W05D5

Midterm Project Kickoff

Instructor: Mark Cassar
Adapted from slides by: Eric Elmoznino

Outline for today

- Introductions
- Project descriptions
  - Predicting flight delays
  - Clustering NYC neighborhoods
- Project skeleton
  - How to plan the project
  - How to divide tasks
- GitHub essentials
- Evaluation criteria

Project descriptions

Predicting flight delays

- Supervised Learning:
    - **Regression Problem**: Predict delay of flights 1 week in advance
    - **(Stretch) Multiclass Classification**: Predict type of delay it will be
    - **(Stretch) Binary Classification:** Predict if the flight will be cancelled.
- Feature Engineering
- Sampling
- JDBC connection to a Postgres database
- Future role as **Data Scientist** or **Machine Learning Engineer**
- Repository

Predicting flight delays: the data

- Four tables:
    - ***flights***: Departure and arrival information for flights in US from 2018 and 2019
    - ***fuel_comsumption***: Consumption of different airlines from 2015-2019 aggregated per month
    - ***passengers***: Total passengers on different routes from 2015-2019 aggregated per month
    - ***flights_test***: Departure and arrival information for flights in US from January 2020.
- Other APIs you find (e.g. a weather API)
- Postgres database (credentials in compass activity):
    - Don’t: SELECT * FROM aviation without a WHERE clause
    - Do: Save result of query as CSV

Predicting flight delays: workflow

1. Pull a subset of the database and save as csv
2. Do some data exploration (guidance from this notebook)
3. Engineer features and train models (guidance from this notebook)
4. Submit results in the form of sample_submission.csv
    - use test data only at the very end

Clustering NYC neighborhoods

- Unsupervised Learning
  - **Clustering Problem**: Cluster using cultural, geographic, and transport features
  - **Insights**: See if clusters are predictive of economic indicators and/or demographics
- Data Visualization
- Using APIs for data enhancements
- JSON parsing
- Future role as **Data Analyst/Scientist**
- Repository

Clustering NYC neighborhoods: the data

- Parse nyc_geo.json into dataframe with:
    - Borough
    - Neighborhood
    - Latitude
    - Longitude
- Join data with features from APIs:
    - Cultural: Foursquare, Yelp, Google, Meetups
    - Transport: Uber
    - Other: NYC Open Data
    - Any others you think of

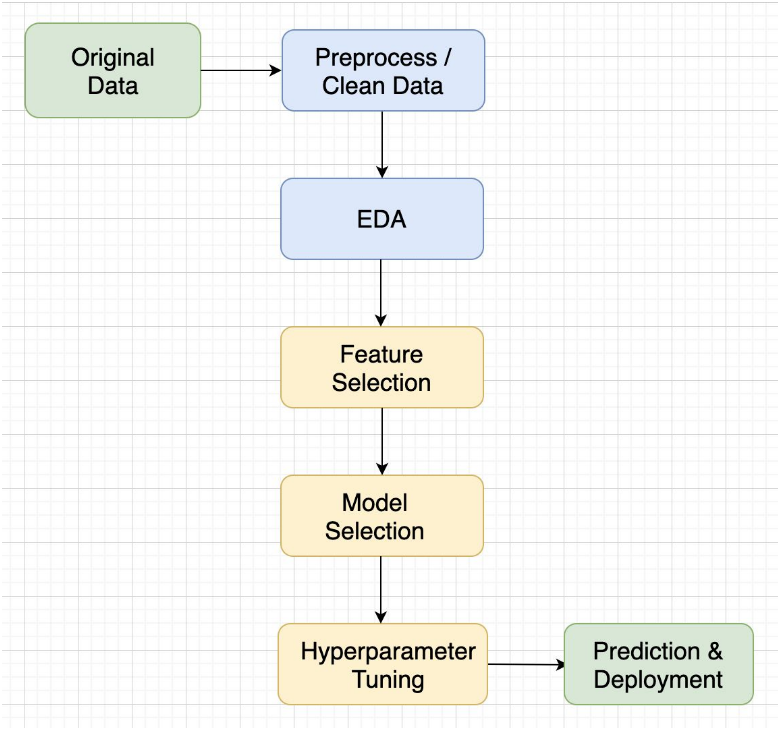
Clustering NYC neighborhoods: where to start

