GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Replication and transparency

Stata Self-Learning Course

Overview

Replication and transparency

Overview

Replication and transparency

General remarks

- Minimum aim: computational reproducibility
- Better: Other people understand your data & code and can use it

- There are no universal guidelines for coding in Stata
- What follows is a compilation of different guidelines & own experience
- No “best solution”, no guarantee for completeness

General remarks

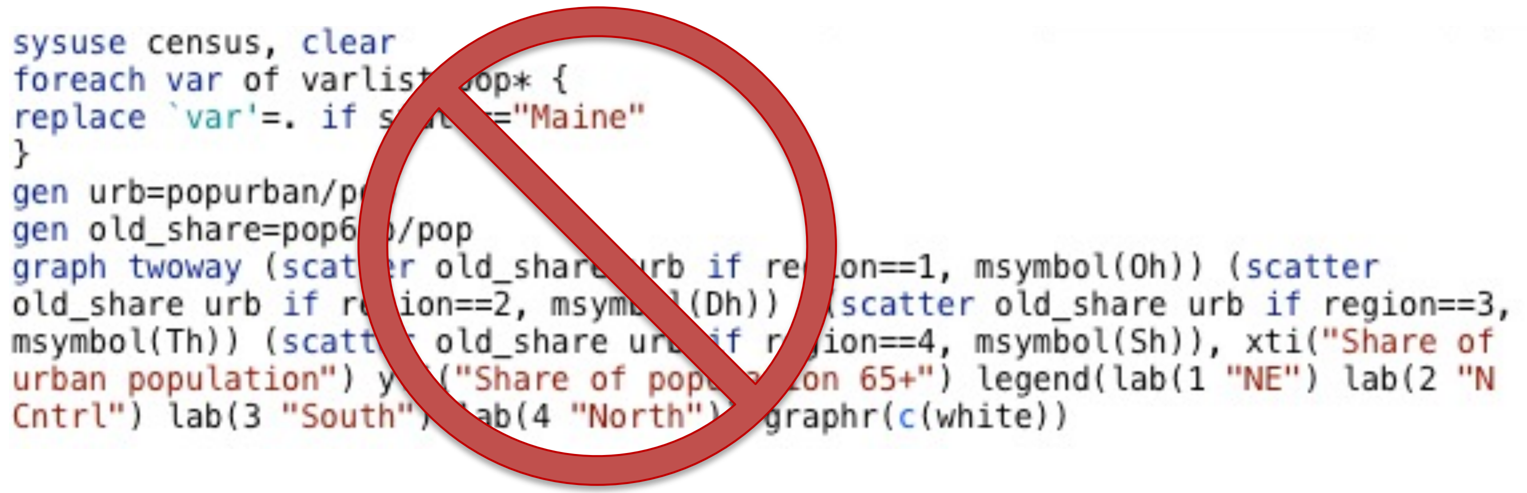
Some key principles

- Code and data should be consistent
    - ➢ Agree on standards such as naming conventions within your team
- Code and data should be as self-explanatory as possible & sufficiently commented/documented
    - ➢ Choose meaningful names for objects, comment workflow/decisions from the beginning
- Code should be as simple & short as possible
    - ➢ Only keep what is needed, structure your code

Overview

Replication and transparency

Readability of your code



```
sysuse census, clear
foreach var of varlist pop* {
replace `var'=. if state=="Maine"
}
gen urb=popurban/pop
gen old_share=pop65p/pop
graph twoway (scatter old_share urb if region==1, msymbol(Oh)) (scatter
old_share urb if region==2, msymbol(Dh)) (scatter old_share urb if region==3,
msymbol(Th)) (scatter old_share urb if region==4, msymbol(Sh)), xti("Share of
urban population") yti("Share of population 65+") legend(lab(1 "NE") lab(2 "N
Cntrl") lab(3 "South") lab(4 "North")) graphr(c(white))
```

- Structure your code to enhance readability
  - Use line breaks & indentation
  - Use * /* // for headings & comments
  - Break code into multiple lines with /// or #delimit
  - ➢ Agree with team on a style

Replication and transparency

Title: Project & title of do-file

Introduction: What does this file do?

