Anatomy of a Data Science Project

Lecture 2, DSC 180A

Announcements

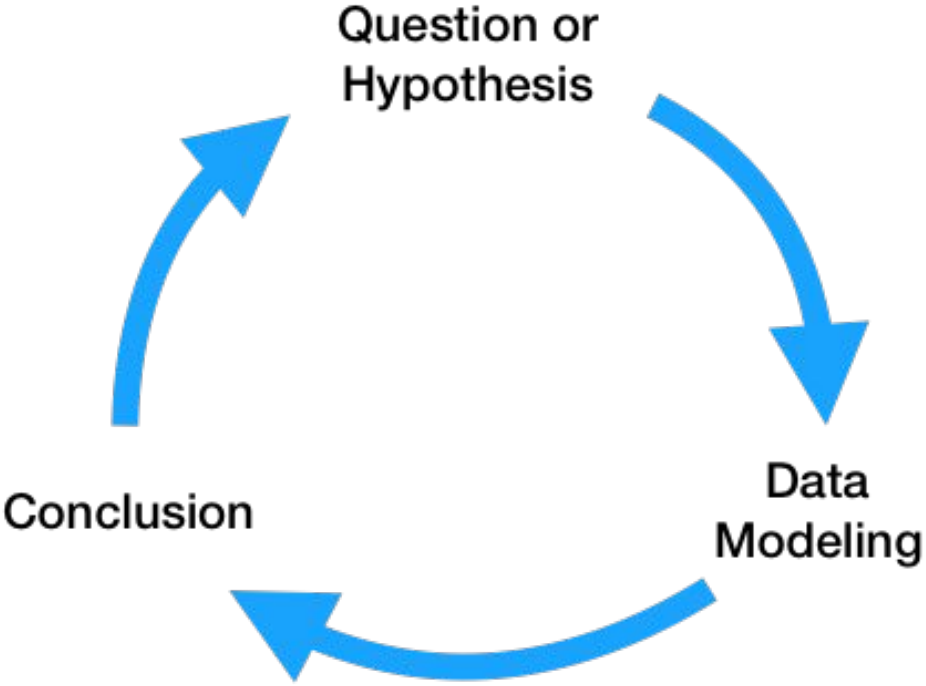
Friday Sections -- what are they for? Mandatory or not?

Lecture (Methodology) HW -- release first HW Tuesday. Do it in Friday lab.

First assignment -- “Data and the DGP” up soon on Domain webpage.

Career Readiness Survey

Data Science Lifecycle



The code for an investigation must:

- Be flexibly written.
- Clearly documented.
- Accessible to others.

In order to adapt to successive iterations through the lifecycle.

The *Real* Data Science Lifecycle

Poorly developed code results in:

- Fewer iterations and slower progress on the project
- Higher likelihood of mistakes in the results
- Difficult to understand conclusions
- The project fading into obscurity…

Better code ⇒ higher chance of success

Tools/Libraries for Managing Project Components

Managing Project Components

Too many tools to learn and new ones everyday!

Instead, learn the core issues:

- What contract does one component need to speak to another?
- Maximize the isolation of each component to enable easy code changes.
- Components relationship to computational graph:
  - When to recompute a step...
  - When can steps run in parallel?
- How can different components scale as the project or data grows in scope?
- Best use of ‘configuration files’ to manage and track iterations.

Domain Research

Domain Research informs the bulk of a project's structure:

- Why you made certain design decisions
- The context behind the quantities of interest
- The subset and kind of data used
- The cleaning logic and any simplifications in modeling

Understanding these choices requires:

- Extensive narrative documentation
- Code comments to explain specific instances requiring context

Documenting Domain Research

- Markdown for exposition
  - GitHub Rendered Markdown (e.g. README.md)
  - Jupyter Notebook Reports (embedded markdown)
  - Python Library auto-build documentation (Sphinx)
  - R-markdown
  - Hosted Webpage
- Code Comments
  - When domain expertise justifies detailed coding decisions.
  - E.g. `... # filter non-voters in data, as case doesn't apply`

Question / Hypothesis

The question being investigated changes as a project evolves.

When the questions are similar:

- Write code parameterized to handle *all* questions simultaneously.
- Each choice of parameters ⇔ a different question.
- Parameters are kept in configuration files (e.g. json, ini, cfg, yaml).
  - E.g. Configuration file is instructions to run 10 instances of the investigation, for 10 different questions, simultaneously on 10 different servers (e.g. questions by year for 2010-2020)
- Strive to write (and rewrite) code to parameterize many possible questions!

