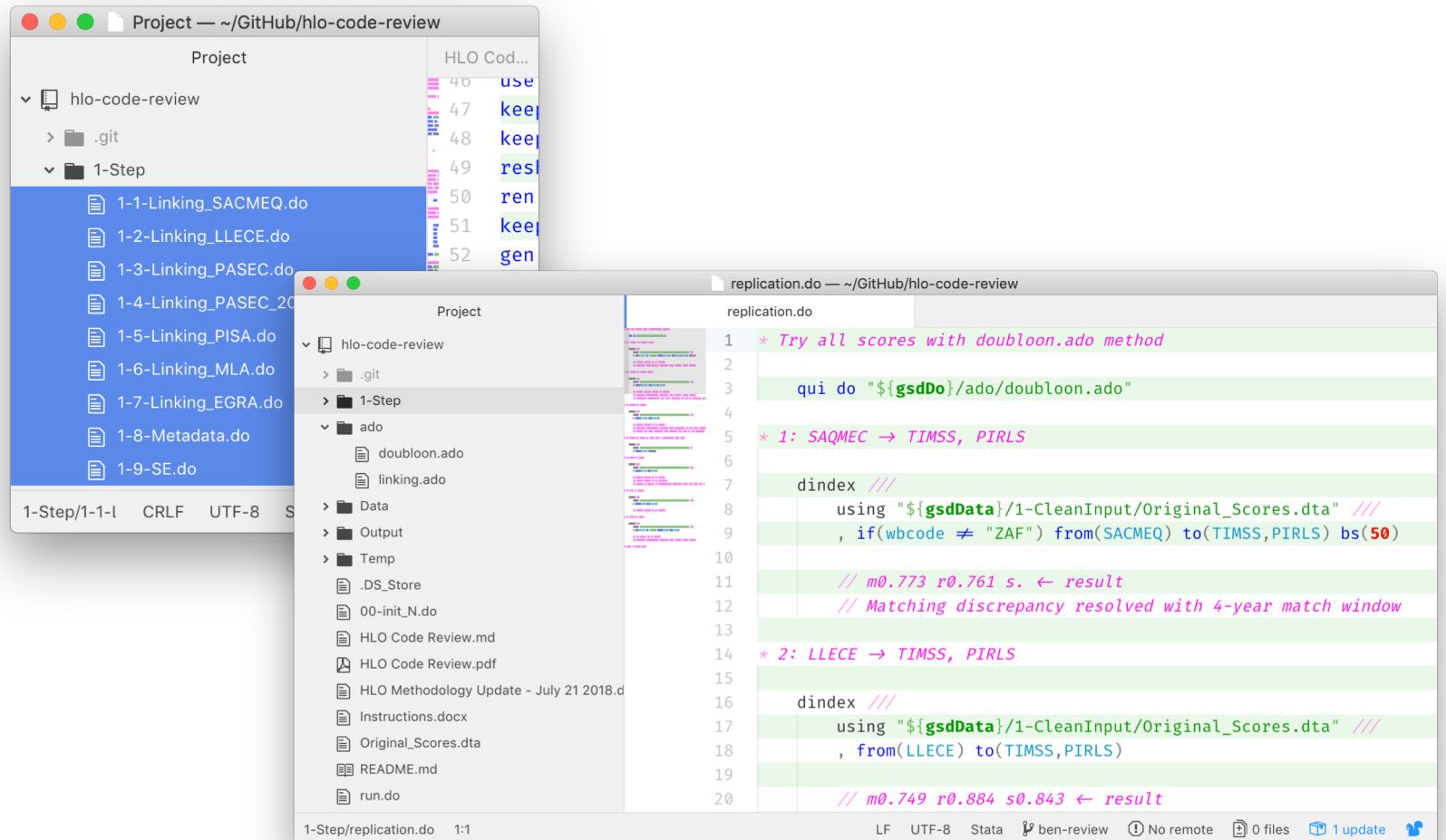


The screenshot shows a Stata IDE interface. On the left, a project tree displays a directory structure for 'dime-msie-track2-solutions' containing subfolders like '.git', 'DataWork', 'Lab2', 'Lab3', 'Lab4', 'Lab5', 'Lab6', 'Lab7', 'Lab8', 'MasterData', '.DS_Store', 'global_setup.do', 'Project_MasterDofile.do', 'LICENSE.md', and 'README.md'. The 'Analysis' folder is expanded, showing files such as 'lab7-analysis.do', 'Cleaning', 'Construct', 'Lab7_analysis_MasterDofile.do', 'Lab7_cleaning_MasterDofile.do', 'Lab7_construct_MasterDofile.do', 'Output', 'Questionnaire', and 'Lab7.MasterDofile.do'. The right pane shows the contents of 'lab7-analysis.do'. The code is color-coded with pink for comments and blue for other text. It includes sections for visualization setup, histogram tasks, and final histogram saving. A status bar at the bottom indicates the file is 'DataWork/Lab7/Dofiles/Analysis/lab7-analysis.do' and the time is '19:13'.

```
* Lab 7: Visualization
*
* Easy graph defaults (milege varies by version)
*
global graph_opts ///
    title(, justification(left) color(black) span pos(11)) ///
    graphregion(color(white) lc(white) lw(med) la(center)) /// ← Delete la(center) for version < 15
    ylab(,angle(0) nogrid) xtit(,placement(left) justification(left)) ///
    yscale(noline) xscale(noline) legend(region(lc(none) fc(none)))
*
global graph_opts1 ///
    title(, justification(left) color(black) span pos(11)) ///
    graphregion(color(white) lc(white) lw(med) la(center)) /// ← Delete la(center) for version < 15
    ylab(,angle(0) nogrid) ///
    yscale(noline) legend(region(lc(none) fc(none)))
*
* Task 1: Histogram
*
* Load data
use "${Lab7_dtFin}/hh_roster.dta" ///
if tag_hh == 1, clear
*
* Find a reasonable viewing range.
histogram hh_ag_16_x_16_1
histogram hh_ag_16_x_16_1 if hh_ag_16_x_16_1 < 10000
histogram hh_ag_16_x_16_1 if hh_ag_16_x_16_1 < 200
*
histogram hh_ag_16_x_16_1 ///
if hh_ag_16_x_16_1 < 200 & hh_ag_16_x_16_1 > 0 ///
, freq w(5)
*
* Make a final histogram and save it
*
histogram ///
hh_ag_16_x_16_1 ///
if hh_ag_16_x_16_1 < 200 & hh_ag_16_x_16_1 > 0 ///
, $graph_opts freq w(5) ytit("Number of households") ///
lc(black) lw(thin) fc(maroon) ///
xtitle("Days spent on land preparation {&rarr;}")
*
graph save "${Lab7_outRaw}/task1.gph" , replace
```

Let the computer do the work.

- You shouldn't be copy-pasting the same things
- Computers are for boring and repetitive tasks
- You are for figuring out how to make the computer do it!



