

Stata Coding for Reproducible Research

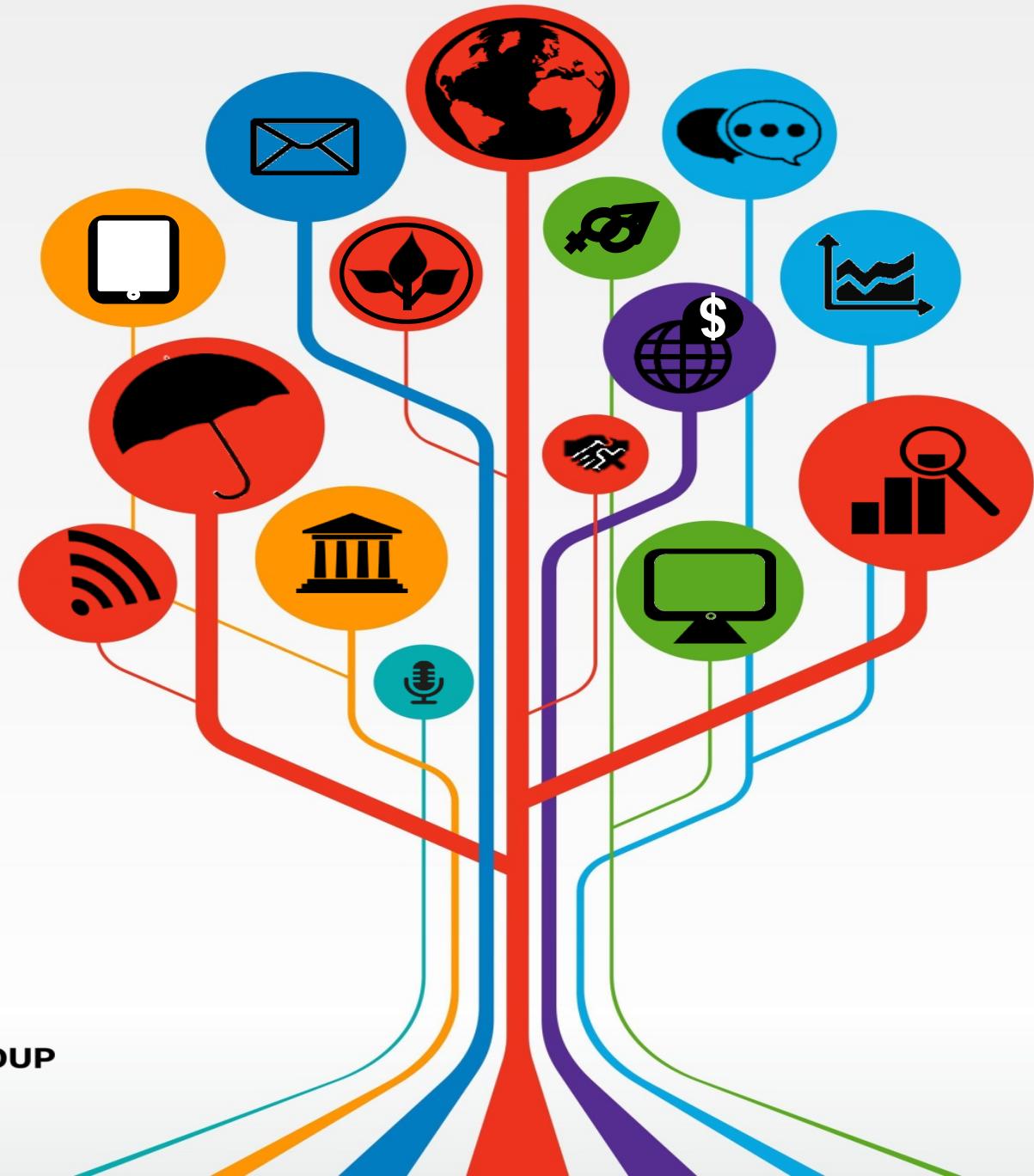
Research Assistant Onboarding

Prepared by DIME Analytics

dimeanalytics@worldbank.org

Presented by Benjamin Daniels

bdaniels@worldbank.org



Introduction: Stata Coding

Stata coding is part of a reproducible research workflow.

- It should be easy to read and re-adapt
- This means in terms of *structure, syntax* and *style*
- Code should be modularized as much as possible
- Anything that might be used again can be saved as an “adofile”

Production

Collection

Analysis

Publication



Survey
Forms

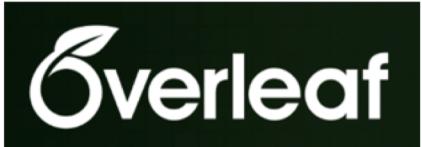
Raw
Data



Cleaned
Data &
Outputs

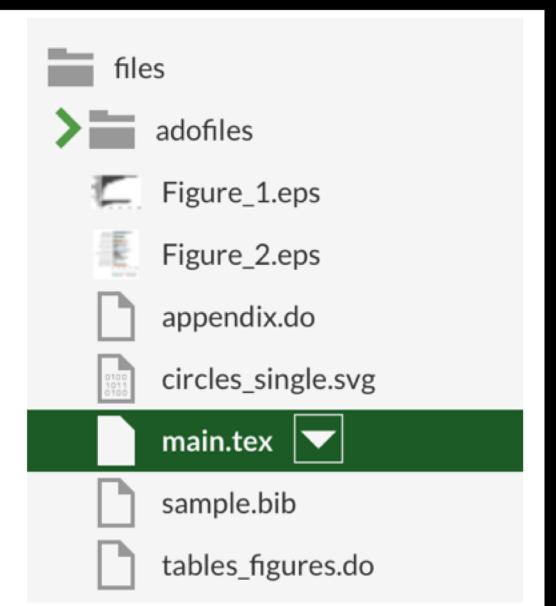
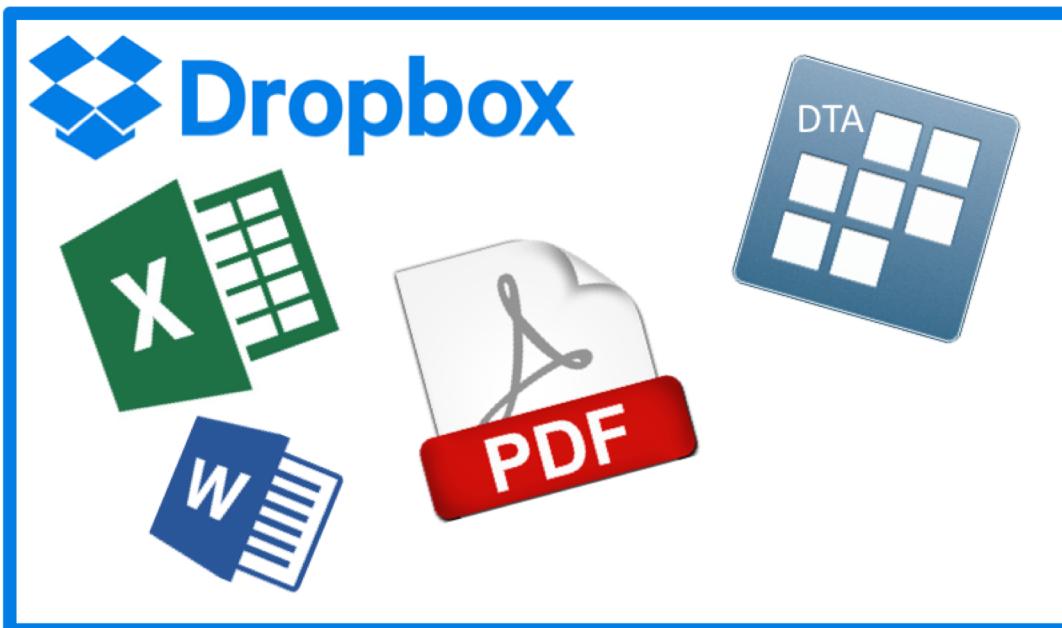
Raw
Data

