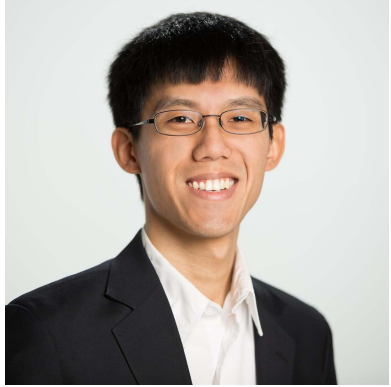
Reproducible Data Science with Open Source Tools

Jay Qi

Slides will be available online

http://github.com/jayqi/talks

Hi, I'm Jay



- Lead Data Scientist at DrivenData
- 10+ years in data science, machine learning, and scientific computing
- Active open source maintainer and contributor
  - Data science tools
  - Python utilities
  - Civic tech projects

DRIVENDATA

Data Science + Social Impact

Machine Learning Competitions • Data Science Consulting • Open Source Software

www.drivendata.org

@drivendata.org

github.com/drivendataorg

DrivenData publishes many different open source projects, such as...

drivendataorg / **competition-winners**

Winning models for our competitions

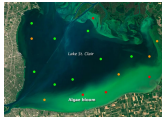
ZAMBA

Computer vision for wildlife conservation

://cloudpathlib

Pathlib for cloud storage services

CyFi: Cyanobacteria Finder



Detect cyanobacteria from satellite imagery

erdantic

Entity relationship diagrams for data classes

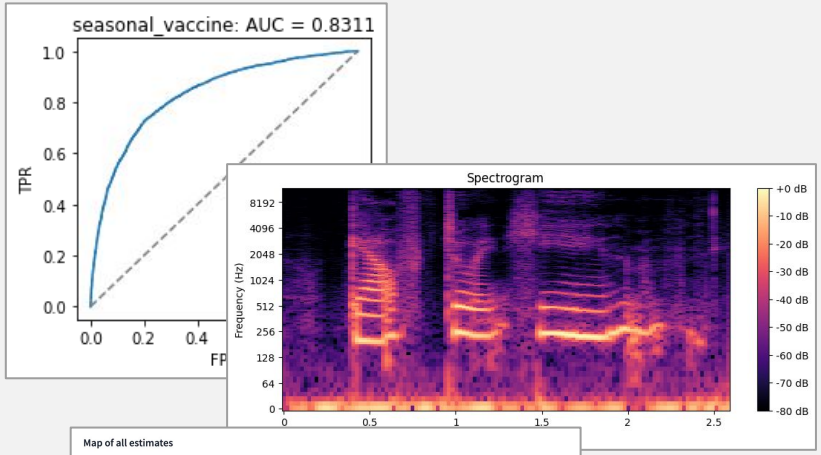
deon

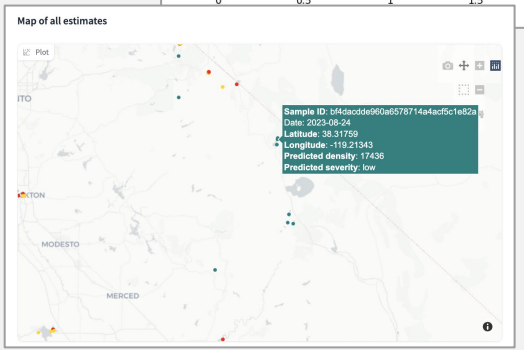
Data science ethics checklist

... and more

Why care about reproducible data science?

So, you did some data science…





But your code is kind of messy...

```
.
├── analysis.py
├── analysis_broken.py
├── analysis_broken_fixed.py
├── data_v1_clean.csv
├── data_v1_processed.csv
├── data_v2_final_final.csv
├── data_v4_final_for_real_this_time.csv
├── final_old.py
├── process.py
├── process_clean.py
└── results_v99.csv
```

So what?

Why care about code quality?

