THE WORLD BANK
IBRD • IDA | WORLD BANK GROUP
Development Economics • Impact

DIME ANALYTICS

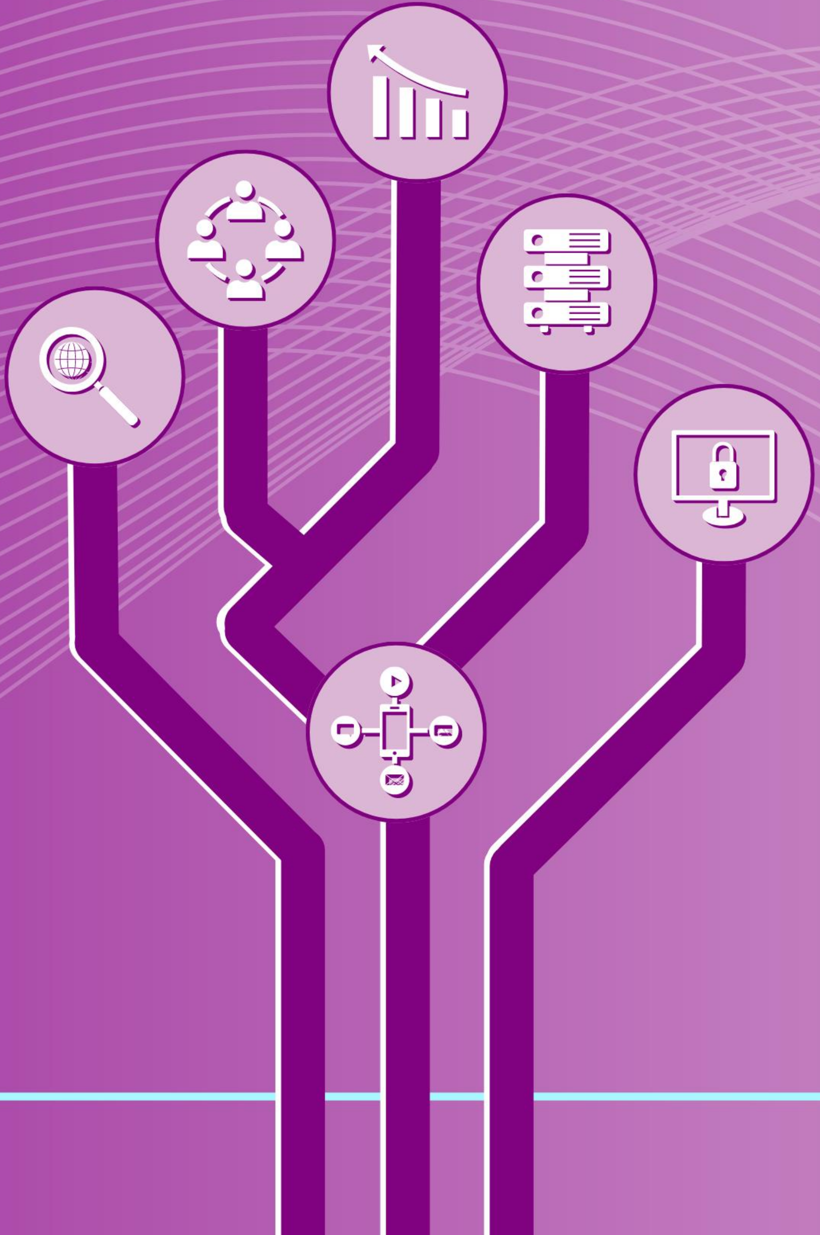
repkit

Stata tools for reproducible coding and automated verification

Luis Eduardo San Martin

Junior Data Scientist

World Bank



About this work

- Our team has verified 200+ reproducibility packages for World Bank research
  - 77% of works reviewed used Stata
  - Only 18% are reproducible at first try
- Introducing repkit: a Stata package to address common reproducibility challenges (and make verification easier)

BRIEF

World Bank Policy Research Working Papers

Questions • Methods • Findings • Implications

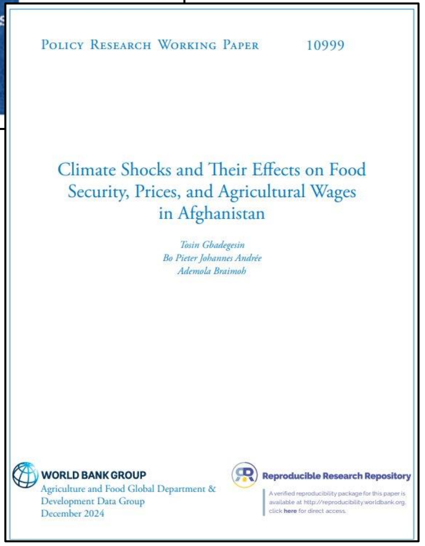
POLICY RESEARCH WORKING PAPERS

www.worldbank.org/prwp

WORLD BANK GROUP

Search the Entire Series

You can search the series by author, country, region, title, topic...etc.