The screenshot shows a Stata IDE interface with two main windows. The left window displays a project structure for 'hlo-code-review' with a '1-Step' folder containing nine .do files: 1-1-Linking_SACMEQ.do, 1-2-Linking_LLECE.do, 1-3-Linking_PASEC.do, 1-4-Linking_PASEC_2011.do, 1-5-Linking_PISA.do, 1-6-Linking_MLA.do, 1-7-Linking_EGRA.do, 1-8-Metadata.do, and 1-9-SE.do. The right window shows the content of 'replication.do'. The code in 'replication.do' is as follows:

```
* Try all scores with doublon.ado method
qui do "${gsdDo}/ado/doublon.ado"

* 1: SAQMEC → TIMSS, PIRLS
dindex ///
    using "${gsdData}/1-CleanInput/Original_Scores.dta" ///
    , if(wbcode != "ZAF") from(SACMEQ) to(TIMSS,PIRLS) bs(50)

// m0.773 r0.761 s. ← result
// Matching discrepancy resolved with 4-year match window

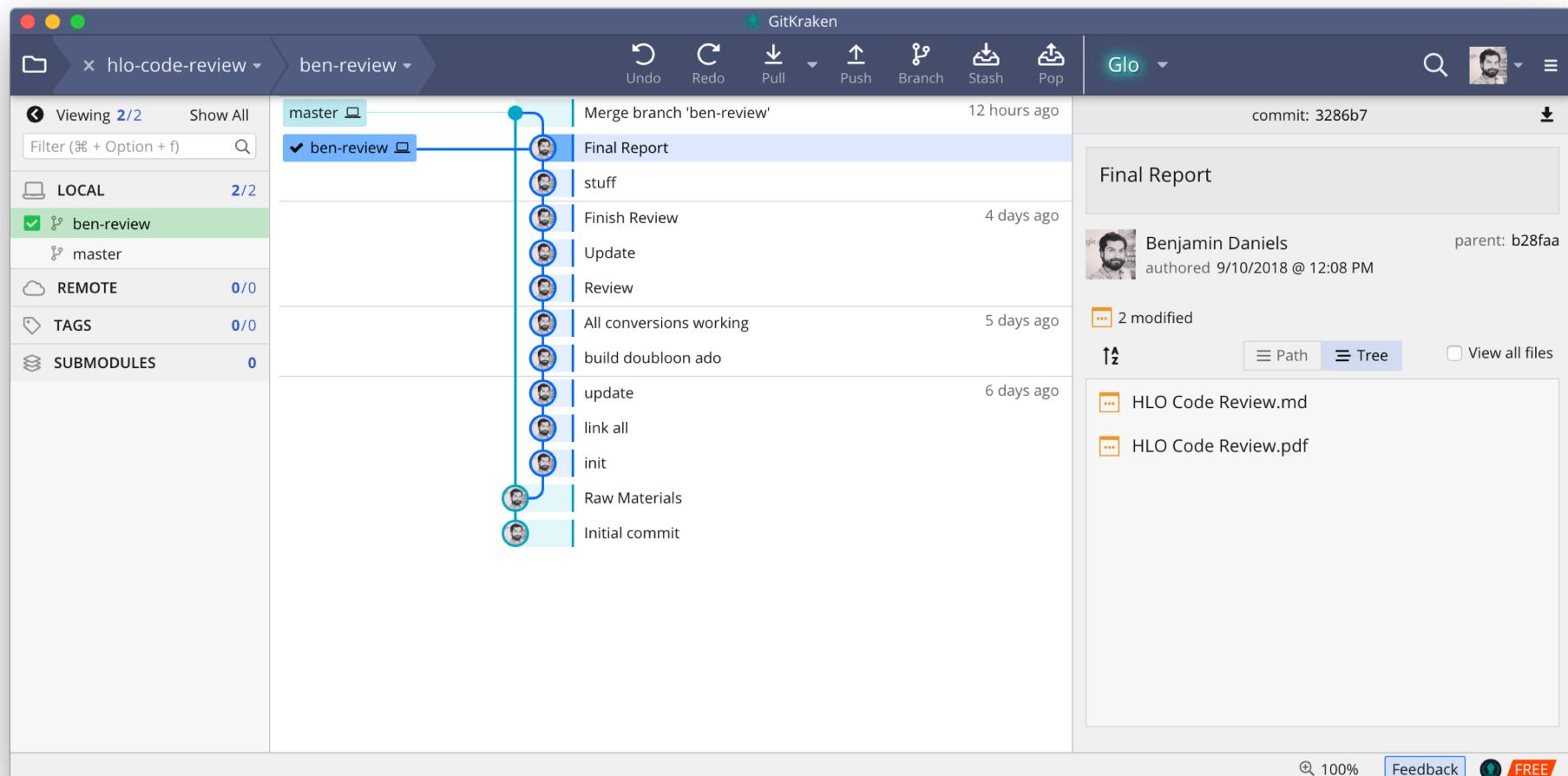
* 2: LLECE → TIMSS, PIRLS
dindex ///
    using "${gsdData}/1-CleanInput/Original_Scores.dta" ///
    , from(LLECE) to(TIMSS,PIRLS)

// m0.749 r0.884 s0.843 ← result
```

Make incremental changes.

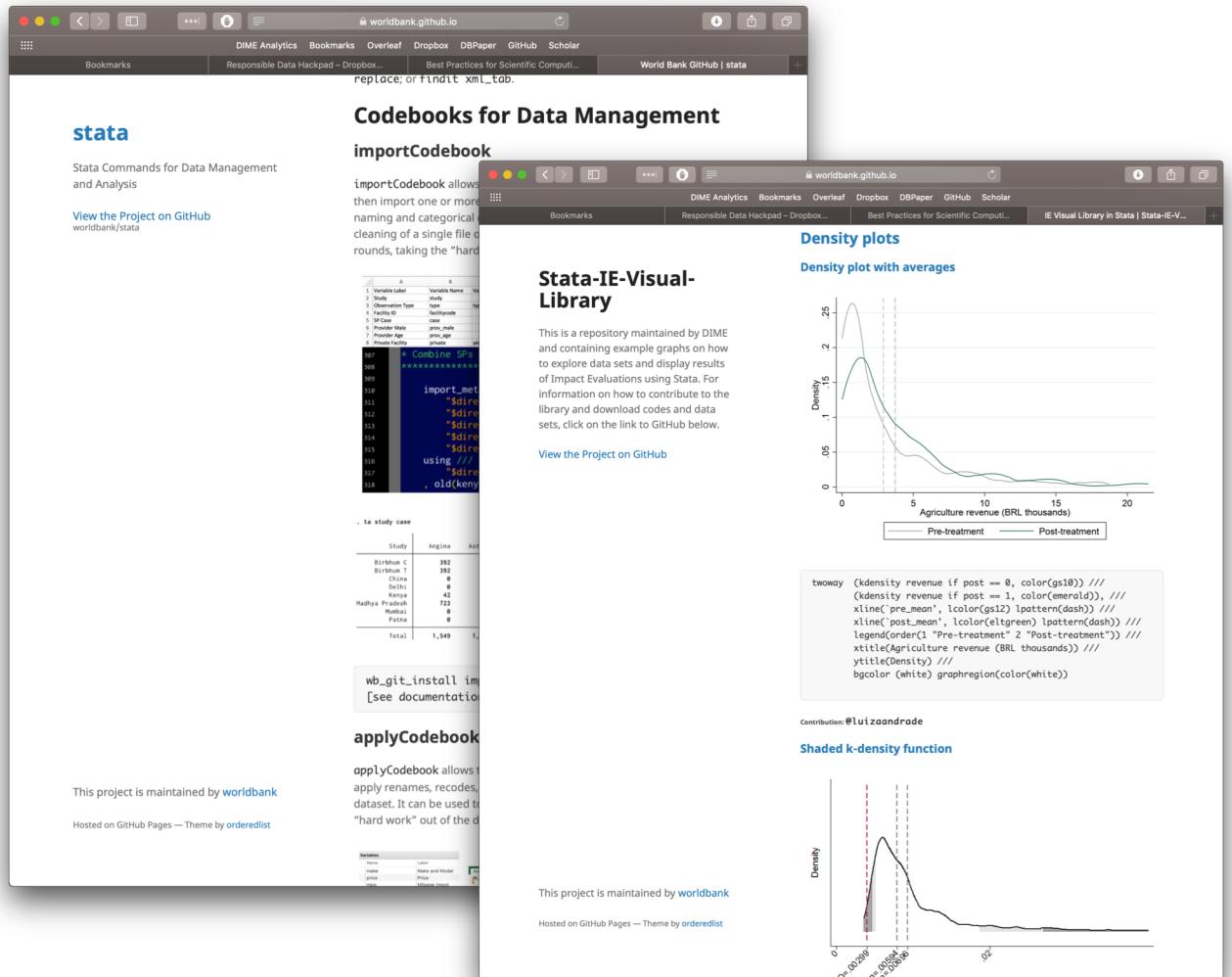
Avoid “task entanglement”

- One task at a time
- Easy method for switching between tasks



Don't repeat yourself (or others).

