Meeting 1 – Introduction and reproducible research

Glenn Magerman
Spring 2020

Learning objectives

- **Learning outcomes**
  - learn the course framework
  - how to work on the course projects
  - what is reproducible research and why it matters
  - data and code organization basics

- **Concepts**
  - **reproducible research:** data preparation, reproducibility, replication, scientific method, research pipeline, literate statistical programming,
  - **code etiquette:** folder management, automation, version control, relational keys, abstraction, code documenting, task management, code etiquette and style.

- **References**
  - Short course on reproducible research from Johns Hopkins
  - Gentzkow and Shapiro's Code and Data for the Social Sciences: A Practitioner's Guide

Learning objectives

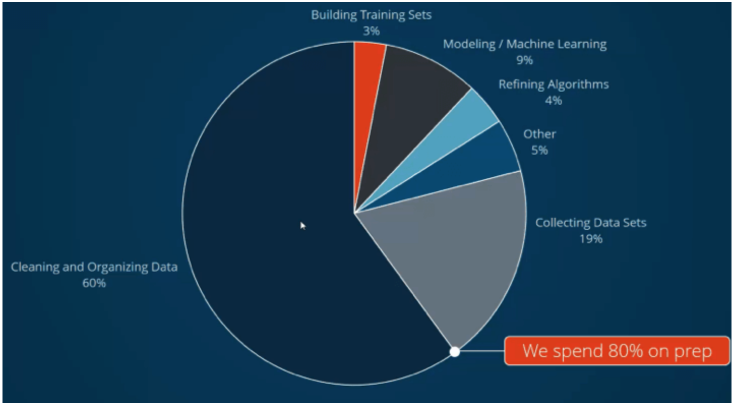
- **The content in this meeting is straightforward**
  - no coding today
  - nothing is new
  - hopefully at the end you say "this is all very, very obvious"
  - but makes a huge difference if not implemented yet

Overview

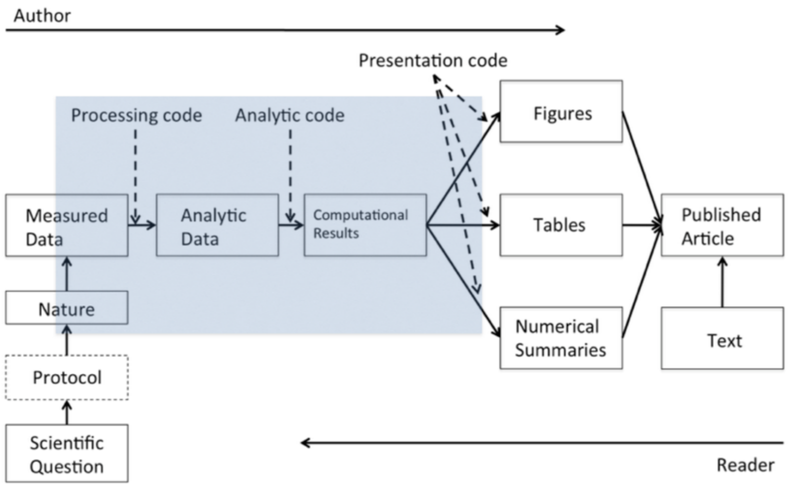
Why this course?

- **Data preparation**
  - I spend 60-80% of my time on data handling/cleaning before analysis
  - similar estimates in institutions and companies
  - important step, but time consuming and not all that fun

Data Science is 99% preparation, 1% misinterpretation – Big Data Borat



Why this course?



Why this course?

- *"In trying to replicate the estimates from an early draft of a paper, we discover that the code that produced the estimates no longer works because it calls files that have since been moved. When we finally track down the files and get the code running, the results are different from the earlier ones."*
- *"In the middle of a project, we realize that the number of observations in one of our regressions is surprisingly low. After much sleuthing, we find that many observations were dropped in a merge because they had missing values for the county identifier we were merging on. When we correct the mistake and include the dropped observations, the results change dramatically."*
- *"A referee suggests changing our sample definition. The code that defines the sample has been copied and pasted throughout our project directory, and making the change requires updating dozens of files. In doing this, we realize that we were actually using different definitions in different places, so some of our results are based on inconsistent samples."*
- *"We are keen to build on work a research assistant did over the summer. We open her directory and discover hundreds of code and data files. Despite the fact that the code is full of long, detailed comments, just figuring out which files to run in which order to reproduce the data and results takes days of work. Updating the code to extend the analysis proves all but impossible.* ***In the end, we give up and rewrite all of the code from scratch****."*

Why this course?

- **We want to ask good questions, analyze and report our findings**
  - but most of the time we write code
- **The cost of bad data/code is extremely high**
  - wrong predictions
  - longer model development
  - non-convergence of models/algorithms
  - non-robust results
  - garbage in, garbage out: 75% C-level exec not confident in their data quality (Trifacta, 2020)
  - models are useless at best, detrimental at worst, with bad quality data
- **We write code for a living, but most of us are not trained**
  - not learned in standard courses
    (mostly start from nice exercise datasets)
  - we learn on the fly, through shame, and sometimes from peers
  - projects get dumped after several years of work, people quit their jobs

Why Python?

