# data management theory

adam okulicz-kozaryn

adam.okulicz.kozaryn@gmail.com

this version: Thursday 10th March, 2022 17:42

## outline

the golden rule

basic theory

programming principles by computer scientists

the zen of Python

# outline

# Know thyself

- old proverb; can google, see wiki at home
  - *https://www.google.com/search?q=Know+thyself*
  - `https://en.wikipedia.org/wiki/Know_thyself`
- fascinating book `http://www.hup.harvard.edu/catalog.php?isbn=9780674013827`
- but in this class, something else is even more important

# Know Your Data

- simply cannot manage it well if you don't know it well
- again, be prepared to invest a lot of time into your data
  - use data that either is of your interest
  - or that can make $ (say use in future career)
  - or ideally both!
- and use descriptive stats
  - des sum tab edit list inspect, and especially graphs!
- think about it! don't be mindless!
  - ask questions, be investigative
- double check, cross check, give to others to check

# the silver rule

- keep it as simple as possible
  - especially if overwhelmed or struggling
- say retain only 10var and 100obs
  - much easier to manage such data!

# the three key rules

- simplicity transpancy clarity:
  - use fancy code: macros, loops and ados iff they simplify
- have chunks of code only once
  - use root .do, macros, loops, ados to accomplish that
- code it all from raw to final (replication principle)

## all rules in simple words

- the fancier the code, the more time/effort to write it
- don't do fancy things unless they save time in the long run
- it's all about managing complexity
- automate as much as you can
- simplify and be clear
- have general modules (sections or separate dofiles)
  - that can be reused for different projects
- don't reinvent the wheel–google often

## things usually overlooked

- have chunks that you do not use but may be useful (commented out)
- clarity and logical organization; clear sections

# outline

## accuracy or correctness

- it's fundamental and obvious: code cannot be wrong
- we'll cover some commands/tricks (eg `assert`)
  - to make sure stata did what you think it did
- the bottom line and best advice:
  - double check (if not 100% sure or always for rookies)
  - especially at the beginning do not assume things
  - double/triple check the whole dofile once finished
  - use as much des stats as possible

## efficiency: few lines of code do many things

- efficiency==programming (macros, loops, ados)
- but also think how you can optimize your code
  - do more in fewer lines, drop unnecessary things
- reorganize and rewrite!
  - just like your papers: you print them out
  - and move paragraphs and words around
  - and you simplify and strike out unnecessary words
- do the same with code! drop everything you can!
- code should be "tight"
  - as few lines as possible to perform given task

## efficiency: on the other hand

- but you also want to be extensive in a way
- in a good way...
- like with free writing, so with code
  - do "free writing"
- be expressive and dump your ideas into dofile
- just be organized so that you know what is going on!
- yes, by all means, be efficient–drop unnecessary things
- but do not drop things that may be useful
  - say in the future or other projects
  - may comment them out (useful!)

# rewrite/revise

- do “free writing” with code, too (i often come up with some idea out of sudden, and then just write it down...)
- start simple and keep on adding things
- rewrite/revise your code
- improve, add, modify, optimize
- (there is often a tendency to over optimize, i.e. spending weeks on small chunk of code that does not really matter that much)

## simplicity: different, often opposite, from efficiency

- people don't realize this!
- be as simple as possible in writing the code (papers, too)
- the more code you have and the more complicated it is:
  - the more likely you have mistakes
  - and the more difficult it is to find them
- do not complicate your code for the sake of fanciness
  - yes, we do it all the time! don't do it! simpler is better

## standardization (see my template organize.do)

- standardization helps to make fewer mistakes
  - and make your code more transparent
- whole research process should be standardized; eg:
  - have the same style for graphs, tables (more later)
  - have the same tables of descriptive statistics
- you should have a template for a dofile (and for a paper)!
  - why waste time on tedious boring sections and parts
  - you could use your time on creative and fun parts instead!
  - research production is like car production
  - don't do everything by hand every time!

# modularity

- break large tasks into small (manageable) blocks/components
  - (like in dissertation–don't overwhelm yourself doing everything at once)
- the components are like sections in a paper, step-by-step
- it is easy then to reuse these components

# automation (closely related to standardization)

- everything should be coded
- no copy-paste, point-and-click, etc
- automate as much as possible!
- practical reason: much faster!
- technical reason: computers *never* make mistakes
- programming (macros, loops) help a grade deal

# documentation

- you may want to have notes...but mostly:
- documentation is just about having a commented dofile
- difficult to overestimate the dofile comments
- note, typically, i undercomment, too

## singularity

- as discussed in organization and documentation class:
  - have only one chunk of code and one file in one place
- this principle is often overlooked
- LaTeX (now even ms word) and html with css do it:
  - take out the (common) formatting
- do a similar thing in dofile: take out the common code
- otherwise, it's inefficient, and leads to errors
- take out the common code and put into common
  - (root or parent) dofile
- make programs (.ado) (more later)

## singularity example

```
<font size=2 face="Helvetica" color=red>formatted text
</font> regular text <font size=2 face="Helvetica"
color=red> formatted text again</font>


----
aokTag1{font size=2; face="Helvetica";color=red;}
-----

<aokTag1>formatted text<aokTag1> regular text
<aokTag1>formatted text again</aokTag1>

%% then you can just change tag definition and all
 intsatnces in 150 files are changed automatically !
```