1. Parse neighbourhood JSON data
2. Explore APIs and join features to dataframe
3. Do some data exploration (guidance from this notebook)
4. Engineer features and train models (guidance from this notebook)
5. Visualize clusters and relationships to economic/demographic variables
6. Submit results as submission.csv with columns “neighborhood”, “cluster_id”

Project skeleton

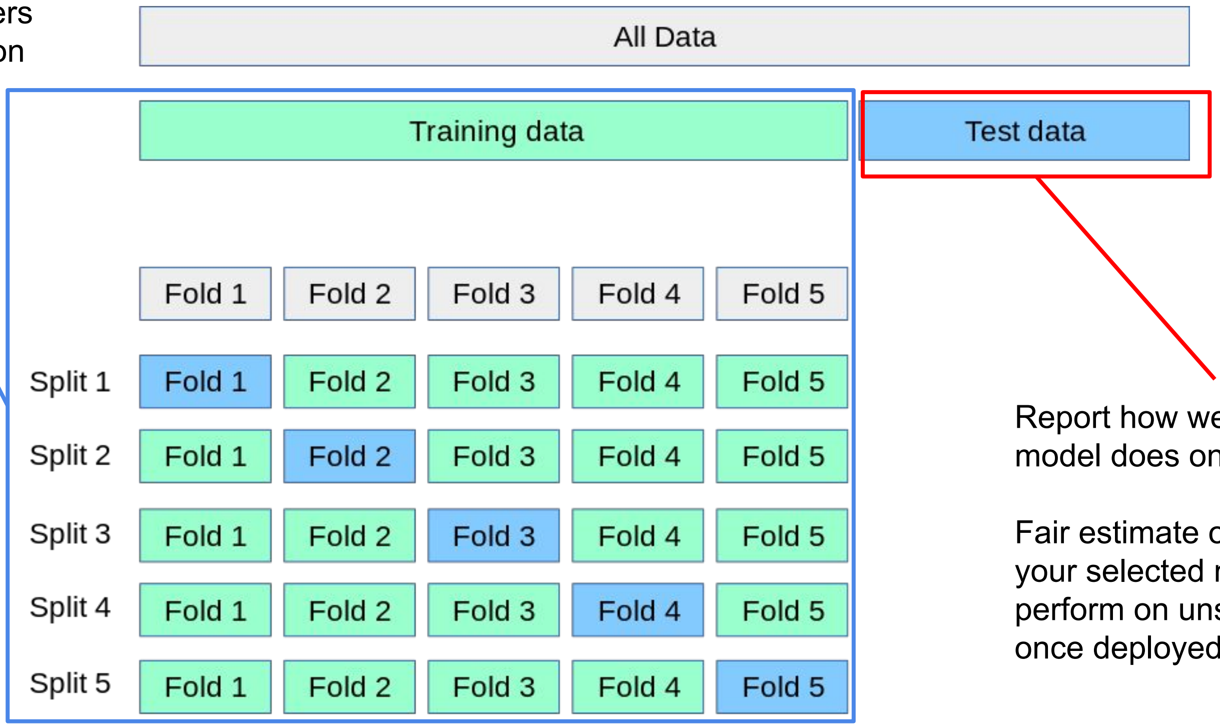
Workflow

- No exact recipe
- Only use exploratory analysis to inform decisions
- Model selection through cross-validation on the *training set*.
- **Test set untouched until submission**



