



DSC Capstone Sequence

Lecture 06
Data Science Software
Development



Lecture Outline

- Anatomy of a DS Project, Revisited
- Build Scripts
- Configuration vs. Code



Anatomy of a DS Project, Revisited



Anatomy of a DS project, continued...

Last Time:

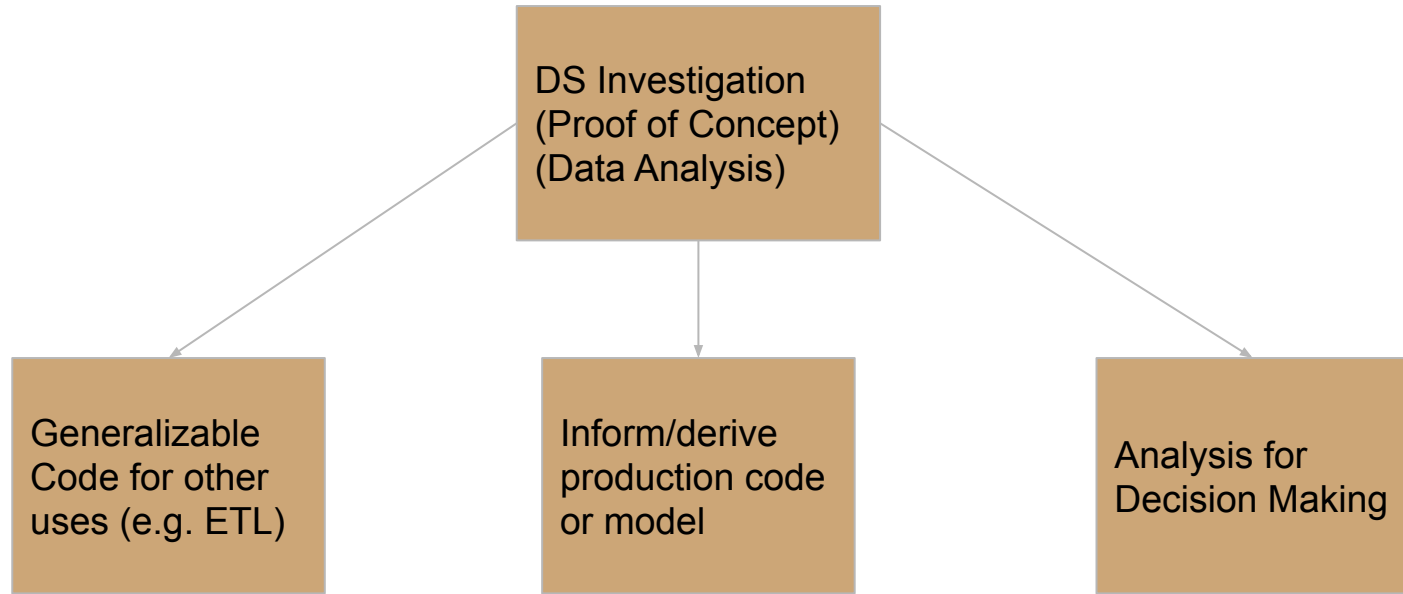
- Discussed the structure of a DS project
- Which behaviors that structure is meant to encourage.

This Time:

- Discuss typical ways DS projects are used
- Understand in more detail, how two components help w/usage:
 - Build scripts (helping reproduce and update results)
 - Configuration files (helping parameterize experiments)

Focusing on a DS investigation

How is a data-driven project used?

