Lecture 2
Reproducibility

Pierre Biscaye
Université Clermont Auvergne

Data Science for Economics

Note: Materials for this lecture are drawn from Ted Miguel's Development Economics course at UC Berkeley

Agenda

1. Overview of reproducibility and transparency
2. Organizing files
3. Organizing code
4. Coding transparency and portability
5. Writing code in Python
6. More Python basics

Key references

- Casey, Glennerster, and Miguel. (2012). “Reshaping Institutions: Evidence on Aid Impacts Using a Pre-analysis Plan”, Quarterly Journal of Economics, 127(4), 1755-1812.
- Miguel et al. (2014). “Promoting Transparency in Social Science Research”, Science, 10.1126/science.1245317.
- Christensen and Miguel. (2018). “Transparency, Reproducibility, and the Credibility of Economics Research”, Journal of Economic Literature, 56(3), 920-980.
- Ferguson et al. (2023). “Survey of open science practices and attitudes in the social sciences”, Nature Communications, 14.
- Christensen, Freese, and Miguel. (2019). Transparent and Reproducible Social Science Research: How to Do Open Science, University of California Press.

Threats to validity of research

- **Fraud**: undermines public trust in science
  - Open data and code can help uncover
- **Publication bias**: missing studies/ "file-drawer" problem
  - May lead to wasted research effort, misguided policy decisions
  - May incentivize author manipulation/"p-hacking"
  - Pre-registration can reduce scope for manipulation, promote publication
- **Failure to replicate**: within study (reproduction) and across settings (replication)
  - Increasing journal data posting requirements
  - Difficulty of getting funding or publishing replications of studies

What do transparency and reproducibility mean in economics research?

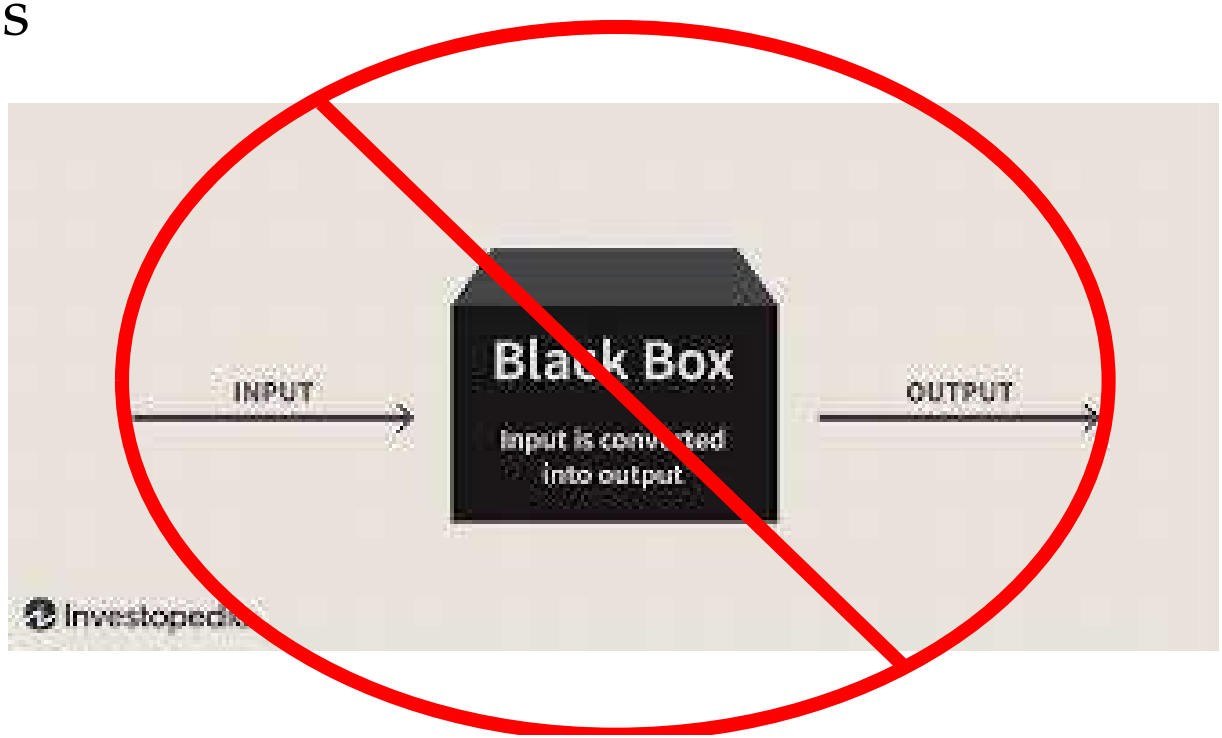
- **Transparency**:
  - Ensuring that all data, methods, and analyses are openly shared and clearly documented, allowing others to understand and evaluate the research process.
- **Reproducibility**:
  - The ability of others to replicate the results of a study using the original data and code provided by the researchers.
  - Critical for validating findings, building trust, and advancing knowledge.
- **Benefits**:
  - Results that can be verified and shown to be largely free of investigator bias are more convincing.
  - More publication of null or controversial results to broaden policy discourse.