Data ETL (extract-transform-load)

As a project evolves, the data may change or new data may be added…

- Keep schema and parameters in configuration (beware: magic numbers!)
- Is the source stable? (DB, API, Scrape)
  - Separate the data ingestion from any transformations
- Unnecessary computation wastes time and resources:
  - Problem: don’t re-pull data because your cleaning code changed!
  - Answer: write intermediate files to disk (or a personal file store)
- Write processing code that is agnostic to the computer running it:
  - ‘git clone => run’ on your laptop or DataHub; scale up only when needed
  - Even better, is the intermediate data accessible from both? (and when do you want that?)

Model Building

Choosing the best model involves exploring *many* parameters!

- Keep track of parameters and results in configuration files.
- Use frameworks that enable ‘pipelining’ (e.g. sklearn, spark, tensorflow)
- Often need to scale-up processing on different servers

Continued Prediction / Inference

Once a model is built, a project often still lives on…

- Is the finished model being used for live predictions?
  - How does a scikit-learn model get called by a Java backend website?
  - `mdl.predict` may be called via HTTP-requests (RESTful interface).
- Model Quality Reporting:
  - If inference: is the project easily rerun on a new dataset? Can it be automated?
  - For live predictions: are the distributions of predictions stable? What is their quality? Can you create automated reporting?
- What if someone else uses their model? Does it work?
  - Package as a python module or in a Docker Container.

Conclusions / Decision / Report

Once a model is built to your satisfaction…

- Document and explain your results (e.g. in markdown).
- Justify any decisions made from the model
- Create reporting from the model
  - Update the reporting from new data, by rerunning project from scratch.
  - Email the compiled reporting (markdown=>HTML) automatically from a server.

Your project will fade into GitHub obscurity without good documentation!

A Template for Encouraging Best Practices

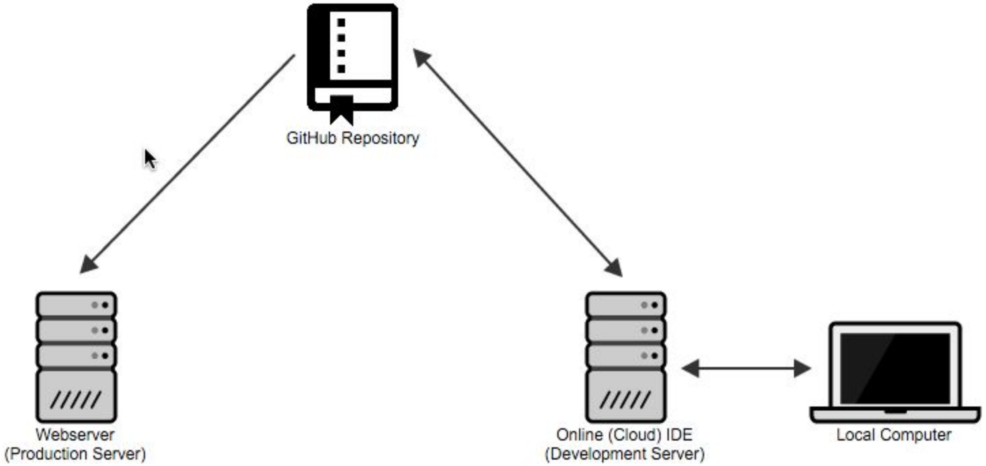
We will follow general opinion of Cookie cutter data science.

An organized project structure encourages a clean project!

```
├── LICENSE
├── Makefile           <- Makefile with commands like `make data` or `make train`
├── README.md          <- The top-level README for developers using this project.
├── data
│   ├── external       <- Data from third party sources.
│   ├── interim        <- Intermediate data that has been transformed.
│   ├── processed      <- The final, canonical data sets for modeling.
│   └── raw            <- The original, immutable data dump.
│
├── docs               <- A default Sphinx project; see sphinx-doc.org for details
│
├── models             <- Trained and serialized models, model predictions, or model summaries
│
├── notebooks          <- Jupyter notebooks. Naming convention is a number (for ordering),
│                         the creator's initials, and a short `-` delimited description, e.g.
│                         `1.0-jqp-initial-data-exploration`.
│
├── references         <- Data dictionaries, manuals, and all other explanatory materials.
│
├── reports            <- Generated analysis as HTML, PDF, LaTeX, etc.
│   └── figures        <- Generated graphics and figures to be used in reporting
│
├── requirements.txt   <- The requirements file for reproducing the analysis environment, e.g.
│                         generated with `pip freeze > requirements.txt`
│
├── setup.py           <- Make this project pip installable with `pip install -e`
├── src                <- Source code for use in this project.
│   ├── __init__.py    <- Makes src a Python module
│   │
│   ├── data           <- Scripts to download or generate data
│   │   └── make_dataset.py
│   │
│   ├── features       <- Scripts to turn raw data into features for modeling
│   │   └── build_features.py
│   │
│   ├── models         <- Scripts to train models and then use trained models to make
│   │   │                 predictions
│   │   ├── predict_model.py
│   │   └── train_model.py
│   │
│   └── visualization  <- Scripts to create exploratory and results oriented visualizations
│       └── visualize.py
│
└── tox.ini            <- tox file with settings for running tox; see tox.testrun.org
```

