**Introduction to Data Management – EEB698**

Adapted from Abhijit Dasbupta webinar on Best Practices for Data Analysis

1. **Project organization**
   1. Use a template to organize each project. Create a standard hierarchy of folders & files you always start each project with.
      1. ▸ Set up a particular folder structure where
         1. ▸ you know what goes where
         2. ▸ you can have canned scripts/packages set things up
   2. Structure (Look at Jeff Leek’s datasharing repo: <https://github.com/jtleek/datasharing>)
      1. Background
         1. Store papers/reports relevant to this project.
      2. Data
         1. Raw (read-only, storage, not to be touched, pristine backup)
         2. Rda (intermediate and final R datasets, could also be called working)
      3. Docs
         1. “Study overview” document
            1. section called “code book” or “data dictionary” describes the purpose and type of data for each variable in the database.
            2. section called “study design” or “protocol” describes exactly how the study was conducted.
         2. Generated documents from analyses - Rmarkdown, word docs, ppt.
      4. Graphs (pdf, png, svg)
      5. Lib (enables you to do the common things once.)
         1. R - custom R functions written for the project or used in this project
         2. Tests- unit testing files for custom functions
         3. Packages.R - state what other packages needed for this project
         4. Reload.R - function which automatically loads functions in lib and all packages, so one command produces everything needed to run the project.
      6. DataWrangling
         1. DataAcquisition.R – script for compiling all data files into a single dataset
         2. DataMunging.R – script for munging data into a usable database
      7. FiguresScripts
         1. Figures.R - holds code for final figures for paper
      8. AnalysisScripts
         1. Modeling.R – script(s) for analysis/modeling
      9. Report
         1. Rmarkdown files with notes from final analysis
      10. ReadMe
          1. Write down the driving questions and purposes of the project, as well as any notes important for running the code/remembering what you did and need to do. (you won't remember in 3 months!)
2. General best practices
   1. File Naming Conventions
      1. Be explicit about what a file does in the name.
   2. Document code & functions as much as you can.
      1. Comment a lot
      2. For a function - write down what the arguments mean and the purpose of the function. These files are all searchable, if you write down.
3. How to ingest & store data
   1. Excel is okay only for data transport and storage. Enter, then keep in read-only mode.
   2. Better to keep it in text or put it in a database; makes data sharing very easy, does not limit access to data, operating system agnostic
      1. Csv - stores data beautifully; TSV
      2. Software carpentry - look it up. They have lots of great ideas on this.
4. Importing data
   1. Use scripts
   2. Verify size of data (did it read correctly) and types of variables (did it change anything when importing?)
5. Data munging and wrangling
   1. Manipulate data with care.
      1. Keep a pristine copy.
      2. Use scripts to manipulate so you can reproduce. Catch analyst errors & fix them.
      3. Systematically verify and clean following your own standard of practice.
      4. Document what you find and do (lab notebook) - Carl Boettiger has an example of how to create a lab notebook in R ecosystem. <http://www.carlboettiger.info/2012/09/28/Welcome-to-my-lab-notebook.html>
      5. Test your data at every stage to see if the structure you expect is still there.
   2. Prefer tidy datasets
      1. Each variable is own column, each observation is own row, each value has its own cell. (no values in headers of columns).
      2. Tidyverse - a set of R packages that R Studio has developed to create & analyze tidy data.
      3. Package called group?) by david robinsion -makes results of analysis into tidy datasets
   3. Code suggestions
      1. Comment your code- why did you make each choice?
      2. Develop naming conventions -
         1. Follow established coding styles, we suggest Google’s Style Guide for R <https://google.github.io/styleguide/Rguide.xml>
         2. Under\_scored or CamelCase - which one are you going to use for multi-word names?
         3. Be explicit in your variable names
   4. Forgiving mistakes
      1. Use version control system
         1. Use git for code & small data sets - can rewind with git
            1. Can use git to comment on next steps for a project
         2. Use dropbox for larger datasets
      2. Avoiding mistakes - when want to do something new that might corrupt data and analyses