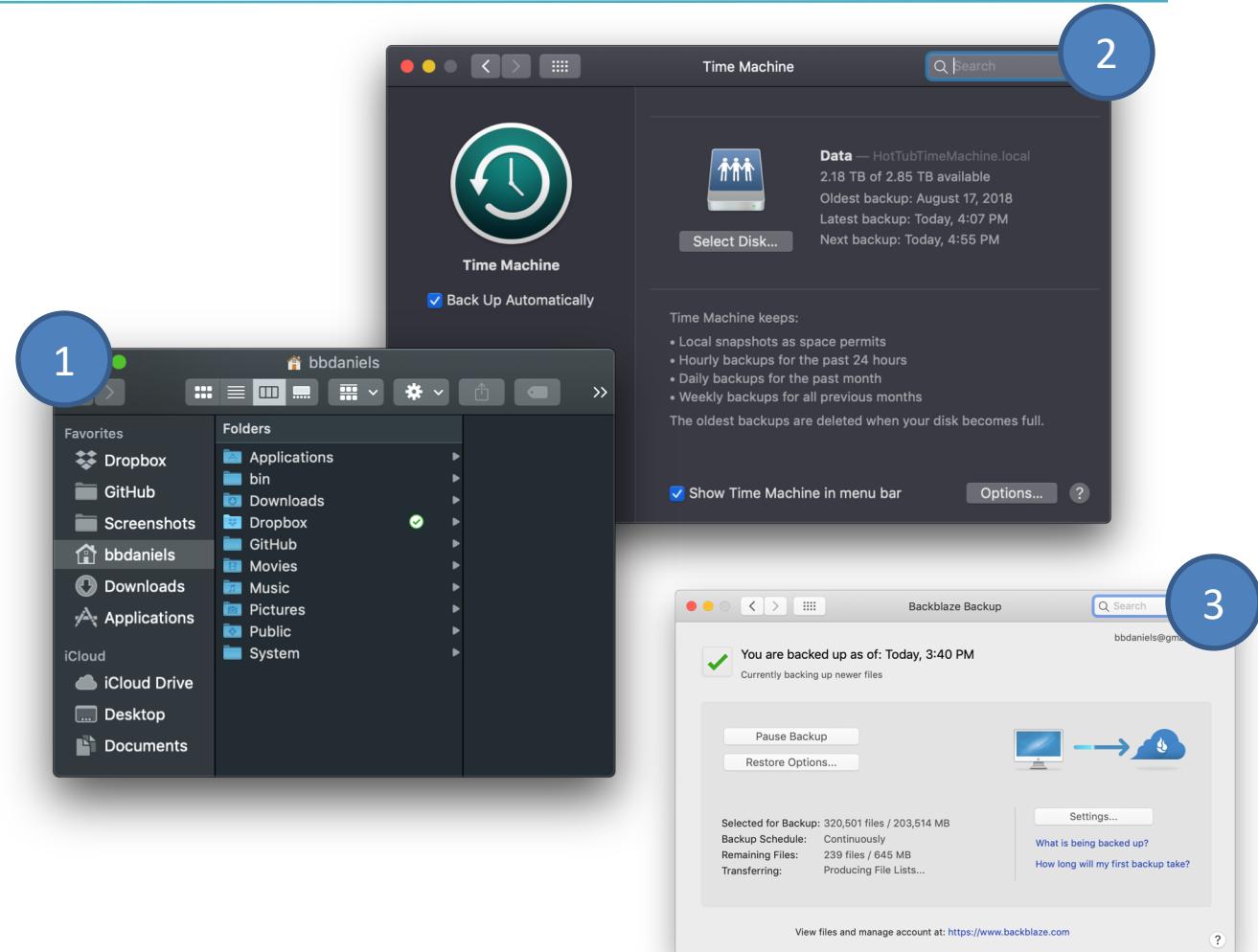
- Maintain a “code library” of your own good work and visit those of others
- Use DIME Wiki
- Write or borrow ado-files for common tasks
- Clean data *before* analysis files begin



Plan for mistakes.

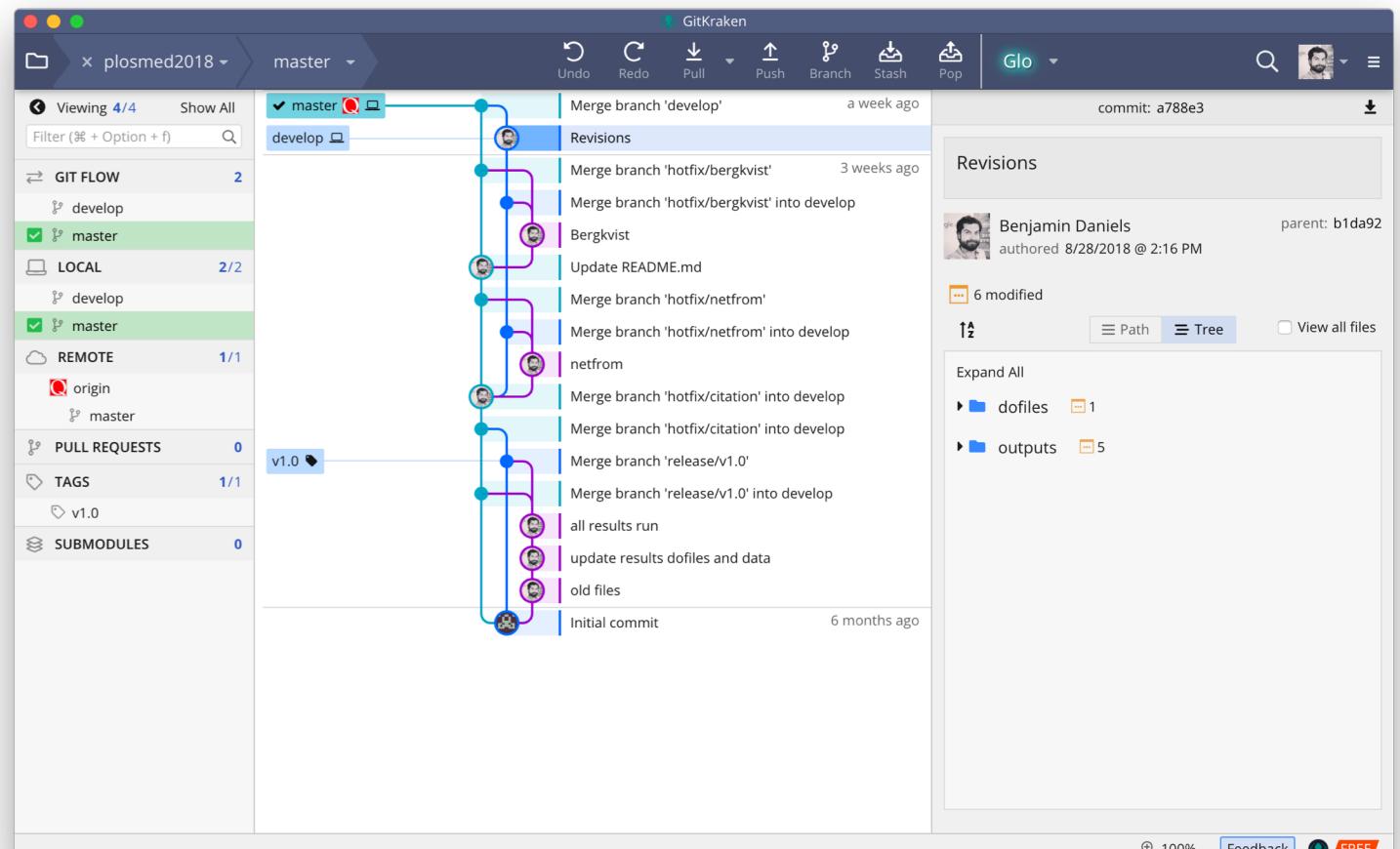
3-2-1-0 Backup Strategy:

- 3 total copies of your data
- 2 on different devices you own
- 1 copy off-site
- 0 Dropbox does not count



