10 ways your data project can fail

(and how to avoid them)

Isaac Slavitt
isaac@drivendata.org
AC 297r, Spring 2017

First, some motivation

- Capstone projects are excellent practice for being a quantitative professional in the “real world”
- You will learn from this experience
- This is a difficult **technical** and **professional** real-world project that you can discuss at an expert level
  - “Tell me about a data project that you’ve done with a team. What did you add to the group?”
  - “Tell me about a dataset that you’ve analyzed. What techniques did you find helpful and which ones didn’t work?”
  - “Can you give an example of how you have approached missing data?”

http://www.datasciencequestions.com/

Data scientist assets

- Creative
- Optimistic
- Detail oriented
- Highly technical
- Makes a plan and sticks to it
- Rigorous and concerned with correctness
- Can work without seeking external guidance
- Sense of craftsmanship and takes pride in work
- Immediately thinks of edge cases and failure modes

Data scientist liabilities

- Creative
- Optimistic
- Detail oriented
- Highly technical
- Makes a plan and sticks to it
- Rigorous and concerned with correctness
- Can work without seeking external guidance
- Sense of craftsmanship and takes pride in work
- Immediately thinks of edge cases and failure modes

Pitfalls specific to smart & conscientious people

1. **Scope.** Your scope will be too ambitious, vague, and open ended.
2. **Data quality.** You will not get data early enough and it will be messy.
3. **Data familiarity.** You will not spend enough time exploring the data.
4. **Perfectionism.** You will spend too long looking for the “perfect” solution.
5. **Priorities.** You will spend too long worrying about non-value adding details.
6. **Communication.** You will not ask questions and get feedback early enough.
7. **Time management.** You will not start working on final deliverables early enough.
8. **Organization.** Your research and code will be hard to maintain, explain, reproduce.
9. **Simplicity.** Your solution will be overly complicated and under documented.
10. **Delivery.** You will not communicate your solution clearly enough.

1. Scope

Your scope will be too ambitious, vague, and open ended

Problems

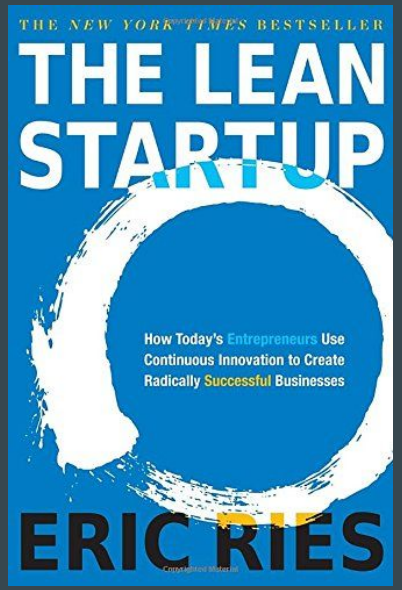
- The beginning of a project is the optimism danger zone
- Most problems have many plausible solutions and promising methods
- “This should only take a few days” syndrome

https://xkcd.com/1425/

1. Scope

Your scope will be too ambitious, vague, and open ended

Ways to avoid this



- First, brainstorm all possible ideas and deliverables
  - Write down all the most promising research directions
  - Cut it down to a plausible number for ~4 weeks of work
  - Now, cut it down again — **seriously, you will not have time for it**
  - You don't have to delete ideas, they're now called "nice to haves"
- Define the **MVP**
  - Create a short (1-2 page) scoping document
  - "SMART": specific, measurable, achievable, relevant, and time-bound
  - Also explicitly define what is **not in scope**
  - You don't have to delete ideas, they're now called "nice to haves"
  - Get **explicit feedback and then agreement** on this scope **before you start work**

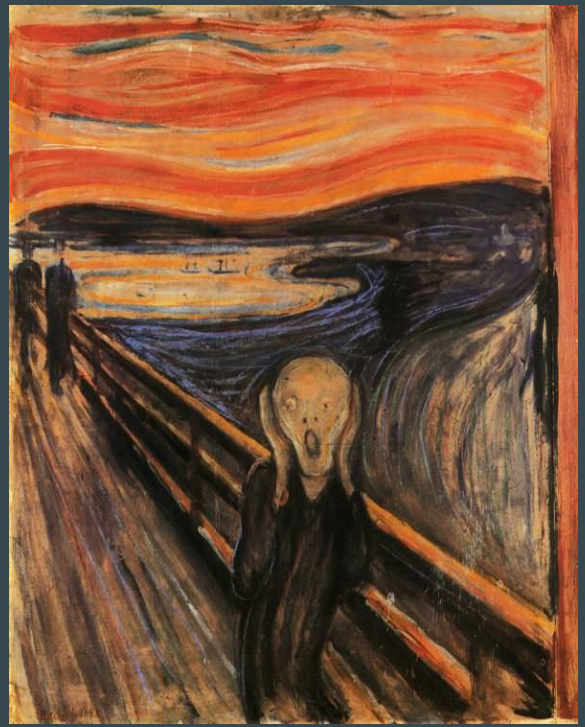
Choose your own adventure: _____promise and _____deliver