- **What is Python**
  - high-level programming language
  - developed by Guido Van Rossum (1989)
  - open source contributions
  - official language at Google
  - principles of elegance, simplicity and readability (The Zen of Python)
- **Why Python**
  - fast growing programming language
  - free, open source and pre-installed in Linux/Unix OS
  - 3rd most popular, predicted to overcome Java and C in a few years
  - efficient coding (vs Java and C)
  - huge community and backing
  - great for beginner programmers, used for teaching
  - general purpose: not only statistics (R) or web development (PHP)
  - ages well: flexible and evolves with new technologies (internet, AI, ...)
- **Some cons**
  - high memory usage, careful coding required
  - multithreading to be implemented through packages

Course overview

- **Goals of the course**
  - collection and analysis of large and/or live datasets
  - reproducible research
  - raising standards for coding practices
  - joint learning

- **The course has 3 key components**
  - Code etiquette: transparent, standardized, proper coding
  - Python: data collection, preparation and handling (software)
  - GitHub: version control, repository, collaborative tools (infrastructure)

- **A big thank you to Fabrizio and Federico for co-organizing this!**

DISCLAIMER: I'm learning with you as we go along.
More importantly: do as I say, not as I do!

Organization

- **12 meetings**
  - M1: Introduction and reproducible research
  - M2: GitHub
  - M3-M5: Python basics
  - M6-M7: Collecting data (API, scraping)
  - Easter break (larger project)
  - M8: project discussion
  - M9-10: Scientific analysis
  - M11-M12: additional topics (visualization, introduction to ML)
- **This course is NOT**
  - regressions in Python
  - a machine learning course
  - application/field specific

Practicalities

- **All content embedded in GitHub**
  - https://github.com/Python-do-ECARES
  - learning by doing (use GitHub and learn GitHub)
  - learn from each other (upload and share code)
  - learning curve, break even in around 2 months
- **2 people lead each meeting**
  - email me to lead a meeting
  - summarize and present content
  - go through exercises
  - discuss problems, alternative methods, best practices etc.
- **Everyone prepares content/exercises before next lecture**
  - upload code to GitHub
  - need to get your hands dirty to learn

Overview

What is reproducible research?

- **Think of a symphonic orchestra**
  - orchestral pieces have a score: all parts + instructions on how to play
  - reproducible research is how to develop the score for data analysis
  - so other people can reproduce ("perform") the analysis you have done
- **Generally, reproducibility is**
  - making available data and code used in the study
  - obtain the same results
  - also outside research: processes to preserve something so someone else in the organization can repeat what you did
- **But there is no agreed upon notation system for communicating data analysis**
  - it is done differently by everyone
  - its definition is different for different people

Reproducibility and the scientific method

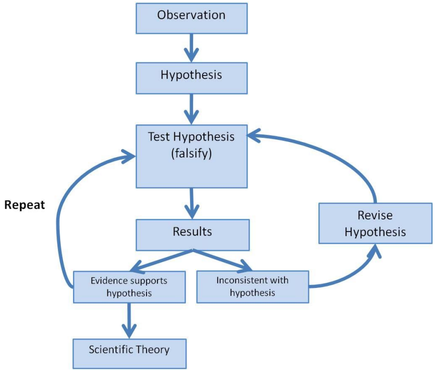
- **Reproducibility**
  - *"closeness of the agreement between the results of measurements of the same measurand carried out with same methodology described in the corresponding scientific evidence"*
  - i.e. use the authors' data and methods and get the same results.

- **Replication**
  - *"ability to independently achieve non-identical conclusions that are at least similar, when differences in sampling, research procedures and data analysis methods may exist"*
  - i.e. apply the same method on other datasets or similar methods on the same dataset and get similar results.

(Note: what many people call replication studies are actually reproduction studies)

Reproducibility and the scientific method

- **Reproducibility/replication are main tools of the scientific method**
  - Boyle (1660s): "by repeating the same experiment over and over again, the certainty of fact will emerge"
  - Karl Popper (1934): "non-reproducible single occurrences are of no significance to science"
  - dogma in modern science that reproducibility is a necessary condition (not necessarily sufficient) for establishing a scientific fact


Observation
Hypothesis
Test Hypothesis (falsify)
Repeat
Results
Revise Hypothesis
Evidence supports hypothesis
Inconsistent with hypothesis
Scientific Theory

The replication crisis

- **Reproducibility is crucial for many types of research**
  - leading theories
  - policy impact, many stakeholders
- **Awareness since 2010s**
  - many scientific studies cannot be replicated or reproduced
  - most severely in medicine and social sciences
  - publication bias
  - major policy at leading journals
  - battling fake news
- **Why can studies not be replicated?**
  - expensive to set up experiment again (e.g. follow people over 20 years, CERN)
  - very large datasets and large required computing power (e.g. HPC)
  - in large datasets, even impossible to get a grip on all dimensions of the data. how to trust the data?
  - proprietary data
  - unique timing, population or event (external validity)
  - no time: publish-or-perish pressure

