Stats 285, Lecture 7

Painless Data Pipelining with Kedro

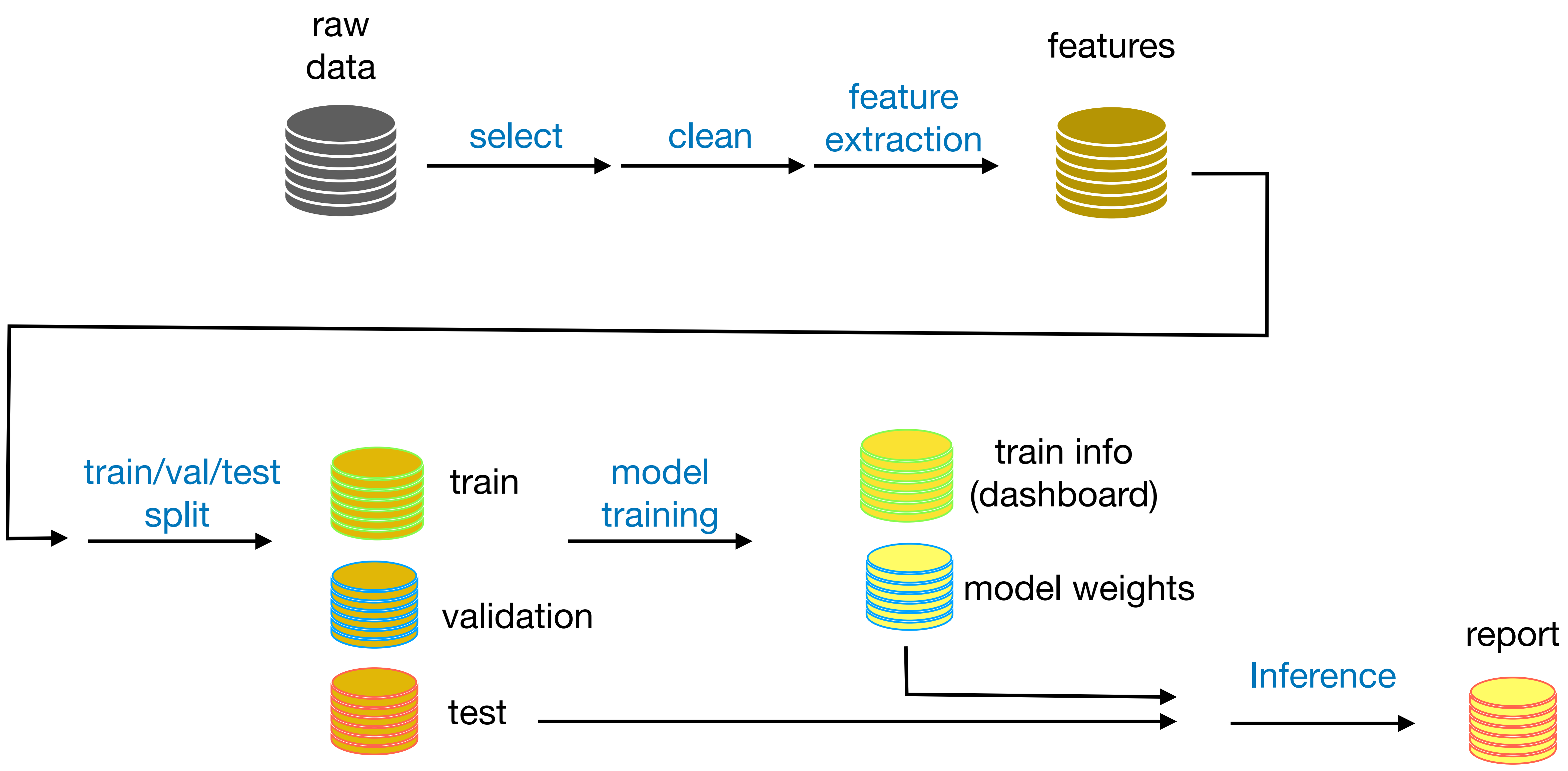
Alon Kipnis

26 April 2020

Overview

- Introduction
    - Pain points in Data Science experiments
- Kedro
    - Features
    - Examples