2. Data quality

You will not get data early enough and it will be messy

Problems

- Different data formats even within the same field
- Weird file formats
- Unic  de import/export errors
- Lack of consistent primary keys
- Way less data than originally implied
- Data is not stored consistently
- Values are pre-processed or pre-aggregated in some way (e.g. somebody already averaged, rounded, or grouped)
- Misleading types (field looks numeric but they're actually ordinal or numbers representing category)
- Lots of duplicate values



http://www.edvardmunch.org/the-scream.jsp

2. Data quality

You will not get data early enough and it will be messy

Ways to avoid this

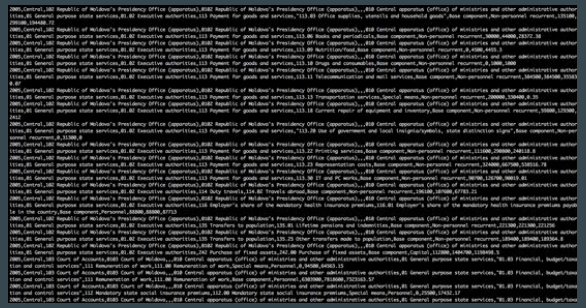
- Get the data **first**
  - It **will** be missing something important
  - ... that is why they need you!
- **Don't panic**!
  - Find a **principled** (not perfect) way to get a sensible working proxy
  - Hint: this is called an **assumption** — it's your job to document *why* you chose it, *what* implications it might have on your analysis, and *how* they can potentially improve this on their end
- Consistency is usually more important than accuracy; being consistently wrong in the right direction is usually OK

3. Data familiarity

You will not spend enough time exploring the data

Problems

- Confusing or misleading data fields
- Data storage or entry practices changed over time
- Patterns that suggest non-random entry or label contamination



3. Data familiarity

You will not spend enough time exploring the data

Ways to avoid this

- Ask a lot of questions **early on**, especially about how the data was collected and what things mean
  - Don't make assumptions about what things mean unless you absolutely have to
  - Ask if they have a "data dictionary" or any other internal docs you can see
- Do a ton of the boring, exploratory work **before** you even think about applying algorithms
  - Do lots of value counts (e.g. `df.value_counts().sort_index()`) and pivot tables
  - Do lots of **basic** visualization like boxplots and histograms
  - Get a sense of imbalanced classes and missing values
- Forget your fancy tools until you really know what's going on
  - Suck it up and open Excel
  - Yes, you are allowed to use Excel sometimes, nobody will retract your "data scientist" title

"There is no such thing as clean or dirty data, just data you don't understand."
— Claudia Perlich, Chief Scientist at Dstillery

4. Perfectionism

You will spend too long looking for the “perfect” solution

Problems

- It feels wrong to make decisions without doing in depth analysis of all the alternatives
- Edge cases or data problems make it seem like there’s no way forward
- All the “but what if” questions are demotivational

4. Perfectionism

You will spend too long looking for the “perfect” solution

Ways to avoid this

- Ignore edge cases until later
  - Consider if you have to even deal with these
  - Many times they’re just not that relevant to the client, so focusing on them is a huge waste of time
- Ignore polished aesthetics until later
- Just remember:
  - You could write an entire master’s thesis for any of these problems
  - The literature review alone would take longer than you have for this project
- If you spend too long thinking about perfect solutions, **your end result will actually be worse** than if you made some early decisions knowing that you can revisit them as time permits

5. Priorities

You will spend too long worrying about non-value adding details

Problems

- yak shaving

- bikeshedding

[1] https://medium.com/@firehoseproject/a-guide-to-yak-shaving-your-code-d30f98dc759
[2] http://bikeshed.org

5. Priorities

You will spend too long worrying about non-value adding details

Problems

- **yak shaving** [MIT AI Lab, after 2000: orig. probably from a Ren & Stimpy episode.] Any seemingly pointless activity which [seems] necessary to solve a problem which solves a problem which, several levels of recursion later, solves the real problem you're working on.
  — The Jargon File, edited by Eric S. Raymond, http://www.catb.org/~esr/jargon/html/Y/yak-shaving.html