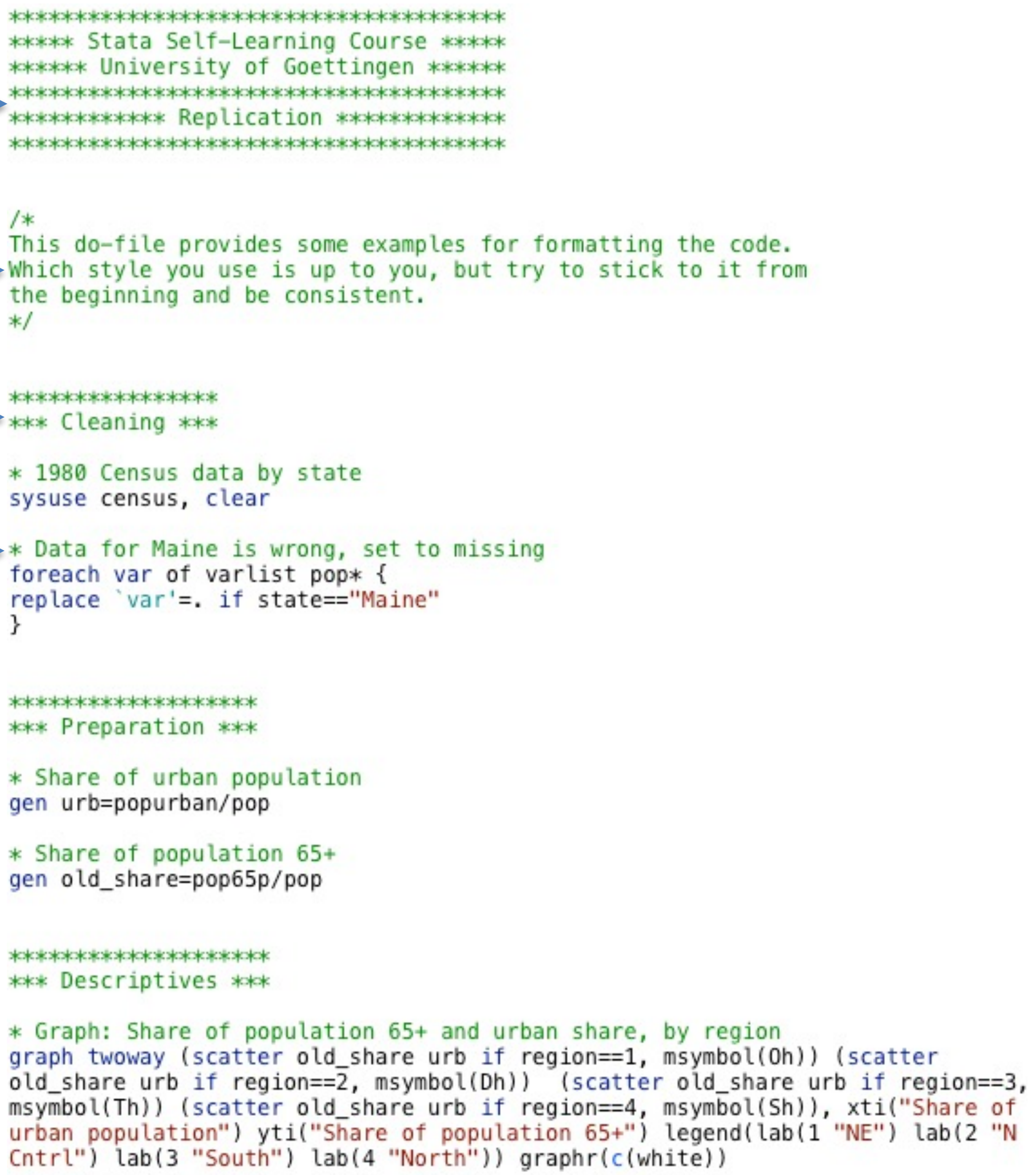
Meaningful headings to structure the file

Comment on decisions

The granularity of comments should balance between explanation of the code and readability: Which information is needed? Which congests the code?