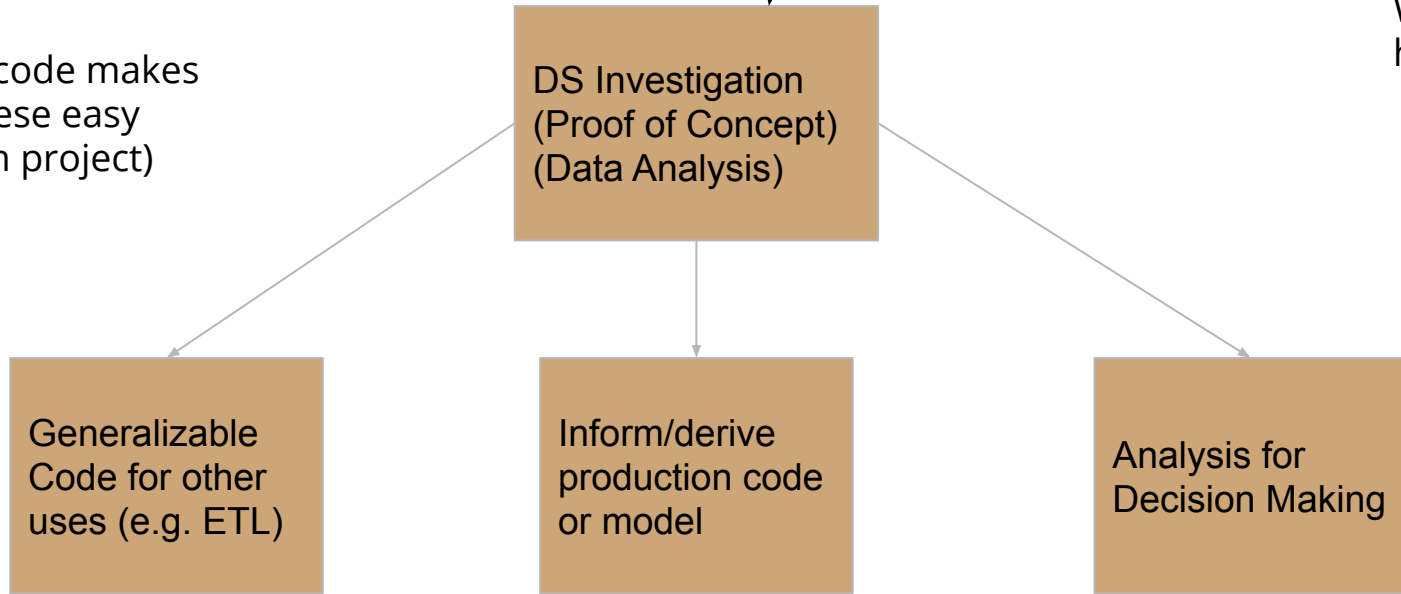


Focusing on a DS investigation

How is a data-driven project used?

Good project code makes
developing these easy
(may do this in project)

We focus
here!



Structuring a DS Project

A project's code should be reproducible, flexible, and clear.

A project should be:

- Understandable by others (verifiably correct).
- Usable by others (extensible).
- Adaptable by you (flexible).

Using a template (e.g. Cookie-Cutter) encourages these practices!

The Template Organizes a DS Investigation

All Data Science investigations connect to:

- Data-driven analysis, to further understanding and inform decision making.
- Data-driven development, to build something that takes action using the analysis.

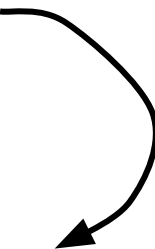
Goal: your data science project is a *usable artifact* that leads to action; its correctness is accountable to others.

Example: Building a Recommender System

Project: develop and train a recommender system.

Primary Output: a (trained) model

These will belong to different repositories!



How the project is used:

- model (output) is executed on a website to give recommendations.
- model architecture (code) may become its own general purpose library

The project also includes:

- A data analysis (e.g. model evaluation) that justifies the project's worth
- Reproducible build, which automates updates to model (e.g. new data).

Example: Develop Browser Instrumentation

Project: develop browser instrumentation for measuring engagement

Primary Output: parameters and logic for measuring browsing behavior.

How the project is used:

- A set of javascript functions to run on website.

The project also includes:

- A data analysis that informs JS code development and justifies the project's budget.

Example: HR Study in Hiring

Project: Perform a study to determine how many people to hire.

Primary Output: a report with recommendations on hiring

How the project is used:

- Analyses are collected into a report for decision making about business.

The project also includes:

- Transparent secondary data analyses (how you arrived at your decision)
- Reproducible build, for easy updating with new information

Project Flexibility

As a bonus, flexible project design makes the process of extracting "useful components" easier!