- **bikeshedding**. "... a metaphor to illuminate Parkinson's Law of Triviality. Parkinson observed that a committee whose job is to approve plans for a nuclear power plant may spend the majority of its time on relatively unimportant but easy-to-grasp issues, such as what materials to use for the staff bikeshed, while neglecting the design of the power plant itself, which is far more important but also far more difficult to criticize constructively."
  — Wiktionary, "bikeshedding," https://en.wiktionary.org/wiki/bikeshedding

5. Priorities

You will spend too long worrying about small details

Ways to avoid this

- First, get all the pieces working — use placeholders or mocks wherever necessary
- Start with data samples, making something run on “big data” is a often huge waste of time if you’re still exploring
- Remember the rules of optimization:
  - Make it work
  - Make it right
  - Make it fast (<- hint: you probably won’t need to even get to this one)
- Remember: “**YAGNI**”
- Don’t worry about aesthetics and minor style tweaks until the very end
- **Always go back to the scoping document — remember your MVP**

[1] http://wiki.c2.com/?MakeItWorkMakeItRightMakeItFast
[2] https://martinfowler.com/bliki/Yagni.html

6. Communication

You will not ask questions and get feedback early enough

Problems

- Faulty assumptions lead you to **do the wrong thing**
- “Going dark” — sitting on problems, questions, and bad news until it’s too late
- Clients have no idea what you’re doing and then have issues with the final product, e.g.:
  - Delivering a solution that **solves the wrong problem**
  - Delivering a **technically advanced but impractical solution**
  - Delivering a **technically advanced solution that nobody understands**

6. Communication

You will not ask questions and get feedback early enough

Ways to avoid this

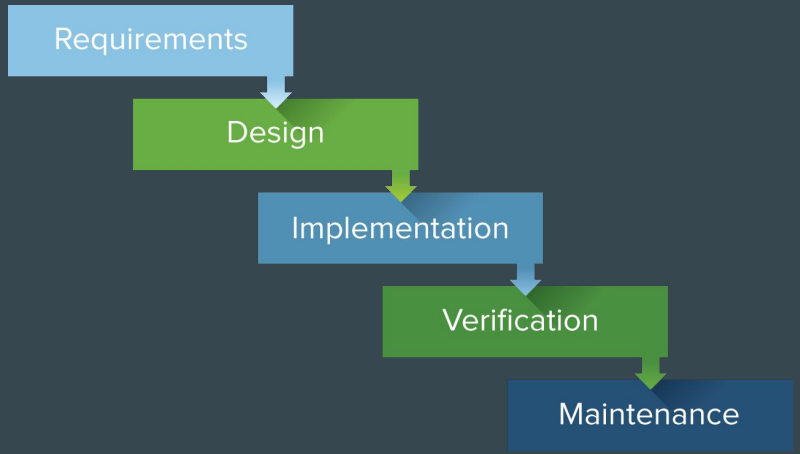
- **Do not start writing code** until you understand what you're trying to do
  - Have the client get you up to speed
    - What is the **greater context** for their problem?
    - Ignore the tools and techniques for a minute — what are their actual **goals**?
    - Ask questions like "If you had _____, how would you use it to make different decisions?"
  - Don't be embarrassed to ask basic questions or push back to get clarification
    - Protip: this is like not knowing somebody's name... if you ask early on, it's less embarrassing
- Stay in contact with your client and **over-communicate**
  - Give relatively frequent progress updates
  - Run concepts and questions by them to see if you're on the right track

7. Time management

You will not start working on final deliverables early enough

Problems

- By the time you're ready to "prep things for delivery" you are out of time
- The ninety-ninety rule: "The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time."
  — Tom Cargill, Bell Labs, via Wikipedia



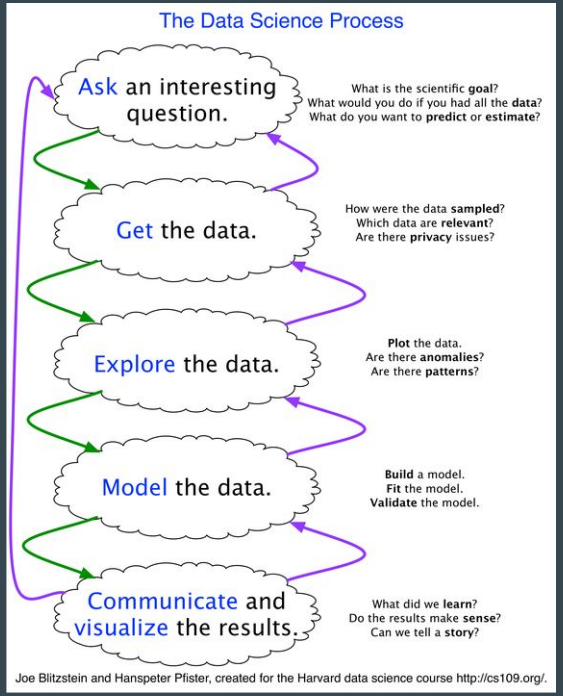
https://www.smartsheet.com/agile-vs-scrum-vs-waterfall-vs-kanban