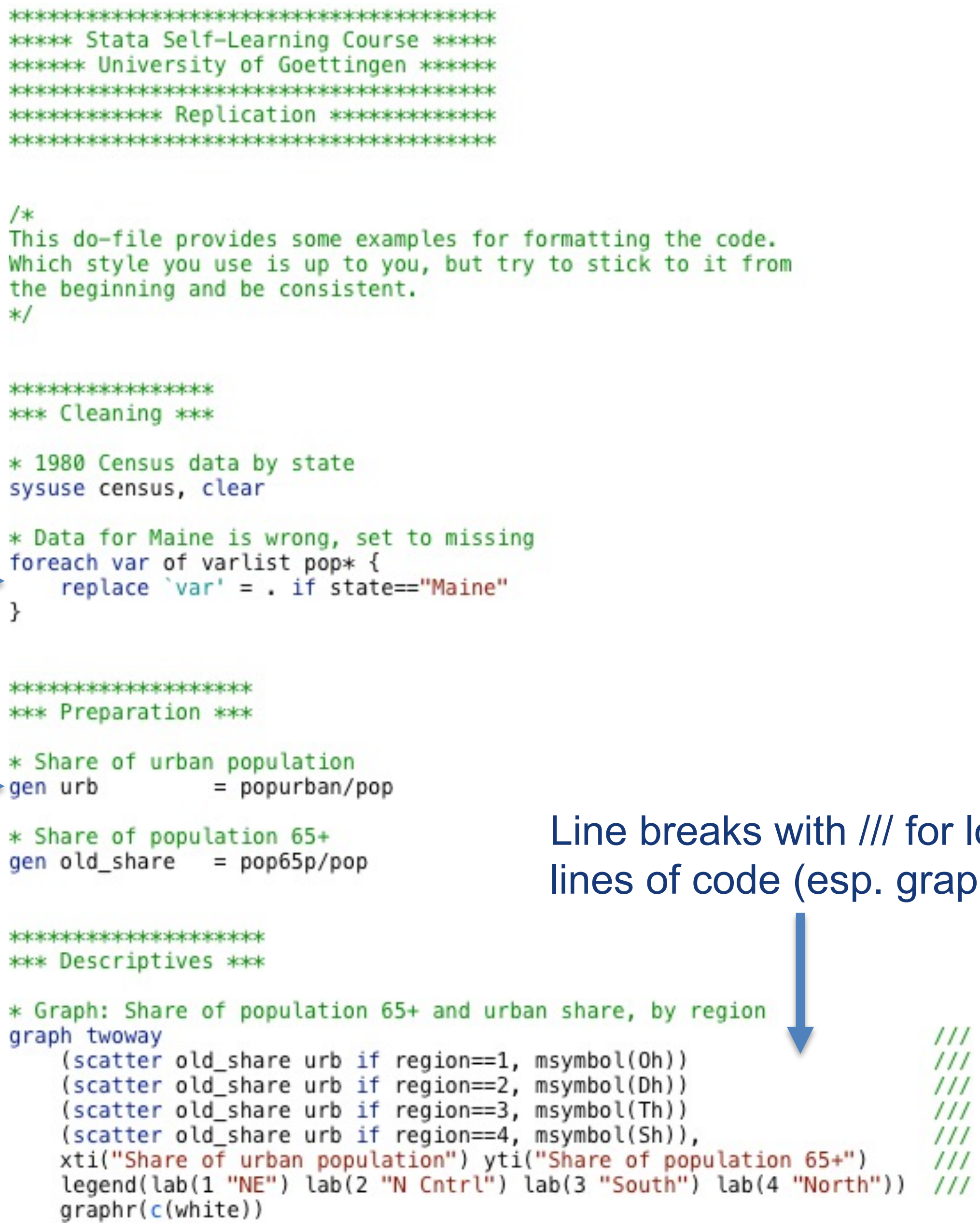
Marking open/important decisions might be useful (e.g. //!\\)