Data science work is only useful if it's **correct** and **valid**.

How can anyone know if it is? By **reproducing** it.

Reproducible and well-organized code lets you

- Collaborate more easily with others
- Share your learnings
- Feel confident about your conclusions

Best practices

Typical software engineering best practices apply:

1. Use version control
2. Write tests
3. Use a linter and auto-formatter

But data science has its own specific considerations.

Data science best practices

The next part of the presentation will cover these principles:

1. Analysis is a directed acyclic graph
2. Notebooks are for exploration, source files are for repetition
3. Build from the environment up
4. Keep secrets and configuration out of version control

We'll also discuss some recommended open source tools that you can research further on your own.

Data analysis is a directed acyclic graph

Each step in your analysis should be a node in a **directed graph with no loops**.

This means you can:

- Run the process forwards to recreate any analysis output
- Trace backwards from an output to know the code and data that created it

This also means **raw data must be treated as immutable**. Never modify it in place—always write out changes to a new copy.

Each step in your analysis is a node in a graph like this.

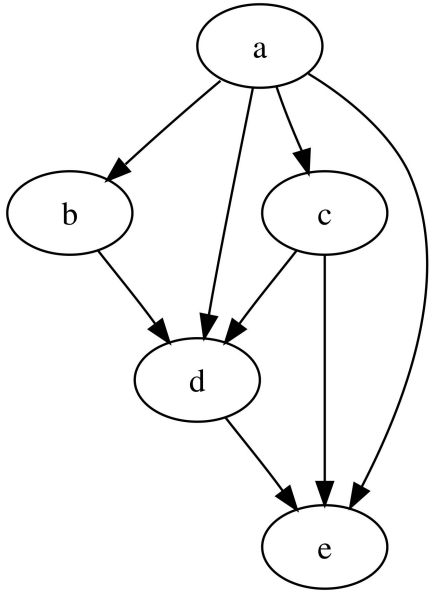


Diagram from Wikipedia

Data analysis is a directed acyclic graph

Each step in your analysis should be a node in a **directed graph with no loops**.

This means you can:

- Run the process forwards to recreate any analysis output
- Trace backwards from an output to know the code and data that created it

This also means **raw data must be treated as immutable**. Never modify it in place—always write out changes to a new copy.

TOOLS

For simple workflows, just keep track of the steps. You can combine them into a shell script.

For something a little more complicated, use:

GNU Make / Makefiles

It's comes preinstalled on Linux and Macs. On Windows, install with package manager like chocolatey.

If your workflow is really complicated, there are many good tools from data engineering:

- Airflow
- Luigi
- Prefect
- Dagster

Notebooks are for exploration and communication

Notebooks display code, outputs, and documentation together on one screen.

They let you quickly iterate and visualize results.

To ensure reproducibility, **always run your notebooks end-to-end**.

In [10]:
```
# what seasons are the data points from?
metadata["season"] = (
    metadata.date.dt.month.replace([12, 1, 2], "winter")
    .replace([3, 4, 5], "spring")
    .replace([6, 7, 8], "summer")
    .replace([9, 10, 11], "fall")
)
metadata.season.value_counts()
```

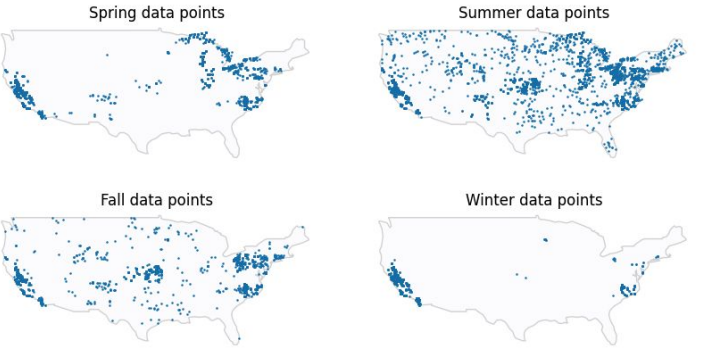
Out[10]:
```
summer    10813
spring     5045
fall       4758
winter     2954
Name: season, dtype: int64
```

Most of the data is from summer. Harmful algal blooms are more likely to be dangerous during the summer because more individuals are taking advantage of water bodies like lakes for recreation.