Model selection

Find the best model+hyperparameters through cross-validation on the *train set*

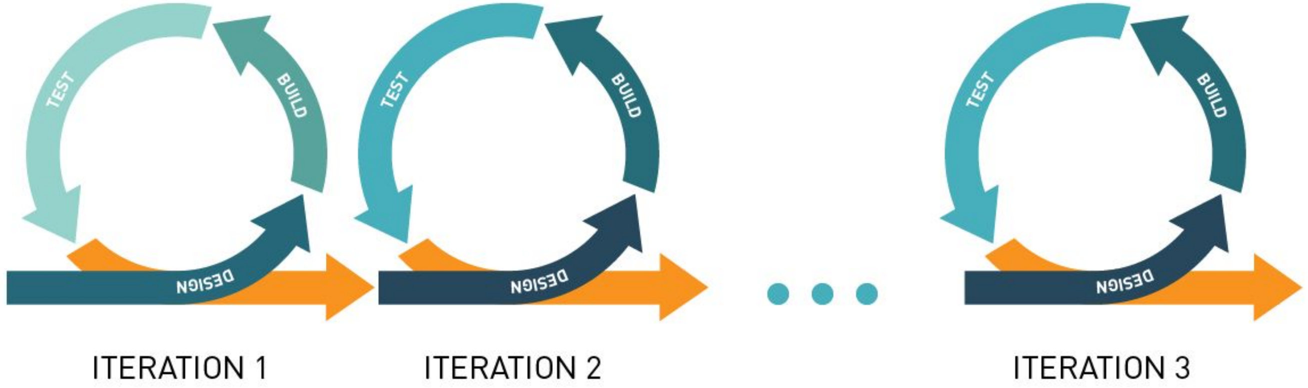


Report how well your best model does on the *test set*

Fair estimate of how well your selected model would perform on unseen data once deployed

Iterative progress

- Get results early and often
- Cover entire pipeline early
- Build on those results
- Refine / Clean up



How to collaborate

- Use GitHub to have 1 unified remote copy with all past versions
    - Dropbox etc. will not have a list of past working milestones to roll back to
    - Dropbox etc. will cause issues if one person is making changes to code that another person is using (due to constant syncing)
- Try to parallelize, not pipeline (don't want anyone to be blocked)
- Ideas for parallelizing between team members:
    - Multiple tables/data sources. Can split EDA of each and come up with interesting insights
    - Can work on different ML models and/or feature engineering strategies
    - Can generate different visualizations/insights from models
- Meet frequently to discuss insights/next steps and maintain accountability

GitHub essentials

Git steps

1. 1 person creates a repository on GitHub (with README.md)
2. Clone the repository

```
cd [directory you want to work in]
git clone [repository url]
cd [repository name]
```

3. Make changes (e.g. create jupyter notebooks, add code, etc.)

```
git add [file path]                       # add any new files to version control
git commit -a -m "[your commit message]"  # register changes as a new version
```

4. Sync with the remote (cloud) copy. Pull others' changes, push your own

```
git pull        # remote changes -> your copy
git push        # your copy -> remote (error if there are changes you haven't pulled)
```

GitHub conflicts

- Git does not constantly sync local+remote automatically like Google Docs.
  It syncs when you pull/push
- What if you and a teammate were modifying the same code?
  - Example: you both worked on the same function, they pushed first, and you pull
- Git will throw an error when you try to pull and will ask you to resolve conflicts
  - You'll have to pick which changes to keep: yours, your teammate's, or a mixture
  - Can be *very* difficult to resolve conflicts
- Best solution: avoid working on the same files, or at least the same code
  - If you pull changes that don't overlap with what you're working on, git doesn't overwrite what you've done or show conflicts

GitHub best practices

- Commit+push working copies (milestone achieved, no errors)
- Add useful commit messages (in case you need to roll back)
- Work on separate features/files to avoid conflicts
- Git pull frequently to avoid conflicts (local version very out of sync)
- If data not too large, add to version control as well
- In README, provide all details a new team member would need to navigate and modify the project

Evaluation criteria

Criteria

1. Presentation
2. Code quality
3. Approach to the problem
4. Accuracy and quality of model

Presentation

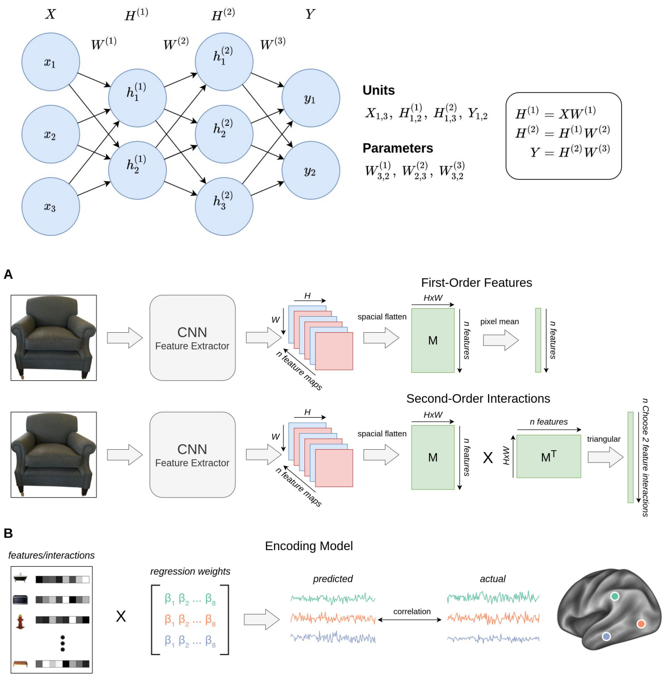
- Present as if to a client (who has some data science knowledge)
- Make it a story
    - What is the problem?
    - What is the dataset?
    - How did you analyze the dataset?
    - What were the findings?
- No code (in general)
- 5 minutes