The Data Scientist’s Work Environment

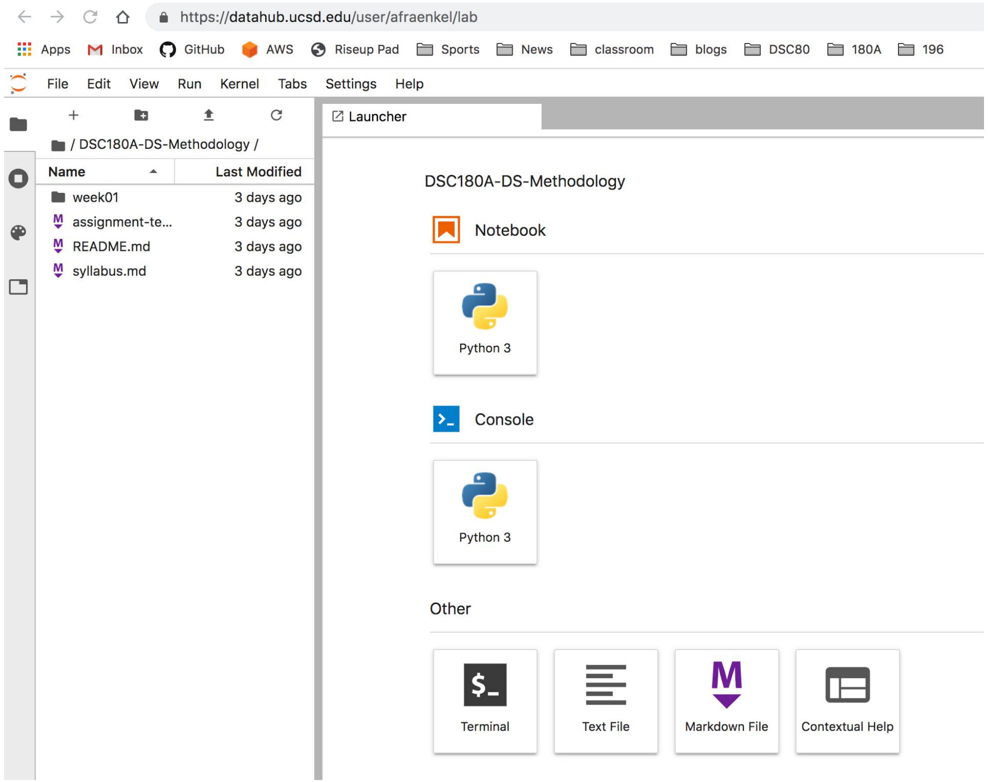
- Local Development on laptop
  - Cheap and convenient
  - Higher efficiency in your ‘home’ environment
- Development on Server
  - More RAM and CPU available
  - Long running programs
- Need ability to do both!
  - Code must run both places
  - Both must have ‘latest’ versions



Remote Servers in DSC 180A

[UCSD datahub](https://datahub.ucsd.edu) as you know it:

- Jupyter portal just a server.
- Open a JupyterLab IDE by natigating to:
  - https://datahub.ucsd.edu/user/<NAME>/lab
- Open a Terminal from Jupyter to pull from git, etc…
- DataHub shuts down after 30 minutes of client inactivity =(



Remote Servers in DSC 180A

Accessing DataHub DSMLP through the Terminal.

- See Documentation: http://go.ucsd.edu/2CZladZ
- Use ssh to log-in remotely to a datahub server.
    - Personal and persistent disk space
    - Shared disk space among entire class (request from staff)
    - Ability to request CPU/GPU and more RAM
    - Ability to specify and update custom environment and software
    - Kick-off long running programs
- Use your own personal server; up your cloud-computing skills!

Logging into DSMLP Servers

- `ssh` user@dsmlp-login.ucsd.edu (your school username)
  - Logs you into your home directory in a jump-box.
- `launch-scipy-ml.sh` (launches a server with 8GB RAM)
  - Your home directory is also available from here (called a “Volume”)
  - Other scripts `launch-XXX-XXX.sh` launch different server configurations
  - Can open a Jupyter Notebook from this server, if on campus network or VPN
- The launch scripts can take a Dockerfile that configures the environment.