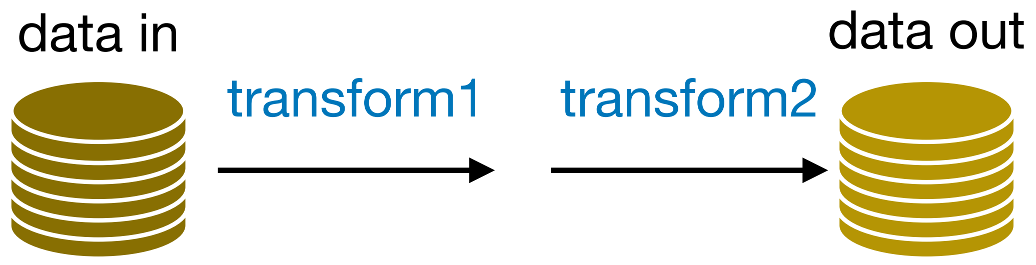
Motivation: Pain Points in Data Science Experiments



- Pains in **data management**: loading, storing, versioning
- Pains in **data processing**: compute time, writing and maintaining code

Kedro's Big Idea #1: Think Pipelines

data science project —> acyclic graph



pipeline = serial transformation of datasets

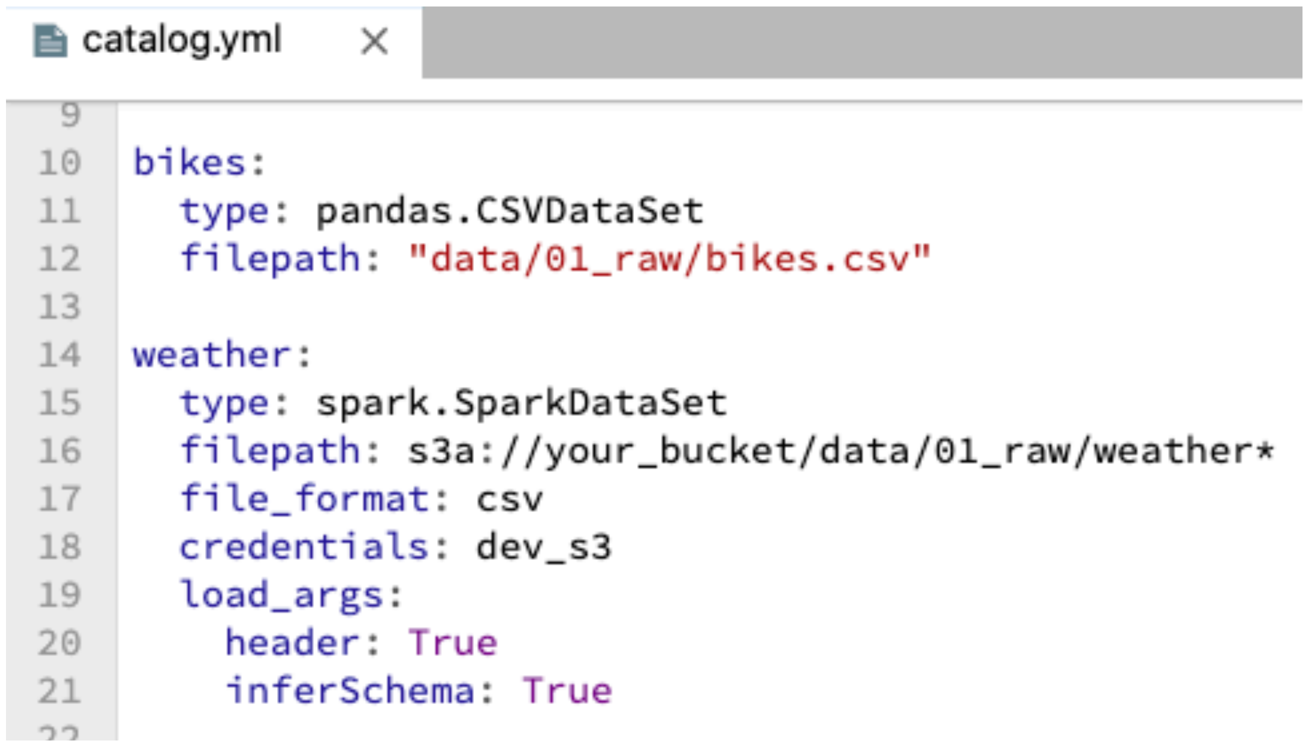
- **Two types of operations:**
  1. Reading/writing data
  2. Transforming data