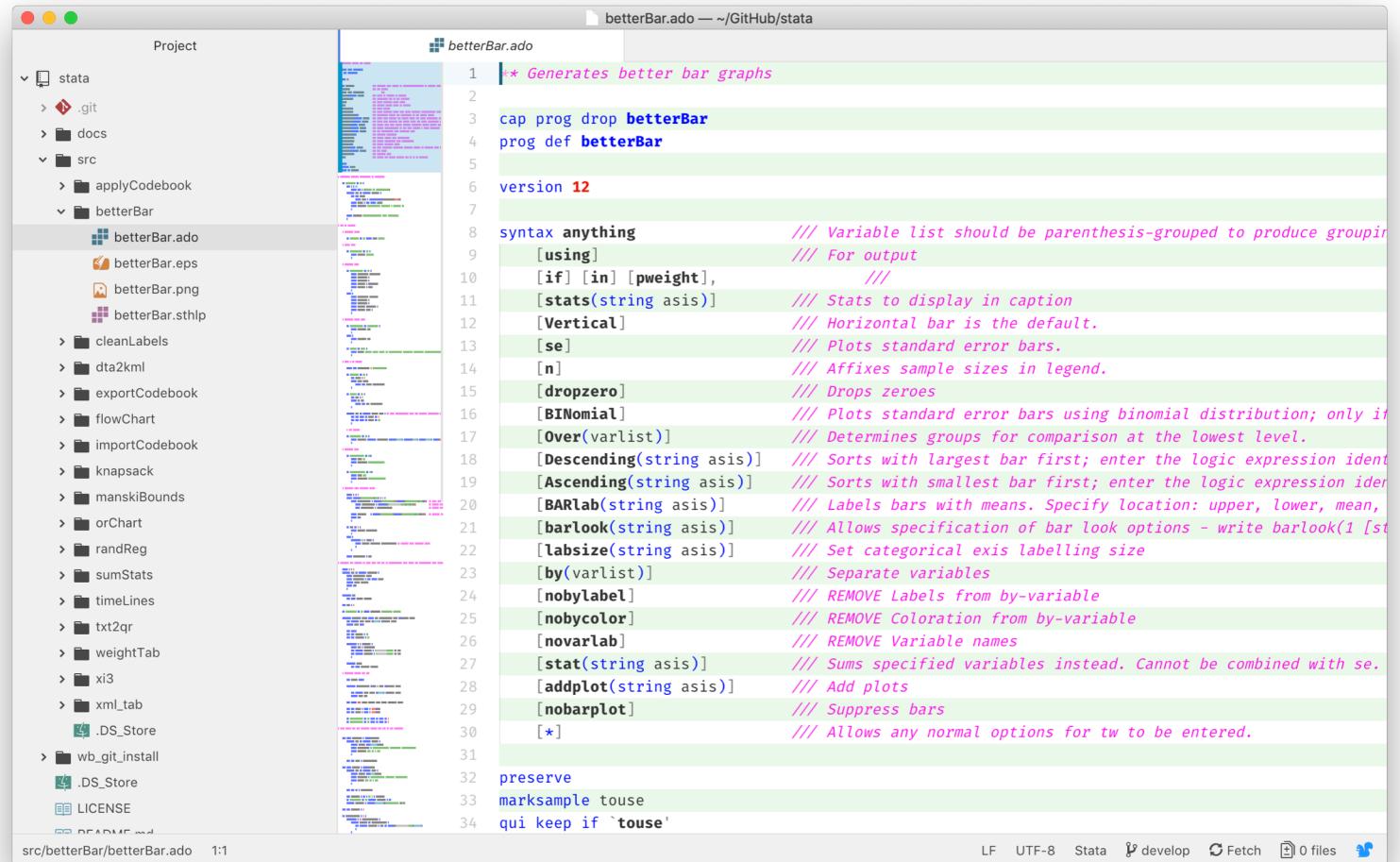
Optimize only when it works.

- First drafts are rarely perfect
- FINAL drafts *never* are
- So don't re-generate stuff by hand all the time
- Nothing needs to look nice until you are ready to go to press
- Raw outputs need not be pretty
- At press time you can clean everything up and delete cruft



Document design and purpose, not mechanics.

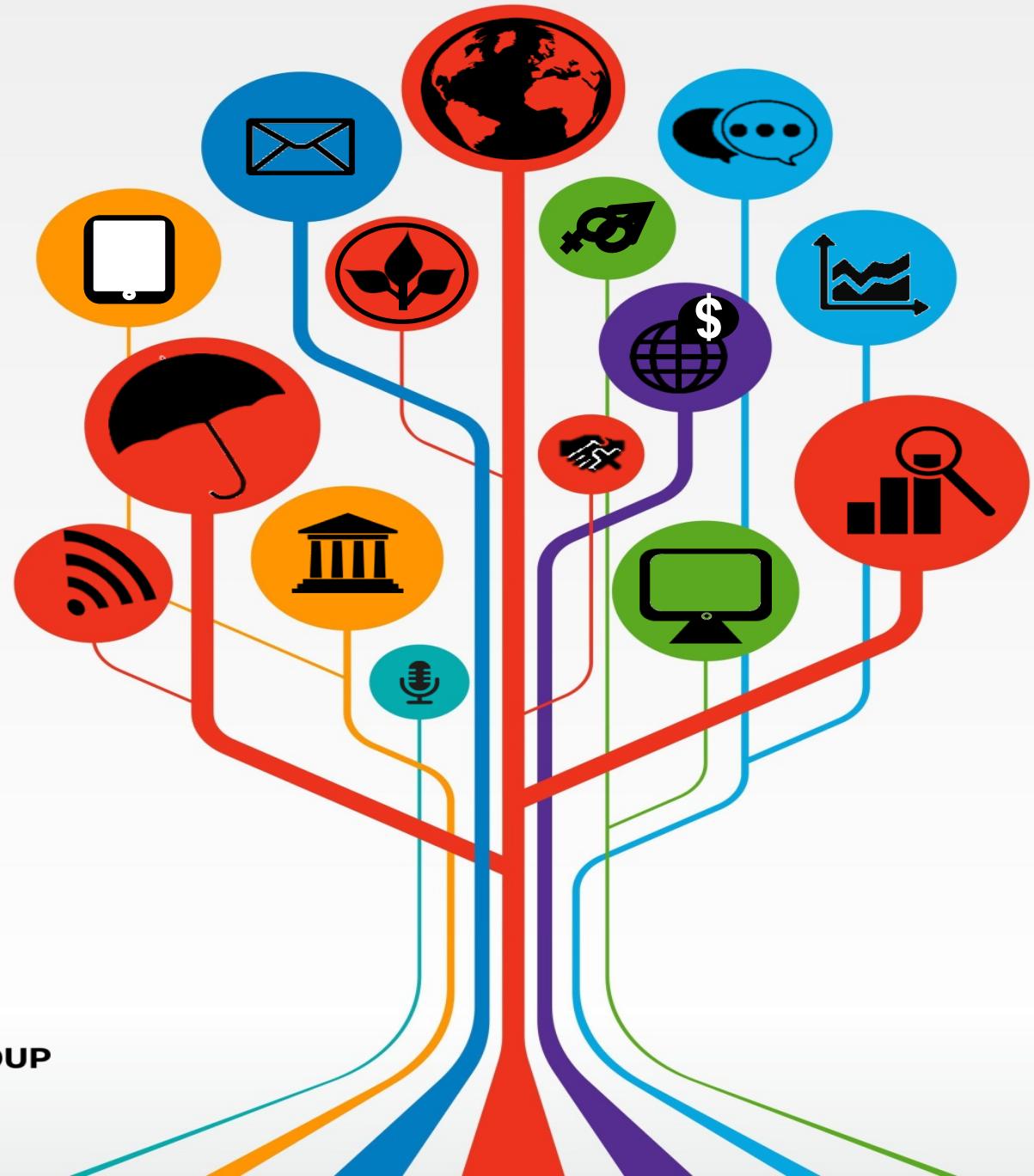
- Don't make us read the code!



```
Project
  stata
    > .git
    > docs
    > src
      > applyCodebook
      > betterBar
        betterBar.ado
        betterBar.eps
        betterBar.png
        betterBar.sthlp
      > cleanLabels
      > dta2kml
      > exportCodebook
      > flowChart
      > importCodebook
      > knapsack
      > manskiBounds
      > orChart
      > randReg
      > sumStats
      > timeLines
      > txt2qr
      > weightTab
      > xi3
      > xml_tab
      .DS_Store
    > wb_git_install
      .DS_Store
      LICENSE
      README.md

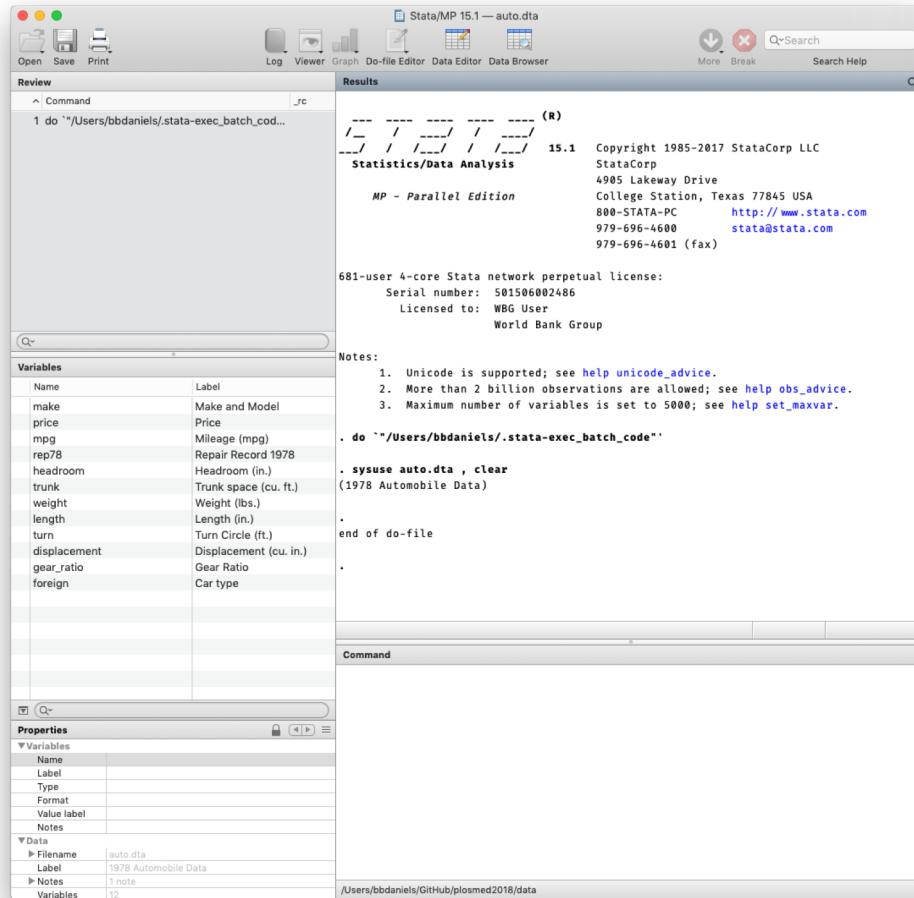
betterBar.ado — ~/GitHub/stata
1  /* Generates better bar graphs
2
3  cap prog drop betterBar
4  prog def betterBar
5
6  version 12
7
8  syntax anything           /// Variable list should be parenthesis-grouped to produce groupings
9  [using]                   /// For output
10 [if] [in] [pweight],       ///
11 [stats(string asis)]     /// Stats to display in caption
12 [Vertical]                /// Horizontal bar is the default.
13 [se]                      /// Plots standard error bars.
14 [n]                        /// Affixes sample sizes in legend.
15 [dropzero]                 /// Drops zeroes
16 [BINomial]                 /// Plots standard error bars using binomial distribution; only if
17 [Over(varlist)]            /// Determines groups for comparison at the lowest level.
18 [Descending(string asis)]  /// Sorts with largest bar first; enter the logic expression identifying
19 [Ascending(string asis)]   /// Sorts with smallest bar first; enter the logic expression identifying
20 [BARlab(string asis)]     /// Labels bars with means. Specify location: upper, lower, mean,
21 [barlook(string asis)]     /// Allows specification of bar look options - write barlook(1 [st
22 [labsize(string asis)]    /// Set categorical axis labelling size
23 [by(varlist)]              /// Separate variables
24 [nobylabel]                 /// REMOVE Labels from by-variable
25 [nobycolor]                 /// REMOVE Coloration from by-variable
26 [novarlab]                  /// REMOVE Variable names
27 [stat(string asis)]         /// Sums specified variables instead. Cannot be combined with se.
28 [addplot(string asis)]      /// Add plots
29 [nobarplot]                  /// Suppress bars
30 [*]                         /// Allows any normal options for tw to be entered.
31
32 preserve
33 marksample touse
34 qui keep if `touse'
```