7. Time management

You will not start working on final deliverables early enough

Ways to avoid this

- Just **get something working** then worry about making it correct
  - Break your solution up into loosely coupled pieces
  - Use placeholders at first, but get everything assembled
  - Work iteratively to improve pieces until you run out of time
- As early as possible, reach the “shape” of your **final deliverable**
  - There may still be a laundry list of things to improve — “future work” is a thing, and clients actually appreciate it
  - But in theory, you could hand it off **today** and still meet the requirements in your scoping doc
  - Guess what: now it’s ready in principle and you can work on the “over delivering” part if you have time
- Note: if you thrive on stress and time anxiety, ignore this advice



https://www.quora.com/What-is-it-like-to-design-a-data-science-class/answer/Joe-Blitzstein

8. Organization

Your research and code will be hard to maintain, explain, and reproduce

Problems

- “We can clean this up later”
- `data_final-version_AGGREGATED-(EDITSv2)~v3_draft!USE_THIS_ONE!.xls`
- Most data science projects look like this:

```
.
├── Step\ 1\ -\ install\ necessary\ software\ and\ packages.txt
├── Step\ 2\ -\ one-off\ step\ to\ create\ postgresql\ server\ instance\ and\ a\ database.txt
├── Step\ 3\ -\ one-off\ step\ to\ create\ tables\ and\ views\ in\ postgresql.py
└── Step\ 4\ -\ The\ only\ file\ to\ run\ when\ you\ want\ to\ run\ models\ and\ generate\ new\ scores.py
```

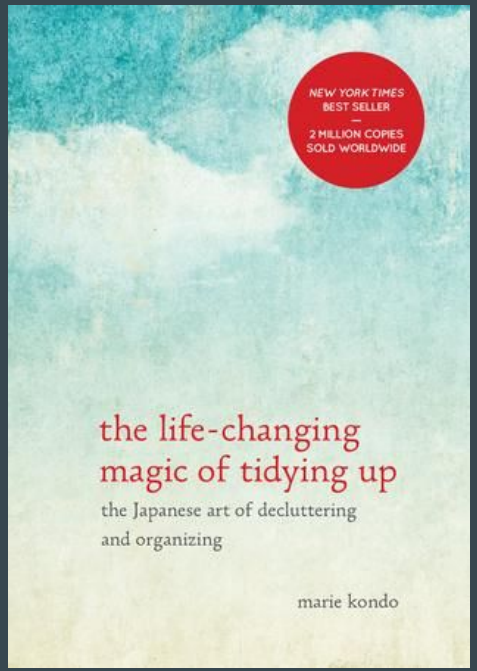
```
.
├── AllViolations.csv
├── BusinessClass.py
├── GenLearningData.py
├── GenTestingData.py
├── InspectionClass.py
├── LearnTest.py
├── PhaseIISubmissionFormat.csv
├── PhaseIISubmissionFormat_final.csv
├── PhaseIISubmissionFormat_test.csv
├── README.txt
├── ReviewClass.py
├── restaurant_ids_to_yelp_ids.csv
├── yelp_boston_academic_dataset
└── yelp_duplicate_ids.csv
```

```
.
├── Inspection_count_min.jpeg
├── README.Rmd
├── README.html
├── dd_dictionary.csv
├── mallet.rar
├── scripts\ and\ data
│   ├── AllViolations.csv
│   ├── PhaseIISubmissionFormat.csv
│   ├── build_rev_tm.R
│   ├── docsAsTopicsProbs_noStopwords.txt
│   ├── feature_eng.R
│   ├── features_test_phase2.csv
│   ├── features_train_phase2.csv
│   ├── learning_final.R
│   ├── negative-words.txt
│   ├── positive-words.txt
│   ├── rand_neg.txt
│   ├── restaurant_ids_to_yelp_ids.csv
│   ├── rev_tm.txt
│   ├── review_sentiscored.csv
│   ├── run.R
│   ├── sentiment_script.R
│   ├── sub_2_PhaseII_h20.csv
│   ├── yelp.stops
│   └── yelp_academic_dataset_business.json
├── varimp_gbm1.jpeg
├── varimp_gbm2.jpeg
└── varimp_sev.jpeg
```