- **Advantages:**
  - Code reusability
  - Easy to redirect component

Kedro's Big Idea #2: Standardize I/O

No read/write commands

- Maintain a **Data Catalog**:
  - Automatically handles:
    - Read/Write
    - Versioning
    - Remote connection
      - Authentication
    - Compression
    - …

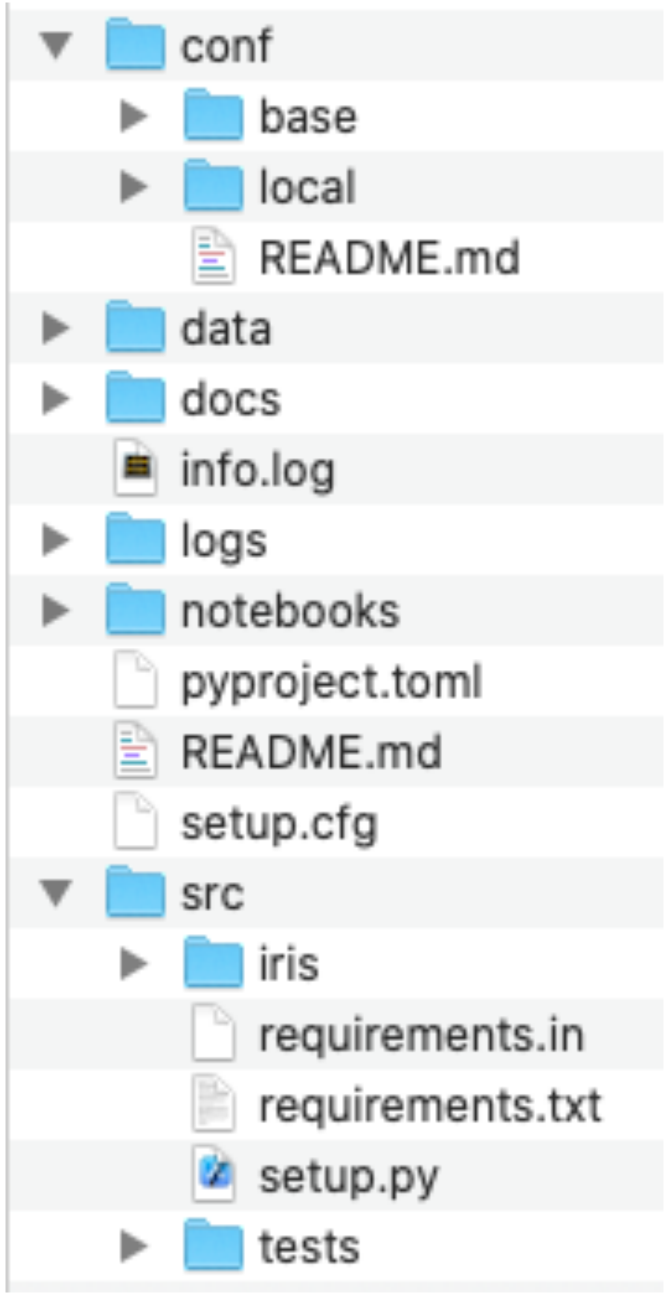

catalog.yml

```yaml
bikes:
  type: pandas.CSVDataSet
  filepath: "data/01_raw/bikes.csv"

weather:
  type: spark.SparkDataSet
  filepath: s3a://your_bucket/data/01_raw/weather*
  file_format: csv
  credentials: dev_s3
  load_args:
    header: True
    inferSchema: True
```