POLICY RESEARCH WORKING PAPER 10999

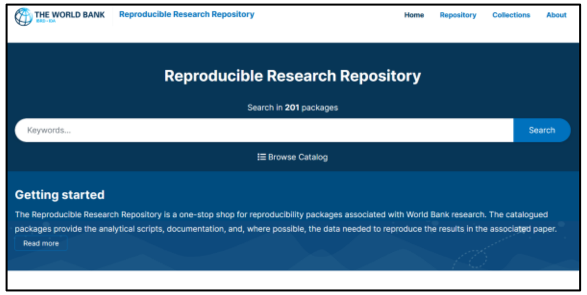
Climate Shocks and Their Effects on Food Security, Prices, and Agricultural Wages in Afghanistan

Tosin Gbadegesin
Bo Pieter Johannes Andrée
Ademola Braimoh

WORLD BANK GROUP
Agriculture and Food Global Department & Development Data Group
December 2024

Reproducible Research Repository

A verified reproducibility package for this paper is available at http://reproducibility.worldbank.org. click **here** for direct access.


THE WORLD BANK | Reproducible Research Repository — Home — Repository — Collections — About

Reproducible Research Repository

Search in **201** packages

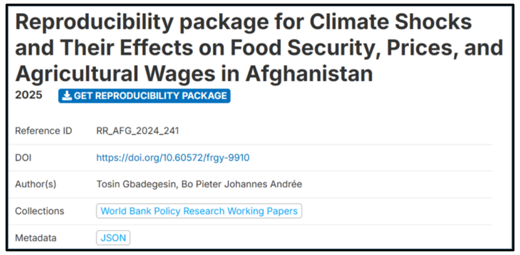
Keywords... | Search

Browse Catalog

Getting started

The Reproducible Research Repository is a one-stop shop for reproducibility packages associated with World Bank research. The catalogued packages provide the analytical scripts, documentation, and, where possible, the data needed to reproduce the results in the associated paper.

Read more


Reproducibility package for Climate Shocks and Their Effects on Food Security, Prices, and Agricultural Wages in Afghanistan

2025 GET REPRODUCIBILITY PACKAGE

Reference ID	RR_AFG_2024_241
DOI	https://doi.org/10.60572/frgy-9910
Author(s)	Tosin Gbadegesin, Bo Pieter Johannes Andrée
Collections	World Bank Policy Research Working Papers
Metadata	JSON

https://www.worldbank.org/en/research/brief/world-bank-policy-research-working-papers

https://reproducibility.worldbank.org

repkit

Commands included

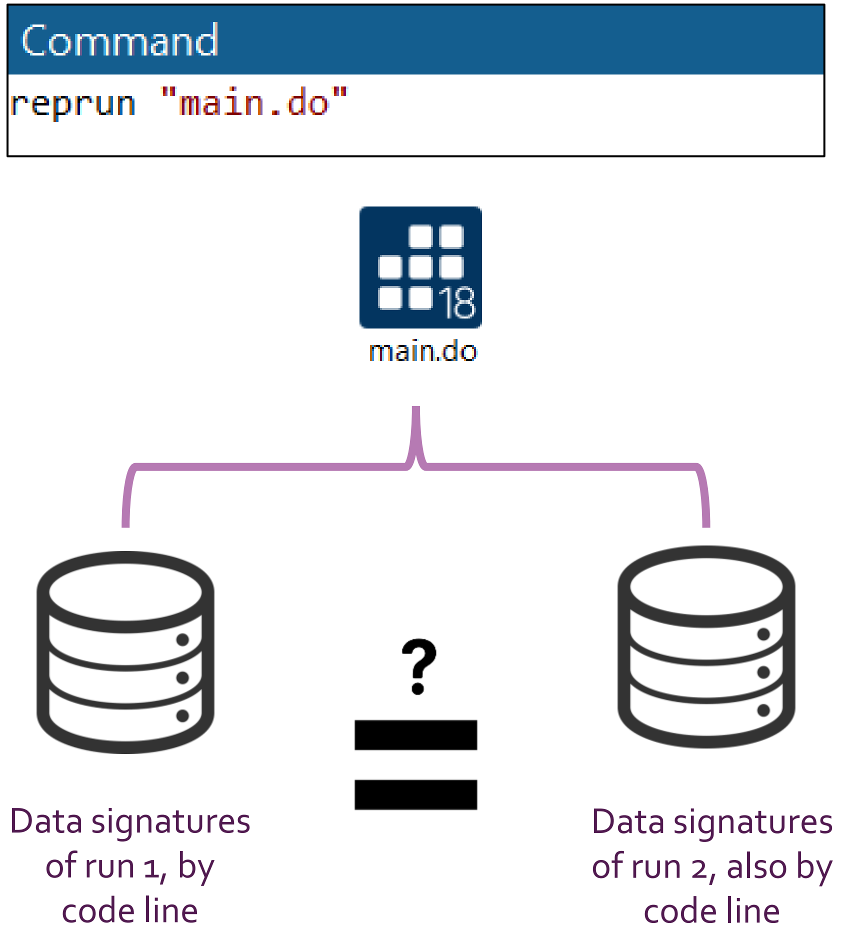
1 – Check for instabilities in your code: `reprun`

