Streamline your Data Science projects with Ploomber

Eduardo Blancas, JupyterCon 2020

Twitter: @edublancas / Website: ploomber.io

Data Science is all about experimentation

- Three stages for each experiment:
  1. Thinking
  2. Coding
  3. Execution
- **More experiments** = higher chance of **success**

Image: https://unsplash.com/photos/JeInkKlI2Po

Poor man's solution

- Code project in a (long) **single file**
- Problem: **Not maintainable**

  1. No clear boundaries among tasks

  2. Leads to unwanted interactions

- We need **reproducible** and **maintainable** projects

Image: https://www.flickr.com/photos/44674593@N00/7308867666

Why bother about maintainability?

Without a lot of structure, we forget what we've done in the past, we can't read each others' work, and we can't test whether what we've done is correct.[1]

— *Patrick Ball*

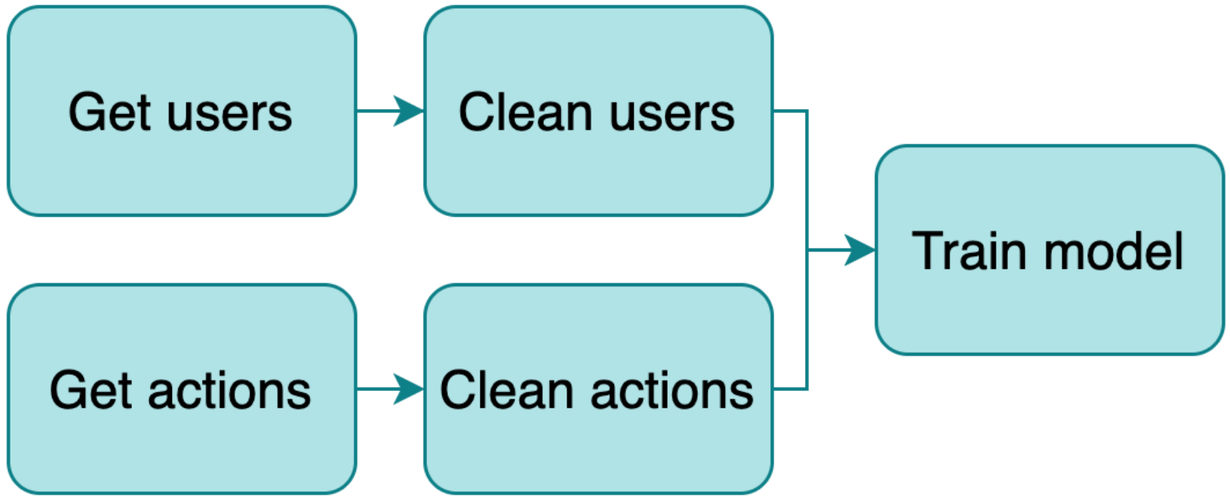
Image: https://unsplash.com/photos/r6mBXuHnxBk

[1] https://hrdag.org/2016/06/14/the-task-is-a-quantum-of-workflow/

Build a data pipeline

- **Break down** logic in small tasks
- **Outputs become inputs** of *downstream* tasks
- **New challenge**: Manage structure