Dofiles

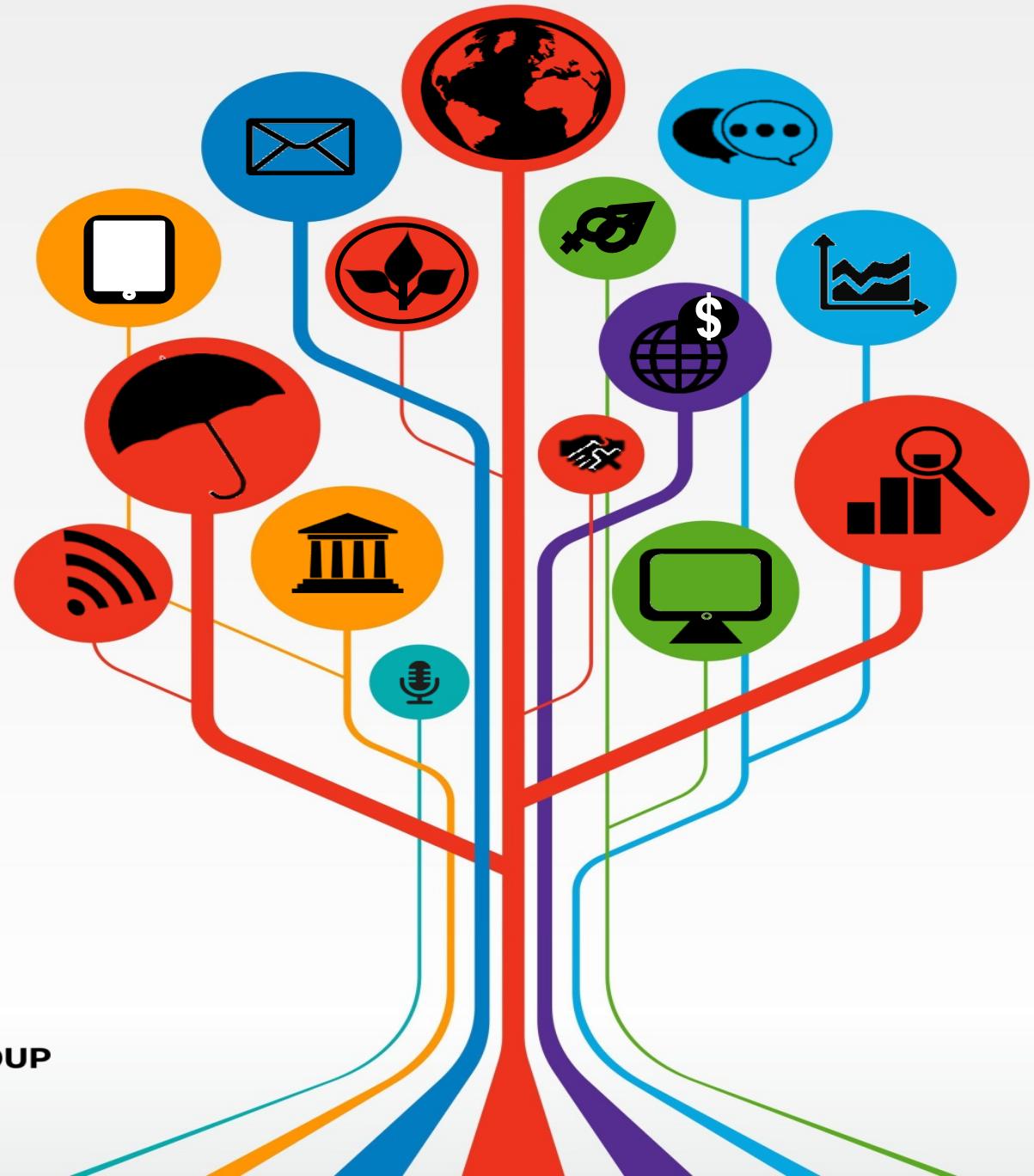


LaTeX
Files and
Bibliography

PDF



Stata Coding: Structure



Production

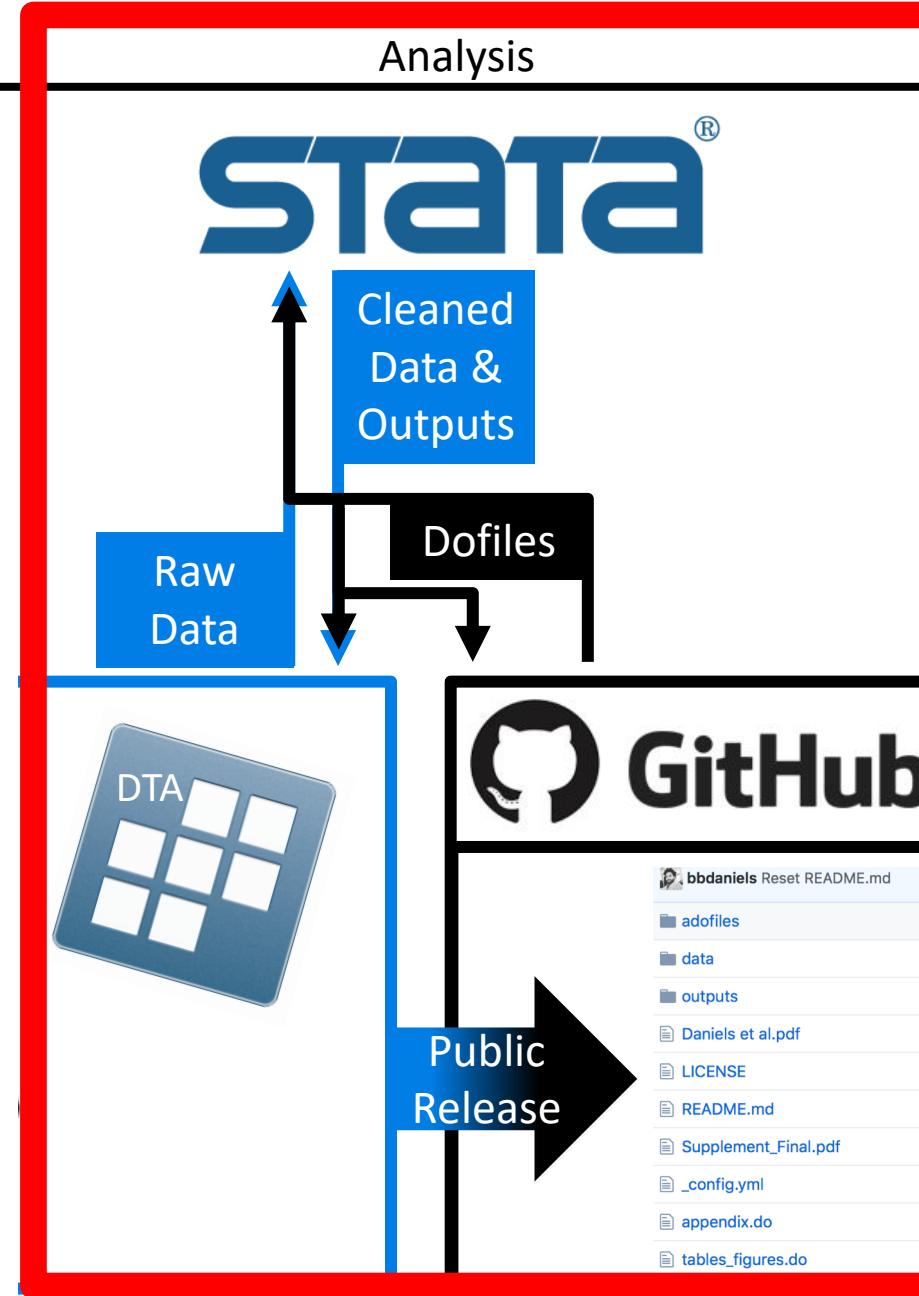
Collection

Analysis

Publication

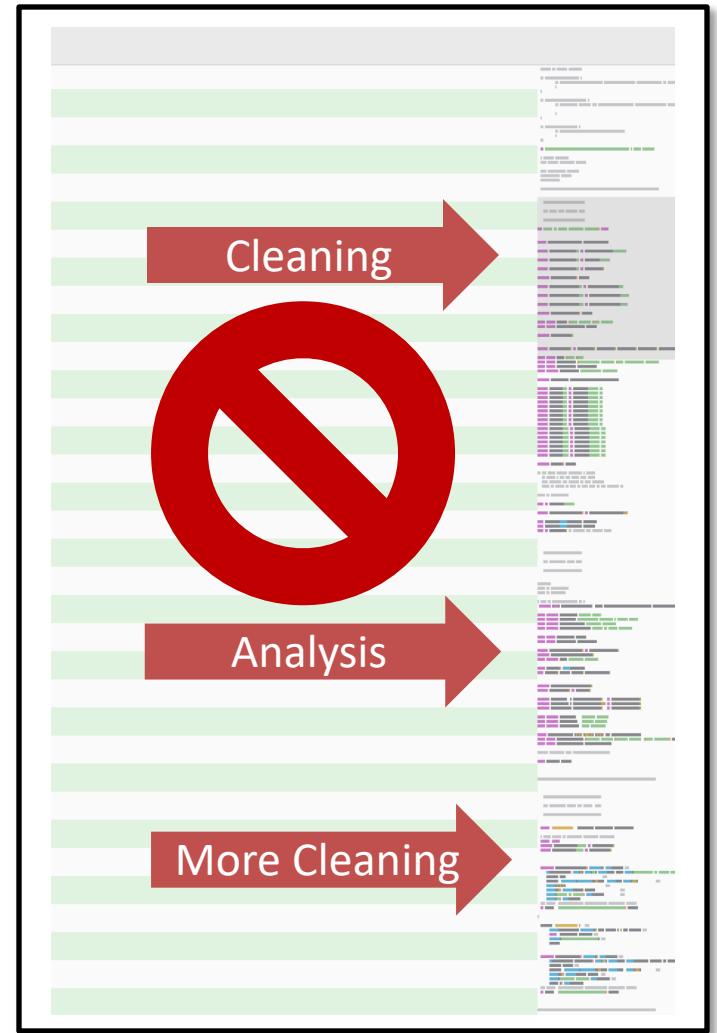
**Stata structure is the environment
your code lives in.**

- Publishing a paper is not enough!
- Code and data to reproduce results is often required by Open Access agreements or journals themselves.
- And even if it isn't, others may want or need to use or reuse your code in the future, so it is good academic citizenship.



Modular organization is constant preparation

- It is much easier to maintain files **modularly** from the outset than to clean up “everything dofiles”.
- Cleaning is **separated** from analysis so that each analysis step begins with [use].
- The final product then only consists of keeping the analysis files that are used in the paper and **archiving** the rest.



What does this look like in practice?

The diagram illustrates the relationship between a project's file structure and its corresponding Stata Do-file script.

Left Side: Project File Structure

- Project name: Water-When-It-Counts
- Root folder contains:
 - _config.yml
 - Data
 - Analysis
 - Master data sets
 - Do-files
 - Analysis
 - Attrition test.do
 - Balance tests.do
 - Descriptive statistics.do
 - Numbers in main text.do
 - Plots.do
 - Regressions.do
 - Master.do
 - Output
 - Raw files
 - Balance tests
 - Descriptive statistics
 - Plots
 - Regression results
 - README.md