The problem:

- Reproducibility issues arise from Stata commands that introduce unnoticed/uncontrolled randomness
- If hundreds or thousand of lines of code, costly to detect where the instability starts

Solution:

- `reprun` runs a do-file twice and compares intermediate results **line-by-line** across runs, flagging inconsistencies
- Works smoothly with sub do-files and loops
- Word of caution: it's only as fast as your actual code



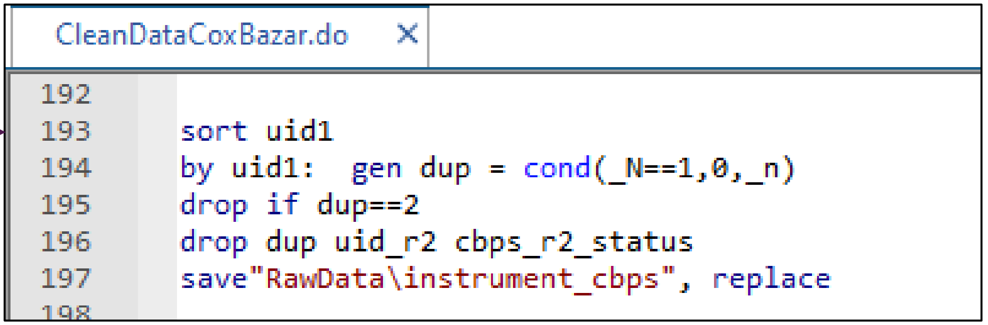
Command

```
reprun "main.do"
```

main.do
?
Data signatures of run 1, by code line
Data signatures of run 2, also by code line

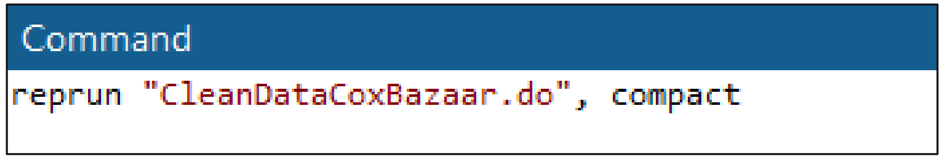
1 – Check for instabilities in your code: `reprun`

Lines 193-195: Sorting on non-unique variable and dropping observations based on sorting



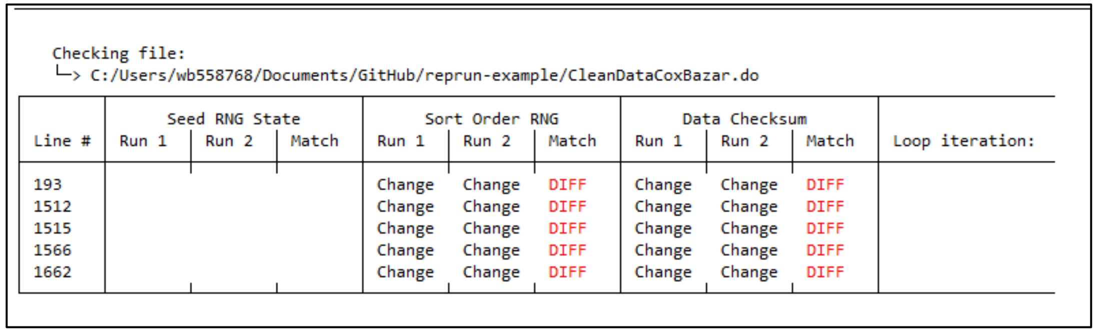
CleanDataCoxBazar.do

```
192
193    sort uid1
194    by uid1:  gen dup = cond(_N==1,0,_n)
195    drop if dup==2
196    drop dup uid_r2 cbps_r2_status
197    save"RawData\instrument_cbps", replace
198
```



Command

```
reprun "CleanDataCoxBazaar.do", compact
```



```
Checking file:
 └─> C:/Users/wb558768/Documents/GitHub/reprun-example/CleanDataCoxBazar.do
 ------------------------------------------------------------------------------------------------
        |     Seed RNG State      |     Sort Order RNG      |      Data Checksum      |
 Line # | Run 1 | Run 2 | Match   | Run 1  | Run 2  | Match | Run 1  | Run 2  | Match | Loop iteration:
 ------------------------------------------------------------------------------------------------
 193    |       |       |         | Change | Change | DIFF  | Change | Change | DIFF  |
 1512   |       |       |         | Change | Change | DIFF  | Change | Change | DIFF  |
 1515   |       |       |         | Change | Change | DIFF  | Change | Change | DIFF  |
 1566   |       |       |         | Change | Change | DIFF  | Change | Change | DIFF  |
 1662   |       |       |         | Change | Change | DIFF  | Change | Change | DIFF  |
 ------------------------------------------------------------------------------------------------
```