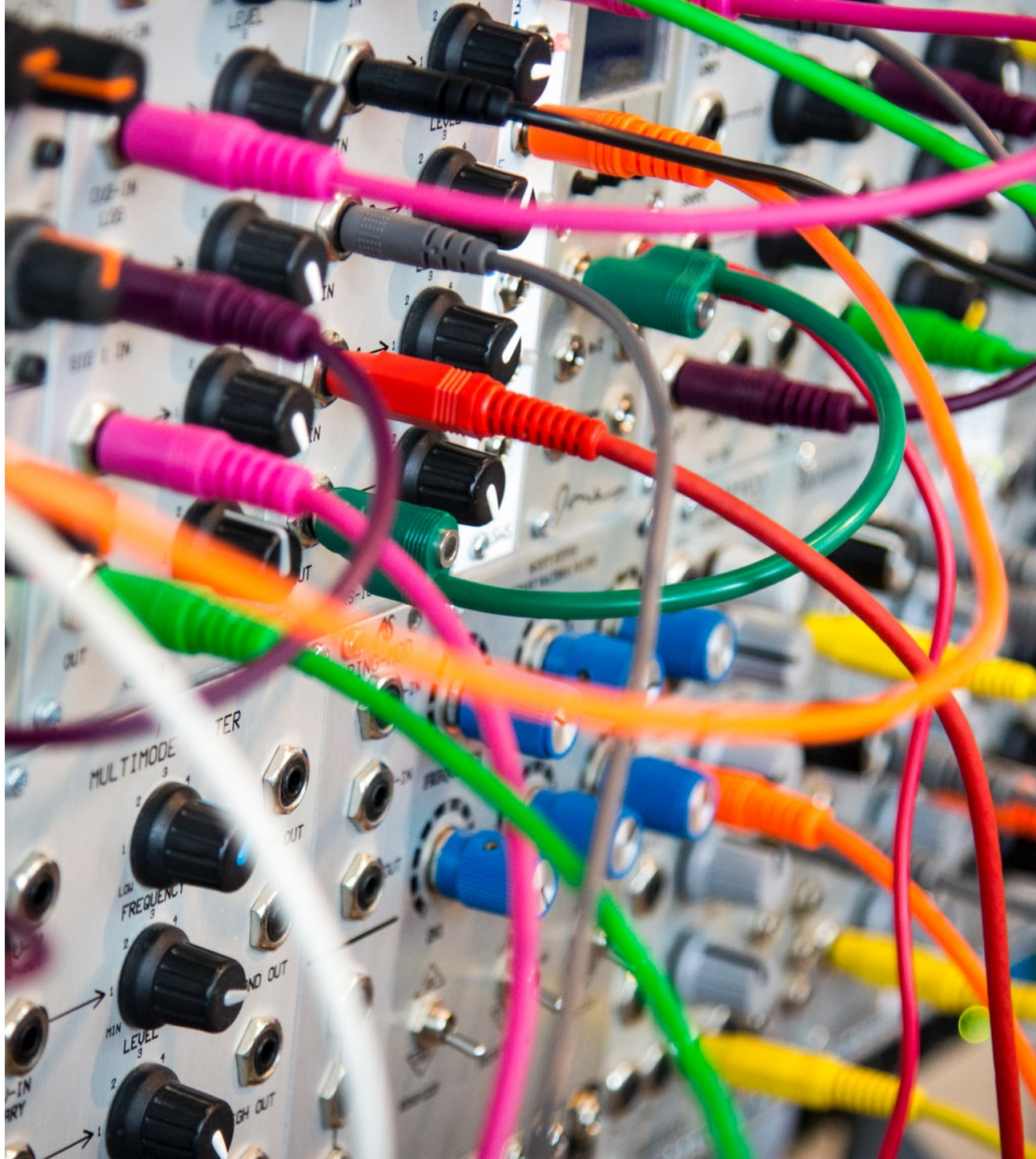
  1. Multiple files
  2. Route outputs
  3. Orchestrate execution



Current approach: Workflow managers

- Frameworks to develop pipelines (Make, Airflow, Luigi, etc)
- Problems:
  1. Learn a new tool
  2. Write pipeline code
  3. Adds unnecessary complexity

Image: https://unsplash.com/photos/l090uFWoPaI



Ploomber

Adheres to a *convention over configuration* philosophy, to allows us to **write pipelines with a focus on usability.**

Three conventions:

1. Each task is a script

2. Scripts declare dependencies via an `upstream` variable

3. And outputs via a `product` variable

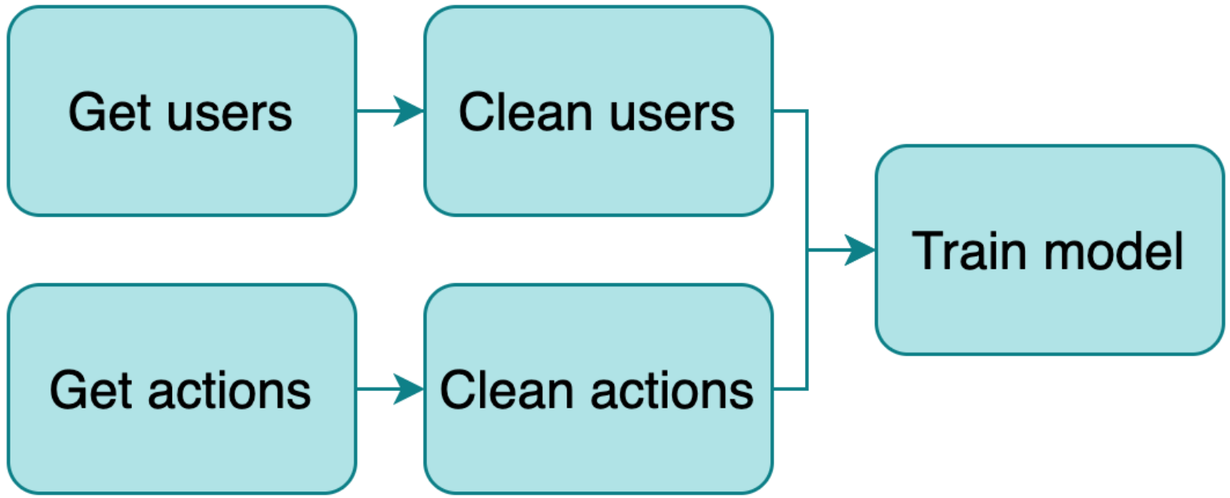
Python example

`train-model.py`:

```
import pandas as pd

# + tags=["parameters"]
upstream = ['clean-users', 'clean-actions']
product = {'nb': 'output/model-evaluation.ipynb',
           'model': 'output/model.pickle'}

# +
users = pd.read_parquet(
    upstream['clean-users']['data'])
actions = pd.read_parquet(
    upstream['clean-actions']['data'])
# code continues...
```