Examples:

- Create a reusable library of ETL code for general use (from project code)
- New ML or Feature library for anyone to use

When working on your project, it may be reasonable to create a new repository for extracted "general use libraries" and merely import it into existing project.

Expectations for a project, according to usage

Building a Project (Usage Expectation #1)

- Can someone else 'git clone' your project, run the code, and understand the results with ~1 command?
- This requires identifying the logic of the project ("big steps") and separating it from the implementation/details ("small steps")

Flexible Development (Usage Expectation #2)

- Easy code reuse (notebooks, build scripts, new project -- use library code)
- This requires separating code, from data, from details likely to change (configuration)



Build Scripts



Build Scripts

A build script:

- Runs code in project to "build" desired output.
- Strings together library code.
 - Contains "over-arching" logic only! It should not be complicated.
- Build scripts are standard for installing software:
 - Makefiles (C / C++ / Bash) -- also great for multilingual data science projects!
 - Ant/Maven/Gradle (Java)
 - `setup.py` (general python)
 - `luigi` (python data pipelines)

We will create our own build script `run.py` (or `run.sh`, `run.R`, etc)

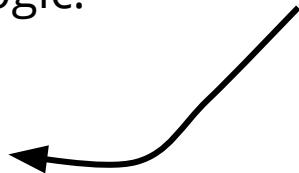
Targets in Build Scripts

- A *target* specifies what to build.
- Usually a string describing the output: `all`, `data`, `analysis`, `train`
 - You should create a target for all desired outputs
- Executing `python run.py data` on the command-line prepares data for the project (e.g. downloads data and runs ETL code)
- Executing `python run.py features` should build features for the project.
- Target names are specific to your project and logic; a target corresponds to a major "step" in the workflow (often an intermediate dataset).
 - There are also "standard" target names.

Example Code development workflow

- Write data ingestion code in `src/etl.py`
- Import `etl` in `run.py` to include data logic:
 - `python run.py data` then "builds the data"
- work in notebooks to develop features
- write finished features in `src/features.py`
- add feature creation call to `run.py`
- `python run.py features` "builds features"
 - w/o rebuilding data, preferably!

When code is no longer "experimental", move it from notebook to `src` and create a target in the build script for its output.



Standard Targets

- ``all`` runs *all* targets (`python run.py all`)
- ``test`` runs all targets on test data (`python run.py test`)
 - We will work on this more in a later week
- ``clean`` deletes all build files (`python run.py clean`)
 - Reverts to a *clean* repository

Build Script Features

- String together library code to create output for project (Basic)
- Define targets that both:
 - clarify the logic of the project, and (basic)
 - build intermediate states of the project (basic)
- Does not do unnecessary computation (intermediate)
 - if data already exists to step K , start at step $K+1$
- Runs tasks in parallel, when possible (advanced)
 - Makefiles do this!

Build Script References

- Makefiles for Data Science
 - <https://bost.ocks.org/mike/make/>
 - <http://zmjones.com/make/>
- Python Workflow managers (e.g. Luigi, Airflow)
 - <https://towardsdatascience.com/data-pipelines-luigi-airflow-everything-you-need-to-know-18dc741449b7>
- Better managing targets for run.py with argparse:
 - <https://docs.python.org/3/howto/argparse.html>



Configuration vs Code



Flexible Project Development

- Build scripts lay out "big picture" logic of a project
- Library code contains implementation logic of the pieces of the build.
- What if the implementation details are likely to change?
 - Parameterize this details as inputs to the library functions
 - Input parameters <=> Configuration
 - Pass configuration files into build script to quickly change input

What is configuration (detail) vs code (logic) depends on context.

- Ask: is the value likely to change as the project evolves?