In [11]:
```
# where is data from for each season?
fig, axes = plt.subplots(2, 2, figsize=(10, 5))

for season, ax in zip(metadata.season.unique(), axes.flatten()):
    base = world[world.name == "United States of America"].plot(
        edgecolor="gray", color="ghostwhite", alpha=0.3, ax=ax
    )

    sub = metadata[metadata.season == season]
    geometry = [Point(xy) for xy in zip(sub["longitude"], sub["latitude"])]
    gdf = gpd.GeoDataFrame(sub, geometry=geometry)
    gdf.plot(ax=base, marker=".", markersize=2.5)
    ax.set_xlim([-125, -66])
    ax.set_ylim([25, 50])
    ax.set_title(f"{season.capitalize()} data points")
    ax.axis("off")
```



`train_labels.csv`

Let's look at the labels for the training data.

In [12]:
```
train_labels = pd.read_csv(DATA_DIR / "train_labels.csv")
train_labels.head()
```

Notebooks are for exploration and communication

Notebooks display code, outputs, and documentation together on one screen.

They let you quickly iterate and visualize results.

To ensure reproducibility, **always run your notebooks end-to-end**.

TOOLS

Python: Jupyter Notebooks

R: R Markdown

More specialized or advanced tools:

- Quarto — publishing production-quality reports
- Observable — write Javascript, specialized for data visualization
- Marimo — newer project, more app-like, stronger interactivity

Notebooks are for exploration and communication

Source files are for repetition

If you're copy-pasting code between notebooks, that is a sign for you to refactor.

Move the good parts of your analysis into **source files** that can be imported as library modules.

- More portable
- Reusable
- Easier to test

```python
from src.data.multilabel import (
    multilabel_sample_dataframe,
    multilabel_train_test_split,
)
from src.features.SparseInteractions import SparseInteractions
from src.models.metrics import multi_multi_log_loss
```

master box-plots-sklearn / src / data / multilabel.py

Code Blame 83 lines (62 loc) · 3.05 KB

```python
    return np.concatenate([sample_idxs, remaining_sampled])


def multilabel_sample_dataframe(df, labels, size, min_count=5, seed=None):
    """ Takes a dataframe `df` and returns a sample of size `size` where all
        classes in the binary matrix `labels` are represented at
        least `min_count` times.
    """
    idxs = multilabel_sample(labels, size=size, min_count=min_count, seed=seed)
    return df.loc[idxs]


def multilabel_train_test_split(X, Y, size, min_count=5, seed=None):
    """ Takes a features matrix `X` and a label matrix `Y` and
        returns (X_train, X_test, Y_train, Y_test) where all
        classes in Y are represented at least `min_count` times.
    """
    index = Y.index if isinstance(Y, pd.DataFrame) else np.arange(Y.shape[0])

    test_set_idxs = multilabel_sample(Y, size=size, min_count=min_count, seed=seed)
    train_set_idxs = np.setdiff1d(index, test_set_idxs)

    test_set_mask = index.isin(test_set_idxs)
    train_set_mask = ~test_set_mask

    return (X[train_set_mask], X[test_set_mask], Y[train_set_mask], Y[test_set_mask])
```

Build from the environment on up

The first step in reproducing an analysis is replicating the **computational environment** it was run in — the same tools, libraries, versions.

It's easy to lose track of this while you're experimenting. Make sure your dependencies are **always written down**.

For added reliability, write out and commit a **lock file** to version control. A lock file includes everything in your environment and should be produced by a tool.