`reprun` reports an inconsistency in the data checksum (data signature) of runs 1 and 2 that starts in line 193

2 – Avoid absolute file paths: `reproot`

The problem:

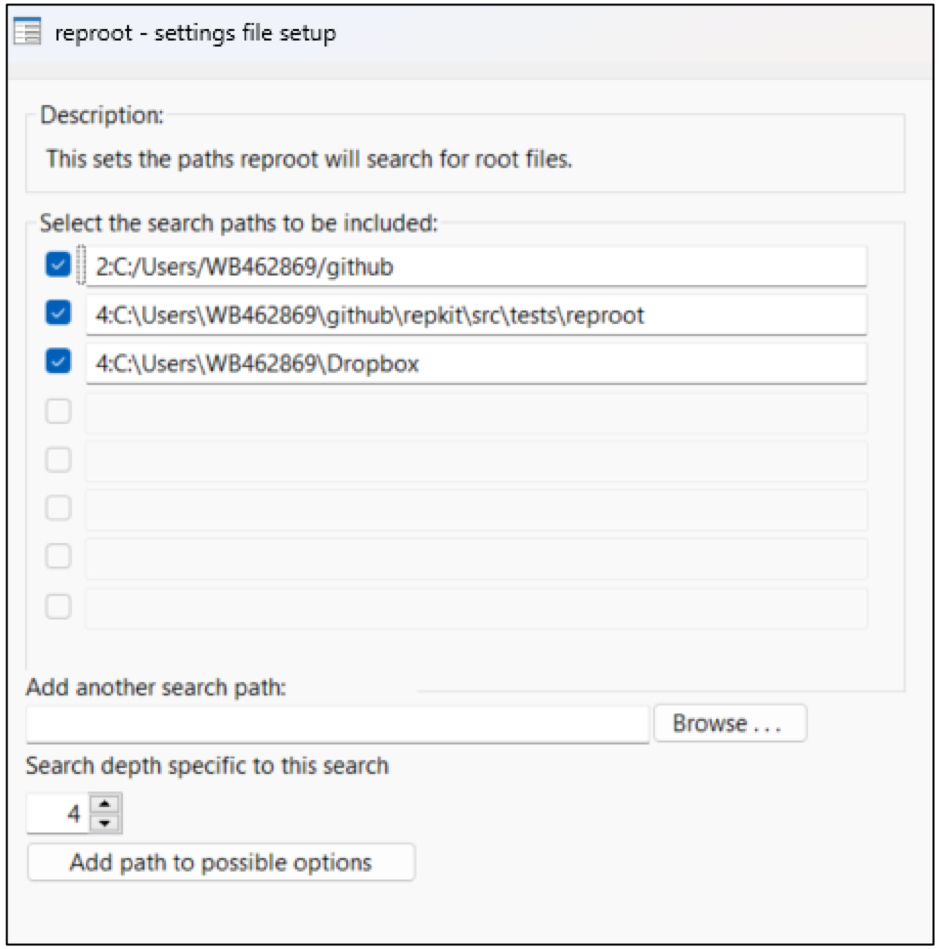
- Code that relies on the use of absolute file paths won't work in a new computer until you adjust file paths, at least once per project

```
* Kristoffer's root path
if "`c(username)'" == "wb462869" {
    global data "C:\Users\wb462869\Dropbox\Projects\ProjectA"
    global code "C:\Users\wb462869\GitHub\ProjectA"
}
* Ben's root path
if "`c(username)'" == "bbdaniels" {
    global data "/Users/bbdaniels/Dropbox/ProjectA"
    global code "/Users/bbdaniels/GitHub/ProjectA"
}
```

Solution:

- `reproot` manages file paths in the backend, looking in pre-defined folder locations for projects
- Needs to be set up only **once per computer**
- Works with multi-rooted projects. For example:
  - Data is on OneDrive or Dropbox
  - Code is in a GitHub repository

2 – No more absolute file paths: `reproot`


reproot - settings file setup

Description:

This sets the paths reproot will search for root files.

Select the search paths to be included:

- [x] 2:C:/Users/WB462869/github
- [x] 4:C:\Users\WB462869\github\repkit\src\tests\reproot
- [x] 4:C:\Users\WB462869\Dropbox
- [ ]
- [ ]
- [ ]
- [ ]
- [ ]

Add another search path:

Search depth specific to this search: 4

```
* Use reproot to get root paths
reproot, project("proj-a") ///
  roots("code data") prefix("prja_")
* Load data
use "${prja_data}/data/raw.dta"
* Run analysis
do  "${prja_code}/code/tab1.do"
do  "${prja_code}/code/tab2.do"
```

