Structuring a DS Project

Lecture 3, DSC 180A

Announcements

Assignment 1 Comments: Comment on Grading; Code Review Friday.

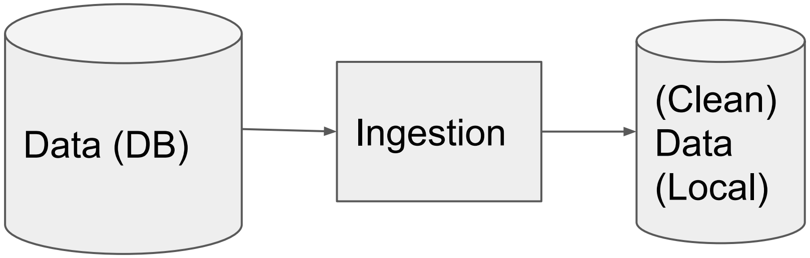
Working on projects:

- Work a little every day (things always go wrong!)
- Work in parallel on multiple pieces
- Don’t let a single step be a ‘blocker’.

Data Ingestion

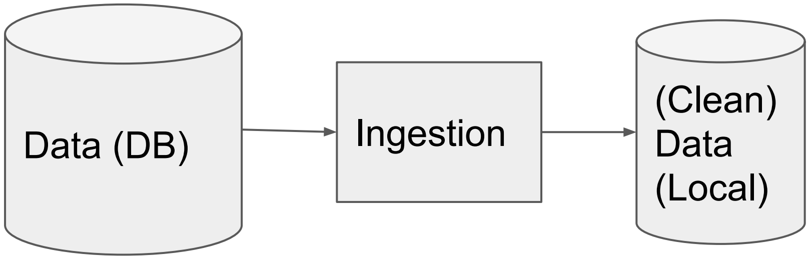
- Task: get data from internet to computer.
- Make it easy to (incrementally) change data ingested.
- Rerun to ‘refresh’ with new data.
- Run data pull on different servers to reproduce results

How to organize code to achieve this?



Data Ingestion: `etl.py`

- Library code: contains functions for importing by other processes.
- Good for interactive use; reusable.
- Written as generically as practicable.
- Contains logic not necessary for a *consumer* of the project to know.
- Contains data collection logic other *developers* might want to expand on, when forking a project.
- Library code know nothing of what calls it!




```
Project
├── README.md
├── data-params.json
├── etl.py
└── run.py
```

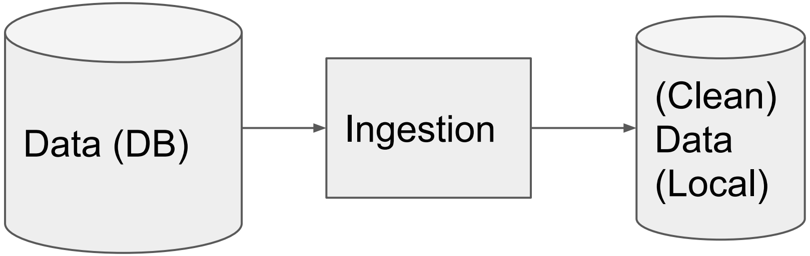
Data Ingestion: `etl.py`

- Library code: contains functions for importing by other processes.
- Written as generically as practicable.
- Good for interactive use; reusable.
- Contains logic not necessary for a *consumer* of the project to know.
- Contains data collection logic other *developers* might want to expand on, when forking a project.
- Library code know nothing of what calls it!

```
'''
etl.py contains functions used to download tables for different
teams and years.
'''

def get_season(team, year):
    '''
    return a table of season statistics for a
    given team and year.
    '''
    ...
    return ...


def get_data(years, teams, outdir):
    '''
    downloads and saves tables at the specified output
directory
    for the given years and teams.

    :param: years: a list of seasons to collect
    :param: teams: a list of teams to collect
    :param: outpath: the directory to which to save the data.
    '''
    ...
    return
```

Data Ingestion: `data-params.json`

- Configuration: parameters for different investigations and experiments.
- E.g. Parameterize across time/space.
- Used by the *consumer* of the project. Shouldn't require a knowledge of the source code!
- Helps log the results of different experiments.




```
Project
├── README.md
├── data-params.json
├── etl.py
└── run.py
```