Fundamentals of Stata



Scientific Computing with Stata

1. Systematic naming conventions
 2. Subsetting your data
 3. Handling repetitive tasks
 4. *[foreach]* and *[forvalues]*
 5. *[bys:]* and *[egen]*
 6. Programs
-
- Review: [Manage Successful Impact Evaluations Stata Labs](#)
 - Nick Cox. “[Speaking Stata: how to repeat yourself without going mad](#)”.



Naming in Stata: datasets, dofiles, variables

- Names should [*order* *, *seq*] sensibly, so same rules as before
- Raw data should match questionnaire (including order) – many people use varnames like s01_q01
- Labels should be English only so that other code does not break, and should be in “Proper Case”

Variables	
Name	Label
providerid	Provider ID
sp1_h_1	Duration of Cough
sp1_h_2	Sputum
sp1_h_3	Past TB
sp1_h_4	Family TB
sp1_h_5	Blood in Sputum
sp1_h_6	Cough Throughout Day
sp1_h_7	Fever
sp1_h_8	Fever Type
sp1_h_9	Family Symptoms
sp2_h_1	Duration of Cough
sp2_h_2	Sputum
sp2_h_3	Type of Doctor SP Saw
sp2_h_4	Medicine Taken
sp2_h_5	For How Long Medicines...
sp2_h_6	Family TB
sp2_h_7	Provider Saw X-Ray Film
sp2_h_8	Provider Read X-Ray Re...
sp2_h_9	Blood in Sputum
sp3_h_1	Duration of Cough

Constructed: more descriptive, still systematic

- Constructed data should stay sensible: crop_type crop_amount
- Note that this often means reversing English adjective order!

Utilities

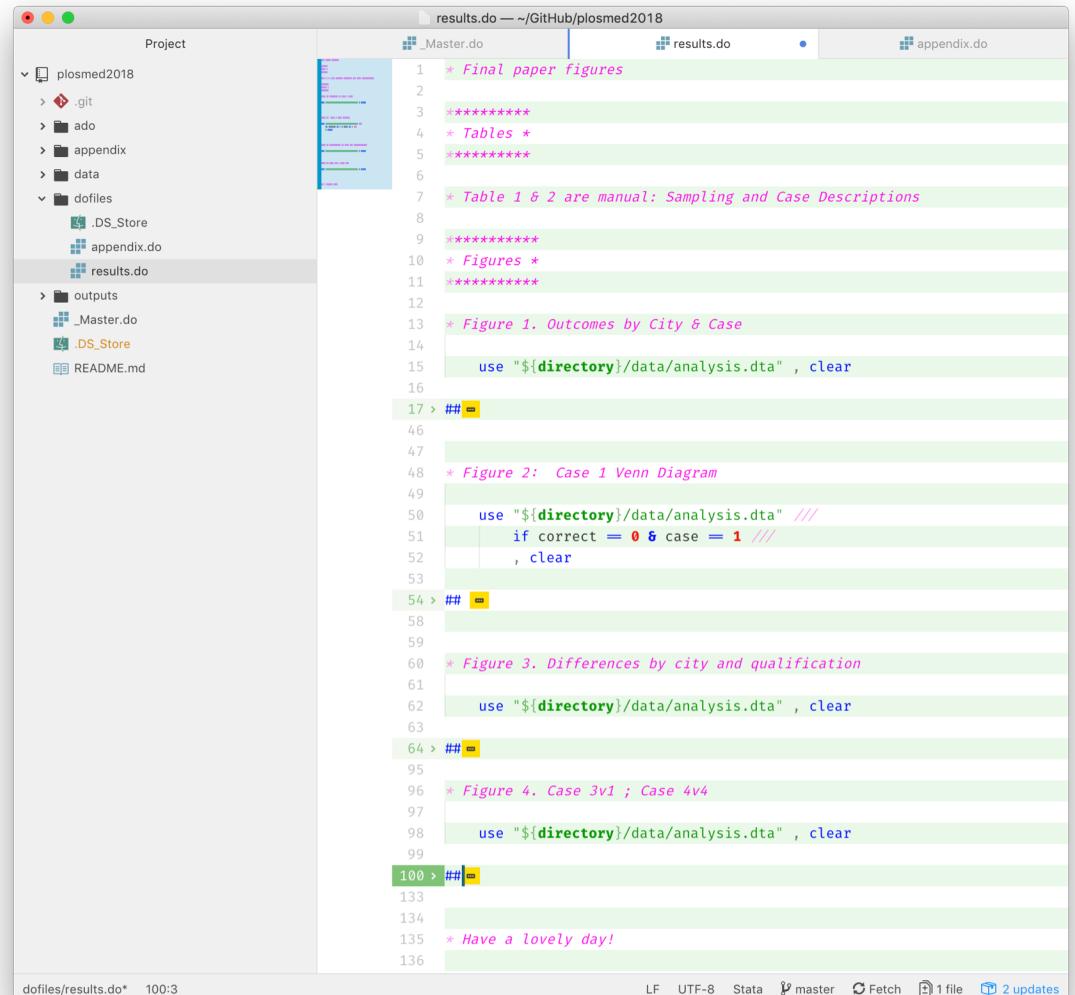
- [rename] command takes wildcards (acceptable in Construct dofiles)
 - [rename *_h_1 h01_*]
 - [rename *_h_* h0*[2]_*[1]]
- [lookfor] (try [return list] after)

Variables	
Name	Label
sp4_spur_1	SP brought a sputum report
sp_id	SP ID
t_12	Ask to Return
t_12a	Return if Symptoms Persist
t_12b	Return if Symptoms Worsen
t_12c	Return to Get Medicines
t_12d	Return for Test Results
t_12e	Return After So Many Days
treat_cxr	Chest X-Ray
treat_refer	Referred Case
type	Sample Type
type_formal	MBBS Provider
visit	Visit
weight	City-Sample Weight
weight_city	Within-City-Case Weight

Subset your data; [use] often

Two major methods:

1. Create alternate datasets
 - “analysis_main.dta”
 - “analysis_main-long.dta”
 - “conf-neudc_prices-wide.dta”
 - “supplement_female-only.dta”
2. Subset with [use]
 - [use *id* correct case
using "\${dtFin}/analysis.dta"
if case == 1, clear]
 - Takes varlist and if/in conditions



The screenshot shows a Stata DO file editor window titled "results.do — ~/GitHub/plosmed2018". The left pane displays a project structure for a GitHub repository named "plosmed2018". The right pane shows the content of the "results.do" DO file, which contains Stata code for generating figures. The code includes sections for "Final paper figures", "Tables", "Figures", and "Differences by city and qualification". It uses the "use" command to subset the "analysis.dta" dataset based on case values (0 or 1) and includes "clear" commands to manage memory.