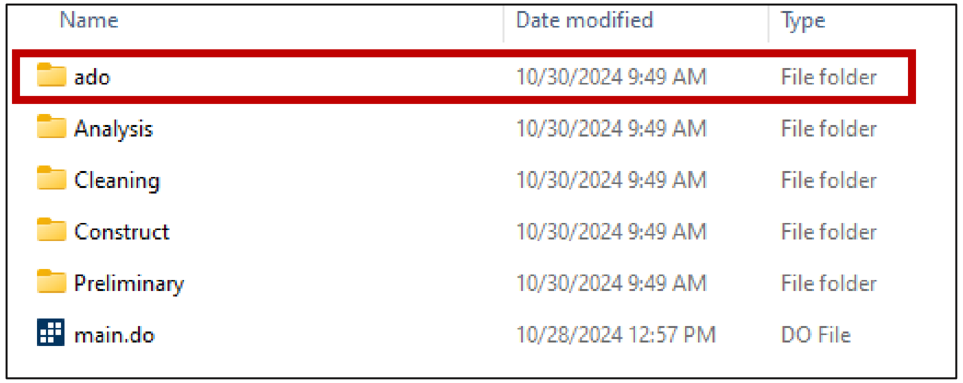
3 – Manage dependencies simply: `repado`

The problem:

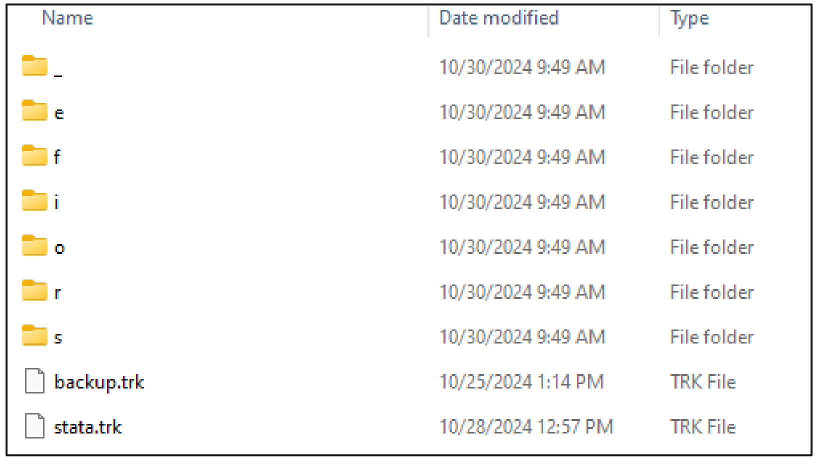
- Missing Stata dependencies stop code execution
- Using different versions of Stata dependencies can produce different results for the same code, causing problems in reproducibility

Solution:

- Use `repado` to temporarily change the dependencies folder (ado) in a main do-file
- Use one ado folder by project and share it with your team
- When creating a reproducibility package, include the ado folder in it