The replication crisis

- **Nature (2016): poll of 1,500 scientists**
  - 70% failed to reproduce at least one other scientist's experiment
  - 50% failed to reproduce at least one of their own experiments

- **Psychology**
  - low statistical power in general
  - $>$ 50% admitted questionable practices: p-hacking, sample selection, p-value rounding to 5%, manipulation of outliers, post-hoc storytelling
  - Science (2015): 100 replicated studies in top 3 journals (i) 36% significant vs 97% in original, (ii) mean effect size around 1/2 compared to published studies

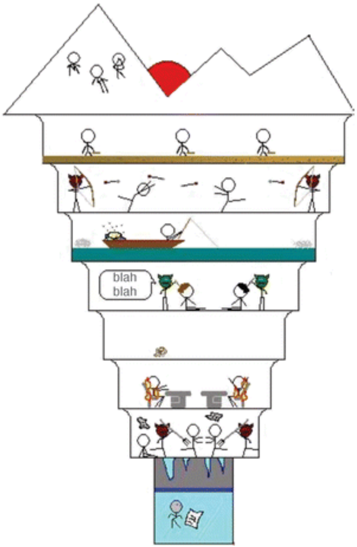
- **Medicine**
  - Nature (2012): only 11% of pre-clinical cancer studies could be replicated

The replication crisis

- **Economics**
  - Science (2016): 6 out of 18 studies in AER or QJE could not be replicated
  - Economic Journal (2017) over 6700 studies:
    - ⋆ median statistical power is 18%.
    - ⋆ those with adequate power (>80%) reveal that
    - ⋆ nearly 80% of the reported effects in these empirical economics literatures are exaggerated
    - ⋆ on avg by a factor of two; one-third inflated by a factor of four or more

The replication crisis

- **The nine circles of scientific hell** (with apologies to Dante)



I	Limbo
II	Overselling
III	Post-Hoc Storytelling
IV	P-Value Fishing
V	Creative Outliers
VI	Plagiarism
VII	Non-Publication
VIII	Partial Publication
IX	Inventing Data

Reproducible research requirements

- **Analytic data made available**
  - data used for analysis
  - not the same as raw data (sample selection, cleaning, too large)
- **Analytic code made available**
  - i.e. code applied to analytic data
  - used for key results (e.g. regression modeling)
- **Documentation of code and data**
  - description, logs, metadata
- **Standard means of dissemination**
  - easily accessible to users/readers

Challenges

- **Often considerable effort to make data/results available**
  - storage and online access to data and code
- **Readers might not have same resources as authors**
  - computing power, different operating systems and softwares
  - access to proprietary data
  - the “score” can get lost in translation (documentation, style,...)
- **In reality**
  - authors put stuff on the web (dreaded “online supplements”)
  - readers download and try to figure out what happened

Opportunities

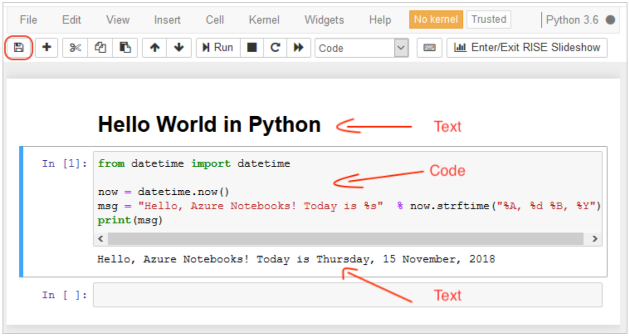
- **Small but growing toolbox for reproducible research**
  - markdown languages (in R, Stata, Python,...)
  - code repositories (GitHub, BitBucket, CRAN)
  - data warehouses (World Bank, DHS, NBB, Eurostat, ...)
- **General improvement of research pipeline**
  - adjust sample selection in reproducible way
  - robustness tests
  - start of new analysis

Literate statistical programming

- **Literate programming**
  - Donald Knuth (1986) (consolidator of CS and inventor of TeX): "a computer program is given an explanation of its logic in a natural language, such as English, interspersed with snippets of macros and traditional source code, from which compilable source code can be generated"
  - people best understand computer programs in a different order than compilers do
  - distinguishes LP from heavily commented programs
  - it forces programmers to explicitly state the thoughts behind the program, making poorly thought-out design decisions more obvious
  - literate programming provides a first-rate documentation system, which is not an add-on, but is grown naturally in the process of exposition of one's thoughts during a program's creation

Literate statistical programming

- **Literate (statistical) programming**
  - analysis code divided into **text** (LaTeX, Markdown, HTML) and **code**
  - code blocks load data, compute results, create output
  - text blocks describe data, explain analysis, present results
  - don't separate code and documentation
  - 1 document for human readable output and machine readable process
  - e.g. markdown languages, Jupyter notebooks (kind of - semantics)
  - Question: not sure if LSP goes well with IDEs?