Cell injection

```python
import pandas as pd

# + tags=["parameters"]
upstream = ['clean-users', 'clean-actions']
product = {'nb': 'output/model-evaluation.ipynb',
           'model': 'output/model.pickle'}

# +
users = pd.read_parquet(
    upstream['clean-users']['data'])
actions = pd.read_parquet(
    upstream['clean-actions']['data'])
# code continues...
```

```python
import pandas as pd
```

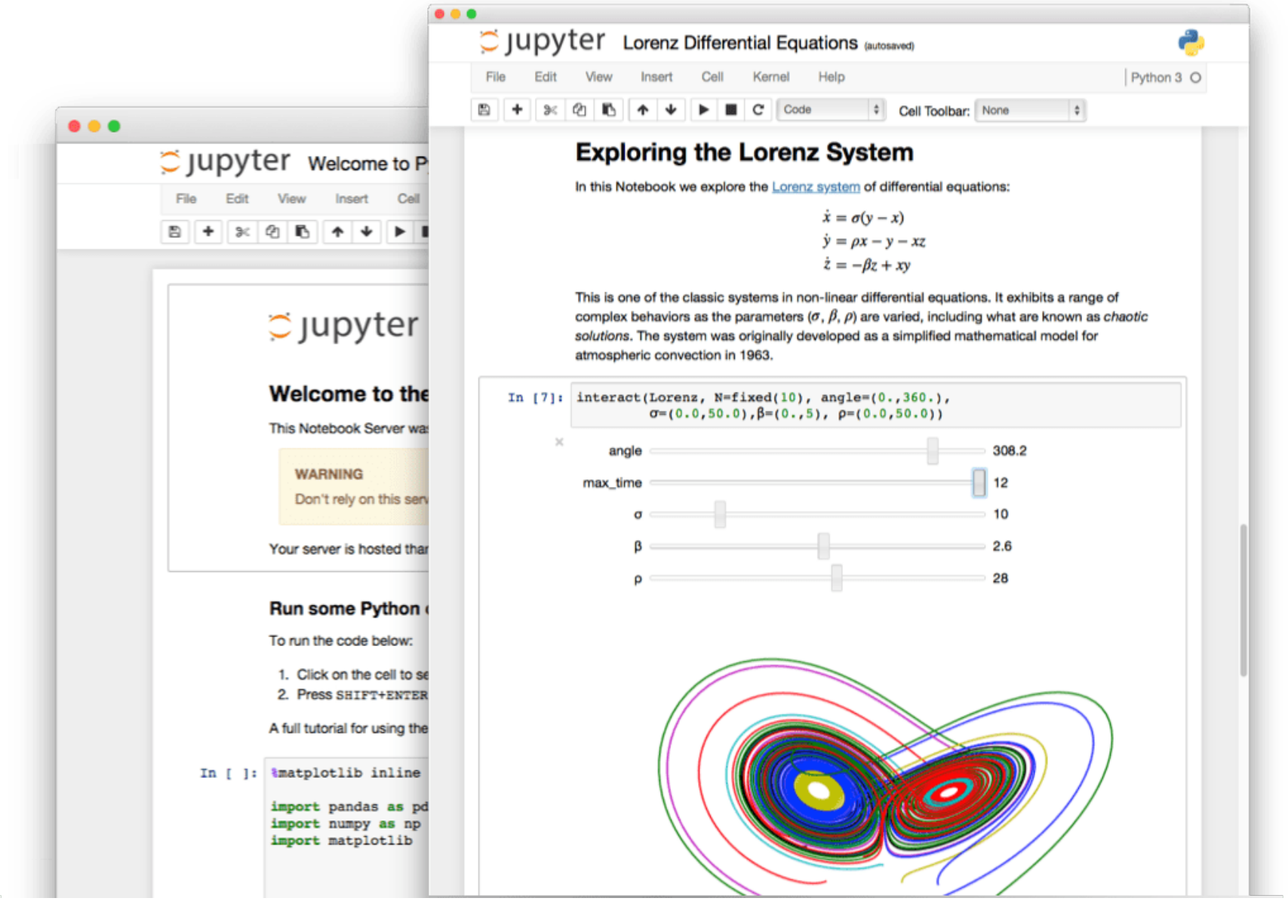
```python
upstream = ['clean-users', 'clean-actions']
product = {
    'nb': 'output/model-evaluation.ipynb',
    'model': 'output/model.pickle'
}
```

```python
# Injected cell
upstream = {
    'clean-users': {
        'nb': 'output/clean-users.ipynb',
        'data': 'output/clean-users.parquet'
    },
    'clean-actions': {
        'nb': 'output/clean-actions.ipynb',
        'data': 'output/clean-actions.parquet'
    }
}
```

```python
users = pd.read_parquet(
    upstream['clean-users']['data'])
actions = pd.read_parquet(
    upstream['clean-actions']['data'])
```