≈

8. Organization

Your research and code will be hard to maintain, explain, and reproduce



8. Organization

Your research and code will be hard to maintain, explain, and reproduce

OPEN ACCESS Freely available online

PLoS COMPUTATIONAL BIOLOGY

Education

A Quick Guide to Organizing Computational Biology Projects

William Stafford Noble[1,2]*

1 Department of Genome Sciences, School of Medicine, University of Washington, Seattle, Washington, United States of America, 2 Department of Computer Science and Engineering, University of Washington, Seattle, Washington, United States of America

Introduction

Most bioinformatics coursework focuses on algorithms, with perhaps some components devoted to learning programming skills and learning how to use existing bioinformatics software. Unfortunately, for students who are preparing for a research career, this type of curriculum fails to address many of the day-to-day organizational challenges associated with performing computational experiments. In practice, the principles behind organizing and documenting computational experiments are often learned on the fly, and this learning is strongly influenced by personal predilections as well as by chance interactions with collaborators or colleagues.

The purpose of this article is to describe one good strategy for carrying out com-

understanding your work or who may be evaluating your research skills. Most commonly, however, that "someone" is you. A few months from now, you may not remember what you were up to when you created a particular set of files, or you may not remember what conclusions you drew. You will either have to then spend time reconstructing your previous experiments or lose whatever insights you gained from those experiments.

This leads to the second principle, which is actually more like a version of Murphy's Law: Everything you do, you will probably have to do over again. Inevitably, you will discover some flaw in your initial preparation of the data being analyzed, or you will get access to new data, or you will decide that your parameterization of a particular model was not

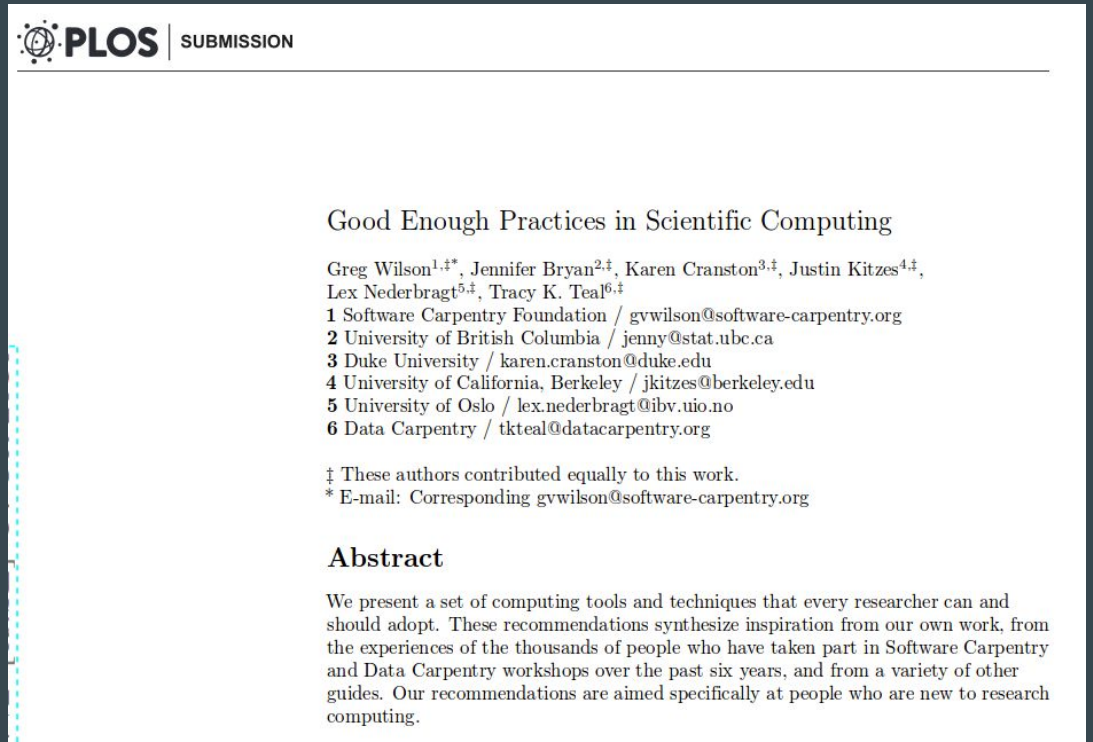
under a common root directory. The exception to this rule is source code or scripts that are used in multiple projects. Each such program might have a project directory of its own.

Within a given project, I use a top-level organization that is logical, with chronological organization at the next level, and logical organization below that. A sample project, called `msms`, is shown in Figure 1. At the root of most of my projects, I have a `data` directory for storing fixed data sets, a `results` directory for tracking computational experiments peformed on that data, a `doc` directory with one subdirectory per manuscript, and directories such as `src` for source code and `bin` for compiled binaries or scripts.