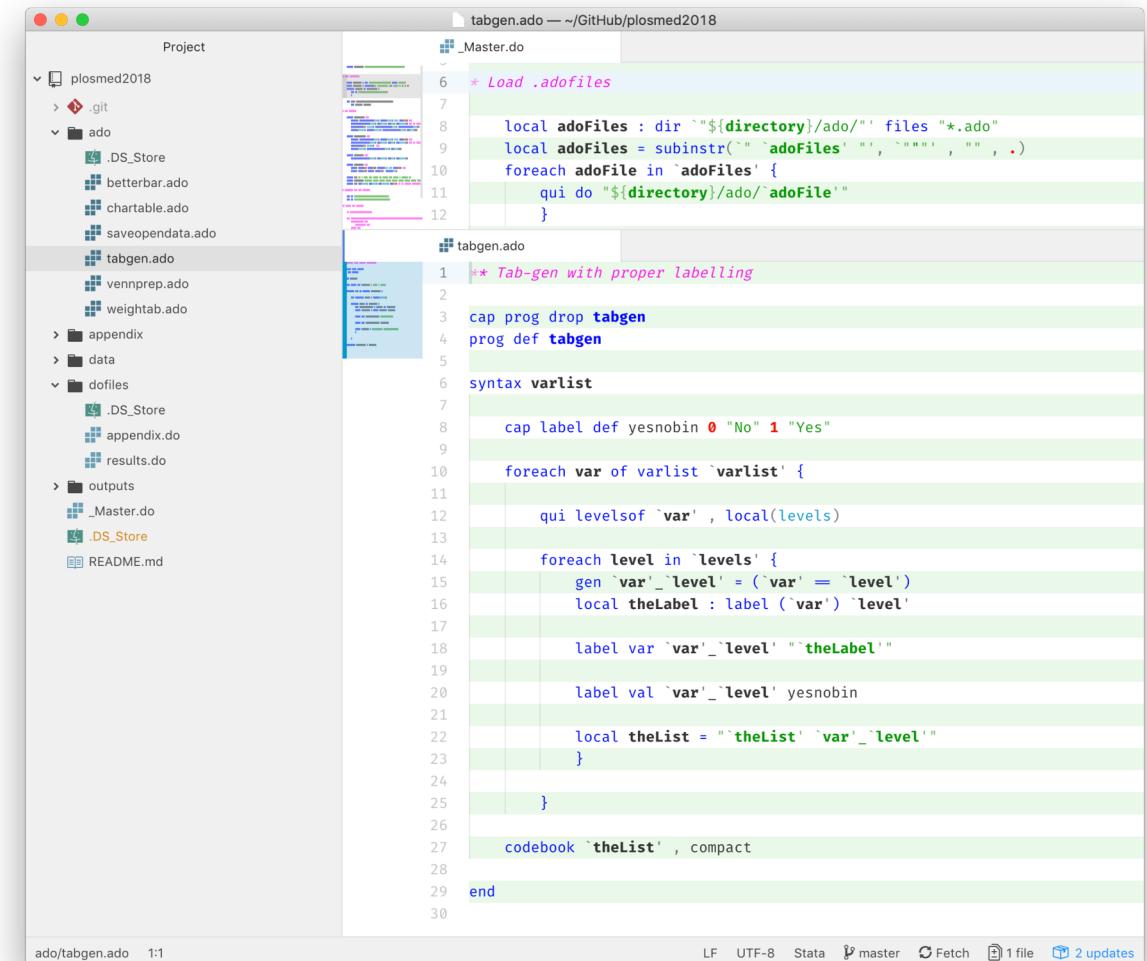
```
* Final paper figures
*****
* Tables *
*****
* Table 1 & 2 are manual: Sampling and Case Descriptions
*****
* Figures *
*****
* Figure 1. Outcomes by City & Case
15    use "${directory}/data/analysis.dta", clear
17 > ##= 46
47
48 * Figure 2: Case 1 Venn Diagram
49
50 use "${directory}/data/analysis.dta" ///
51   if correct == 0 & case == 1 ///
52   , clear
53
54 > ##= 58
58
59
60 * Figure 3. Differences by city and qualification
61
62 use "${directory}/data/analysis.dta", clear
63
64 > ##= 64
64
65 * Figure 4. Case 3v1 ; Case 4v4
66
67 use "${directory}/data/analysis.dta", clear
68
69
70 > ##= 100
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135 * Have a lovely day!
136
```

Repetitive tasks

Fool me thrice / rule of three:

- The first time you do something, do it carefully and correctly
- The second time you do something, think about how you would generalize it
- The third time, implement the generalization

The fifth through 100th times, pat yourself on the back



The screenshot shows a Stata IDE interface. On the left, a 'Project' tree view displays a directory structure for a project named 'plosmed2018'. The 'ado' folder contains several ado files: betterbar.ado, chartable.ado, saveopendata.ado, tabgen.ado, vennprep.ado, and weightab.ado. Other folders include '.DS_Store', 'appendix', 'data', 'dofiles', 'outputs', '_Master.do', and 'README.md'. The right side of the interface shows two code editors. The top editor is titled 'tabgen.ado' and contains Stata do-file code for generating tables. The bottom editor is titled 'Master.do' and also contains Stata do-file code, including a section for defining a 'tabgen' command. The status bar at the bottom indicates the file is 'ado/tabgen.ado' at line 1:1, with encoding 'UTF-8', Stata version 'master', and 1 file, 2 updates.