# portability

- your code should run easily on other computers
- say version 14
- use macros for paths
- always install needed packages
- say where data come from and load from url
- usually repost on your site, say goog drive
  (data at source may change)

## tradeoffs: life is not so simple

- simplicity is sometimes inversely related to efficiency
  - say in programming (loops, macros, ados)
- simplicity is usually inversely related to automation
- so make some choices
- the more serious you are about coding
  - the more you should care for automation and efficiency
- the more data management you do
  - the more automation/efficiency actually simplifies
- like stata v excel: excel simpler for simple tasks
  - but stata is simpler for complicated tasks

## a matter of style

- apart from all these rules, different people have different styles of programming
- just use whatever you like–a matter of taste
  - eg i do not use global macros (i work on linux), you may find them useful on windows
  - i use foreach loops, but not while loops
  - i have few big dofiles, but why not have many small ones ?
- still, all dofiles must be clear and replicable

# outline

the golden rule

basic theory

programming principles by computer scientists

the zen of Python

## intuition

- it occurs to me that this class really is more like computer science than social science
  - CS have classes about c, python, etc.
- we have a class about stata
- but we still do programming, just in different language
  - so i've read actual computer science lit
  - and what i found useful is in this section
  - great reference!
  - essp Box 1 Summary of Best Practices–let's see it!

`http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1001745`

## more principles

- some more programming principles follow
- these are rather general programming principles
- they are applicable to any programming,
  not only stat software; e.g. c, python, php, etc.
- yes, there is some repetition/reformulation of the earlier rules
  - but these are really important, so doesn't hurt to repeat
- these principles come from 2 books about general programming (classics and free!)

  `http://catb.org/esr/writings/taoup/`

  `http://www.htdp.org/2003-09-26/Book/curriculum-Z-H-1.html`

  and free mit courses *http://ocw.mit.edu/courses/*

## clarity

- "design for transparency and discoverability"
  - write clean code
  - avoid fancy code
  - fancy code is buggier
  - clarity is better than cleverness
- eg:
  - group logical chunks together
  - more than twice nested loops gets confusing
  - if your code is mostly loops and macros, consider ado file

## modularity

- "write simple parts that are cleanly connected"
- "controlling complexity is the essence of computer programming"
  - debugging dominates development
- eg:
  - better many small loops that each do one thing than one huge (>100 lines) loop that does everything
  - clear sections of one dofile
  - or many dofiles instead of one dofile without sections

# modularity

- code should be organized logically not chronologically
  - do free writing, but then reorganize
  - like with papers, code should be rewritten, eg:
  - no data management in data analysis part
  - move "generate, recode" to the beginning

# composition

- "design programs to be connected to other programs"
- dofile will produce output for another dofile
- eg: you clean up data in one dofile to make data ready for another dofile to analyze it
  - or just have one big file
- but the workflow needs to be logically organized
  - use master dofile if many dofiles

## optimization (fancier, fewer lines)

- yes, but "get it working before optimizing" !
- eg:
  - recode data using simple commands
  - then make it into macros
  - then into loops
  - then into ado
- if you are advanced you may skip some steps
  - but make sure it is time efficient
  - do not spend hours on fancy loops for sake of fanciness
  - (hours spent on ado files are fine because you will reuse them in the future)

# extensibility

- "design for the future because it will be sooner than you think"
  - you will reuse your code in the near future
  - so write it clean
  - have sections, etc
  - use lots of comments
  - reorganize, rewrite
  - optimize

## silence

- "when a program has nothing surprising to say, it should say nothing"
- drop unnecessary code
  - if you think it may be useful in the future comment it out, or better yet commit in git and delete
- do not generate unnecessary output, do not lose your reader in unnecessary clutter, eg use silently
  - eg: do not present all the descriptive statistics that stata produced
  - only the meaningful output
  - if the output has nothing to say it should be dropped (or commented out)

## automation (again)

- "rule of generation: avoid hand-hacking"
- because humans make mistakes and computers don't, computers should replace humans wherever possible
- automate anything that you can
- your data management/analysis is repetitive and involves few if...then...
  - write a program that can do it and do more creative tasks instead
- don't assume things... use confirm and assert
- write ado programs – they are not that difficult
- write other programs – start with python or bash

## save time: reuse, don't reinvent the wheel

- if someone has already solved a problem once, reuse it !
- it is very unlikely you are doing something completely new
- if anything, the problem is that people do not share their code
- usually all you need to do is to adjust somebody else's code or your old code

## save time: reuse, don't reinvent the wheel

- ask people for code:
  - your supervisor
  - journal article authors
  - your colleagues, friends, etc
- share your code
  - you may want to protect some parts of it
  - (critical, innovative research ideas, etc)
  - but share as much as possible
- acknowledge others' work–then they will be happier to share

# defensive programing

- ◇ "people are dumb-make program bullet-proof"
  - you will find negative income, age over 200, people change gender over time etc...
  - numbers saved as strings, etc
- ◇ think of all possibilities/instances; especially if you suspect some specific problems...
  and make your program bullet-proof, e.g.:
  - confirm numeric variable price
  - assert sex == 0 | sex == 1

# construct functions

- construct your own functions
  in stata these are called ados
- especially if you have lots of code (>1k lines)
  - write functions (new primitives) to perform common tasks
- then a bunch of your code will be your functions
- and you will be calling (using) them to manipulate your data

# outline

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.

- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one– and preferably only one –obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.