         1. Create git branch
         2. Develop, validate
         3. Merge
   5. Track data provenance through the pipeline
      1. Raw data 🡪 intermediate data 🡪 final data 🡪 data for sub-analyses 🡪 data for final table and figures
      2. Catalog where you created each dataset and where it's ingested. Script files have a habit of multiplying.
   6. Explore your data - take your time
      1. Summarize, visualize, discerning relationships. Be fluent & creative in this.
         1. Will save time later catching errors
      2. Share preliminary analysis for a sniff
         1. Are data types what you'd expect, data ranges, distributions, relationships? Are anomalies reasonable? Or errors? Or outlier? Or wrong theoretical framework, or study design?
6. Computation and Analyses
   1. Scripts
      1. Script/function/macro can be verified
      2. Don't hard-code numbers
   2. If doing something multiple times, create a function, put in a separate file, store in a particular place.
      1. Don't hide in general script files where other analyses are going on.
      2. Name file logically
      3. One function or set of related functions (modules) per file
      4. Write documentation when you write function.
   3. Test test test
      1. Write tests for your functions and check. Test early not down the road because errors are harder to catch later. "testthat" in R is a unit testing framework <https://github.com/hadley/testthat>
7. Modeling and applications
   1. Think
      1. Does my understanding of the data support the models and tests I planned to run? Do I need to modify the standard stuff? Do I need help?
   2. Statistical testing
      1. No situation is ideal. Think if test and data match. Leveral computational testing (bootstraps, permutations) if needed.
      2. Reflect on whether the test results match what you feel for the data
   3. Modeling
      1. Don't do one model. Try many modeling looks at the data. If different models are saying qualitatively the same thing, maybe gain confidence in what you're seeing.
      2. Resist temptation to do cookie-cutter models and be satisfied. This is the case when use menu-driven programs (SPSS).
         1. Watch out for confounders
         2. Think about implications of your results
         3. Diagnostics are key
            1. Adequate fit, reasonable functional form, external validity (training sets)
      3. Feedback loops
         1. Modeling is inherently a feedback loop- may loop all way back to the basic questions you pose.
      4. Compare model results
         1. Tidy'ing up results using "broom" or "estout" can help (david robinson, u Chicago - good figure for this in 2016 JSM)
         2. Use tables & graphs to evaluate models.
   4. Reporting and visualization
      1. Know where final tables and figures come from
         1. Create separate files for creating figures and tables for final paper.
         2. Make sure you're using the right data to create them. easy to get confused.
         3. Metadata helps - lab notebook, documentation.
      2. Literate programming rocks
         1. RMarkdown - code and text intertwined. (like EndNote for bibliography management); other options are Sweave, Jupyter notebooks, Matlab notebooks.
         2. Makes reports automated, reliable.
         3. Prevents hard coding or copy/paste, which are error-prone.
      3. Keep visualizations succinct
         1. Don't need to be fancy, colorful. Just clear, to-the-point, make the points you're making obvious.
         2. Sometimes need dynamic graphs (e.g. nytimes, guardian, fivethityeight, wapo)
         3. Read tufte at least once.
      4. Bad visuals
         1. Learn when bad.
            1. Look at good and bd examples

Flowing data

The D3.js website

* + - 1. If you have to make effort to understand a visualization, change the visualization.
    1. Tables
       1. Report #'s to 2-3 decimal points.
       2. Formatting is important.
       3. Packages in R allow you to create nice tables. But can also do in word/excel.
    2. Reproducible presentations
       1. Use literate programming to create presentations. Automate. Makes it easy to create periodic reports.
       2. Look at Frank Harrell's website at Virginia using Sweave, LaTex.

1. Closing thoughts
   1. Adopt a "Mack Truck" or sharing mentality
      1. If someone else had to do this analysis - could they do it from what I have?
      2. Could they use what I have developed?
   2. Validate Validate Validate
      1. Recognize potential for errors
      2. Ensure provenance , veracity of all data. Original and derived.
      3. Use code instead of menus.
   3. This topic is open, fluid and ever-changing.