Key principles of reproducibility

- **Accessible Data**: Provide well-documented, cleaned, and publicly available datasets (where ethically and legally possible).
- **Open Code**: Share analysis scripts and code in a version-controlled repository (e.g., GitHub) with clear instructions for execution.
- **Clear Methodology**: Document all steps in data collection, preprocessing, and analysis to ensure clarity and consistency.
- **Replication Workflow**: Design workflows that allow others to re-run analyses seamlessly, ensuring compatibility across systems and tools.

Takeaways for this course

- The quality and clarity of the research process matters
- Focus of this lecture: organizing and documenting research methods



Organizing files

The first step to a clear research process is file organization. There should be a clear structure to your folder hierarchy and file names.

```
README.pdf
data/
   raw/
      cps0001.dat
   analysis/
      combined_data.dta
      combined_data.csv
      combined_data_codebook.pdf
code/
  01_create/
      01_readcps.R
      02_readfred.R
  02_analysis/
      01_table1-5.R
      02_figures1-4.R
results/
  table1.tex
  table2.tex
  ...
  figure1.pdf
  figure2.pdf
```

Source

Example: Jigawa Floods Project

Basic structure:

1. Code
    1. High-frequency checks for data collection
    2. Analysis: separate scripts for different tasks
2. Data
    1. Raw
    2. Clean
3. Output
    1. Figures
    2. Maps
    3. Tables

Example: Folder of data sources

1. Sub-folders for main data sources/data types
2. Clear organization within folders
3. Example: LSMS-ISA
    1. Country sub-folders
    2. Year/round sub-folders
    3. Data, Questionnaires, Resources sub-folders
    4. Zipped raw data and unzipped folders
        1. Always keep a copy of the original data as a backup!
4. Readme documents
    1. Notes on where/when data were accessed
    2. Example: ARES data

https://databank.illinois.edu/datasets/IDB-1107366

Retrieved 2/14/23

1. ARES, crop-specific exposure to temperature and moisture shocks
    a. 0.25 degree cells by year by crop
    b. 1961-2014
    c. /Users/pierrebiscave/Dropbox/Data/Spatial/ARES
    d. .nc files with crop layers

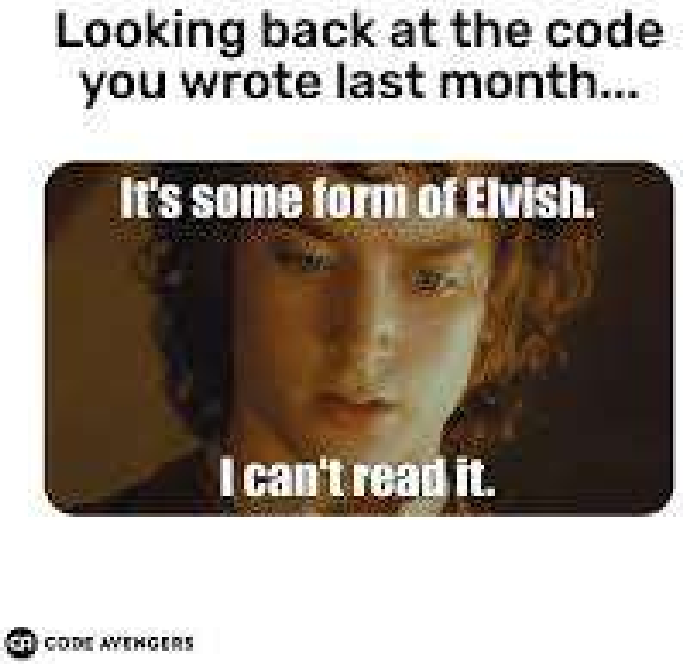
File organization in this class

- Folders for each section
- Subfolders for data, images
- Separate Jupyter Notebooks for different topics
- Clear file naming conventions

Reproducibility of code

- Comment and document thoroughly
- Use modular design
- Adopt consistent naming conventions
- Version control
- Ensure code is portable

This is important for both your future self and for potential collaborators and reviewers!