Within the `data` and `results` directories, it is often tempting to apply a similar,

8. Organization

Your research and code will be hard to maintain, explain, and reproduce



PLOS | SUBMISSION

Good Enough Practices in Scientific Computing

Greg Wilson[1,‡*], Jennifer Bryan[2,‡], Karen Cranston[3,‡], Justin Kitzes[4,‡], Lex Nederbragt[5,‡], Tracy K. Teal[6,‡]
1 Software Carpentry Foundation / gvwilson@software-carpentry.org
2 University of British Columbia / jenny@stat.ubc.ca
3 Duke University / karen.cranston@duke.edu
4 University of California, Berkeley / jkitzes@berkeley.edu
5 University of Oslo / lex.nederbragt@ibv.uio.no
6 Data Carpentry / tkteal@datacarpentry.org

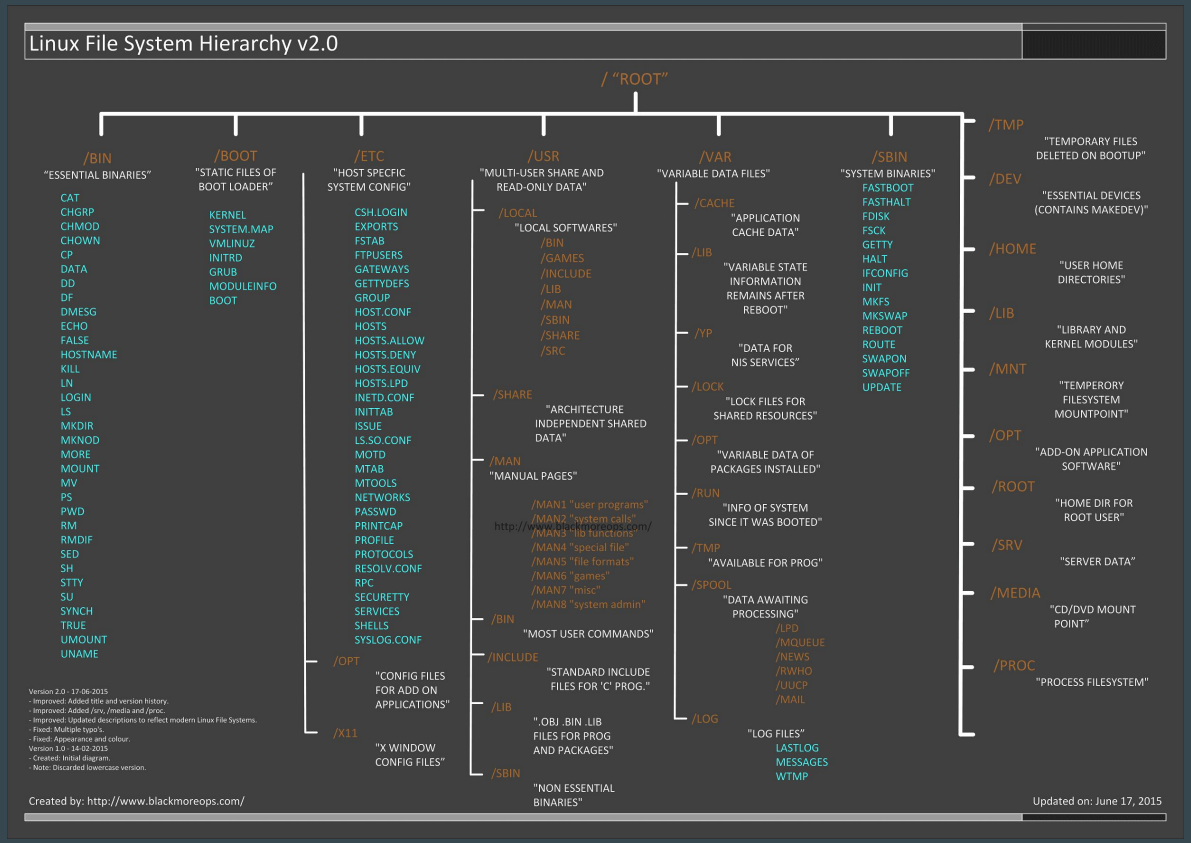
‡ These authors contributed equally to this work.
* E-mail: Corresponding gvwilson@software-carpentry.org

Abstract

We present a set of computing tools and techniques that every researcher can and should adopt. These recommendations synthesize inspiration from our own work, from the experiences of the thousands of people who have taken part in Software Carpentry and Data Carpentry workshops over the past six years, and from a variety of other guides. Our recommendations are aimed specifically at people who are new to research computing.