Hello World in Python

```
In [1]: from datetime import datetime

        now = datetime.now()
        msg = "Hello, Azure Notebooks! Today is %s"  % now.strftime("%A, %d %B, %Y")
        print(msg)

        Hello, Azure Notebooks! Today is Thursday, 15 November, 2018

In [ ]:
```

Steps in data analysis

1. Define the question
2. Define the ideal dataset
3. Determine what data you can access
4. Obtain the data
5. Clean the data
6. Exploratory data analysis
7. Statistical modeling
8. Interpretation of results
9. Challenge results
10. Synthesize/write up results

Steps in data analysis

- **Step 1 – Define the question**
  - always under incomplete information: what problem did you ever solve that had all information you needed?
  - **the most powerful dimension reduction tool you'll ever have**
  - look at particular dimensions of the data only, reduce noise
  - STOP data mining (Pearl, 2018): data + model is needed to answer a question (e.g. correlation global warming and pirates)
  - people who tell me "let's see what we can do with the data" are immediately defriended

- **Step 2 – Define *ideal* dataset**
  - descriptive: whole population
  - exploratory: random sample with many features
  - inferential: right population, randomly sampled
  - predictive: training and test datasets from same population
  - causal: data from randomized study

Steps in data analysis

- **Step 3 – Determine data *you can access***
  - free, bought from provider, create data yourself
  - absolutely amazing data repositories (e.g. UCI Machine Learning Repository, SNAP, Kaggle ...)
  - make sure your data can still answer the question
  - if not, quit or change now **(it will NOT get any better)**
  - people who tell me "I know it's crazy, but this is the best we can do with the data we have" are also immediately defriended

- **Step 4 – Obtain the data**
  - cite the source
  - comply with any licenses
  - script time and url if online
  - static (download once) vs dynamic (call data from code every time)

- **Step 5 – Clean the data**
  - raw data needs to be pre-processed (document this!)
  - if it is pre-processed by someone else, make sure you understand how
  - understand source of data (sampling methods, experiment protocols,...)
  - subsample data if very large
  - determine if data are good enough - if not, quit or change now

Steps in data analysis

- **Step 6 – Exploratory data analysis**
  - understand data dimensions, formats and distributions, connections to other datasets
  - transform variables if needed
  - if needed: split in test/train *now*
  - univariate and multivariate plots and tables (histograms, box plots, scatters, correlations, cluster analysis...)
- **Step 7 – Statistical modeling**
  - insert domain knowledge here
- **Step 8 – Interpret results**
  - use appropriate language (correlation vs causation)
  - interpret coefficients
  - give an explanation

Steps in data analysis

- **Step 9 – Challenge your results**
  - if you don't do it, someone else will
  - challenge all steps: question, data sources, processing, analysis, conclusion
  - think of potential alternative analyses, mechanisms
- **Step 10 – Synthesize/write up**
  - lead with the question
  - summarize analysis into the story
  - don't include every analysis in paper
  - order analysis according to story, not chronologically
  - a picture is worth a 1000 words

Overview

Coding etiquette

- **Coding etiquette**
  - coding conventions that recommend programming style, practices and methods
  - improve readability of code
  - make maintenance and updates easier
  - can be formalized in a documented set of rules that an entire team or company follows
- **Goals**
  - transparent: time saving coding standards
  - clear: for your future self and other users
  - flexible: open to updates and improvements
  - open: to share with other contributors

Disclaimer: no universal way exists, and it often depends on a particular setting/project. But happy to share and hear best practices.

Gentzkow and Shapiro's guidelines

- Draws from Code and Data for the Social Sciences: A Practitioner's Guide
- **Automation**
  - automate everything that can be automated
  - write a single script that executes all code from start to end
- **Version control**
  - store code and data under version control
  - run the whole directory before checking it back in
- **Directories**
  - separate directories by function
  - separate files into inputs and outputs
  - make directories portable
- **Keys**
  - store cleaned data in tables with unique, non-missing keys
  - keep data normalized as far into code pipeline as possible

Gentzkow and Shapiro's guidelines

- **Abstraction**
  - abstract to eliminate redundancy
  - abstract to improve clarity
  - otherwise, don't abstract
- **Documentation**
  - don't write documentation you will not maintain
  - code should be self-documenting
- **Management**
  - manage tasks with a task management system
  - e-mail is not a task management system

Automation

- **Example**
  “We used to routinely export files from Excel to CSV by hand. It worked ok until we had a project that required exporting 200 separate text files from an Excel spreadsheet. We followed our usual practice and did the export manually. Some time later, the provider sent us a new Excel file reflecting some updates to the data. We had learned our lesson.”

- **Rule 1 – Automate all that can be automated**
  - there is a cost of automating things
  - but if you need to do the same thing more than once, most probably that cost is larger
  - in most cases you don’t do something only once
  - in the end, automation saves time
  - QED.