Comment and document thoroughly

- Include clear comments in your code to explain the purpose of each section and the logic behind key steps.
- Best practice: Use a README file to provide an overview of the project, dependencies, and how to run the code.

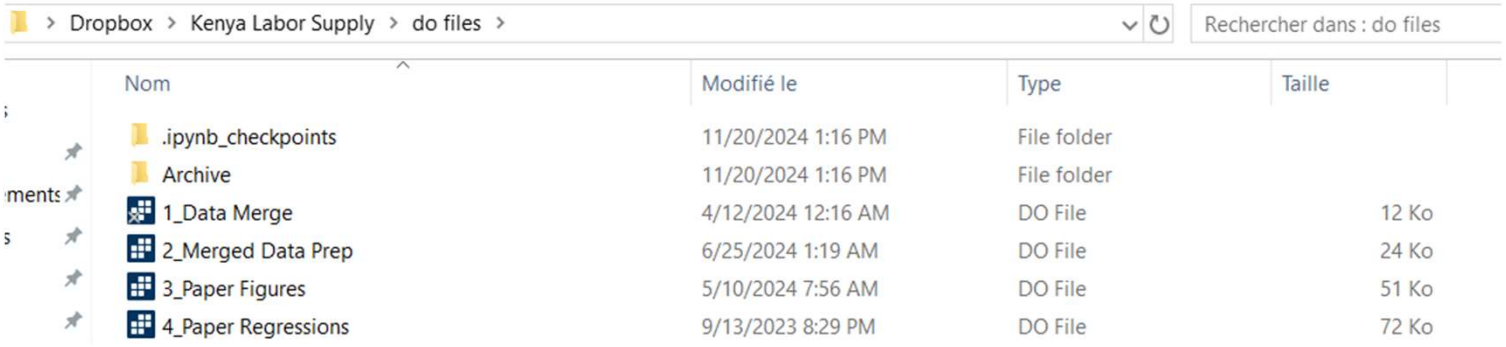
Source

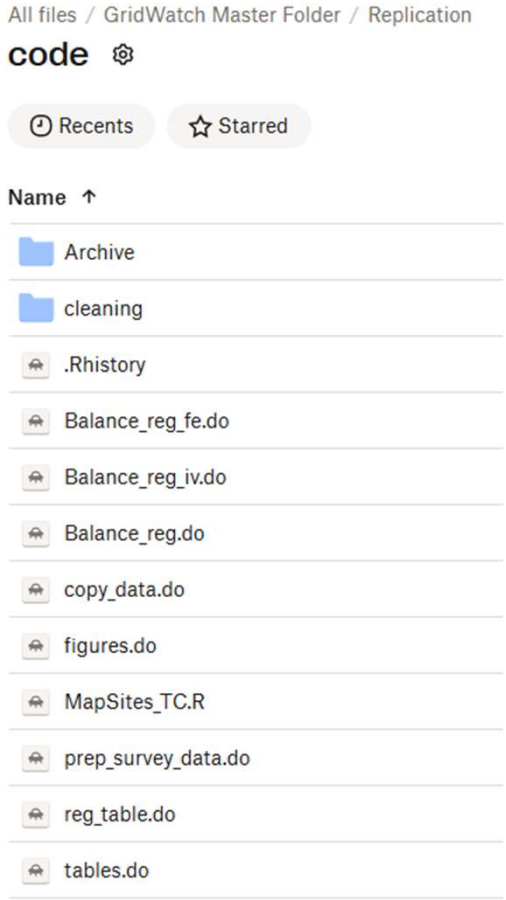
Commented code

```
//
// Prep spatial datasets
//

* Locust Swarms
{
* Data from FAO Locust Hub, coordinates and dates of locust swarm observations globally from 1985-2023
* Some additional detail included there but lots of missing data so focus just on swarm location and timing
import delimited "$data/Locust Hub/Retrieved 2.13.23/Swarms.csv", encoding(UTF-8) clear
drop if objectid==.
drop if missing(x) | missing(y)
destring x, replace
destring y, replace
gen double lat=round((y+0.125)*4)/4 - 0.125
gen double lon=round((x+0.125)*4)/4 - 0.125
gen date = date(substr(startdate,1,10), "YMD")
format date %td
gen year=year(date)
gen month=month(date)

* Output for mapping
preserve
keep x y year
export delimited "$clean/mapping_swarms.csv", replace
restore

* Match to countries, identify countries in Africa and Arabian Peninsula with at least 10 swarms in analysis period