8. Organization

Your research and code will be hard to maintain, explain, and reproduce



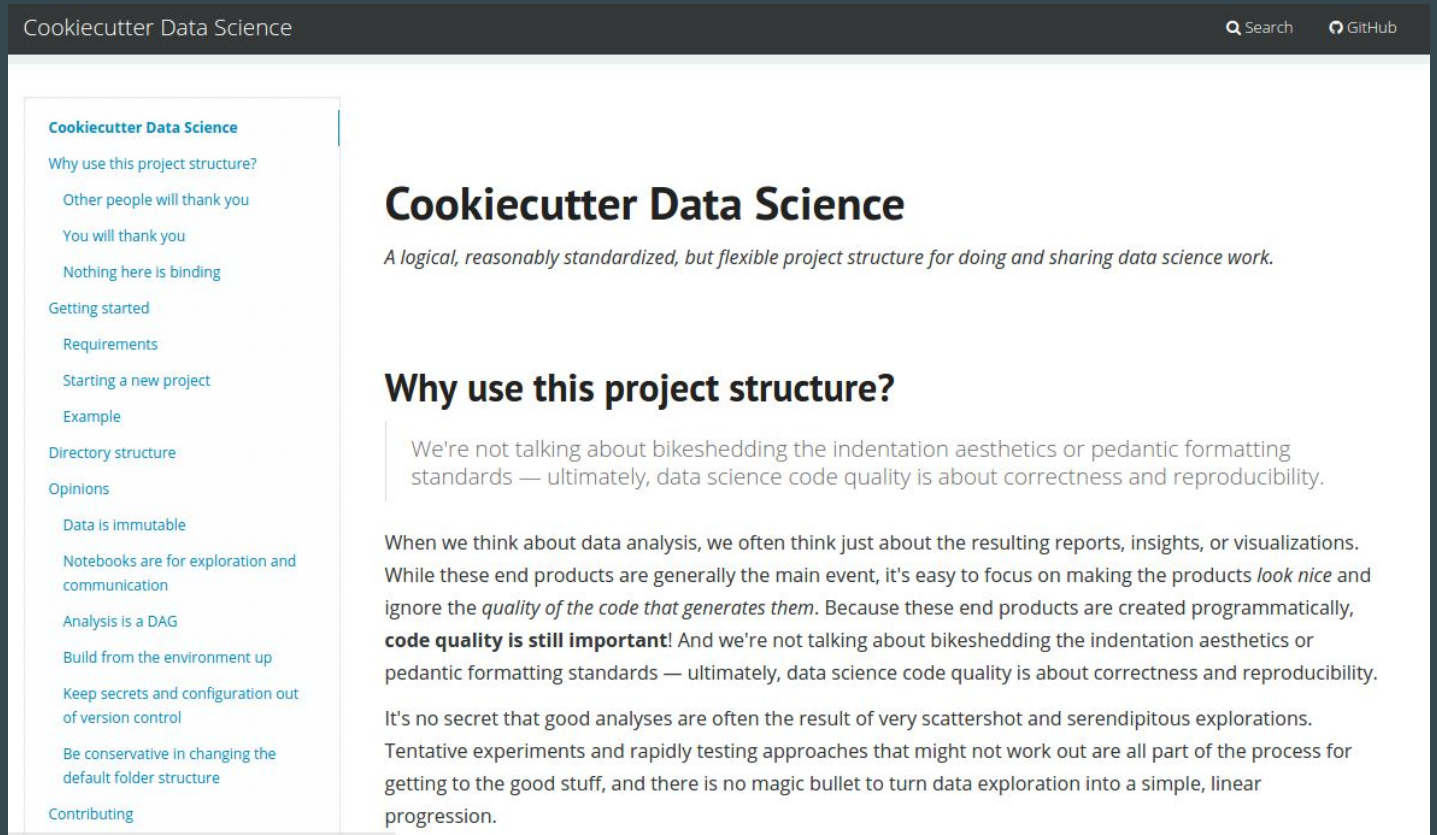
8. Organization

Your research and code will be hard to maintain, explain, and reproduce

```
# an example Django project layout
myproject/
    manage.py
    myproject/
        __init__.py
        urls.py
        wsgi.py
        settings/
            __init__.py
            base.py
            dev.py
            prod.py
    blog/
        __init__.py
        models.py
        views.py
        urls.py
    static/
         css/
         js/
     templates/
         base.html
         index.html
     requirements/
         base.txt
         dev.txt
         prod.txt
```