This is only an example. The exact style is not important, as long as it is clear.



```
**************************************
***** Stata Self-Learning Course *****
****** University of Goettingen ******
**************************************
************ Replication *************
**************************************


/*
This do-file provides some examples for formatting the code.
Which style you use is up to you, but try to stick to it from
the beginning and be consistent.
*/


****************
*** Cleaning ***

* 1980 Census data by state
sysuse census, clear

* Data for Maine is wrong, set to missing
foreach var of varlist pop* {
replace `var'=. if state=="Maine"
}


*******************
*** Preparation ***

* Share of urban population
gen urb=popurban/pop

* Share of population 65+
gen old_share=pop65p/pop


********************
*** Descriptives ***

* Graph: Share of population 65+ and urban share, by region
graph twoway (scatter old_share urb if region==1, msymbol(Oh)) (scatter
old_share urb if region==2, msymbol(Dh))  (scatter old_share urb if region==3,
msymbol(Th)) (scatter old_share urb if region==4, msymbol(Sh)), xti("Share of
urban population") yti("Share of population 65+") legend(lab(1 "NE") lab(2 "N
Cntrl") lab(3 "South") lab(4 "North")) graphr(c(white))
```

Replication and transparency



```
**************************************
***** Stata Self-Learning Course *****
****** University of Goettingen ******
**************************************
************ Replication *************
**************************************


/*
This do-file provides some examples for formatting the code.
Which style you use is up to you, but try to stick to it from
the beginning and be consistent.
*/


****************
*** Cleaning ***

* 1980 Census data by state
sysuse census, clear

* Data for Maine is wrong, set to missing
foreach var of varlist pop* {
    replace `var' = . if state=="Maine"
}


*******************
*** Preparation ***

* Share of urban population
gen urb         = popurban/pop

* Share of population 65+
gen old_share   = pop65p/pop


********************
*** Descriptives ***

* Graph: Share of population 65+ and urban share, by region
graph twoway                                                    ///
    (scatter old_share urb if region==1, msymbol(Oh))          ///
    (scatter old_share urb if region==2, msymbol(Dh))          ///
    (scatter old_share urb if region==3, msymbol(Th))          ///
    (scatter old_share urb if region==4, msymbol(Sh)),         ///
    xti("Share of urban population") yti("Share of population 65+")    ///
    legend(lab(1 "NE") lab(2 "N Cntrl") lab(3 "South") lab(4 "North"))  ///
    graphr(c(white))
```

Indentation for loops, if-branches etc.

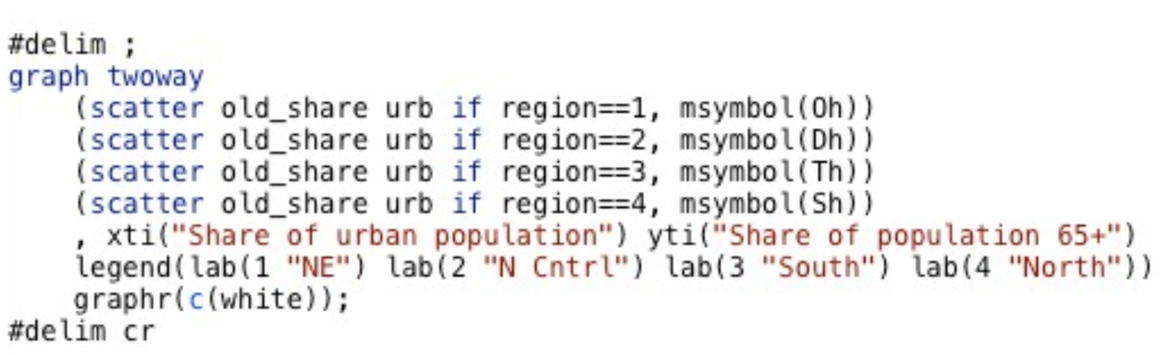
Might use tabs within a line (but too many can make it worse)

Line breaks with /// for long lines of code (esp. graphs)

This is only an example. The exact style is not important, as long as it is clear.