Right Side: Do-file Script

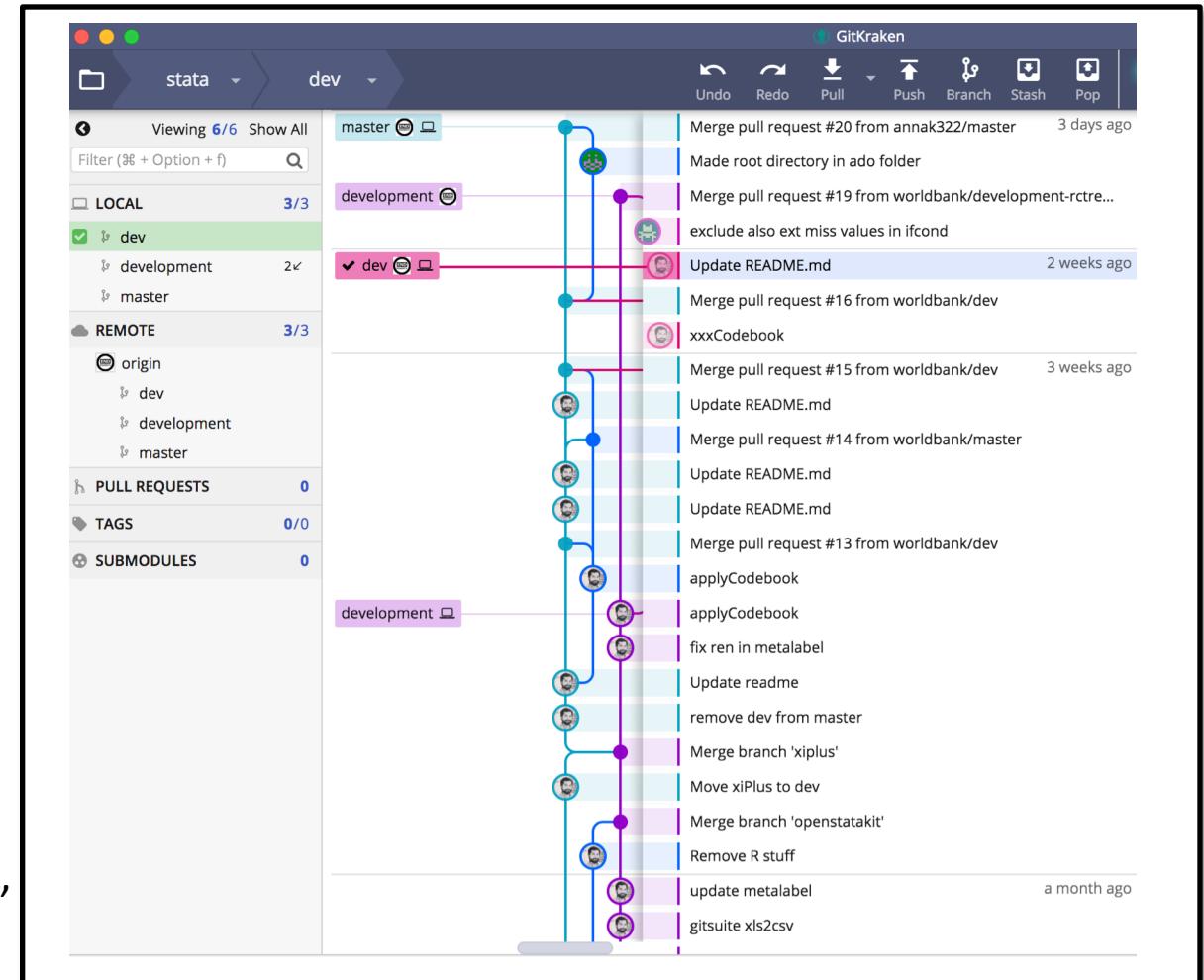
```
Master.do
1  ****
2 *      MOZ PROIRRI          *
3 *      REPLICATION MASTER DO-FILE   *
4 *          2018           *
5 ****
6
7 ****
8 *      SELECT PARTS TO RUN          *
9 ****/
10 ****
11 * select which parts of this do-file to run
12 local packages    1 // Install packages -- only needs to be ran once in each computer
13 local attrition    1 // Run attrition test
14 local balance_tables 1 // Create balance tables
15 local descriptives   1 // Create descriptive statistics graphs
16 local graphs        1 // Create graphs
17 local regressions   1 // Run regressions and export results
18 ****
19
20
21 ****
22
23 *      PART 1: Set standard settings and install packages      *
24 ****/
25 ****
```

Modular data analysis with Git(Hub)

Git is a great workflow structure for this because components can be *added*, *deleted*, and *recovered* without information loss, even if you are working alone.

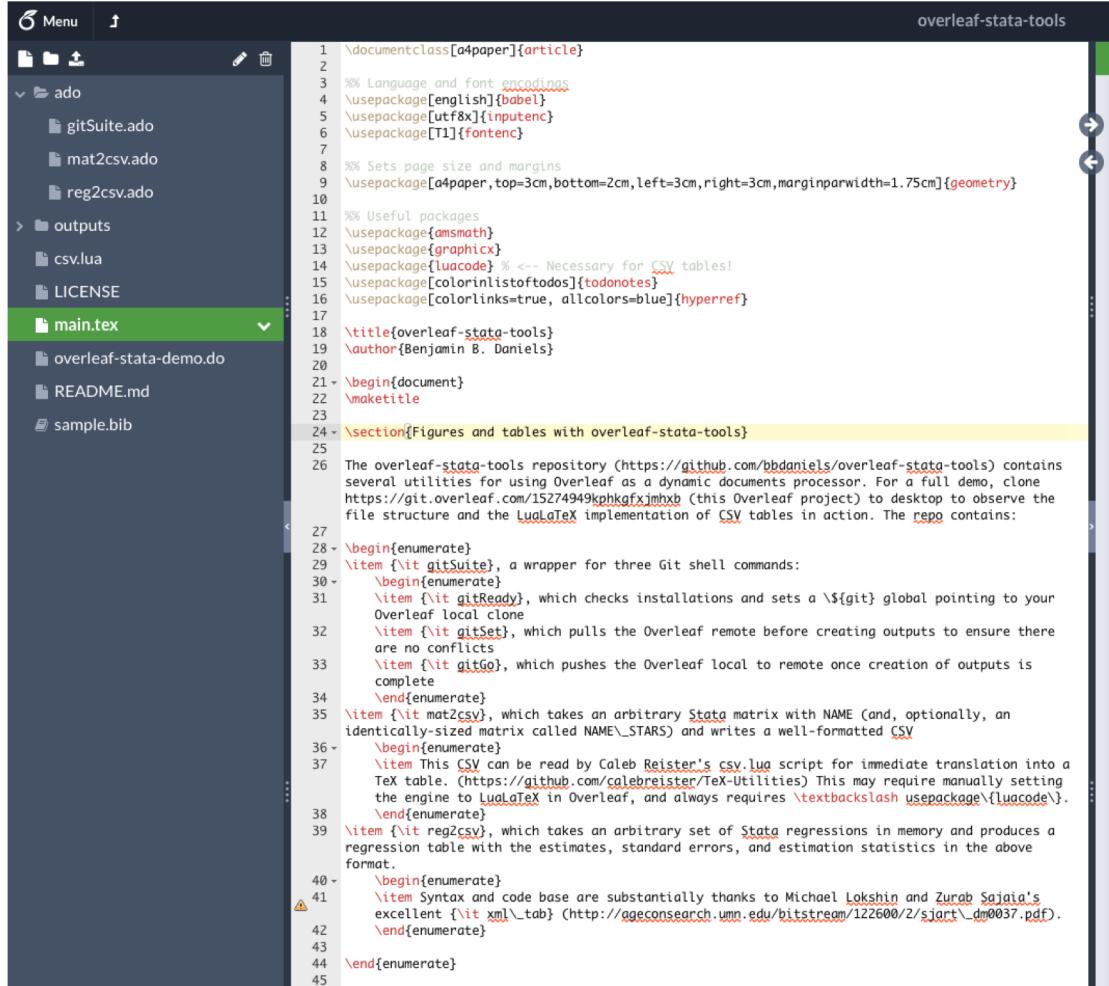
Every version of everything is available **everywhere**.

- **Version Control** – Analysts can always find “that code that did that one thing”, and the current master version will have various experimental “branches” until they are confirmed functional and merged in.
- **Efficiency** – This allows simultaneous distributed editing of dofiles, text files, bibliographies, etc., but is not appropriate for Office/.dta/.pdf files.
- **Privacy** – Must use paid account (such as github.com/worldbank) to have “private” or “secret” folders.



Writing: LaTeX Documents (Overleaf + Git)

Overleaf is
kept up to
date via link
to GitHub



The screenshot shows the Overleaf web interface. On the left, the file tree for the project "overleaf-stata-tools" is visible, containing files like gitSuite.ado, mat2csv.ado, reg2csv.ado, outputs, csv.lua, LICENSE, main.tex (which is selected), README.md, and sample.bib. The main area displays the LaTeX code for "main.tex". A large bracket on the left side groups the "Overleaf is kept up to date via link to GitHub" text with the Overleaf interface.

```
\documentclass[a4paper]{article}

% Language and font encodings
\usepackage[english]{babel}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}

% Sets page size and margins
\usepackage[a4paper,top=3cm,bottom=2cm,left=3cm,right=3cm,marginparwidth=1.75cm]{geometry}

% Useful packages
\usepackage{amsmath}
\usepackage{graphicx}
\usepackage{luacode} % <- Necessary for CSV tables!
\usepackage{colorinlistoftodos}{todonotes}
\usepackage[colorlinks=true, allcolors=blue]{hyperref}

\title{overleaf-stata-tools}
\author{Benjamin B. Daniels}

\begin{document}
\maketitle

\section{Figures and tables with overleaf-stata-tools}

The overleaf-stata-tools repository (https://github.com/bbdaniels/overleaf-stata-tools) contains several utilities for using Overleaf as a dynamic documents processor. For a full demo, clone https://git.overleaf.com/15274949kphkgfxjmhxb (this Overleaf project) to desktop to observe the file structure and the LuaLaTeX implementation of CSV tables in action. The repo contains:

\begin{enumerate}
\item \texttt{gitSuite}, a wrapper for three Git shell commands:
    \begin{itemize}
        \item \texttt{gitReady}, which checks installations and sets a \$git global pointing to your Overleaf local clone
        \item \texttt{gitSet}, which pulls the Overleaf remote before creating outputs to ensure there are no conflicts
        \item \texttt{gitGo}, which pushes the Overleaf local to remote once creation of outputs is complete
    \end{itemize}
\item \texttt{mat2csv}, which takes an arbitrary Stata matrix with NAME (and, optionally, an identically-sized matrix called NAME\_STARS) and writes a well-formatted CSV
    \begin{itemize}
        \item This CSV can be read by Caleb Reister's csv.lua script for immediate translation into a TeX table. (https://github.com/calebreister/TeX-Utilities) This may require manually setting the engine to LuaLaTeX in Overleaf, and always requires \textbackslash usepackage\{luacode\}.
    \end{itemize}
\item \texttt{reg2csv}, which takes an arbitrary set of Stata regressions in memory and produces a regression table with the estimates, standard errors, and estimation statistics in the above format.
    \begin{itemize}
        \item Syntax and code base are substantially thanks to Michael Lokshin and Zurab Sajaia's excellent \texttt{xml\_tab} (http://ageconsearch.umn.edu/bitstream/122600/2/sjart\_dm0037.pdf).
    \end{itemize}
\end{enumerate}
```

Recompile

overleaf-stata-tools

overleaf-stata-tools

Benjamin B. Daniels

May 7, 2018

1 Figures and tables with overleaf-stata-tools

The overleaf-stata-tools repository (<https://github.com/bbdaniels/overleaf-stata-tools>) contains several utilities for using Overleaf as a dynamic documents processor. For a full demo, clone <https://git.overleaf.com/15274949kphkgfxjmhxb> (this Overleaf project) to desktop to observe the file structure and the LuaLaTeX implementation of CSV tables in action. The repo contains:

1. *gitSuite*, a wrapper for three Git shell commands:
 - (a) *gitReady*, which checks installations and sets a \$git global pointing to your Overleaf local clone
 - (b) *gitSet*, which pulls the Overleaf remote before creating outputs to ensure there are no conflicts
 - (c) *gitGo*, which pushes the Overleaf local to remote once creation of outputs is complete
2. *mat2csv*, which takes an arbitrary Stata matrix with NAME (and, optionally, an identically-sized matrix called NAME_STARS) and writes a well-formatted CSV
 - (a) This CSV can be read by Caleb Reister's *csv.lua* script for immediate translation into a TeX table. (<https://github.com/calebreister/TeX-Utilities>) This may require manually setting the engine to LuaLaTeX in Overleaf, and always requires \usepackage\{luacode\}.
3. *reg2csv*, which takes an arbitrary set of Stata regressions in memory and produces a regression table with the estimates, standard errors, and estimation statistics in the above format.
 - (a) Syntax and code base are substantially thanks to Michael Lokshin and Zurab Sajaia's excellent *xml_tab* (http://ageconsearch.umn.edu/bitstream/122600/2/sjart_dm0037.pdf).