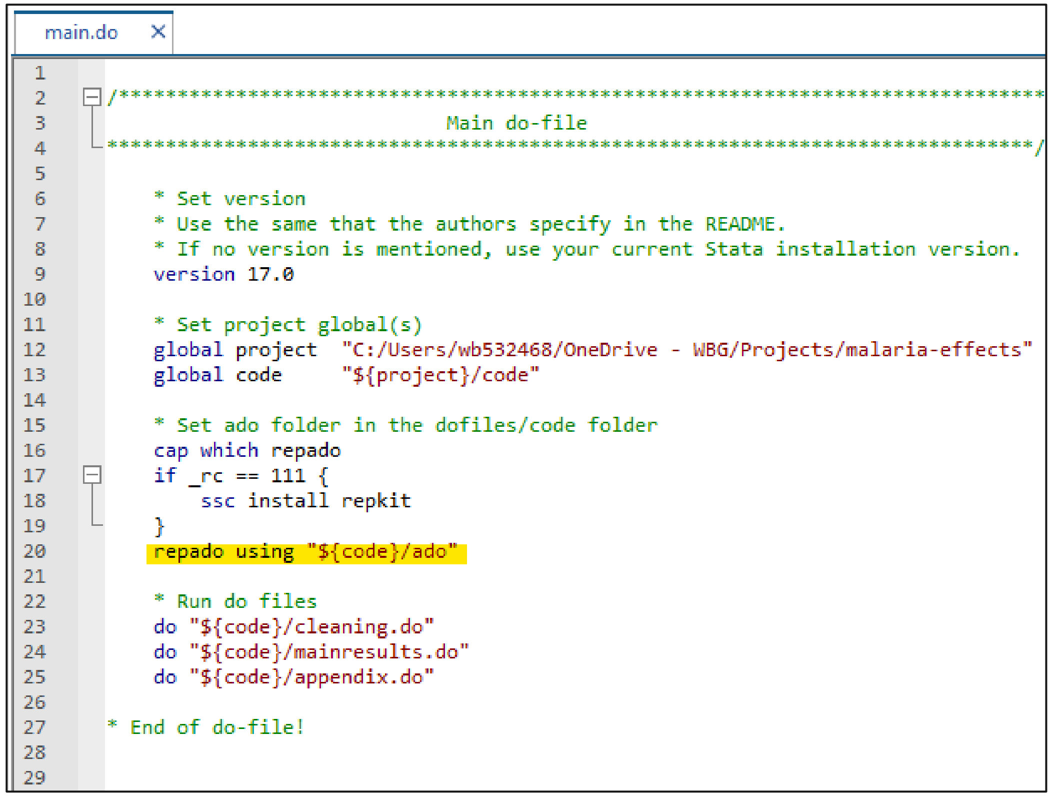


Name	Date modified	Type
ado	10/30/2024 9:49 AM	File folder
Analysis	10/30/2024 9:49 AM	File folder
Cleaning	10/30/2024 9:49 AM	File folder
Construct	10/30/2024 9:49 AM	File folder
Preliminary	10/30/2024 9:49 AM	File folder
main.do	10/28/2024 12:57 PM	DO File



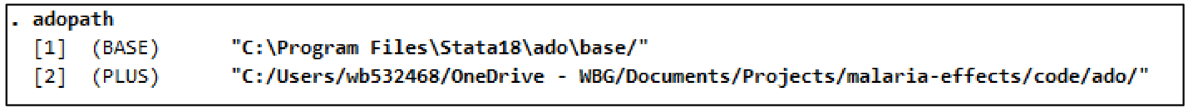
Name	Date modified	Type
_	10/30/2024 9:49 AM	File folder
e	10/30/2024 9:49 AM	File folder
f	10/30/2024 9:49 AM	File folder
i	10/30/2024 9:49 AM	File folder
o	10/30/2024 9:49 AM	File folder
r	10/30/2024 9:49 AM	File folder
s	10/30/2024 9:49 AM	File folder
backup.trk	10/25/2024 1:14 PM	TRK File
stata.trk	10/28/2024 12:57 PM	TRK File

3 – Manage dependencies simply: `repado`



main.do

```
/*******************************************************************************
                              Main do-file
*******************************************************************************/

    * Set version
    * Use the same that the authors specify in the README.
    * If no version is mentioned, use your current Stata installation version.
    version 17.0

    * Set project global(s)
    global project  "C:/Users/wb532468/OneDrive - WBG/Projects/malaria-effects"
    global code     "${project}/code"

    * Set ado folder in the dofiles/code folder
    cap which repado
    if _rc == 111 {
        ssc install repkit
    }
    repado using "${code}/ado"

    * Run do files
    do "${code}/cleaning.do"
    do "${code}/mainresults.do"
    do "${code}/appendix.do"

* End of do-file!
```



```
. adopath
  [1]  (BASE)      "C:\Program Files\Stata18\ado\base/"
  [2]  (PLUS)      "C:/Users/wb532468/OneDrive - WBG/Documents/Projects/malaria-effects/code/ado/"
```

4 – Beautify your Stata code: `lint`

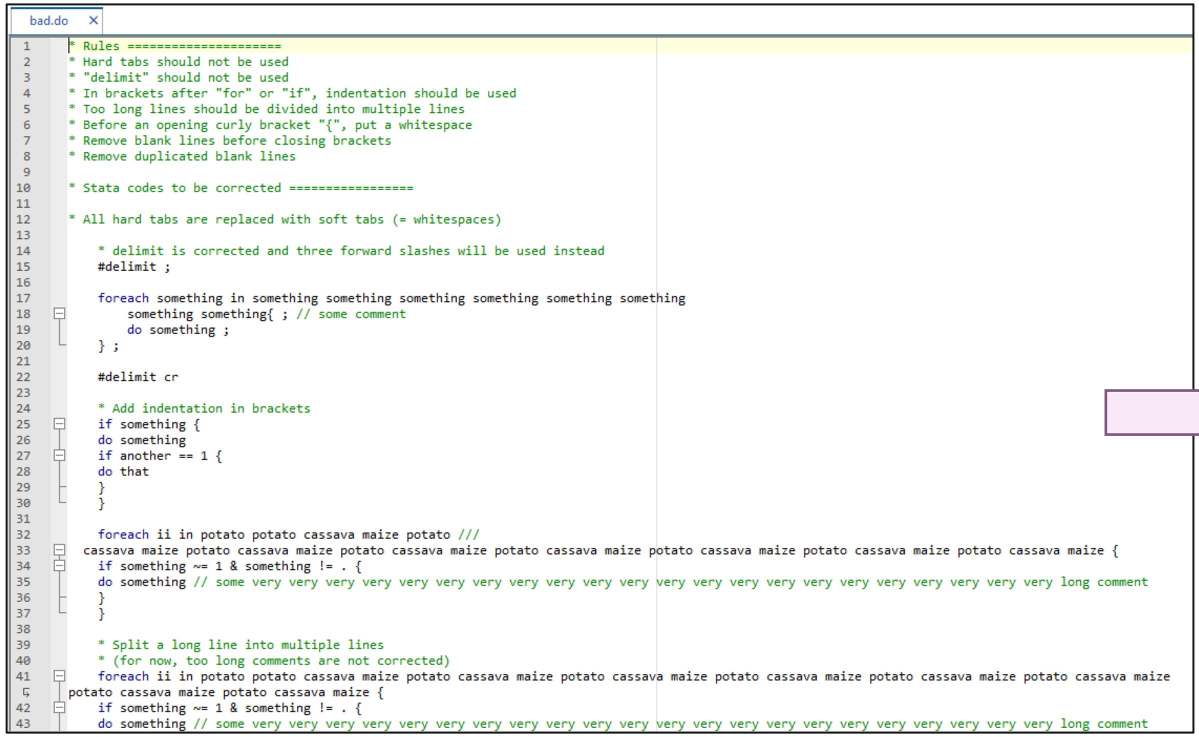
The problem:

- Messy code is hard to understand and error-prone
- More importantly: it's not transparent at all!

Solution:

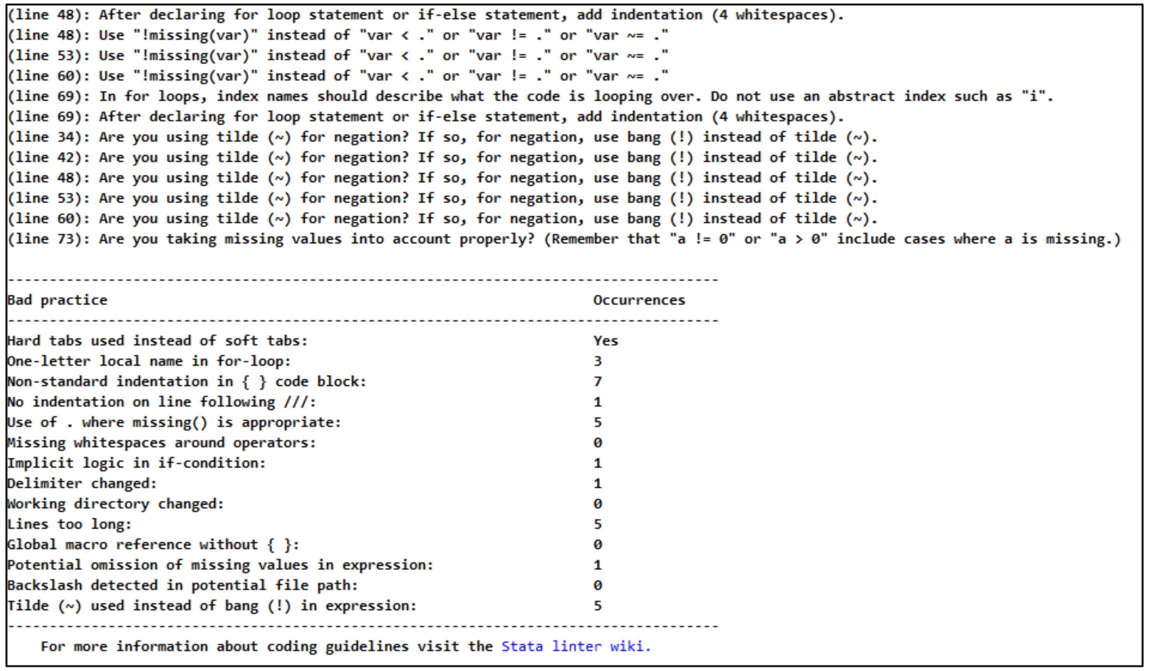
- `lint`: a code linter for Stata
- It scans a do-file and flags coding styles issues, according to the DIME Analytics Stata Style Guide