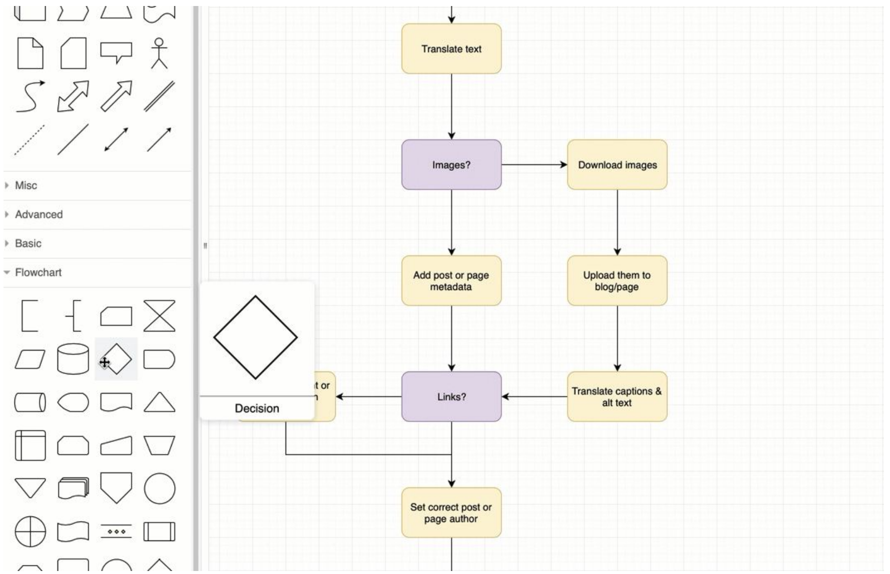
Presentation: structure

- **Motivation**: What is the problem? Why is it important (either business, public good, or research perspective)?
- **Task**: Problem from a technical perspective. Description of the dataset, features and targets, data exploration
- **Modeling**: *Important* aspects of your approach. How did you process the data or engineer features? What model did you use? Use schematics!
- **Results**: Visuals! Show metrics and experiments. Demo (if any)
- **Conclusions**: Did you solve the problem? Why or why not? What next?

Presentation: draw.io figures

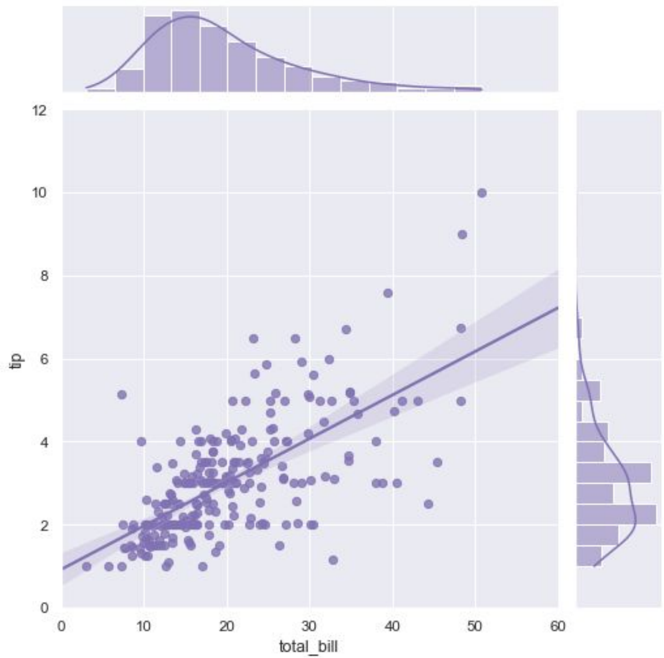
- Good for schematics, model diagrams, shapes, math typesetting, etc.