Example: Code with no Configuration

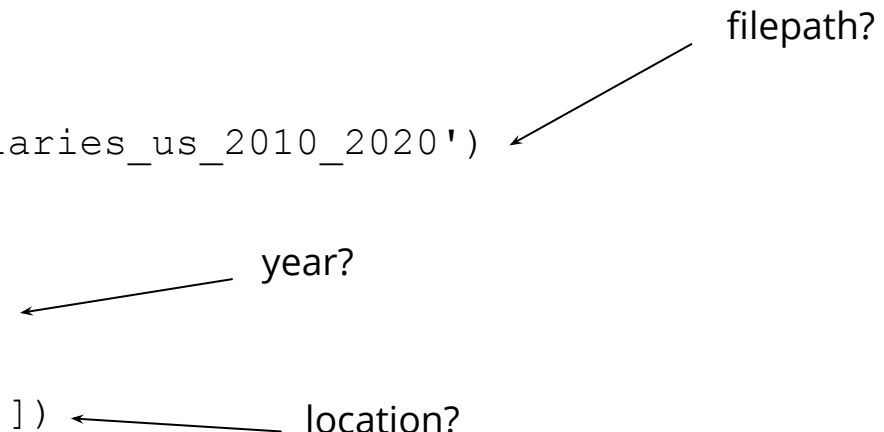
```
def get_data():  
    data = pd.read_csv('data/raw/salaries_us_2010_2020')  
    data = (data  
            .loc[df['year'] == 2015]  
            .loc[df['loc'] == 'CA'])  
    return data
```

What code represents logic unlikely to change? (code)

What may change as project evolves? (configuration)

Example: Code with no Configuration

```
def get_data():  
    data = pd.read_csv('data/raw/salaries_us_2010_2020')  
    data = (data  
            .loc[df['year'] == 2015]  
            .loc[df['loc'] == 'CA'])  
    return data
```



What code represents logic unlikely to change? (code)

What may change as project evolves? (configuration)

Example: configuration as input variables

```
def get_data(fp, years, locs):  
    data = pd.read_csv(fp)  
    data = (data  
            .loc[df['year'].isin(years)]  
            .loc[df['loc'].isin(locs)])  
  
    return data
```

- Natural to use and experiment w/new combinations in notebooks
- Can change project scope easily/quickly (change *input*, not code!)

Configuration files provide input to library code

- Keep input params organized in config files
- Build scripts run library code; pass inputs read from config files.

src/etl.py

```
def get_data(fp,
             years, locs):
    ...
    return data
```

config/etl.json

```
{
  "fp": "data/raw",
  "years": [2015],
  "locs": ["CA"]
}
```

run.py

```
...
cfg = json.loads(
    open(...))
get_data(**cfg)
```

Configuration Observations

- tracking of your experiments (build multiple simultaneously!)
- Contains inputs to functions in library code
 - make the keys to your configuration are the input variables for functions
- Allows user/observer to understand import quantities in project.
- Easy to make quick changes to a project

Application 1: Working on multiple computers

- Project using manually downloaded dataset:
 - On local laptop: `/home/user/datasets/ds.csv`
 - On Server: `/teams/DSC180A_FA20/00coursestaff/ds.csv`
- Ideal to have data accessible in `<project>/data` (same in both places)
- Solution create a symlink (shortcut) from data location to project directory
 - Bonus: multiple people on server can share the same dataset
 - Each person thinks the data lives in their project directory!

Application 1: Working on multiple computers

- Solution create a symlink (shortcut) from data location to project directory

src/env.py

```
def setup_data(ddir):  
    ...  
    # create data dir  
    ...  
    # create symlink  
  
    return
```

config/env-local.json

```
{  
    "ddir": "/home/... /ds.csv",  
}  
  
config/env-dsmlp.json  
  
{  
    "ddir": "/teams/... /ds.csv",  
}
```

run.py

```
...  
cfg = json.loads(  
    open(...))  
setup_data(**cfg)
```

Application 2: Model Parameters

- Config files are useful to record final model parameters
- If you plan on doing a systematic parameter search (good!)...
 - Don't create 10k json files by hand!
 - Use the config file to specify the *bounds* of the search.
- The config files will parameterize different directions of investigation
 - (Ones that aren't easy to codify in clear, logical code.

See example project templates for illustration:

- <https://github.com/DSC-Capstone/project-templates>