Publication: Public Release of Data + Code

Writing code on Git with collaboration public release in mind prepares the project for preservation and open access

- The version history answers seminar questions that start with “what happened when you tried....?”
- The final repository has reusable code that other projects will benefit from
- Picking up the same project to extend or replicate analyses later is a one-click process

<https://worldbank.github.io/Water-When-It-Counts/>

Water-When-It-Counts

Replication files for Water When It Counts: Reducing Scarcity through Irrigation Monitoring in Central Mozambique by Paul Christian, Florence Kondylis, Valerie Mueller, Astrid Zwager and Tobias Siegfried

[View the Project on GitHub](#)

This project is maintained by [worldbank](#)

Water When It Counts: Reducing Scarcity through Irrigation Monitoring in Central Mozambique

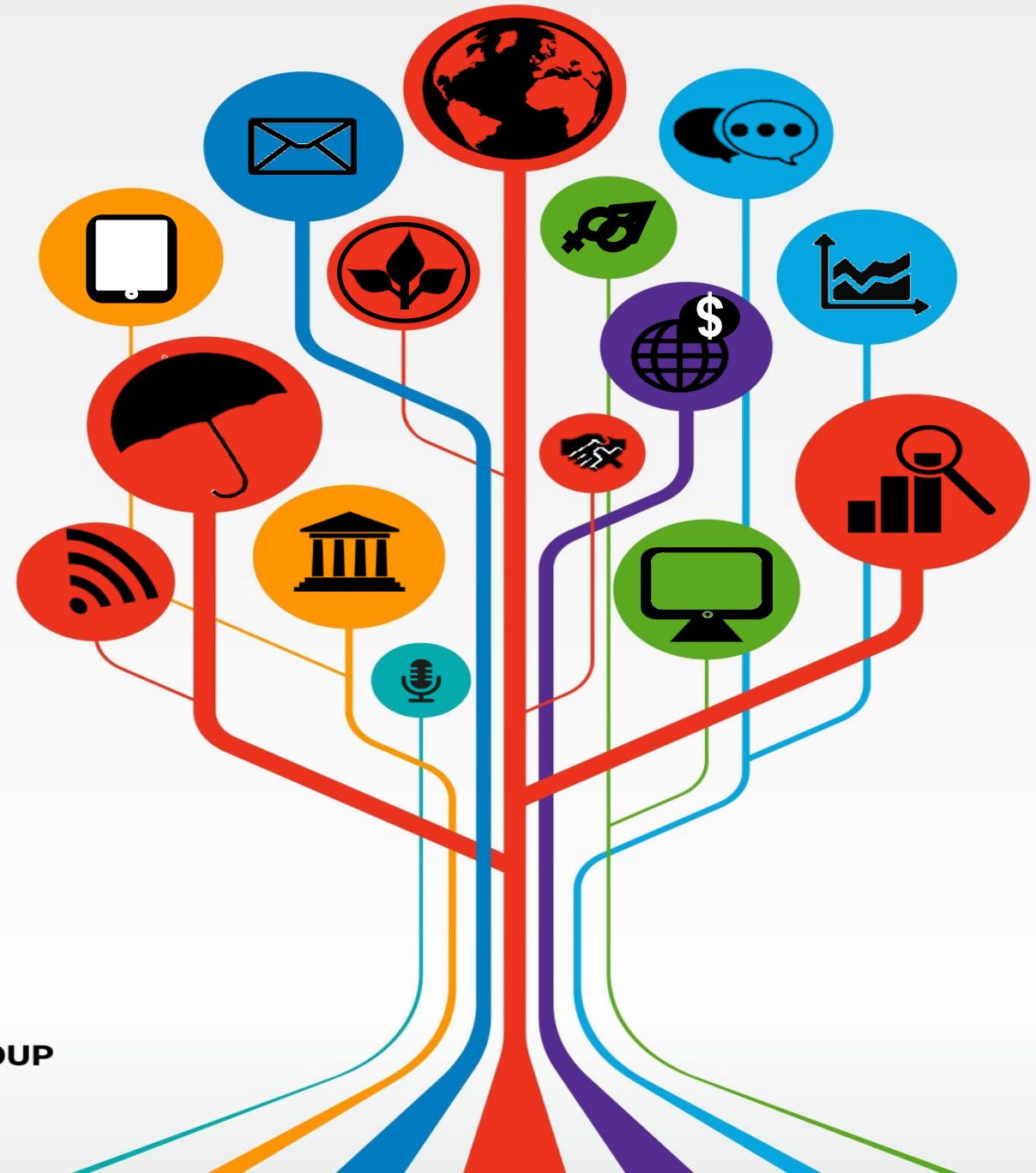
Replication files for Water When It Counts: Reducing Scarcity through Irrigation Monitoring in Central Mozambique by Paul Christian, Florence Kondylis, Valerie Mueller, Astrid Zwager and Tobias Siegfried

Abstract

Management of common-pool resources in the absence of individual pricing can lead to suboptimal allocation. In the context of irrigation schemes, this can create water scarcity even when there is sufficient water to meet the total requirements. High-frequency data from three irrigation schemes in Mozambique reveal patterns consistent with inefficiency in allocations. A randomized control trial compares two feedback tools: i) general information, charting the water requirements for common crops, and ii) individualized information, comparing water requirements with each farmer's water use in the same season of the previous year. Both types of feedback tools lead to higher reported and observed sufficiency of water relative to recommendations, and nearly eliminate reports of conflicts over water. The experiment fails to detect an additional effect of individualized comparative feedback relative to a general information treatment

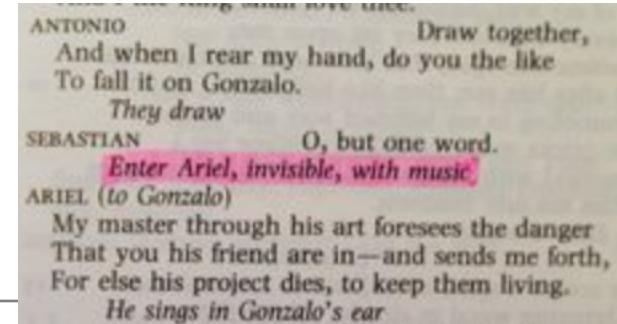
[Full paper](#)

Stata Coding: Syntax



Stata syntax is the language of your code

- Stata code is a *script*, not a *program*
- Think of a play
 - You can *read the script* and get a good idea of what will happen when the instructions are followed
 - This is as important as the actual production (output) because it serves as a record of what you did
- Someone is going to want to read your code!
 - They want to see exactly how you got to the results
 - They want to do something similar but not identical in their own work
- More technical details:
 - Stata does not work like most (“object-oriented”) programming languages
 - These (including R) treat “objects” or “functions” as the fundamental thing that is referenced by code
 - Stata is a *scripting* language for *econometrics*: its basic elements are observations and characteristics (what you know as variables but a programmer will not understand)
 - You cannot [browse] in R because a data.frame is not a basic object there



Functional syntax

- **``Stata quotes''**
 - Will break lots of programming tools because the backtick ` is a special character
 - Are really important to get right
 - `NAME' calls a local macro
 - local NAME "STRING" stores STRING in `NAME'
 - local NAME `"STRING"' stores "STRING" in `NAME'
 - local NAME `=2+2' stores 4 : this works anywhere, like
`yline(`=`beta'*`r(mean)''')`
- **Equals sign =**
 - When used with a macro, they evaluate what comes after
 - local NAME = min(2,3,4) stores 2 in `NAME'
 - local NAME = "min(2,3,4)" stores min(2,3,4) in `NAME'
- **Backslash **
 - *never* use in filepaths: /users/dropbox/
 - “Escapes” functional characters:
 - local NAME = "\`OTHERNAME" stores `OTHERNAME', not the contents of the local `OTHERNAME'

<https://www.stata.com/help.cgi?quotes>

Macros: `local` and \${global}

- “Macros” (this is what programmers call “variables”) hold information within a Stata session
- Difference in *scope* – Use them appropriately according to scope
- Only define globals in the master do-file.
- Use locals everywhere else (varlists, loops, estimation results)... they are deleted after the dofile finishes

```
286 * -----  
287 * Get furrow level data  
288 *-----  
289  
290 * Load data set  
291 use "$analysis_dt\furrow_week.dta", clear  
292  
293 * Collapse to scheme level  
294 collapse post (sum) totwater water_reqday ///  
295 (count) count_totwater = totwater count_water_reqday = water_reqday, ///  
296 by(scheme_id week)  
297  
298 foreach varAux of varlist water_reqday totwater {  
299 replace `varAux' = . if count_`varAux' == 0 // Make sure collapse doesn't  
300 }  
301  
302 * Create water gap  
303 foreach varAux in totwater water_reqday {  
304 gen ln_`varAux' = ln(`varAux') + 1  
305 }  
306  
307 gen water_gap = ln_totwater - ln_water_reqday  
308
```

global: project-level information

local: operation-level information

Macros: naming convention

- Always give a local or a global a name where the reader can tell what it represents.
- Especially in loops, abstract indices can become confusing quickly!
- To reiterate: Stata is a *scripting* language, not a *programming* language.

```
*Before  
forvalues x = 1/3 {  
    forvalues y = 1/10 {  
        forvalues z = 1/6 {  
  
            sum variable cl_harv_c`z'_s`x'_p`y'  
        }  
    }  
}  
  
  
*After  
forvalues seasnum = 1/3 {  
    forvalues plotnum = 1/10 {  
        forvalues cropnum = 1/6 {  
  
            sum variable cl_harv_c`cropnum'_s`seasnum'_p`plotnum'  
        }  
    }  
}
```

Tips for usages of globals

- Only define globals in the main master do-file
- Usages of globals:
 - Root folders
 - Standardize conversion coefficients
 - Varlists commonly used across the project – for example list of controls included in multiple regressions