Replication and transparency

You can also use `#delim` to change the meaning of line breaks. Normally, line breaks mean the end of a command. With `#delim ;` the semicolon means the end of the command, and you can use line breaks for formatting. `#delim cr` changes this back to normal mode



```
#delim ;
graph twoway
    (scatter old_share urb if region==1, msymbol(Oh))
    (scatter old_share urb if region==2, msymbol(Dh))
    (scatter old_share urb if region==3, msymbol(Th))
    (scatter old_share urb if region==4, msymbol(Sh))
    , xti("Share of urban population") yti("Share of population 65+")
    legend(lab(1 "NE") lab(2 "N Cntrl") lab(3 "South") lab(4 "North"))
    graphr(c(white));
#delim cr
```

This is only an example. The exact style is not important, as long as it is clear.

Readability of your code (and data)

- Names should be descriptive/self-explanatory
  - Variables
  - Macros
  - Files

➢ Agree with team which naming conventions make sense

Example:

- Your data is based on a long questionnaire. Should variables be named after question number (q_35_2) or “title” (income_job_2)?
- The first is easier to combine with the supplementary material (and unambiguous)
- The latter is easier to memorize & recognize when coding

Readability of your data

Structure & content of the dataset should be clear:

- Use meaningful variable labels & notes (see next slide)
- Use meaningful value labels (and check their consistency)
- Use meaningful missing values where appropriate (e.g., .d for "don't know", .r for "refused" etc.)

- Order important variables such as identifiers, country names, dates/year at the top
- Check meaningful unique identifier(s)
- Provide further documentation material outside of Stata

Readability of your data

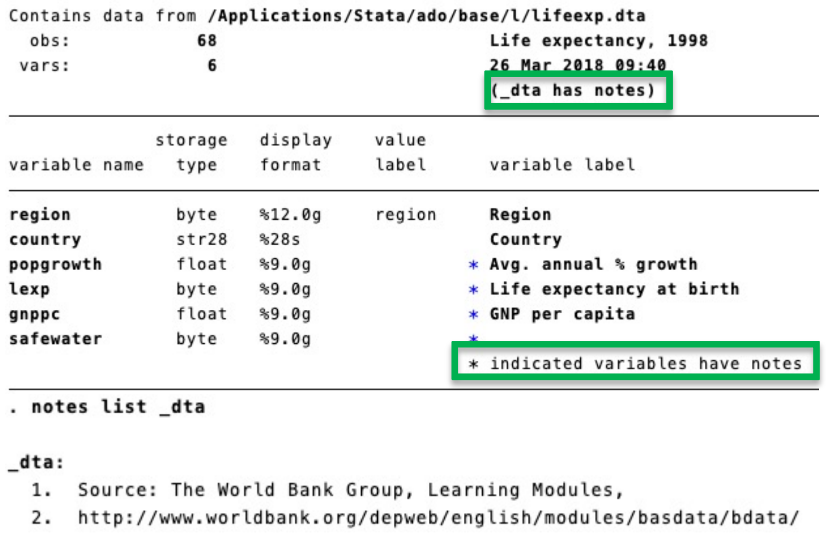
Variable labels

- Very easy to find → quick overview on variable content
- But: Might not want all information in label, as labels are used for outputs such as tables or graphs

Notes/characteristics

- Can be very detailed
- But: not everyone knows them
- Characteristics are a more advanced version of note

`help note`; `help char`



```
Contains data from /Applications/Stata/ado/base/l/lifeexp.dta
  obs:            68                          Life expectancy, 1998
 vars:             6                          26 Mar 2018 09:40
                                              (_dta has notes)
------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
------------------------------------------------------------------------------
region          byte    %12.0g     region     Region
country         str28   %28s                  Country
popgrowth       float   %9.0g               * Avg. annual % growth
lexp            byte    %9.0g               * Life expectancy at birth
gnppc           float   %9.0g               * GNP per capita
safewater       byte    %9.0g               *
                                            * indicated variables have notes
------------------------------------------------------------------------------
. notes list _dta

_dta:
  1.  Source: The World Bank Group, Learning Modules,
  2.  http://www.worldbank.org/depweb/english/modules/basdata/bdata/
```