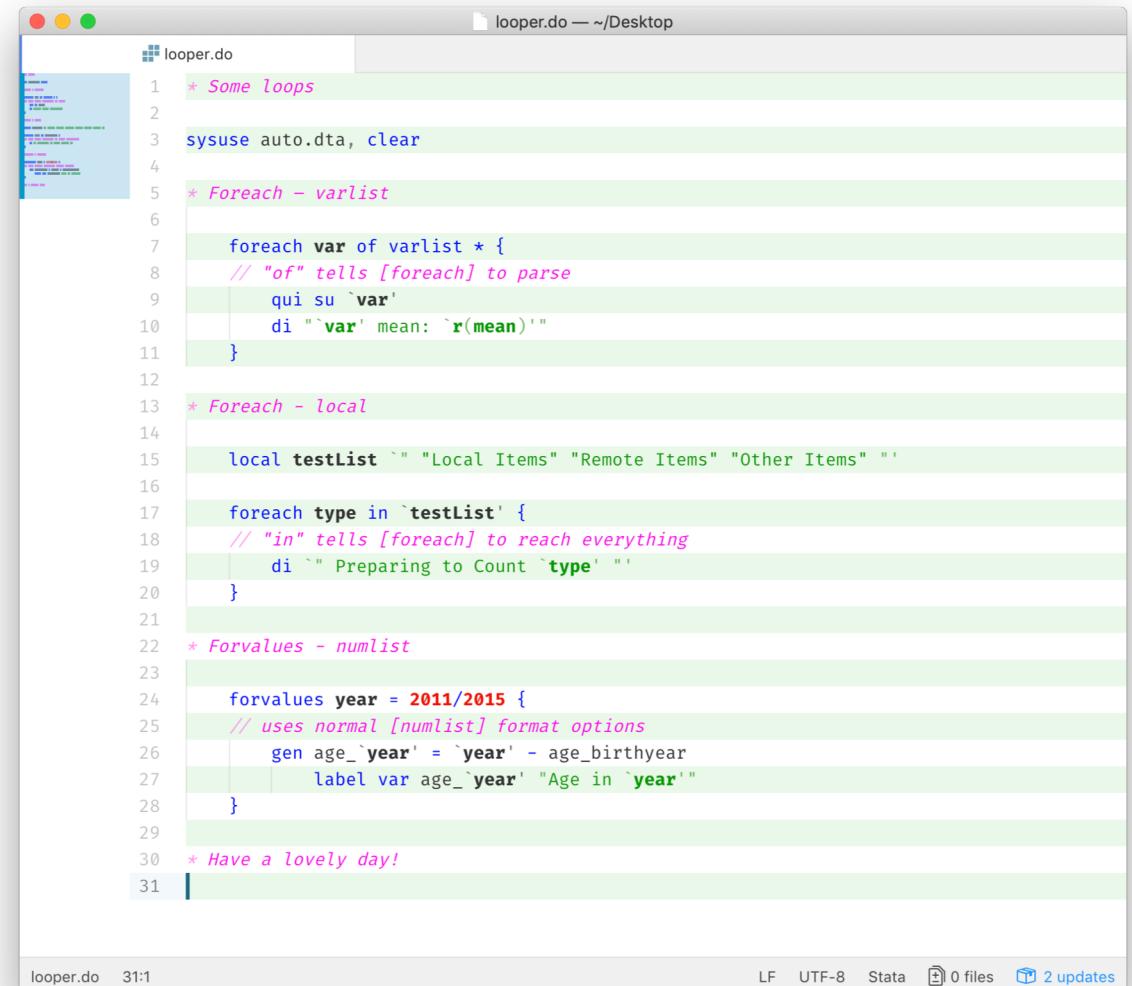
```
tabgen.ado — ~/GitHub/plosmed2018

Master.do
6  * Load .ado files
7
8  local adoFiles : dir "${directory}/ado/" files "*.ado"
9  local adoFiles = subinstr(`adoFiles', " ", " ", , .)
10 foreach adoFile in `adoFiles' {
11     qui do "${directory}/ado/`adoFile'"
12 }

tabgen.ado
1  /* Tab-gen with proper labelling
2
3  cap prog drop tabgen
4  prog def tabgen
5
6  syntax varlist
7
8  cap label def yesnobin 0 "No" 1 "Yes"
9
10 foreach var of varlist `varlist' {
11
12     qui levels `var', local(levels)
13
14     foreach level in `levels' {
15         gen `var'_`level' = (`var' == `level')
16         local theLabel : label(`var') `level'
17
18         label var `var'_`level' ``theLabel''
19
20         label val `var'_`level' yesnobin
21
22         local theList = ``theList' `var'_`level''
23     }
24
25 }
26
27 codebook `theList', compact
28
29 end
```

[foreach] and [forvalues] loop commands

- Setting up loops: use a meaningful `index' local name
- Foreach replace `index' with the list items, in order
 - Variable names
 - Elements of a local
 - Whatever you want
- Be careful about hierarchy/indentation

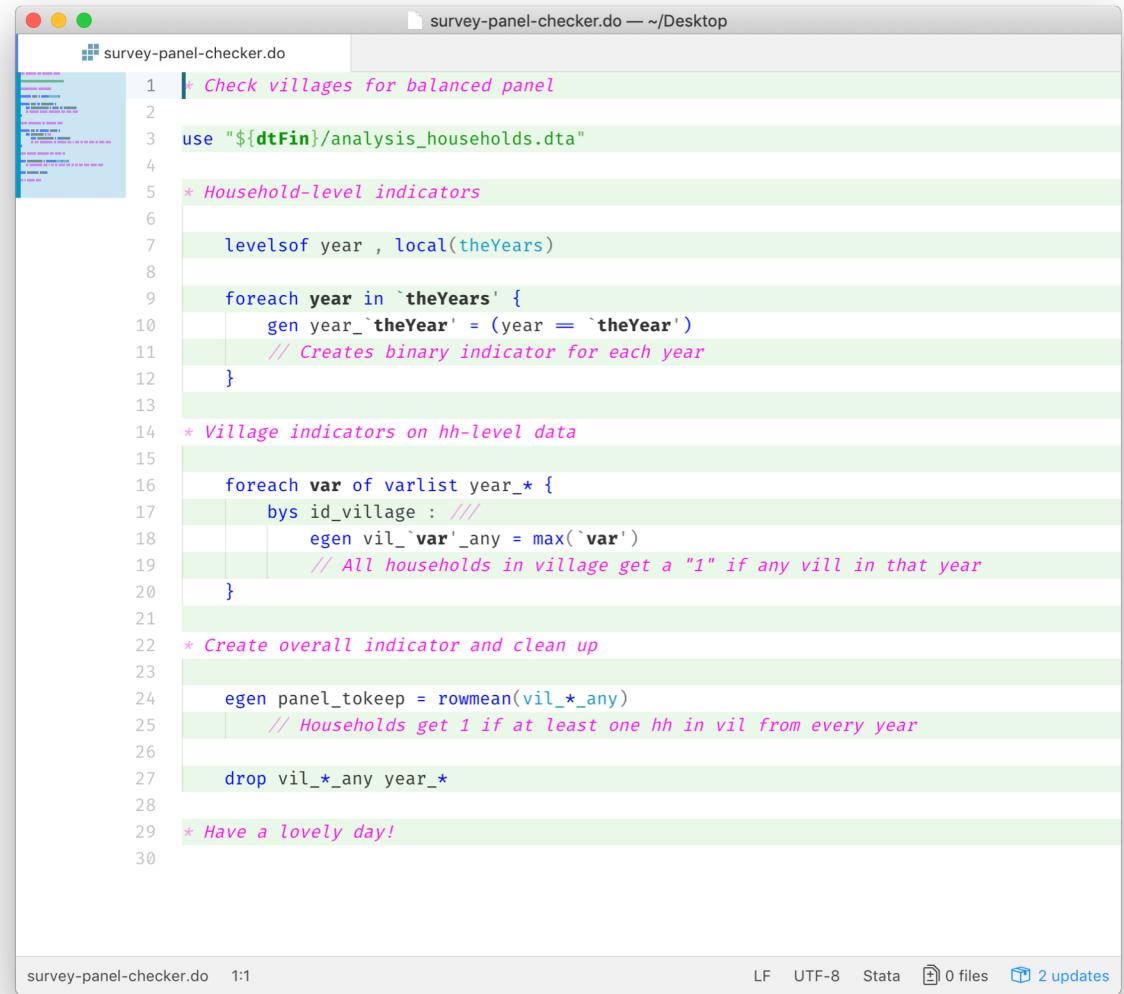


The screenshot shows a Stata do-file named "looper.do" with the following content:

```
1 * Some loops
2
3 sysuse auto.dta, clear
4
5 * Foreach - varlist
6
7 foreach var of varlist * {
8     // "of" tells [foreach] to parse
9     qui su `var'
10    di "`var' mean: `r(mean)'"
11 }
12
13 * Foreach - local
14
15 local testList `"" "Local Items" "Remote Items" "Other Items" ""
16
17 foreach type in `testList' {
18     // "in" tells [foreach] to reach everything
19     di `"' Preparing to Count `type' ''"
20 }
21
22 * Forvalues - numlist
23
24 forvalues year = 2011/2015 {
25     // uses normal [numlist] format options
26     gen age_`year' = `year' - age_birthday
27     label var age_`year' "Age in `year'"
28 }
29
30 * Have a lovely day!
31
```