Automation

- **Don't do things by hand**
  - editing a spread sheet (remove outliers, sample selection)
  - editing tables or figures (rounding, formatting)
  - clicking URLS to download data (make URL part of code)
  - moving around data on you computer
  - point and click (some progs also generate code then in logs)
  - anything you do by hand is harder to document (if you change your process, you need to update the document)

- **Teach a computer**
  - clear instructions
  - forces the researcher to think what you want to do
  - everything is reproducible

- **Example – download data**
  - by hand: go to website, download by clicking, save as...
  - by code: visit url (whole path), save as (project directory)

Automation

- **Example**
  - your project folder might look something like this
  - staring at it, you might figure out what is what

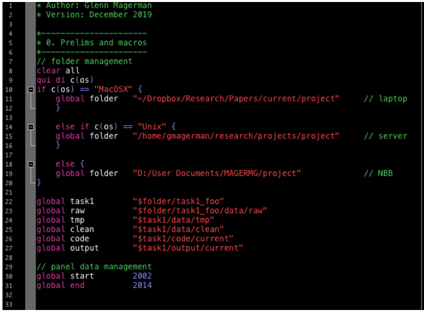
chips.csv	mergefiles.do	tv_potato_submission.pdf
cleandata.do	regressions_alt.do	tv_potato.tex
extract0B.xls	regressions_alt.log	tv.csv
fig1.eps	regressions.do	tvdata.dta
fig2.eps	regressions.log	
figures.do	tables.txt	

- **But**
  - which obs do we drop where?
  - should I first run mergefiles.do or cleandata.do?
  - does it matter if we first run figures.do or regressions.do?
  - is regressions_alt.do output in the paper or is it a leftover?
  - what is in tables.txt? is it produced manually or is it code output?

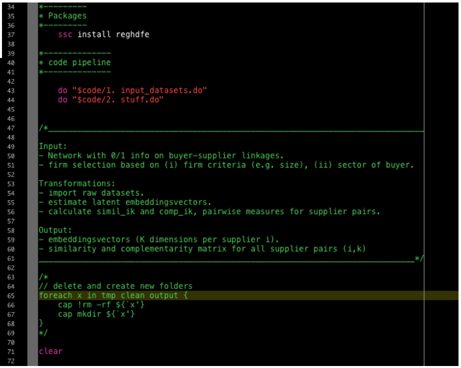
Automation

- **Rule 2 – Write a master script that executes everything**
  - minimizes effort of going through various codes and data versions
  - set up fixed folder structure per project
- **Master file**
  - master file cannot be incomplete, ambiguous, or out of date
  - (unlike a readme file)
  - test: delete all of the output files, and start over again.
  - now, output is reproducible, starting from the raw data and code only

Personal workflow



```
* Author: Glenn Magerman
* Version: December 2019

*----------------------
* 0. Prelims and macros
*----------------------
// folder management
clear all
qui di c(os)
if c(os) == "MacOSX" {
    global folder   "~/Dropbox/Research/Papers/current/project"     // laptop
    }

    else if c(os) == "Unix" {
    global folder   "/home/gmagerman/research/projects/project"     // server
    }

    else {
    global folder   "D:/User Documents/MAGERMG/project"             // NBB
}

global task1        "$folder/task1_foo"
global raw          "$folder/task1_foo/data/raw"
global tmp          "$task1/data/tmp"
global clean        "$task1/data/clean"
global code         "$task1/code/current"
global output       "$task1/output/current"

// panel data management
global start        2002
global end          2014


```



```
*---------
* Packages
*---------
    ssc install reghdfe

*-------------
* code pipeline
*-------------

    do "$code/1. input_datasets.do"
    do "$code/2. stuff.do"


/*______________________________________________________________________________

Input:
- Network with 0/1 info on buyer-supplier linkages.
- firm selection based on (i) firm criteria (e.g. size), (ii) sector of buyer.

Transformations:
- import raw datasets.
- estimate latent embeddingsvectors.
- calculate simil_ik and comp_ik, pairwise measures for supplier pairs.

Output:
- embeddingsvectors (K dimensions per supplier i).
- similarity and complementarity matrix for all supplier pairs (i,k)
______________________________________________________________________________*/

/*
// delete and create new folders
foreach x in tmp clean output {
    cap !rm -rf ${`x'}
    cap mkdir ${`x'}
}
*/

clear

```

Version control

- **Continued example**
  - in the current project, after some work, the directory might look as follows

cleandata_022113.do	cleandata_022613.do	regressions.log
cleandata_022113a.do	cleandata_022613_jms.do	regressions_022413.do
chips.csv	tvdata.dta	regressions_022713_mg.do
regressions_022413.log		

- **versions of files are denoted by**
  - dates (e.g. to compare results and roll back changes if wanted)
  - author (e.g. model specification JMS vs MG)

Version control

- **But it's not helpful at all**
  - which version to continue on, the old or the new? (e.g. 5-author project: regularly had to manually compare and merge individual co-authors' output)
  - what if I forget to correctly tag the files?
  - which is the log file allocated to regressions_022713_mg.do?
  - Did MG forget to tag, overwriting the old .log file? or is there no log?
  - which version of cleandata.do produces the data for regressions_022413?
  - changing file names in all places is a hassle and will lead to errors
  - (there is a cheap trick: use coded dates for all files) (e.g. Stata: filename_‘c(current_date)’)
  - it is too much work to keep track of multiple versions of files, resulting in not being to replicate results later on