Readability: Some remarks

- What’s considered “readable” varies immensely
- Also, there might be trade-offs between what’s considered readable & what’s practical

  For example, some propose to never abbreviate commands. That’s something I personally wouldn’t consider as a huge increase in readability as the abbreviations are so common, and I would have to exert some effort to break my habit of using them. Others find it annoying to put white spaces between “=“, while I find they increase readability. Then again, some propose to keep do-files short and rather use many do-files, while others prefer having less files.

- Always try to make your code readable for others. But: There’s no sense in setting standards if they are not followed through

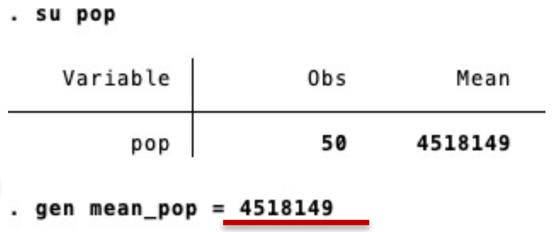
Overview

Replication and transparency

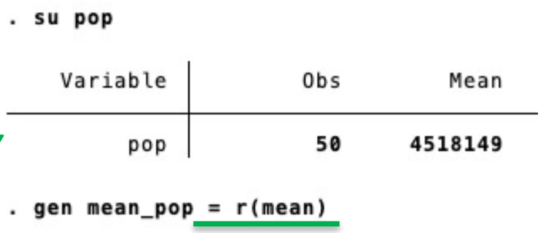
Abstraction & automation

Do everything as abstract as possible

- Try to **never ever** “hard code” values in your code
- Instead, use
  - return objects & ereturn objects & system variables

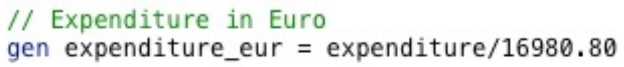


```
. su pop

    Variable |        Obs        Mean
-------------+------------------------
         pop |         50     4518149

. gen mean_pop = 4518149
```

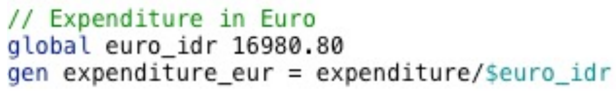


```
. su pop

    Variable |        Obs        Mean
-------------+------------------------
         pop |         50     4518149

. gen mean_pop = r(mean)
```

  - macros & macro functions



```
// Expenditure in Euro
gen expenditure_eur = expenditure/16980.80
```



```
// Expenditure in Euro
global euro_idr 16980.80
gen expenditure_eur = expenditure/$euro_idr
```

- Use automated (export) tables & graphs whenever possible
  - See chapter on advanced graphs & tables and on putdocx

Abstraction & automation

Minimize copy & paste: Definitions etc. should be done **at one point only** to prevent inconsistencies & errors

- Most obvious example: Use loops for repetitive tasks

```
foreach var of varlist pop* {
    replace `var' = . if state=="Maine"
    note `var': "Data for Maine is wrong and was set to missing"
}
```

- Use the same do-file for definitions which re-occur at different steps, e.g., creating an index at base- & endline

```
use "raw/baseline.dta", clear
    /*
    do some cleaning
    */
    run "wealth_quintiles.do"
save "prep/baseline_cleaned.dta", replace
```