[*bys:*] and [*egen*] use groups of observations

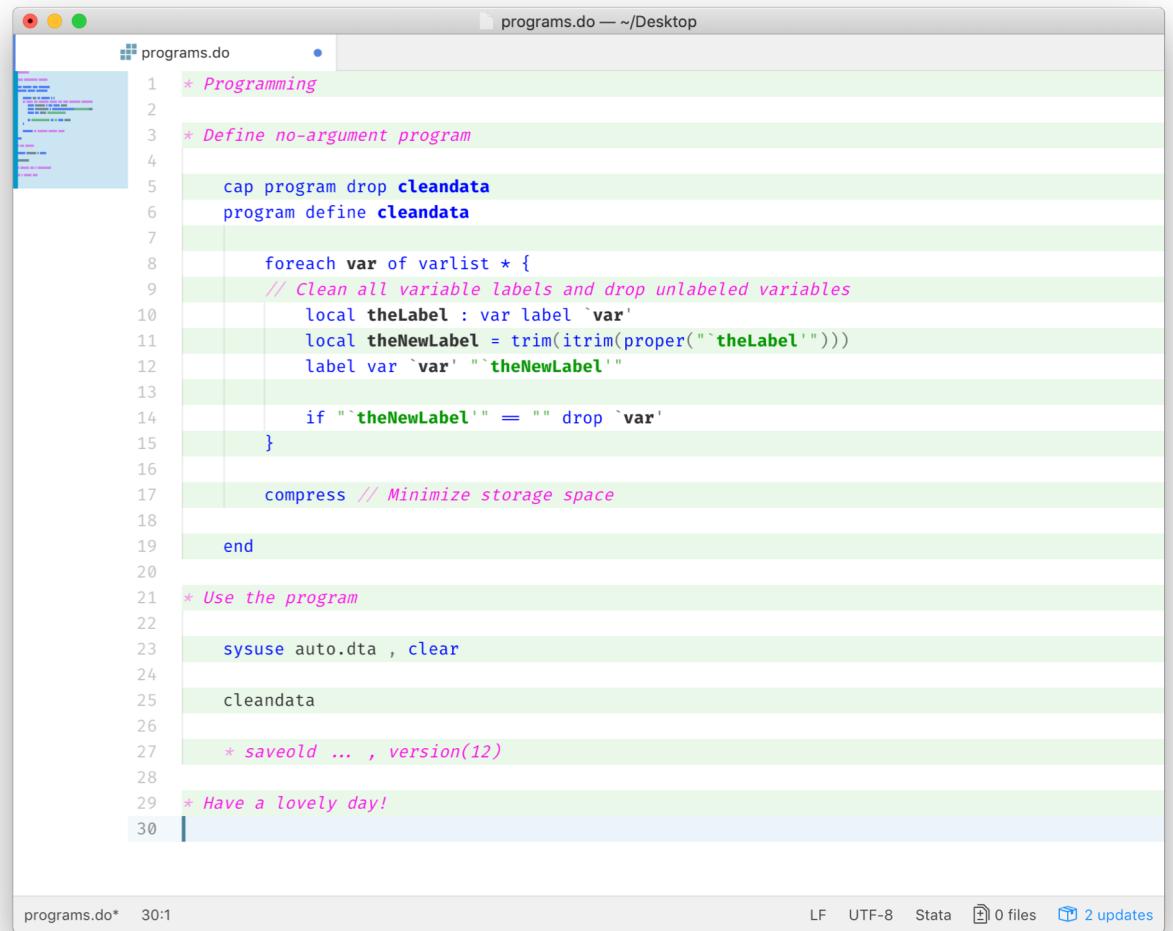
- You have long-format panel data of households covering several years.
- Which *villages* have at least one household in all survey years?
- [*collapse*] and [*merge*] would work with just two years, but an arbitrary number of years is harder...



```
* Check villages for balanced panel
use "${dtFin}/analysis_households.dta"
* Household-level indicators
levelsof year , local(theYears)
foreach year in `theYears' {
    gen year_`theYear' = (year == `theYear')
    // Creates binary indicator for each year
}
* Village indicators on hh-level data
foreach var of varlist year_* {
    bys id_village : ///
        egen vil_`var'_any = max(`var')
    // All households in village get a "1" if any vill in that year
}
* Create overall indicator and clean up
egen panel_tokeep = rowmean(vil_*_any)
// Households get 1 if at least one hh in vil from every year
drop vil_*_any year_*
* Have a lovely day!
```

Generalizing with [program]

- *[program]* takes a loop-like code block and generalizes it into a command
- Allows you to call the same code block on demand
- Better than loops in that it allows multiple index-like entries: there are called “arguments” and “options”
- Can be further generalized into ado-files: more on that in future



The screenshot shows a Stata do-file named "programs.do" open in a text editor. The code defines a program called "cleandata" that iterates over variables to clean labels and drop unlabeled variables, then compresses the data. It also demonstrates how to use the program by running "sysuse auto.dta, clear", "cleandata", and saving the results.

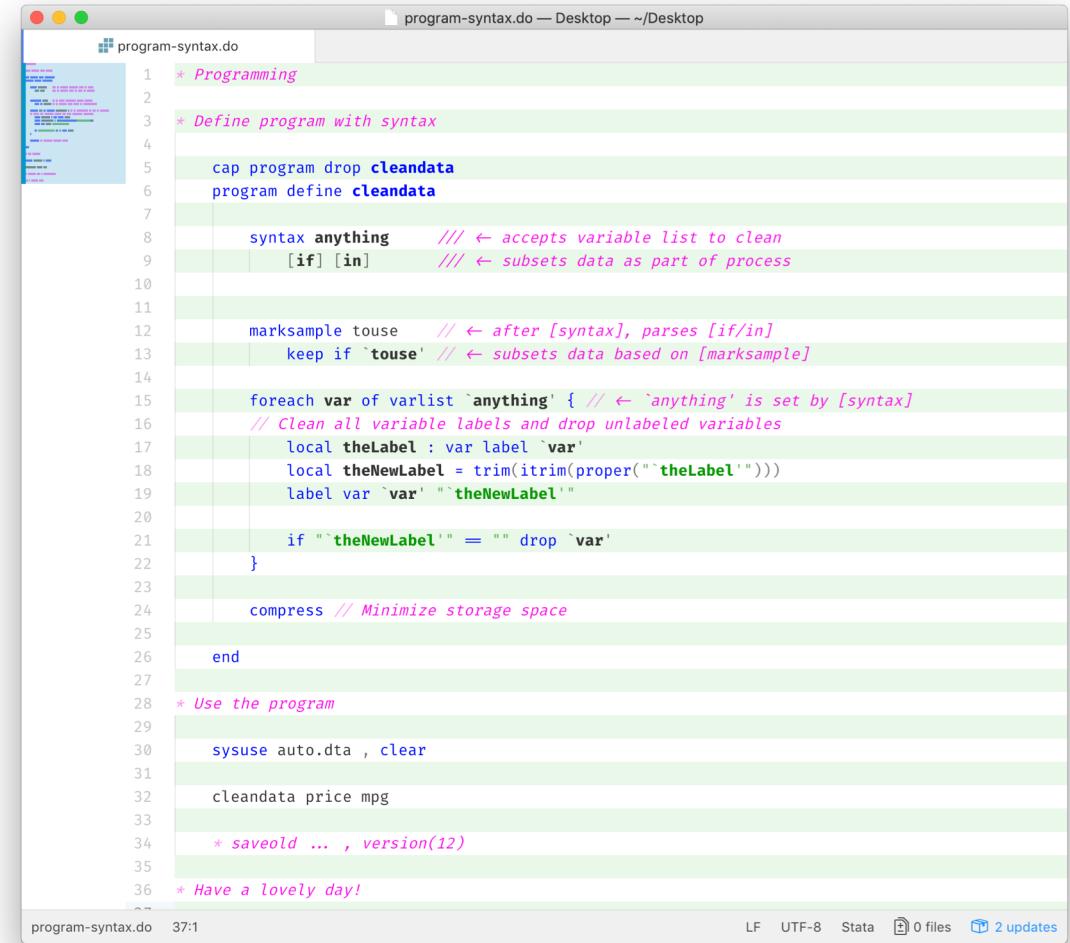
```
* Programming
* Define no-argument program
cap program drop cleandata
program define cleandata
    foreach var of varlist * {
        // Clean all variable labels and drop unlabeled variables
        local theLabel : var label `var'
        local theNewLabel = trim(trim(proper(`theLabel'))))
        label var `var' `theNewLabel'

        if "`theNewLabel'" == "" drop `var'
    }
    compress // Minimize storage space
end

* Use the program
sysuse auto.dta, clear
cleandata
* saveold ... , version(12)
* Have a lovely day!
```

Generalizing with [program]

- *[program]* takes a loop-like code block and generalizes it into a command
- Allows you to call the same code block on demand
- Better than loops in that it allows multiple index-like entries: there are called “arguments” and “options”
- Can be further generalized into ado-files: more on that in future



```
* Programming
*
* Define program with syntax
*
cap program drop cleandata
program define cleandata
    syntax anything      /// ← accepts variable list to clean
                        [if] [in]   /// ← subsets data as part of process
    marksample touse    // ← after [syntax], parses [if/in]
                        keep if `touse' // ← subsets data based on [marksample]
    foreach var of varlist `anything' { // ← `anything' is set by [syntax]
        // Clean all variable labels and drop unlabeled variables
        local theLabel : var label `var'
        local theNewLabel = trim(trim(proper("`theLabel'"))
        label var `var' "`theNewLabel'"
        if "`theNewLabel'" == "" drop `var'
    }
    compress // Minimize storage space
end
*
* Use the program
*
sysuse auto.dta , clear
cleandata price mpg
* saveold ... , version(12)
* Have a lovely day!
```

Thank you!

worldbank.github.io/dimeanalytics