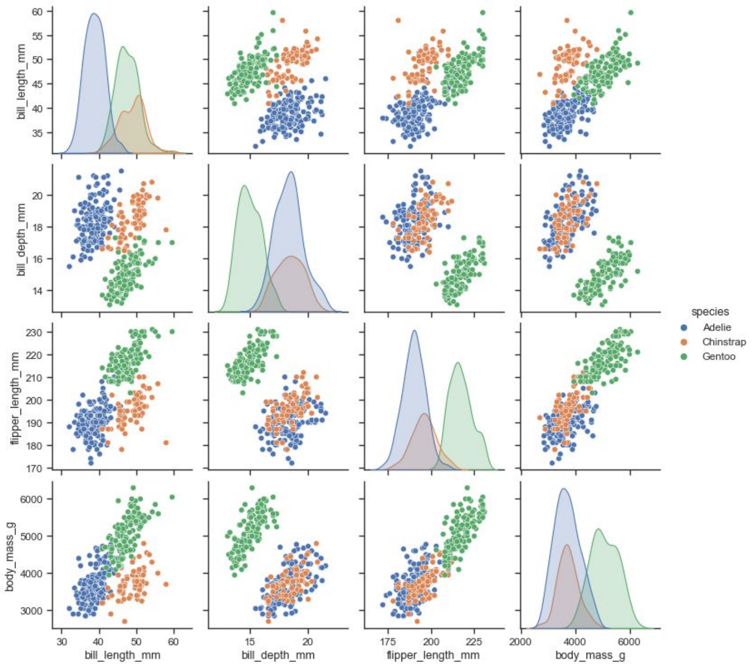




Presentation: python plotting libraries

- Good for displaying information about your dataset and results





Code quality

- Modularization
  - Different files for different steps of the ML pipeline (e.g. data processing, model training, etc.)
  - Functions for unified code (if it has a short english description, it’s probably a function)
- Readability
  - Meaningful variable names
  - Comments for code blocks, docstrings for functions
  - Periodically refactor code (e.g. remove old commented out code)
    - only when things are working and you know you want to keep the code
- Robustness
  - Test your functions
- Efficiency
  - Save processed data/trained model, only recreate when needed

Code quality: modularization

```
repo/
├── data
│   ├── raw_data.csv
│   └── preprocessed_data.csv
├── src
│   ├── modules
│   │   ├── data_preprocessing.py
│   │   └── modeling.py
│   │   └── figure_generation.py
│   ├── tests
│   │   ├── test_data_preprocessing.py
│   │   └── test_modeling.py
│   └── experiments.ipynb
├── output
│   ├── predictions.csv
│   ├── figures
│   │   ├── process_schematic.jpg
│   └── └── cluster_visualizations.jpg
└── README.md
```

```python
# data_preprocessing.py
...
def load_preprocessed_data():
    ...
    return X, y
...
```

```python
# experiments.ipynb
from modules.data_preprocessing import load_preprocessed_data
from modules.modeling import train_models
...
X, y = load_preprocessed_data()
best_model, cv_performance = train_models(X, y)
...
```

Code quality: docstrings

```python
def add_binary(a, b):
    '''
    Returns the sum of two decimal numbers in binary digits.

            Parameters:
                    a (int): A decimal integer
                    b (int): Another decimal integer

            Returns:
                    binary_sum (str): Binary string of the sum of a and b
    '''
    binary_sum = bin(a+b)[2:]
    return binary_sum
```

Code quality: testing

```
def sum(x, y):
    z=x+y
    return z

def test_sum():
    assert sum(x=3, y=1) == 4
```

Code quality: avoid retraining model

```
def train_model(X, y, force_refit=False):
    ...
    if os.path.exists(model_filepath) and not force_refit:
        model = pickle.load(open(model_filepath, 'rb'))
    else:
        model = SVM()
        model.fit(X_train, y_train)
        pickle.dump(model, open(model_filepath, 'wb'))

    y_pred = model.predict(X_test)
    ...
```

[Pickle tutorial]