Tips for usages of locals

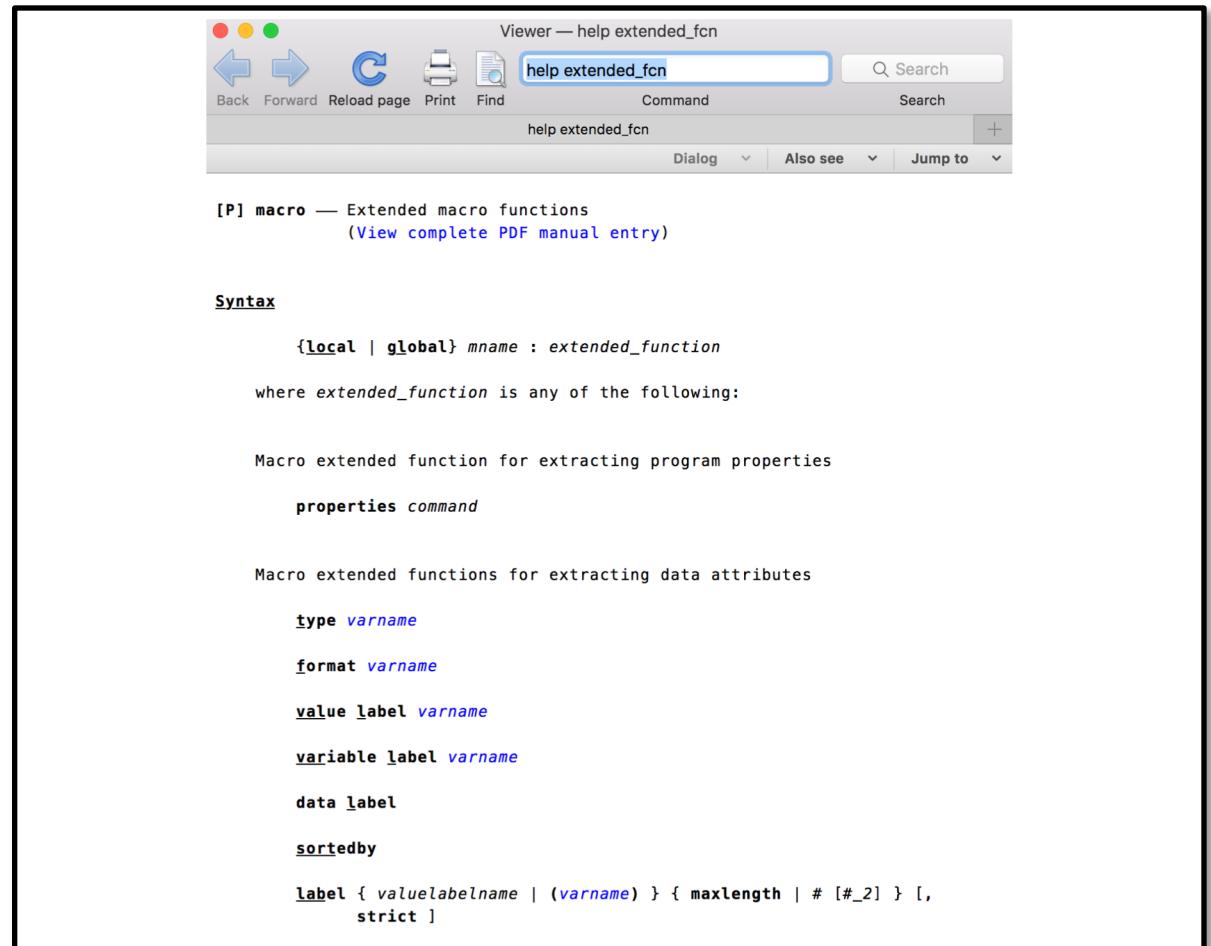
Use locals to shorten variable names and make them more explanatory

```
*Before
sum      c_harv_value          if c_harv_q14_p`plotNum'_s`seasonNum' == 1
reg      c_harv_value fertilizer seedtype labor  if c_harv_q14_p`plotNum'_s`seasonNum' == 1

*After
local   grewRice    c_harv_q14_p`plotNum'_s`seasonNum'
sum      c_harv_value          if `grewRice' == 1
reg      c_harv_value fertilizer seedtype labor  if `grewRice' == 1
```

“Extended functions” in Stata

- Because variables are the core of Stata, it knows a lot about them
- Extended functions access that information
 - *[h extended_fcn]*
- They pull information about variables (types, labels, etc), locals, and other Stata native objects into macros
- *[local theLabel : val lab foreign]*



The screenshot shows the Stata Help viewer window titled "Viewer — help extended_fcn". The search bar contains "help extended_fcn". The main content area displays the help documentation for the `macro` command, specifically for extended macro functions. It includes sections for Syntax, where it shows the command structure `{local | global} mname : extended_function`, and examples of extended functions like `properties command` and `label { valuelabelname | (varname) } { maxlenlength | # [#_2] } [, strict]`.

```
[P] macro — Extended macro functions  
(View complete PDF manual entry)

Syntax
{local | global} mname : extended_function
where extended_function is any of the following:
Macro extended function for extracting program properties
properties command

Macro extended functions for extracting data attributes
type varname
format varname
value_label varname
variable_label varname
data_label
sortedby

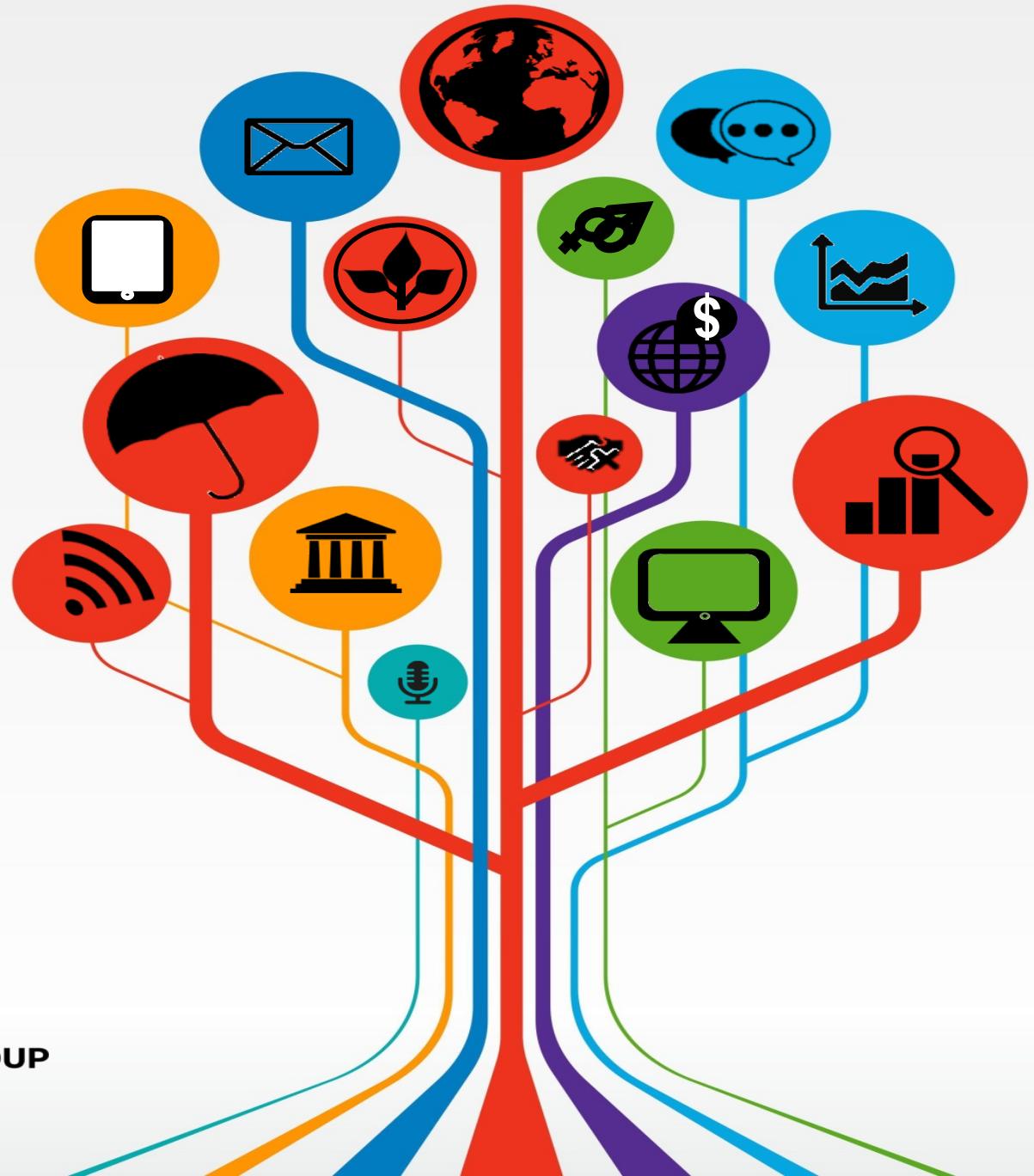
label { valuelabelname | (varname) } { maxlenlength | # [#_2] } [, strict ]
```

Stored results and [return], [ereturn], [creturn]

```
1 clear
2
3 sysuse auto
4
5 reg price mpg rep78 headroom
6 local mpgBeta= _b[mpg]
7 di "`mpgBeta'"
8
9 * return holds outputs
10
11 return list
12
13 mat results = r(table)
14 matlist results
15
16 * ereturn holds estimates
17
18 ereturn list
19 di "`e(N)'"
20 count if e(sample) = 1
21
22
```