Tasks as notebooks

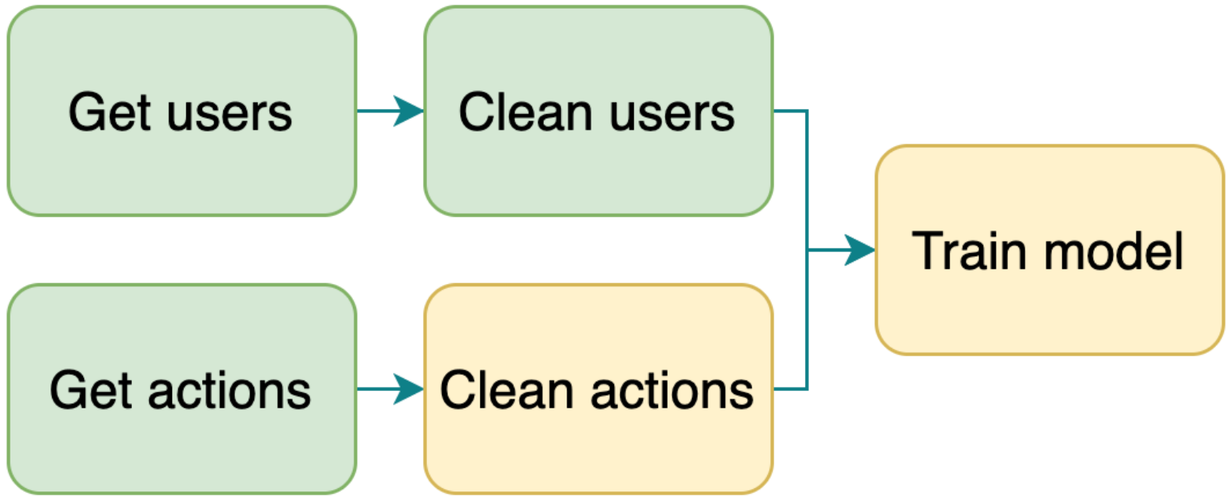
- Tasks are converted to notebooks (`.ipynb`) before execution[2]
- No need to write code to save tables or charts
- Generate logs for each execution



[2] Thanks to papermill! https://github.com/nteract/papermill

Build process

1. **Extract** `upstream` and `product`
2. **Determine execution order** using extracted `upstream` dependencies
3. **Inject code**. Replace the list of dependency names with a map of names to products
4. **Execute** the pipeline (skips up-to-date tasks)



Sample workflow

1. Input

```
get-users.py
clean-users.py
get-actions.py
clean-actions.py
train-model.py
```

2. Execute

```
ploomber build --entry-point .
```

3. Result

```
output/
    get-users.ipynb
    users.parquet

    clean-users.ipynb
    clean-users.parquet

    get-actions.ipynb
    actions.parquet

    clean-actions.ipynb
    clean-actions.parquet

    model-evaluation.ipynb
    model.pickle
```

Integration with Jupyter

- Scripts open as notebooks[3]
- (Temporary) cell injection
- Ensures no hidden state by running `ploomber build`

[3] Thanks to jupytext! https://github.com/mwouts/jupytext

Demo

Main features

- Python, R and SQL[4] supported[5]
- Parametrized pipelines with auto-generated CLI
- Testing (i.e. run data tests upon task execution)
- Debugger integration (pdb and ipdb)

---

[4] Any database with a DBAPI 2.0 client or compatible with SQLAlchemy is supported

[5] Languages that have Jupyter kernels are supported with a few limitations

Resources

- Install: `pip install ploomber`
- For updates/questions/feedback, follow me on Twitter: @edublancas
- Code: github.com/ploomber/ploomber
- Examples: github.com/ploomber/projects
- Website: ploomber.io
- This presentation: blancas.io/talks/jupytercon-20.pdf

Thanks for watching!

Twitter: @edublancas / Website: ploomber.io