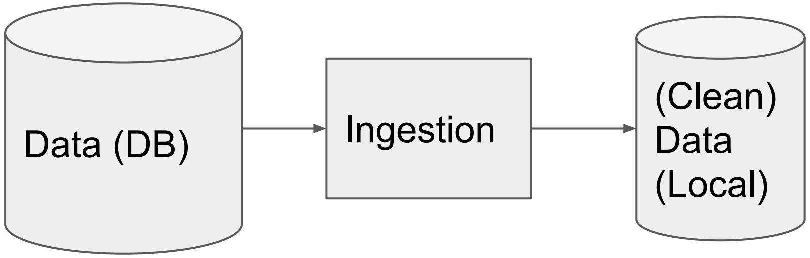
Data Ingestion: `data-params.json`

- Configuration: parameters for different investigations and experiments.
- E.g. Parameterize across time/space.
- Used by the *consumer* of the project. Shouldn't require a knowledge of the source code!
- Helps log the results of different experiments.

```
{
    "years": [2015, 2016, 2017, 2018, 2019],
    "teams": ["sfo", "gnb"],
    "outpath": "data/raw"
}
```

Data Ingestion: `run.py`

- Script: Builds (common portions of) the project.
- Imports and runs library code: gives examples of *code usage*.
- Current: hand-made `run.py`
- Also possible to use specialized tools: python CLI (e.g. argparse), Makefiles, Maven, etc...




```
Project
├── README.md
├── data-params.json
├── etl.py
└── run.py
```