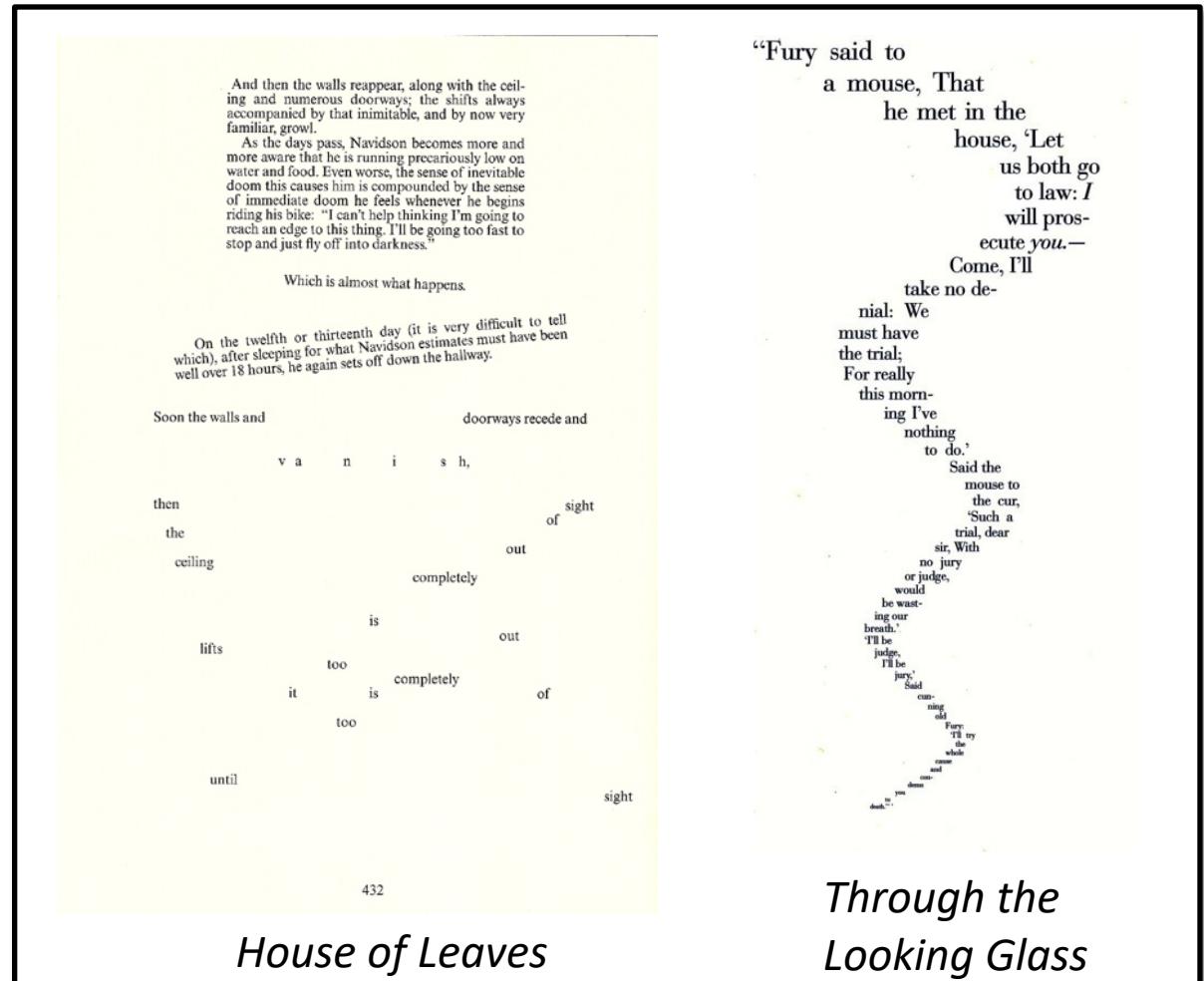
```
23 * creturn holds system information
24
25 creturn list
26
27 foreach letter in `c(alpha)` {
28     di "`letter'"
29 }
30
31 * return is useful!
32
33 qui count
34 forvalues i = 1/`r(N)' {
35     local theName = make[`i']
36     di "`theName'"
37 }
38
39 * Need a blank line at the end
```

Stata Coding: Style



Stata style is the formatting of your code

- “Typesetting”: the organization of the material induces a logic in the eye of the reader
- This can work *for* or *against* the meaning that the code conveys
- Simple but powerful elements: alignment, white space, line breaks, comments



Stata style makes your code “readable”

- **In general, spell out commands and variable names fully**
- This makes sure that:
 1. People in any language can tell what command you are using
 2. Variable lists don't change based on ordering (*var1-var10* and *var_** are particularly bad practice)
 3. Code reads well in English as a narrative for your own sanity



```
Regressions.do
*****
* PART 2: Treatment effect on share of households with negative water gap
*****
74 eststo clear
75
76 * First: simple DID
77 reghdfe water_gap_neg tmt_hh post tmt_hh_post, ///
    vce(cluster hh_id) absorb(pair moyplant)
78
79 eststo
80 estadd local pair "Yes"
81 estadd local month "Yes"
82
83 * Second: with heterogeneity
84 reghdfe water_gap_neg tmt_hh post tmt_hh_post ///
    high tmt_hh_high post_high tmt_hh_post_high, ///
    absorb(pair moyplant) ///
    vce(cluster hh_id)
85
86 eststo
87 estadd local pair "Yes"
88 estadd local month "Yes"
89
90 * Export
91 esttab using "$out_regs/delete_me.tex", ///
    replace label ar2 se(%9.3f) ///
    scalars("pair Randomization pairs fixed-effects" "month Month of planting fixed-effects") ///
    noconstant star(* 0.10 ** 0.05 *** 0.01) ///
    addnotes(Notes: Observations at are at household-round-plot-crop-growth stage level. Sample is re-
    order(post tmt_hh post tmt_hh_post high post_high tmt_hh_high tmt_hh_post_high)
92
93 sleep $sleep
94 filefilter "$out_regs/delete_me.tex" "$out_regs/water_gap_neg.tex", ///
    from("{l}") to("{p}{\BStextwidth}") replace
95
96 erase "$out_regs/delete_me.tex"
97
98
99
100
101
102
103
104
105
```

White Space

- Stata does not distinguish between one empty space and many empty spaces, or one line break or many line breaks
- It makes a big difference to the human eye and we would never share a Word document, an Excel sheet or a PowerPoint presentation without thinking about white space – although we call it formatting

White space used for vertical alignment

```
gen NoPlotDataBL = 0  
replace NoPlotDataBL = 1 if c_plots_total_area >= .  
  
gen NoHarvValueDataBL = 0  
replace NoHarvValueDataBL = 1 if c_harv_value >= .  
  
rename c_gross_yield c1_gross_yield  
rename c_net_yield c1_net_yield  
rename c_harv_value c1_harv_value  
rename c_total_earnings c1_total_earnings  
rename c_input_spend c2_inp_total_spending  
rename c_IAAP_harv_value c1_IAAP_harv_value  
rename c_plots_total_area c1_total_plotsize  
rename c1_cropPlotShare_??? c1_cropPlotShare_all_???  
  
tempfile BL_append  
save `BL_append'
```

```
gen NoPlotDataBL = 0  
replace NoPlotDataBL = 1 if c_plots_total_area >= .  
  
gen NoHarvValueDataBL = 0  
replace NoHarvValueDataBL = 1 if c_harv_value >= .  
  
rename c_gross_yield c1_gross_yield  
rename c_net_yield c1_net_yield  
rename c_harv_value c1_harv_value  
rename c_total_earnings c1_total_earnings  
rename c_input_spend c2_inp_total_spending  
rename c_IAAP_harv_value c1_IAAP_harv_value  
rename c_plots_total_area c1_total_plotsize  
rename c1_cropPlotShare_??? c1_cropPlotShare_all_???  
  
tempfile BL_append  
save `BL_append'
```

White space: Indentations suggest hierarchy

Makes code much more readable!

- Use for preserve/restore, loops and all other commands with curly brackets
- Very easy to see in “minimap” viewer
(all advanced editors can display this – Atom is a particularly good one)



```
41
42 ****
43 *      PART 2: Create variables of interest
44 ****
45
46 * Create conflict and enough water variables
47 *
48 foreach varAux in conflict water {
49
50     egen sum_`varAux' = rowtotal(d_`varAux'_n)
51     egen obs_`varAux' = rownonmiss(d_`varAux'_n)
52
53     bys hh_id plot_id: gen pct_`varAux'_plot =
54     bys hh_id:       gen pct_`varAux' = sum_`varAux'
55
56 }
57
58 * Keep only one observation per household
59 bys hh_id: gen hh1 = _n
60 bys hh_id: egen pct_water_hh = mean(pct_water_
61 bys hh_id: replace pct_water_hh = . if hh1 > 1
62 replace pct_water_hh = . if d_surveyed_hh = 0
```

```

* Renaming pond variables
forvalues pondNo = 1/3 {
    forvalues fishNo = 1/6 {
        foreach varname in ish_code ish_name {
            _buy _buy_tk _harvest _stage f_h
            f_s_u f_s_tk f_current f_ct_n f_c
            m_consume m_c_n m_c_u m_sold m_s
        }

        rename pd`pondNo'_f`varname''fishi
    }

    if "`varname'" == "ish_code" {
        rename d2_pd`pondNo'_f`fishNo
        label var d2_pd`pondNo'_f`fish
        label val d2_pd`pondNo'_f`fish
    }

    *Removing trailing underscores
    if substr("`varname'",length("`varname"),1) == "_"
        local varname = substr("`varname'",1,length("`varname")-1)
        rename d2_pd`pondNo'_f`fishNo
    }

    *Removing trailing underscores
    if substr("`varname'",length("`varname"),1) == "_"
        local varname = substr("`varname'",1,length("`varname")-1)
        rename d2_pd`pondNo'_f`fishNo
    }

    *Dropping variables that
    drop d2_pd`pondNo'_fish
}

*Dropping variables created when
*incorrectly added a 7th fish to
    drop d2_pd?_fish*7

```

Some text editors will also illustrate this alignment for you.