TOOLS — Python

There are too many environment managers in Python, but there has recently been an emerging winner:

uv

TOOLS — R

In R, use **renv** for to create reproducible environments.

Build from the environment on up

The first step in reproducing an analysis is replicating the **computational environment** it was run in — the same tools, libraries, versions.

It's easy to lose track of this while you're experimenting. Make sure your dependencies are **always written down**.

For added reliability, write out and commit a **lock file** to version control. A lock file includes everything in your environment and should be produced by a tool.

TOOLS — Multilingual

If your project is multiple languages or it depends on C/C++ libraries, etc., then use the **conda** ecosystem:

- miniforge — conda manager that works with the open source conda-forge repository
- Pixi — more powerful project manager for conda projects

Keep secrets and computer-specific configuration out of version control

You don't want to leak your passwords. Other people also don't want to deal with hard-coded paths specific to your computer.

Use **environment variables** to specify these things instead.

You can use a **`.env`** file.

There are tools to automatically find and load them. Make sure it's in your `.gitignore` so that it won't ever get committed to version control.

```
# example .env file

DATABASE_URL=postgres://user:pwd@localhost:5432/dbname
AWS_ACCESS_KEY=myaccesskey
AWS_SECRET_ACCESS_KEY=mysecretkey
SOME_PATH=/Users/jay/some_directory/
```

TOOLS

Python: [python-dotenv]

R: [dotenv]

They give you functions like `load_dotenv()` that will automatically find and read `.env` files.

Cookiecutter Data Science

A logical, flexible and standardized project template for doing and sharing data science work.

https://cookiecutter-data-science.drivendata.org

Run a command-line program:

```
$ ccds
```

You'll be prompted to set some configuration options for your project.

bash

```
$ ccds
project_name (project_name): my-project
repo_name (my-project):
module_name (my_project):
author_name (Your name (or your organization/company/team)): Dat A. Scientist
description (A short description of the project.): This is my analysis of the data.
python_version_number (3.10): 3.12
Select dataset_storage
1 - none
2 - azure
3 - s3
4 - gcs
Choose from [1/2/3/4] (1): 3
bucket (bucket-name): s3://my-aws-bucket
aws_profile (default):
Select environment_manager
1 - virtualenv
2 - conda
3 - pipenv
4 - uv
5 - none
Choose from [1/2/3/4/5] (1): 2
Select dependency_file
1 - requirements.txt
2 - pyproject.toml
```

Cookiecutter Data Science

A logical, flexible and standardized project template for doing and sharing data science work.

https://cookiecutter-data-science.drivendata.org

It will produce a project structure that looks something like this, based on your choices and our opinionated best practices.

Primarily for Python projects, but the structure and ideas are still useful for other languages.

```
├── Makefile           <- Makefile with convenience 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.
│
├── 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`.
│
├── pyproject.toml     <- Project configuration file with package metadata for
│                         myproject and configuration for tools like black
│
├── 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`
│
└── myproject          <- Source code for use in this project.
    │
    ├── __init__.py             <- Makes myproject a Python module
    │
    ├── config.py               <- Store useful variables and configuration
    │
```

Cookiecutter Data Science

A logical, flexible and standardized project template for doing and sharing data science work.

https://cookiecutter-data-science.drivendata.org

Open source and available from PyPI or conda-forge.

```
pip install cookiecutter-data-science

# or

pipx install cookiecutter-data-science

# or

uv tool install cookiecutter-data-science

# or

conda install cookiecutter-data-science -c conda-forge
```

Recap

- Reproducibility is important!
- Data science work has specialized best practices
  1. Analysis is a directed acyclic graph
  2. Notebooks are for exploration, source files are for repetition
  3. Build from the environment on up
  4. Keep secrets and configuration out of version control
- Use Cookiecutter Data Science for a well-defined and standard project structure

Thanks! Questions?

Jay Qi

https://cookiecutter-data-science.drivendata.org

Slides available at: http://github.com/jayqi/talks

DRIVENDATA