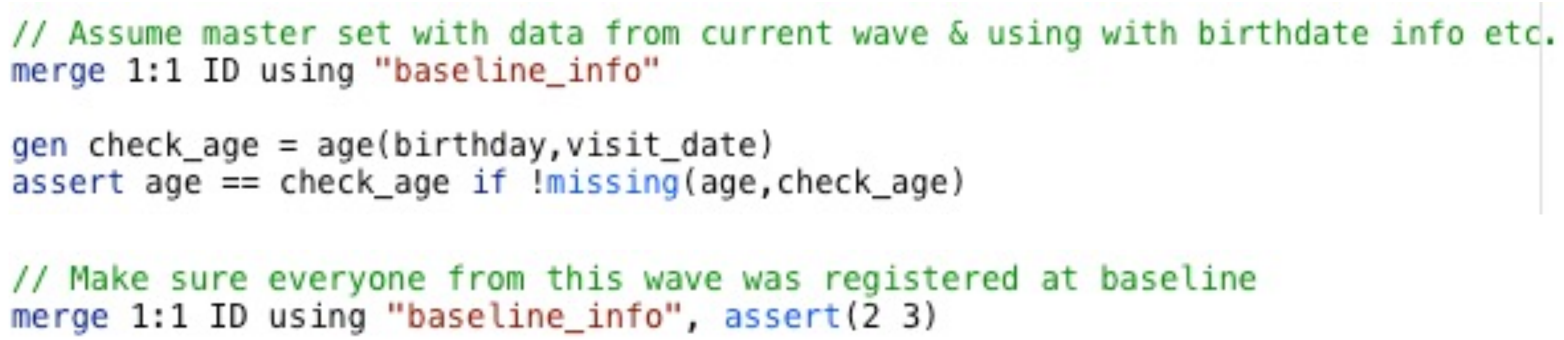
```
use "raw/endline.dta", clear
    /*
    do some other cleaning
    */
    run "wealth_quintiles.do"
save "prep/endline_cleaned.dta", replace
```

- For more complex repetitive tasks: Write programs (.ado-files)

Abstraction & automation

Use (automated) error checks to make sure everything works as intended, using for example:

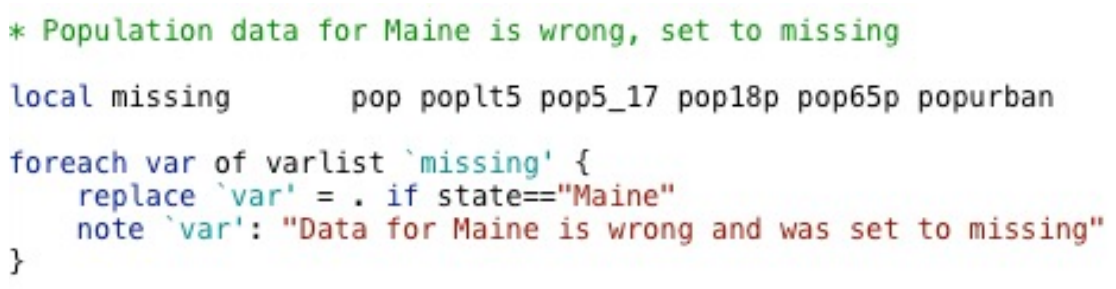
- `isid`
- `confirm`
- `assert`
- merge options



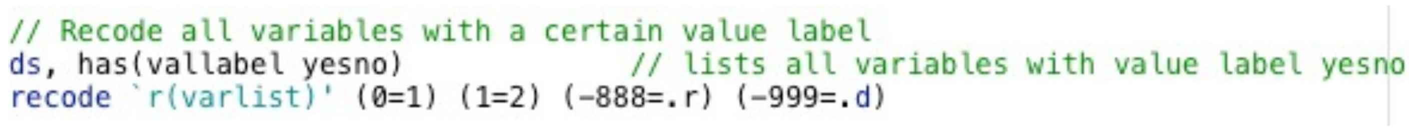
```
// Assume master set with data from current wave & using with birthdate info etc.
merge 1:1 ID using "baseline_info"

gen check_age = age(birthday,visit_date)
assert age == check_age if !missing(age,check_age)

// Make sure everyone from this wave was registered at baseline
merge 1:1 ID using "baseline_info", assert(2 3)
```