4 – Beautify your Stata code: `lint`



bad.do

```
1   * Rules ====================
2   * Hard tabs should not be used
3   * "delimit" should not be used
4   * In brackets after "for" or "if", indentation should be used
5   * Too long lines should be divided into multiple lines
6   * Before an opening curly bracket "{", put a whitespace
7   * Remove blank lines before closing brackets
8   * Remove duplicated blank lines
9
10  * Stata codes to be corrected =================
11
12  * All hard tabs are replaced with soft tabs (= whitespaces)
13
14      * delimit is corrected and three forward slashes will be used instead
15      #delimit ;
16
17      foreach something in something something something something something something
18          something something{ ; // some comment
19          do something ;
20      } ;
21
22      #delimit cr
23
24      * Add indentation in brackets
25      if something {
26      do something
27      if another == 1 {
28      do that
29      }
30      }
31
32      foreach ii in potato potato cassava maize potato ///
33    cassava maize potato cassava maize potato cassava maize potato cassava maize potato cassava maize potato cassava maize potato cassava maize {
34      if something ~= 1 & something != . {
35      do something // some very very very very very very very very very very very very very very very very very very very very very very long comment
36      }
37      }
38
39      * Split a long line into multiple lines
40      * (for now, too long comments are not corrected)
41      foreach ii in potato potato cassava maize potato cassava maize potato cassava maize potato cassava maize potato cassava maize potato cassava maize
    potato cassava maize potato cassava maize {
42      if something ~= 1 & something != . {
43      do something // some very very very very very very very very very very very very very very very very very very very very very very long comment
```



```
(line 48): After declaring for loop statement or if-else statement, add indentation (4 whitespaces).
(line 48): Use "!missing(var)" instead of "var < ." or "var != ." or "var ~= ."
(line 53): Use "!missing(var)" instead of "var < ." or "var != ." or "var ~= ."
(line 60): Use "!missing(var)" instead of "var < ." or "var != ." or "var ~= ."
(line 69): In for loops, index names should describe what the code is looping over. Do not use an abstract index such as "i".
(line 69): After declaring for loop statement or if-else statement, add indentation (4 whitespaces).
(line 34): Are you using tilde (~) for negation? If so, for negation, use bang (!) instead of tilde (~).
(line 42): Are you using tilde (~) for negation? If so, for negation, use bang (!) instead of tilde (~).
(line 48): Are you using tilde (~) for negation? If so, for negation, use bang (!) instead of tilde (~).
(line 53): Are you using tilde (~) for negation? If so, for negation, use bang (!) instead of tilde (~).
(line 60): Are you using tilde (~) for negation? If so, for negation, use bang (!) instead of tilde (~).
(line 73): Are you taking missing values into account properly? (Remember that "a != 0" or "a > 0" include cases where a is missing.)

--------------------------------------------------------------------------------
Bad practice                                                      Occurrences
--------------------------------------------------------------------------------
Hard tabs used instead of soft tabs:                              Yes
One-letter local name in for-loop:                                3
Non-standard indentation in { } code block:                       7
No indentation on line following ///:                             1
Use of . where missing() is appropriate:                          5
Missing whitespaces around operators:                             0
Implicit logic in if-condition:                                   1
Delimiter changed:                                                1
Working directory changed:                                        0
Lines too long:                                                   5
Global macro reference without { }:                               0
Potential omission of missing values in expression:              1
Backslash detected in potential file path:                        0
Tilde (~) used instead of bang (!) in expression:                 5
--------------------------------------------------------------------------------
    For more information about coding guidelines visit the Stata linter wiki.
```

Command

```
lint "test/bad.do", verbose
```

4 – Beautify your Stata code: `lint`

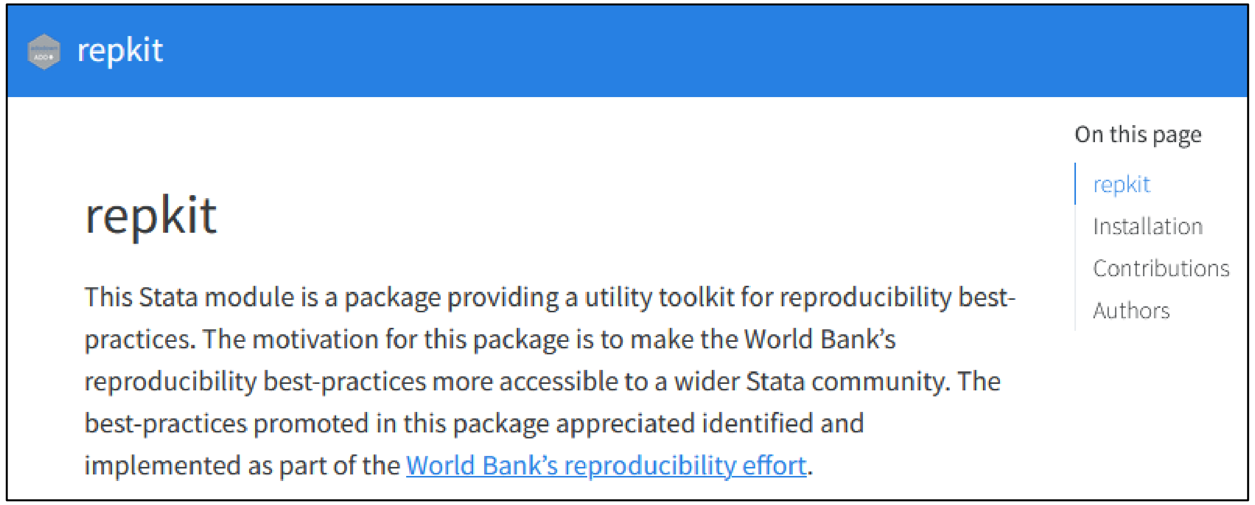
"Programs must be written for people to read, and only incidentally for machines to execute."

—Abelson, Susman, and Susman, Structure and Interpretation of Computer Programs (1985)

Remember: `repkit`

- Package site
- GitHub repository
- Comments or bugs? Here
- Currently available in SSC:

```
ssc install repkit
```



repkit

This Stata module is a package providing a utility toolkit for reproducibility best-practices. The motivation for this package is to make the World Bank's reproducibility best-practices more accessible to a wider Stata community. The best-practices promoted in this package appreciated identified and implemented as part of the World Bank's reproducibility effort.

Thank you!

lsanmartin@worldbank.org