1996-2018
geoinpoly y x using "$data/Country boundaries/Country raw/UIA_World_Countries_Boundaries/WORLD_coor.dta"
merge m:1 _ID using "$data/Country boundaries/Country raw/UIA_World_Countries_Boundaries/WORLD_data.dta", keep(1 3)
gen swarm=1 if year>1995 & year<2019
egen swarms_1996_2018=sum(swarm),by(COUNTRY)
bys COUNTRY: gen first=_n==1
br COUNTRY swarms_1996_2018 if swarms_1996_2018>10 & first==1
* Well over 10: Algeria, Burkina Faso, Chad, Egypt, Eritrea, Ethiopia, Gambia, Guinea, Guinea-Bissau, Libya, Mali,
Mauritania, Morocco/Western Sahara, Niger, Oman, Saudi Arabia, Senegal, Somalia, Sudan, Tunia, Yemen
* Cabo Verde 42, India 127, Iran 19, Israel 11, Kenya 22, Pakistan 108 also meet the swarm count criteria but outside
area of interest or are borderline cases (Israel, Kenya)

* Set target geographic area to trim other datasets, based on countries to target
drop if lat<-2.5
drop if lat>37.5
drop if lon<-17.5
drop if lon>60.25
```

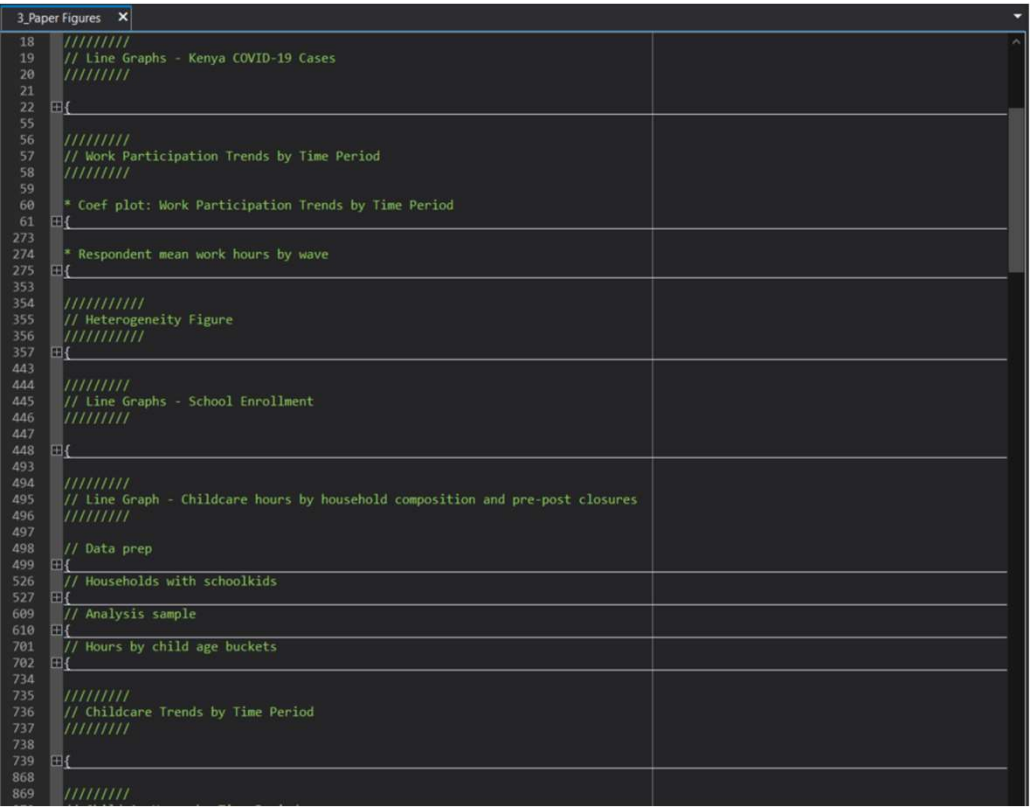
Use modular design

- Break the code into smaller, reusable functions or scripts to make it easier to debug, update, and understand. Separate data cleaning, analysis, and visualization steps logically.