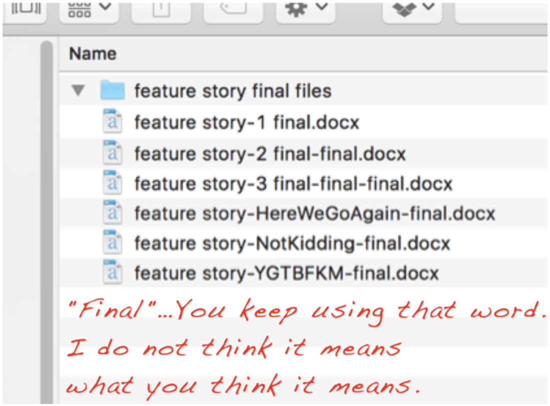
Not one piece of commercial software was written with the “date and initial” method.

Version control

- **Rule 3 – use version control**
  - set up a *repository* (place for storage and safekeeping to deposit) on your pc (or better, in the cloud)
  - if you want to modify a directory, check it out of the repository, make changes elsewhere, check it back in.
  - repository software remembers every version that was ever checked in
  - **don't change file names, add dates or anything**
- **Do you change your mind about something?**
  - ask repository software for a history of changes to the directory
  - color-coded side-by-side comparison of changed content
  - 1 click to go back to any component
  - even if your sneaky co-author changed something without telling you
  - version control is like an undo command for everything.
  - e.g. dropbox, Mac's time machine, track changes in Word, LyX, . . .
  - for our purposes: GitHub (see meeting 2)

Version control

- **Why repositories?**
  - it maintains a single version of the directory at all times
  - you can edit without fear, explore new paths and come back
  - no crazy filenames needed
  - have a data and analysis pipeline: use always same filename as input for further steps



Name
- feature story final files
  - feature story-1 final.docx
  - feature story-2 final-final.docx
  - feature story-3 final-final-final.docx
  - feature story-HereWeGoAgain-final.docx
  - feature story-NotKidding-final.docx
  - feature story-YGTBFKM-final.docx

"Final"...You keep using that word. I do not think it means what you think it means.

Version control

- **Rule 4 – only check in/out whole directory when manipulating**
  - run master file to execute batch of code, manipulations and results

- **Continued example**

rundirectory.bat	tvdata.dta
cleandata.do	regressions.do
chips.csv	regressions.log

- **Same as before, now dropped all dates and initials versions**
  - say someone changes cleandata.do and runs it to overwrite tvdata.dta
  - one might later find that regressions.do breaks when running, because of a change to tvdata.dta that was not expected
  - hence, execute the batch of code from start to finish, and check errors before checking in the directory again
  - all output in regressions.log will be created, no broken items or code
  - you can still go to the old output in regressions.log by using an older version of the directory in the depository

Version control

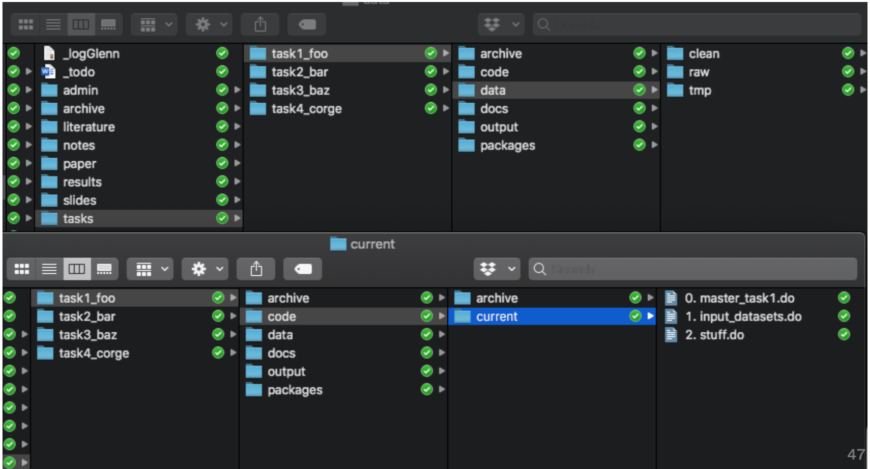
- **Also, keep track of software environment**
  - operating system (and locale/language environment - UTF)
  - software toolchain (compilers, interpreters, programing language, softwares)
  - libraries and packages (and their versions!)
  - external dependencies
  - complex data project rely on many tools that often not always work everywhere

Directories

- **Rule 5 – separate directories by function**
- **Rule 6 – separate files into inputs and outputs**
- **Rule 7 – make directories portable**

Personal workflow

- **Standardized folder structure**
  - always same folders and subfolders
  - admin, archive, literature, notes, paper, results, slides, tasks
  - data: raw, tmp, clean
  - can be automated (Rule 1)