With comments, it points out the section of code

With logic and loops, it helps find matching braces

	250	
	231	qui gen vargroup = 1
	232	
	233	if `n_vargroups' > 1 {
	234	forvalues i = 1/`n_vargroups' {
	235	foreach varname in `vargroup_`i'' {
	236	qui replace vargroup = `i' if regexm(`vargroup_`i'',varname)
	237	}
	238	}
	239	}
	240	

Break up long rows of code

One should never have to scroll horizontally to be able to read code

Two recommended ways to break up lines:

1. [///]
 - Everything on the same row will be interpreted as a comment and the following row will be interpreted as if it was the same row
 - Good for breaking a long line of code into a few rows
2. [#delimit ;] and [#delimit cr]
 - Everything between `#delimit ;` and `#delimit cr` is executed as one line unless it is manually specified using a semicolon
 - Good for breaking a very long line of code into many rows

Example of row breaks

```
local cropcodes    101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116    ///
    117 118 119 120 121 122 123 124 125 126 127 128 129 130 133 138    ///
    139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154    ///
    155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170    ///
    171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186    ///
    187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202    ///
    203 204 205 207 208 209 210 211 212 213 214 215 216 217 218 219    ///
    220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235    ///
    236 237 238 239
```

```
* Code to export Graph
#delimit ;

histogram cons_bread, percent normal
    start(0) bin(10)
    bfcolor ("178 0 80") blcolor("76 0 32")
    ytitle ("Frequency")
    title ("Food Security")
    xtitle ("Number of days")
    subtitle ("Bread Consumption (All Sample)")
    note ("Includes anyone in the HH who consumed bread last week");
    * Saving graph
    graph save "$outputs/Graph1_bread_consum.gph", replace;

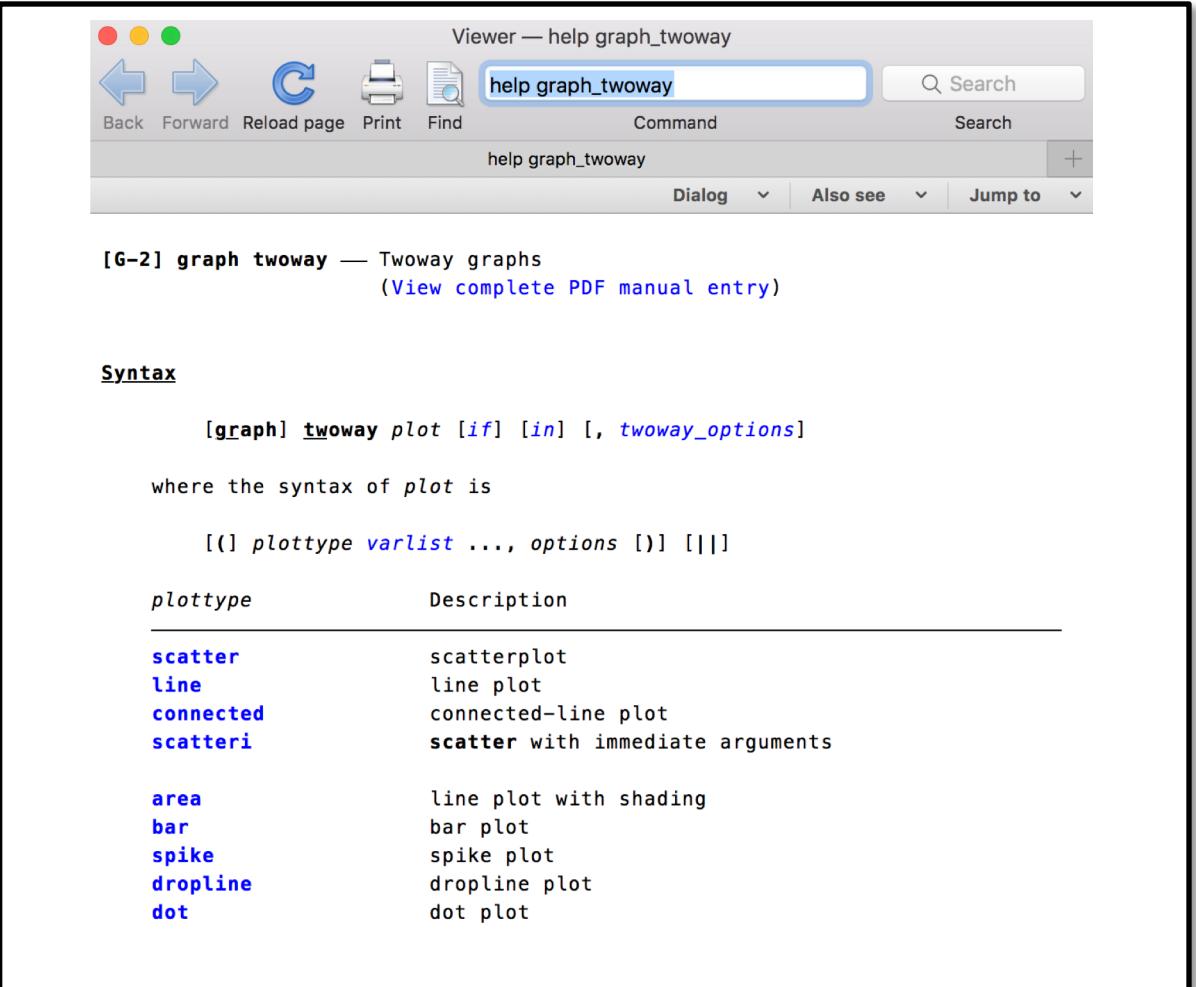
#delimit cr;
```

Line ends with
“delimiter”



Documentation is your best friend!

- Throughout the rest of the training sessions (and your programming life), you will need [*help*]!
- Type [*help commandname*] in Stata at any time
- Or, google “*commandname statalist*” for the user-contributed help group
- Let’s read this one together:



The screenshot shows a Stata help window titled "Viewer — help graph_twoway". The search bar at the top contains "help graph_twoway". Below the title, the text "[G-2] graph twoway — Twoway graphs" is displayed, followed by a link "(View complete PDF manual entry)". A section titled "Syntax" contains the command syntax: `[graph] twoway plot [if] [in] [, twoway_options]`, noting that the syntax of `plot` is `[() plottype varlist ..., options []] [|]`. A table titled "plottype" lists various plot types and their descriptions:

plottype	Description
scatter	scatterplot
line	line plot
connected	connected-line plot
scatter	scatter with immediate arguments
area	line plot with shading
bar	bar plot
spike	spike plot
dropline	dropline plot
dot	dot plot

Resources

[Stata Cheat Sheets](#)

[SSC Stata Commands](#)

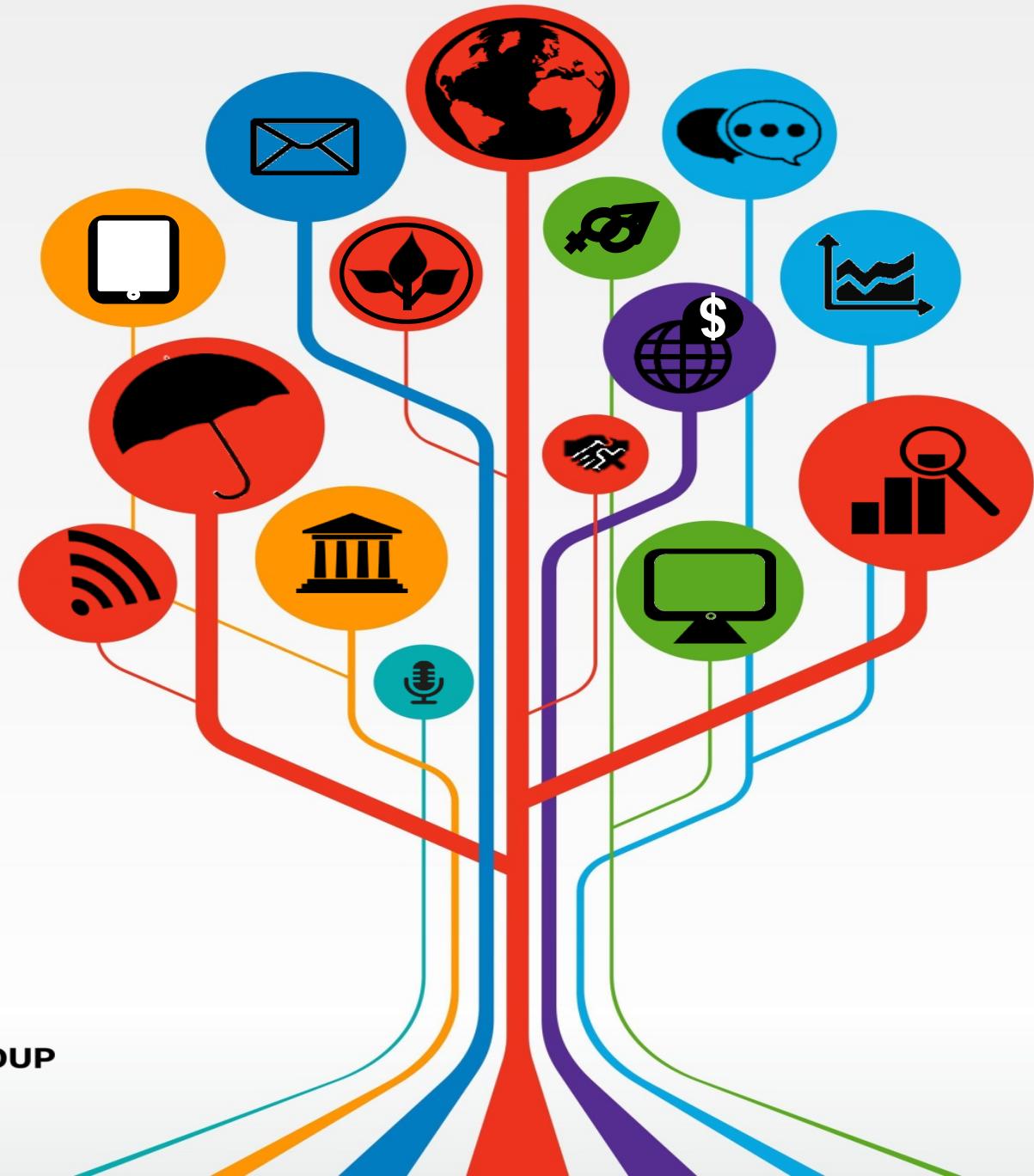
[UCLA Stata Tutorials](#)

[UCLA Visual Library](#)

[Stata Video Library](#)

[Speaking Stata Library](#)

[EGAP Methods Guides](#)



Open access resources from the DIME Analytics team

[Implementation Support](#)

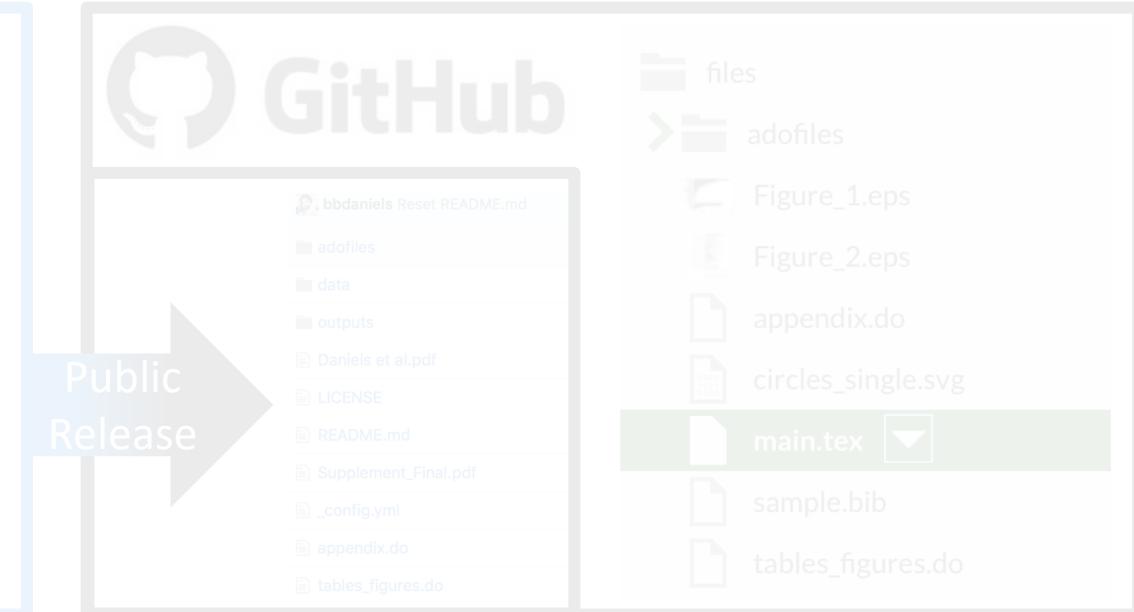
[Import Commands](#)