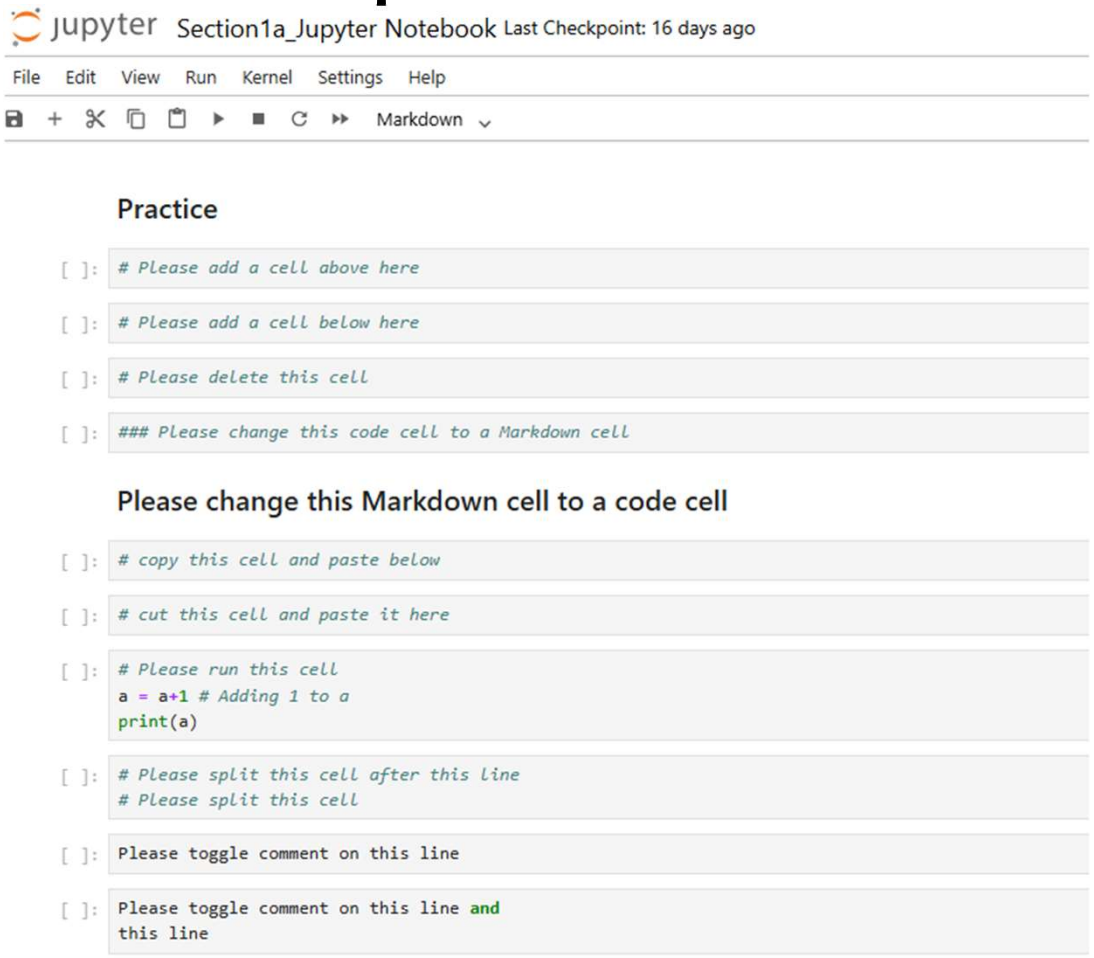




Modular design within scripts



Use clear headings and labels to organize your code



Jupyter Notebook is built to be modular by default

Adopt consistent naming conventions

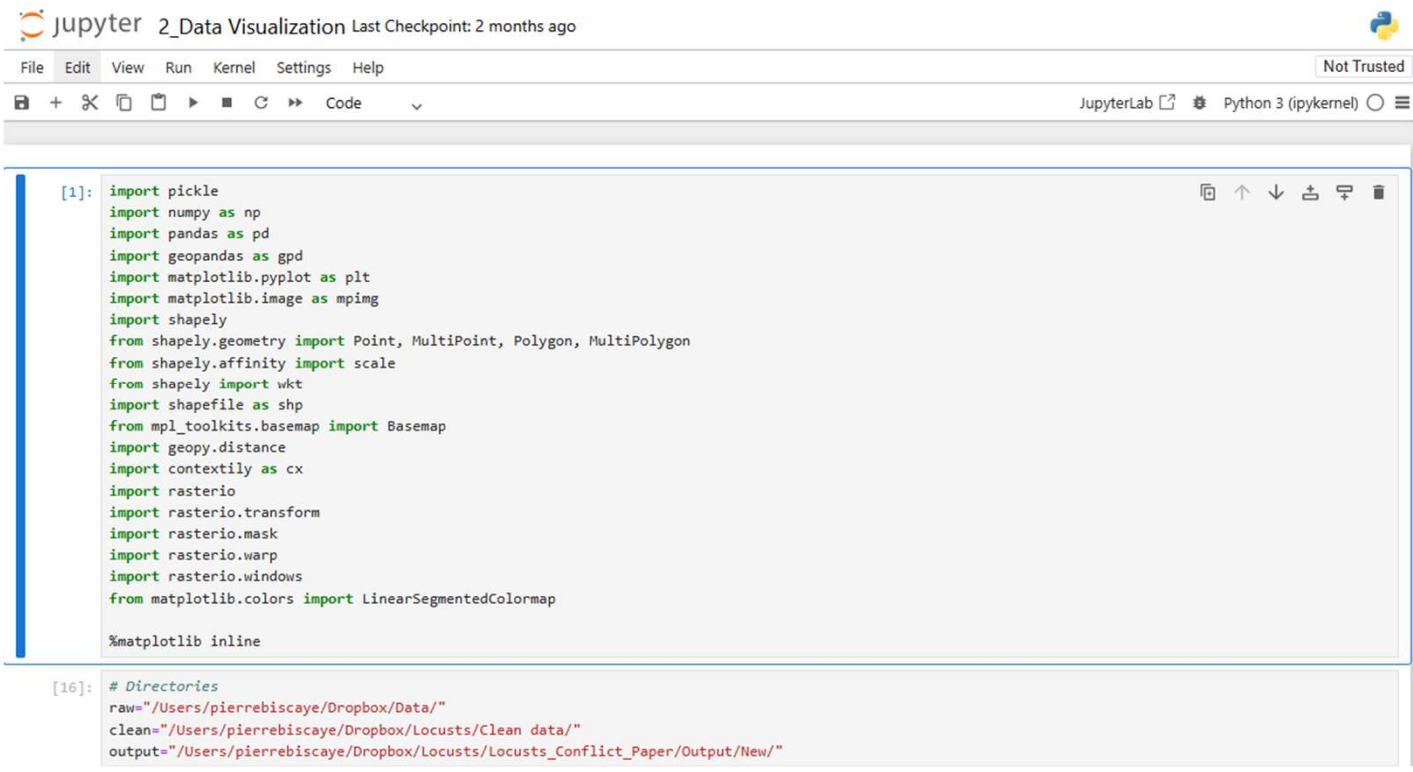
- Use meaningful, consistent names for variables, functions, and files to enhance readability and reduce confusion.
- For example, use clean_data() instead of cd() for function names.

Version control

- Use version control systems to track changes in your code and collaborate efficiently.
- Commit changes with descriptive messages and maintain a well-organized repository structure.
- Gold standard: GitHub repository
- Minimum: clear files names indicating version history, well-structured archive folder
  - Back up your code, don't just always overwrite

Ensure code is portable

- Avoid hardcoding file paths or machine-specific dependencies.
- Use relative paths and specify software environments to ensure others can run the code seamlessly.
- Often useful to have a "0_setup script" with paths and packages to run first.



Writing Python code

- Into Jupyter Notebooks!