Stata style guide

Overview

Questions

- How to name variables?
- What code style do we use?

Objectives

- Use verbose, helpful variable names.
- Make your code accessible to others.

Files

Use forward slash in path names

Write `save "data/worker.dta"`, not ~~`save "data\worker.dta"`~~. The former works on all three major platforms, the latter only on Windows.

Write out file extensions

Write `save "data/worker.dta"` and `do "regression.do"`, not ~~`save "data/worker"`~~ or ~~`do "regression"`~~. Even though some extensions are appended by Stata by default, it is better to be explicit to help future readers of your code.

Put file paths in quotes

Write `save "data/worker.dta"` and `do "regression.do"`, not ~~`save data/worker.dta`~~ or ~~`do regression`~~. Both are correct, but the first is more readable, as most editors readily highlight strings as separate from programming statements.

Use relative path whenever possible

Write `save "../data/worker.dta"`, not `~~save "/Users/koren/Tresorit/research/data/worker.dta~~`. Nobody else will have the same absolute path as you have on your system. Adopt a convention of where you are running scripts from and make paths relative to that location.

Naming

Do not abbreviate commands

Use `generate ln_wage = ln(wage)` and `summarize ln_wage, detail`, not `~~g ln_wage = ln(wage)~~` or `~~su ln_wage, d~~`. Both will work, because Stata allows you abbreviation, but the former is more readable.

Do not abbreviate variable names

Use `summarize ln_wage, detail`, not `~~sumarize ln_w, detail~~`. Both will work, because Stata allows you abbreviation, but the latter is very error prone. In fact, you can turn off variable name abbreviation with `set varabbrev off, permanent`.

Use verbose names to the extent possible

Use `egen mean_male_wage = mean(wage) if gender == "male"`, not `~~egen w1 = mean(wage) if gender == "male"~~`. Your variables should be self documenting. Reserve variable labeling to even more verbose explanations, including units: `label variable mean_male_wage "Average wage of male workers (2011 HUF)"`.

Separate name components with underscore

Use `egen mean_male_wage = mean(wage) if gender == "male"`, not `~~egen meanmalewage = mean(wage) if gender == "male"~~` or `~~egen meanMaleWage = mean(wage) if gender == "male"~~`. The former is more readable. Transformations like mean, log should be part of the variable name.

Do not put units and time in the variable name

Use `revenue`, not `~~revenue_USD~~` or `~~revenue_2017~~`. Record this information in variable labels, though. You *will* change your code and your data and you don't want this detail to ruin your entire code.

It is ok to use short macro names in short code

If you have a `foreach` loop with a few lines of code, it is fine to use a one-character variable name for indexing: `foreach X of variable wage mean_male_wage {`. But if you have longer code and `X` would pop up multiple times, give it a more verbose name.

It is ok to use obvious abbreviation in variable names

If you are hard pressed against the 32-character limit of variable name length, use abbreviation that will be obvious to everyone seeing the code. Use `generate num_customer`, not ~~`generate number_of_customers_of_the_firm`~~ or ~~`generate n_cust`~~.

White space

Include a space around all binary operators

Use `generate ln_wage = ln(wage)` and `count if gender == "male"`, not ~~`generate ln_wage=ln(wage)`~~ or ~~`count if gender=="male"`~~. The former is more readable.

Include a space after commas in function calls

Use `assert inlist(gender, "male", "female")` not ~~`assert inlist(gender,"male","female")`~~. The former is more readable.

Indent code that belongs together

```
foreach X of variable wage mean_male_wage {
   summarize `X', detail
   scalar `X'_median = r(p50)
}
```

not

```
foreach X of variable wage mean_male_wage {
summarize `X', detail
scalar `X'_median = r(p50)
}
```

Each .do file should be shorter than 120 lines

Longer scripts are much more difficult to read and understand by others. If your script is longer, break it up into smaller components by creating several .do files and calling them.