[ietoolkit](#)

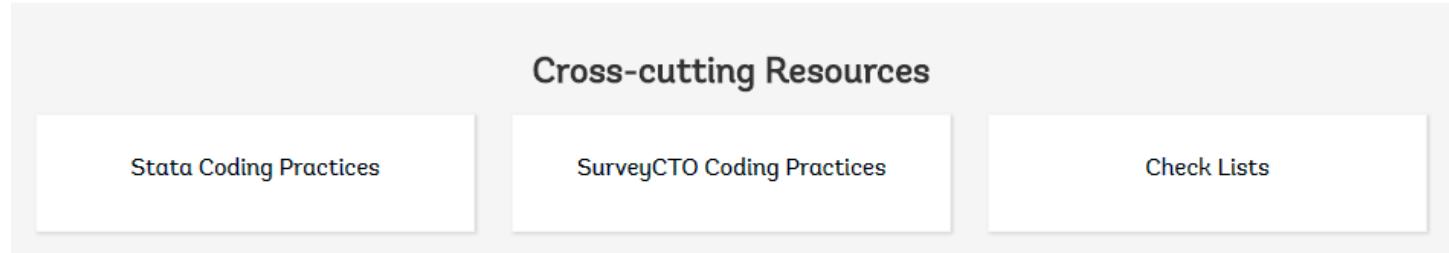
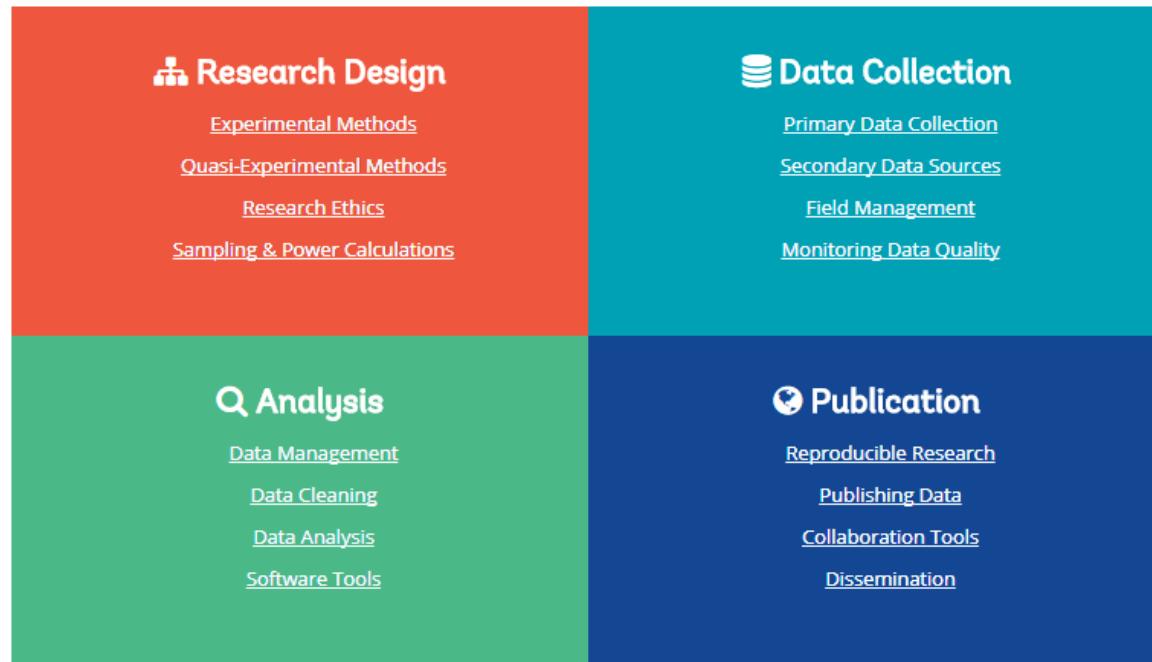
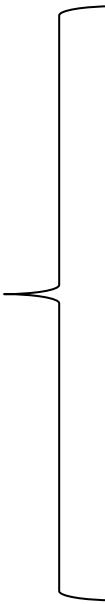
[Stata GitHub](#)

[LaTeX trainings](#)

[GitHub Trainings](#)



DIME Wiki:
structure mirrors
open research
workflow



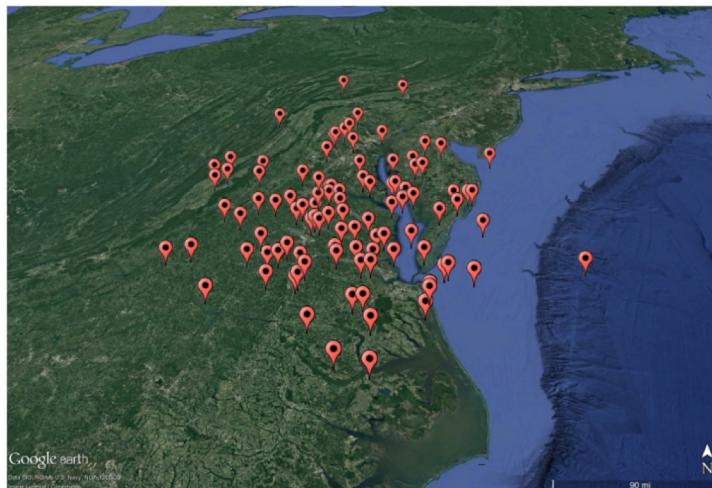
Cross-cutting resources
highlight specific tools
that support best
practices

World Bank GitHub Repos

Code libraries and development repositories on GitHub allow collaborative improvement of software and open-source access to researchers everywhere.

dta2kml

dta2kml outputs decimal lat/lon coordinates into a KML file for visual exploration.



```
wb_git_install dta2kml  
clear  
set obs 100  
gen lat = rnormal() +38  
gen lon = rnormal() -77  
dta2kml using demo.kml , lat(lat) lon(lon) replace
```

stata

Stata Commands for Data Management and Analysis

[View the Project on GitHub](#)

World Bank GitHub

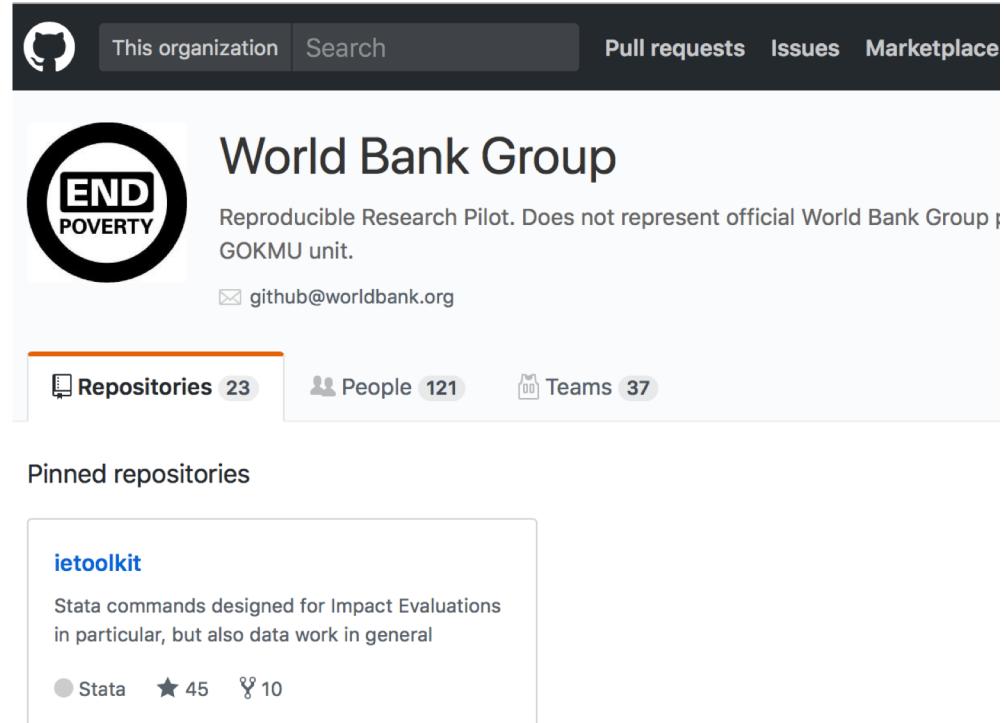
Other World Bank Repositories

- [Impact Evaluations Toolkit](#)
- [Stata Visual Library](#)
- [Distributional Impact Analysis Toolkit](#)

ietoolkit

Stata package routinizing common analytical tasks in
IEs

Widely used in DIME, and endorsed by global
research community



The screenshot shows the GitHub organization page for "World Bank Group". The header includes a GitHub icon, navigation links for "This organization", "Search", "Pull requests", "Issues", and "Marketplace". Below the header is a circular logo for "END POVERTY". The main title is "World Bank Group" with the subtitle "Reproducible Research Pilot. Does not represent official World Bank Group or GOKMU unit." An email link "github@worldbank.org" is provided. At the bottom, there are statistics: "Repositories 23", "People 121", and "Teams 37". A section titled "Pinned repositories" features a box for "ietoolkit" with the description "Stata commands designed for Impact Evaluations in particular, but also data work in general". It shows a Stata icon, a star rating of "45", and a "10" next to a small icon.

Reproducible research tools

LaTeX

- Create dynamic documents
- Export and update results transparently, without manual changes



GitHub

- Transparent research with public codes
- Easier collaboration



Using GitHub for code collaboration

November 2, 2017

DIME DYNAMIC DOCUMENTS TRAINING Exercise 1

Luiza Andrade & Mrijan Rimal



Thank you!

Access lab materials at:
bit.ly/2018-stata2