8. Organization

Your research and code will be hard to maintain, explain, and reproduce

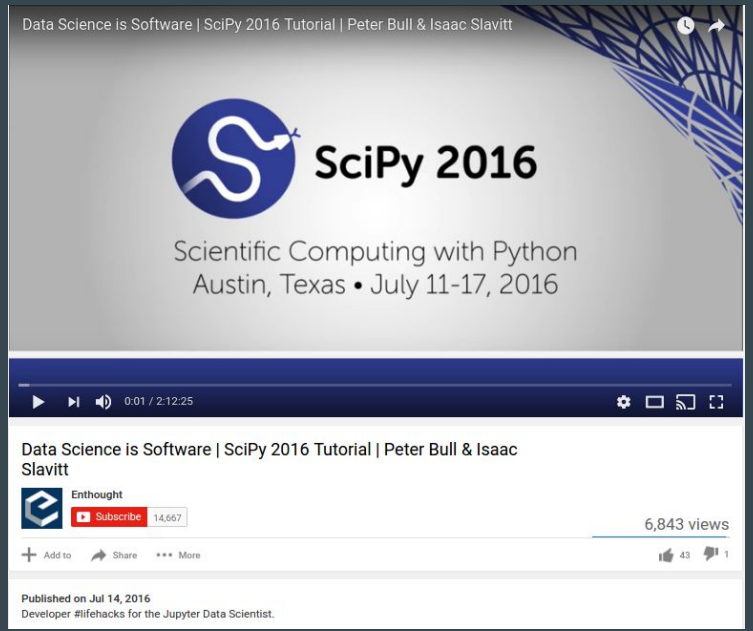


8. Organization

Your research and code will be hard to maintain, explain, and reproduce

Ways to avoid this

- **Start** organized and **maintain discipline**
- **Write the documentation as you go**
- Pull configuration out to specific config files
- Deterministic data flows
    - Everything should be buildable/trainable/runnable from **one interaction**
    - Data is immutable
    - Notebooks are for exploration and communication, not repetitive workflows
    - Keep secrets out of version control
    - USE VERSION CONTROL
- Consider using the Cookiecutter Data Science template
- More unsolicited opinions here: youtu.be/EKUy0TSLg04

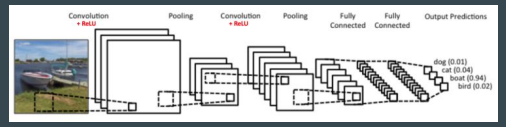


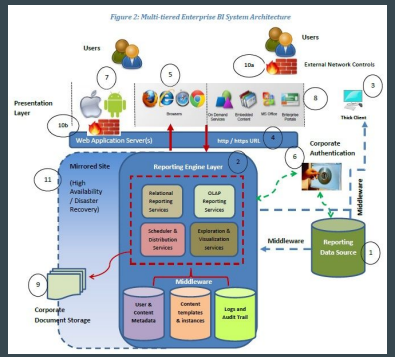
9. Simplicity

Your solution will be overly complicated and under documented

Problems

- What clients say they want





- What they actually want

9. Simplicity

Your solution will be overly complicated and under documented

Ways to avoid this

- Use the least complicated algorithm or model that gets the job done
- Consider using “boring” (read: standard, well understood) tools and libraries
- Think about who will have to use and maintain your code

“Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?”
— Brian Kernighan, The Elements of Programming Style, 2nd edition, chapter 2

“The central enemy of reliability is complexity. Complex systems tend to not be entirely understood by anyone. If no one can understand more than a fraction of a complex system, then, no one can predict all the ways that system could [fail].”
— Geer et al., 2003

10. Delivery

You will not communicate your solution clearly enough

Problems

- What you hand over is surprising or unexpected
  - “We don’t have any GPUs...”
  - “Oh, we don’t have any Linux servers...”
- They don’t understand how it works because
  - You didn’t explain the assumptions
  - You didn’t explain the models or algorithms
- They can’t run your code because
  - It isn’t packaged well
  - They have no idea where to “enter” the code
  - Your results aren’t reproducible
  - You didn’t document the requirements so they can’t get a working dev environment

10. Delivery

You will not communicate your solution clearly enough

Ways to avoid this

- Refer back to the scoping doc (you did do this, *right*?)
- Review the overarching goals of the project
- Describe your process, including challenges, assumptions, and modeling decisions
- Show your solution
- Describe “future work”
- Know your audience
    - Technical or non-technical?
    - **Use plain, clear language**

A few intangibles

- Public service announcement from Pavlos and the teaching staff:
  - Please show up on time
  - Please dress appropriately
  - Be professional
  - Plan ahead (send calendar invites, confirm times, find a quiet place, reserve rooms)
  - Practice what you're going to present in advance
- Framing as it relates to your career
  - "coder" versus "technical professional who solves business problems"
  - "cost center" versus "profit center"

Questions?

Isaac Slavitt
isaac@drivendata.org
AC 297r, Spring 2017