Abstraction & automation

- What to do about variable lists?
  - Can be useful if variables are consistently ordered/named
  - BUT: can also easily lead to errors if order/names change
  - Consider using macros or the `ds` command



```
* Population data for Maine is wrong, set to missing

local missing       pop poplt5 pop5_17 pop18p pop65p popurban

foreach var of varlist `missing' {
    replace `var' = . if state=="Maine"
    note `var': "Data for Maine is wrong and was set to missing"
}
```



```
// Recode all variables with a certain value label
ds, has(vallabel yesno)                 // lists all variables with value label yesno
recode `r(varlist)' (0=1) (1=2) (-888=.r) (-999=.d)
```

- Some commands allow incomplete varnames as input, e.g., “med” instead of “medage” (not to be confused with “med*”)
- This can easily lead to mistakes → use `set varabbrev off`

Overview

Replication and transparency

Folder structure

Have a clear folder structure & file system

- Separate “raw” from prepared data, inputs from outputs, etc.
- Provide a ReadMe-file in the main folder:
  - Contains all information to understand structure & run files
- Master Do-File:
  - Contains settings, globals, etc.
  - Runs all do-files in the correct order
- Recommended: Also provide data & code to build analysis dataset from “raw” (de-identified) dataset

Folder structure

Decide on a system for version control of files & documentation

- Github (e.g. https://github.com/BITSS/wb_reusable_analytics)
- OSF (https://osf.io/)
- Limited version control with owncloud
- Can use `creturn list` to capture date/user/system (see next slide)
- Can use `datasignature` to check whether data changed & `cf` to see how datasets differ

Folder structure

Directories & paths

- **Never** use the Windows „\“ in file paths! They don‘t work on Mac & Linux and cause problems when using globals!
- Two possible ways to define (flexible) filepaths:

A. Set directory (in Master do-file) & use relative filepaths

```
cd  "/Users/anna/ownCloud/Project"
use "data/raw/baseline.dta", clear
```

B. Put directory in a global (in Master do-file) & use global for absolute filepaths

```
global dir  "/Users/anna/ownCloud/Project"
use         "$dir/data/raw/baseline.dta", clear
```

- Possible ways to get the correct filepath automatically
  - profile.do (https://julianreif.com/guide/#stata-profile)
  - creturn list: c(username) (see DIME Master Do-file)
  - creturn list: c(pwd) (IPA cleaning guide)

Overview

Replication and transparency

Version control of Stata & Stata commands

- Stata version control:
  - command "`version`" to set Stata version (set to the lowest version possible to ensure widest application)
  - might use ieboilstart by DIME to also harmonize settings
- Version control of user-written commands
  - There is no automatic version control for user-written commands!
  - Save all used user-written commands in a separate folder such that others can use them in exact the same version you did
  - Run them all in the Master do-file
- Examples: Master do-file by DIME / script by Julian Reif

Overview

Replication and transparency

References & further reading

This was just a selection. You can find more examples & detailed guidelines here:

- Asjad Naqvi. The Stata Guide: Stata and GitHub Integration. Online version of PDF available at https://medium.com/the-stata-guide
- Asjad Naqvi. The Stata Guide: The Stata workflow guide. Online version of PDF available at https://medium.com/the-stata-guide
- DIME Example (including Master do-file): https://github.com/worldbank/rio-safe-space
- DIME Handbook: https://worldbank.github.io/dime-data-handbook/
- Gentzkow, Matthew and Jesse M. Shapiro. 2014. Code and Data for the Social Sciences: A Practitioner's Guide. University of Chicago mimeo, http://faculty.chicagobooth.edu/matthew.gentzkow/research/CodeAndData.pdf, last updated January 2014.
- IPA Data cleaning guide: https://povertyaction.github.io/guides/cleaning/readme/
- IPA Reproducible Research: Best Practices for Data and Code Management: https://www.poverty-action.org/publication/ipas-best-practices-data-and-code-management
- J-PAL Guide to Publishing Research Data (online version of PDF available at https://www.povertyactionlab.org/resource/data-publication)
- Julian Reif: Coding practices: https://julianreif.com/guide/