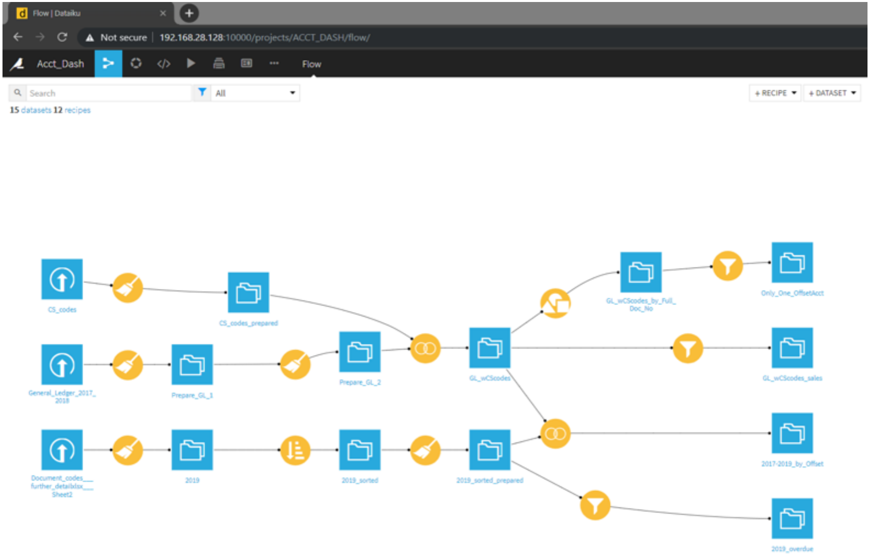


Personal workflow

- **Folder structure**
  - I don't need to think/look up where to source or put things
  - portable (relative paths using macros)
  - master file by task and one "master" master file
  - each major update, I clean whole task except for raw data and code
- **Tasks**
  - divide work and thought processes into smaller pieces
  - input –> task 1 –> output –> task 2 –> output
  - for me personally a huge time saver
  - avoid too many dependencies: finish one part and move on to next
  - easier to debug
  - quasi modular: dependent blocks, but portable

Personal workflow

- **Advanced modularity (DSS)**



Relational keys

- **Consider the following census data**

county	**state**	**cnty_pop**	**state_pop**	**region**
36037	NY	3,817,735	43,320,903	1
36038	NY	422,999	43,320,903	1
36039	NY	324,920	.	1
36040	.	143,432	43,320,903	1
.	NY	.	43,320,903	1
37001	VA	3,228,290	7,173,000	3
37002	VA	449,499	7,173,000	3
37003	VA	383,888	7,173,000	4
37004	VA	483,829	7,173,000	3

- **Issues**
  - How can the population of the state of New York be 43 million for one county but "missing" for another?
  - If this is a dataset of counties, what does it mean when the "county" field is missing?
  - If region is something like Census region, how can two counties in the same state be in different regions?
  - And why is it that all the counties whose codes start with 36 are in New York except for one, where the state is unknown?

Relational keys

- **We can't use these data, because we don't understand what they mean!**
    - don't dive in and ignore this!
    - different programs deal differently with missing values and calculated statistics
    - without looking back at the underlying code, we could never say confidently what every variable is, or even every row. And we can forget trying to merge on attributes from another dataset.
    - how would we know which state goes with county 36040?
    - which region to use for 37003?

- **Rule 8 – save your data as relational databases**
    - large organizations like financial institutions, retailers, and insurers have to manage much more complex data in real time, with huge consequences of mistakes.
    - physical structure of a database should communicate its logical structure

Relational keys

- **Relational databases**
  - Each table (with variables and rows) has a key (variable that uniquely identifies the elements of a table)
  - think of levels/layers/clusters of aggregation
  - e.g. state population is a property of a state, so it cannot live in the county table
  - can be in .txt files etc, no need for additional software!
  - avoids mistakes (e.g. regression with repeated observations)
  - and it saves space (no extra variable needed with repeated entries)
  - classical mistake! typically combine variables defined at several different levels of aggregation

Relational keys

- **Now the ambiguity is gone**
  - every county has a population and a state
  - every state has a population and a region
  - there are no missing states, no missing counties, and no conflicting definitions
  - the database is self-documenting. In fact, the database is now so clear that we can forget about names like county_pop and state_pop and just stick to "population"
- **Remember to merge/join databases before analysis when needed!**

county	state	population
36037	NY	3,817,735
36038	NY	422,999
36039	NY	324,920
36040	NY	143,432
37001	VA	3,228,290
37002	VA	449,499
37003	VA	383,888
37004	VA	483,829

state	population	region
NY	43,320,903	1
VA	7,173,000	3

Data storage

- **Rule 9 – keep data normalized as far into the code pipeline as possible**
- **Step 1 – store raw data**
  - normalize data formats (e.g. import excel sheets and save as .dta, drop excessive sheets, columns etc)
  - preserve the information from the raw data
  - imagine you prepare the data for release to a broad group of users (you probably want to use the data in ways you do not currently anticipate)
- **Step 2 – store tmp data**
  - with manipulations (e.g. variable transformations, additional variables)
- **Step 3 – store clean data**
  - merge with necessary datasets/relational databases, ready for analysis
  - at this stage, your database should still have unique, non-missing keys
  - do no data manipulation in this step
  - if your analysis requires the log of x, calculate it in step 2
- **This ensures the data is formatted, normalized and flexible until the last possible step**