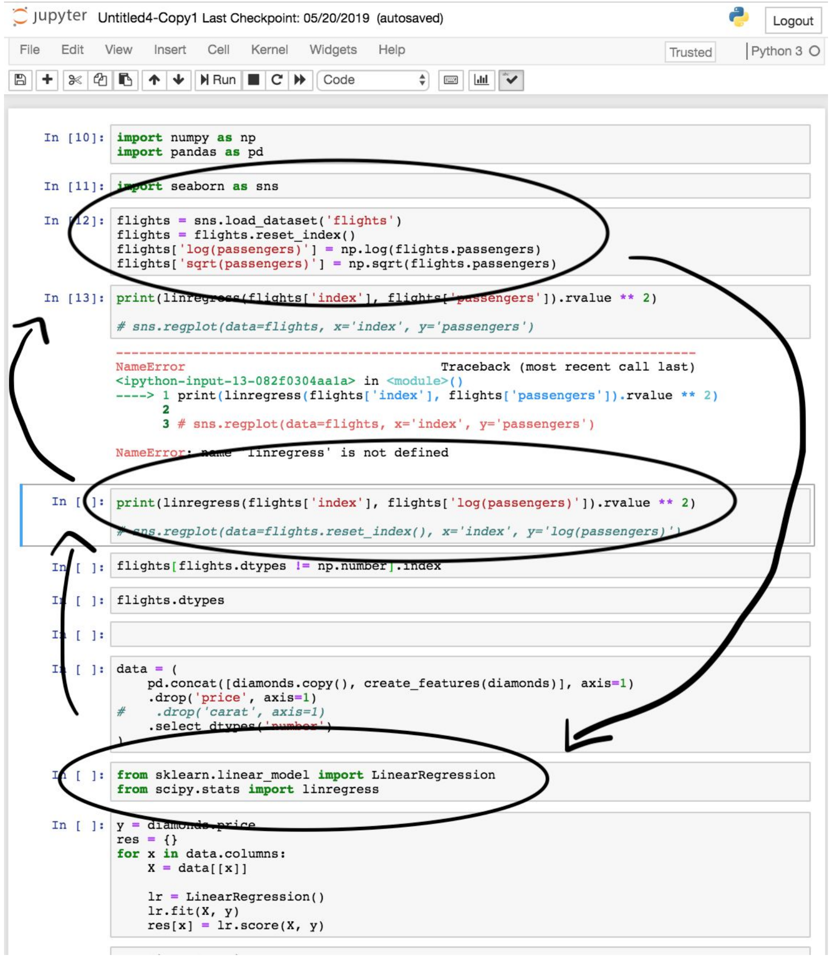
Data Ingestion: `run.py`

- Imports library code (`get_data`)
- Run as a script:
  - `python run.py data`
- Shebang: `#!/usr/bin/env python`
  - Specifies which python interpreter to use.
- `main` function strings together library functions with parameters in config.
- `__name__ == '__main__'...` returns true only when file is run as a script. Should only have minimal code inside.

```
#!/usr/bin/env python

import sys
import json

from etl import get_data


def main(targets):

    if 'data' in targets:
        with open('data-params.json') as fh:
            data_cfg = json.load(fh)

        # make the data target
        get_data(**data_cfg)

    return


if __name__ == '__main__':
    targets = sys.argv[1:]
    main(targets)
```

Project Structure:

- As project grows, so does code complexity!
- As a project grows, it becomes unclear:
    - how code should be run…
    - what the code *does*…
    - if the code is correct…



Why Care About Project Structure?

We're not talking about bikeshedding the indentation aesthetics or pedantic formatting standards — ultimately, data science code quality is about correctness and reproducibility.

Cookie Cutter Data Science

- Clear and consistent project organization encourages software development best-practices and readable code.
- Such habits yield more consistently correct code that’s more easily fixed and adapted to other tasks.

Why Care About Project Structure?

We're not talking about bikeshedding the indentation aesthetics or pedantic formatting standards — ultimately, data science code quality is about correctness and reproducibility.

Cookie Cutter Data Science

- Clear and consistent project organization encourages software development best-practices and readable code.
- Such habits yield more consistently correct code that’s more easily fixed and adapted to other tasks.
- We will follow the opinions of Cookie Cutter Data Science

An *Example* Project Template

```
├── .gitignore          <- files to keep out of version control (e.g. data/binaries)
├── run.py              <- run.py with commands like `make data` or `make train`
├── README.md           <- The top-level README for developers using this project.
├── data
│   ├── temp            <- Intermediate data that has been transformed.
│   ├── out             <- The final, canonical data sets for modeling.
│   └── raw             <- The original, immutable data dump.
├── notebooks           <- Jupyter notebooks (presentation only).
├── references          <- Data dictionaries, explanatory materials.
├── requirements.txt    <- For reproducing the analysis environment, e.g.
│                          generated with `pip freeze > requirements.txt`
├── src                 <- Source code for use in this project.
    ├── 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 make predictions
    │   ├── predict_model.py
    │   └── train_model.py
    └── visualization   <- Scripts to create exploratory and results oriented viz
        └── visualize.py
```

Results are Derived from Immutable Raw Data

- Data is immutable: *never* edit raw data
  - Raw data is always (re)ingested from elsewhere.
  - File-path may be a *symbolic link* (shortcut), if stored locally.
  - Raw data never changes => doesn’t need version control! (.gitignore)
- Final data is always reproducible from raw data (with run.py)
- Temp data holds data ‘useful to keep around’ for development, analysis, debugging, etc…

```
...
├── data
│   ├── temp             <- Intermediate data that has been transformed.
│   ├── out              <- The final, canonical data sets for modeling.
│   └── raw              <- The original, immutable data dump.
...
```

Notebooks are for Analysis and Communication

- Notebooks are great for communication, analysis, and initial development.
    - Use to create up-to-date, reproducible, static HTML reports.
- Complicated code in notebooks are hard to understand and don’t work well with version control and collaboration.
- Notebooks should:
    - Mostly call library functions in src, with very simple code logic.
    - Never “copy-paste” code between notebooks -- if it’s reusable, put it in a library function.
    - Name it something descriptive: `03-fraenkel-prelim-EDA.ipynb`

```
...
├── notebooks          <- Jupyter notebooks (presentation only).
...
```

Build from the Environment Up

- To reproduce a project from scratch, must also reproduce the computational environment on which it was run.
- requirements.txt contains all python libraries needed for running the project.
- `git clone project => mk virtualenv => pip install requirements.txt`
- When a project has more complicated requirements, may need to use container approach (e.g. Docker or Vagrant).

```
...
├── requirements.txt   <- For reproducing the analysis environment, e.g.
│                         generated with `pip freeze > requirements.txt`
...
```