Abstraction

- **Rule 10 – abstract to eliminate redundancy**
- **Rule 11 – abstract to improve clarity**
- **Rule 12 – otherwise, don't abstract**

- **If you need to copy-paste code, consider writing a function**
  - e.g. instead of calculating a moment at the city, state, county level every time
  - write a program that does the analysis
  - apply the program
  - DRY coding: Don't Repeat Yourself
  - DRY code maintains modularity: easier to port parts and manage big programs

- **Reason**
  - avoid redundant code
  - reduce scope for mistakes
  - increases readability of code
  - no need to adjust all code when making a change

Abstraction

- **Bonus**
  - we can use it for other projects
  - make sure to "unit test": test code on toy set to see if it really does what we think it does
- **When to abstract**
  - some recurrent code applies special cases of a more general instance
  - not if code block is used only once

Documentation

- **Rule 13 – don't write documentation you will not maintain**
- **Rule 14 – code should be self-documenting**

- see discussion before

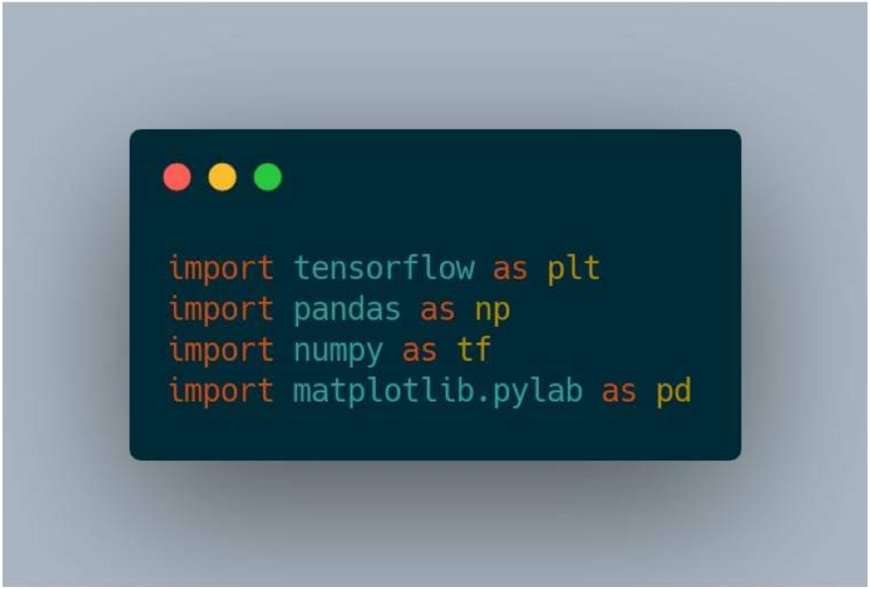
Management

- **Example**
    - Jesse: *Hey Matt, Do you have that robustness check where we control for the amount of ranch dip sold in each county? I am writing the section on dipping sauces and wanted to mention it.*
    - Matt: *Sorry, I thought you were doing that because it's similar to that other thing you were doing with controlling for salsa sales. Let me know if you want to do it or if you want me to take over.*
    - Jesse: *I thought Matt was doing ranch dip and Mike was doing salsa?*
    - Mike: *I did the salsa robustness check two weeks ago. See my e-mail from 8/14, 9:36am.*
    - Jesse: *Right, but in that e-mail you were controlling for the log of salsa consumption. I thought we agreed we wanted the level of consumption?*
    - Mike: *On it!*

Management

- **What's wrong?**
  - ambiguity: Mike thought his task was done, when Jesse thought it was not. In fact, after all these emails, it's still not clear who is doing what.
  - how to follow up on this? looking through email threads? checking code? or output?
- **Rule 15 – manage tasks with a task management system**
  - many free packages exist, often based on Kanban methods
  - often with email integration (auto email when tasks are done/edited)
  - Asana (www.asana.com), Wrike (www.wrike.com) and Flow (www.getflow.com)
  - Here's a comparison of the three (I'll try Asana)
- **Rule 16 – e-mail is not a task management system**

Last one – be nice!



```
import tensorflow as plt
import pandas as np
import numpy as tf
import matplotlib.pylab as pd
```

More coding etiquette

- **I'm sharing a live document for us to contribute to and learn from**
    - it's still a mess...

Wrapping up

- **Data pipeline is a large part of our daily work**
  - not always fun
  - how can we reduce friction and increase speed?
  - all of this has been tackled in more complex settings by CS people
  - we can learn from them
- **Reproducibility is important**
  - for ourselves, for our referees, for science as a whole
- **Standardization is important**
  - optimize our workflows
  - some simple tricks go a very long way
- **hopefully you found this talk very, very obvious**
  - if you rolled your eyes at least 10 times, you're doing very well!

Next meeting

- **Date:** Feb 18, 2020, Room R42.3.103
- **Topic:** Jupyter (Anaconda), Git, GitHub, GitHub Desktop
- **Lead:** Fabrizio Leone
- **TODO before meeting:** install GitHub Desktop and Python. read this guide and follow the steps indicated