Kedro’s Big Idea #3: Project Template

- Standardize project structure
  - Standard files/folders
  - Standard documentation
  - Testing



```
▼ conf
    ▶ base
    ▶ local
      README.md
▶ data
▶ docs
  info.log
▶ logs
▶ notebooks
  pyproject.toml
  README.md
  setup.cfg
▼ src
    ▶ iris
      requirements.in
      requirements.txt
      setup.py
    ▶ tests
```

What is Kedro?

“Kedro is an open-source ***Python*** *framework for creating* ***reproducible****,* ***maintainable*** *and* ***modular*** *data science code“.* [Kedro docs]

“[Kedro] borrows concepts from software engineering best-practice and applies them to machine-learning code; applied concepts include ***modularity****, separation of concerns and* ***versioning****”.*

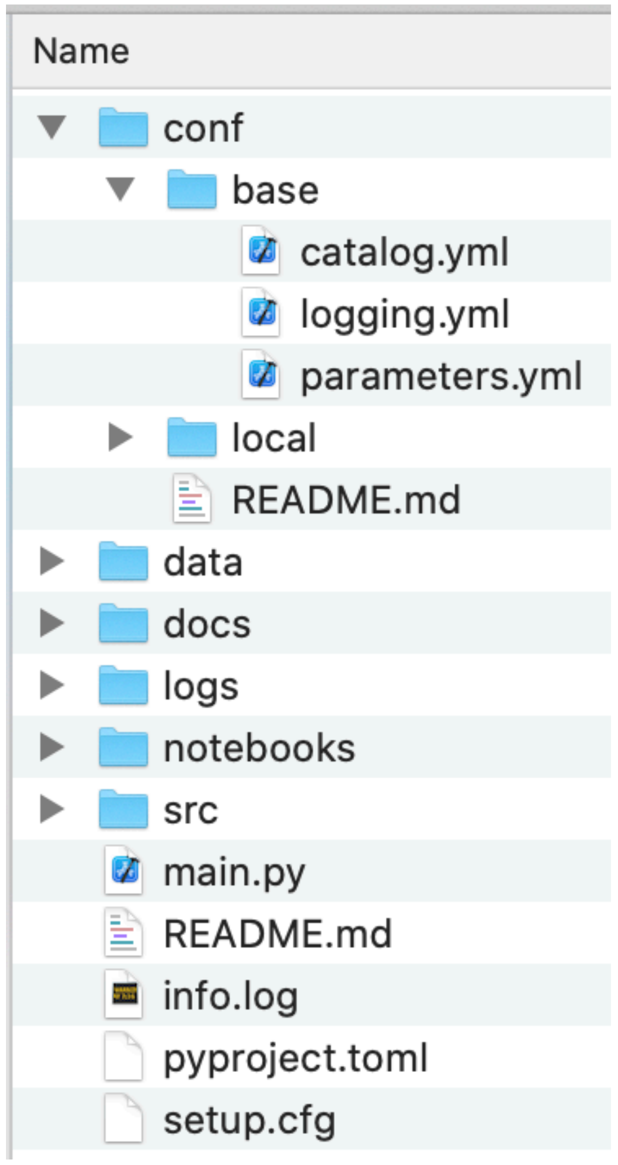
What is Kedro? (Cont'd)

- **Main features:**
    - **The Data Catalog** - extendible collection of datasets and models. Borrows arguments from Pandas, Spark, etc…
    - **Nodes & Pipelines**
    - **Project template** - Files and folders organization. Eases collaboration and code maintenance
    - Isolating all hard-coded parameters in `parameters.yml`
    - Command line interface (CLI) as well as API

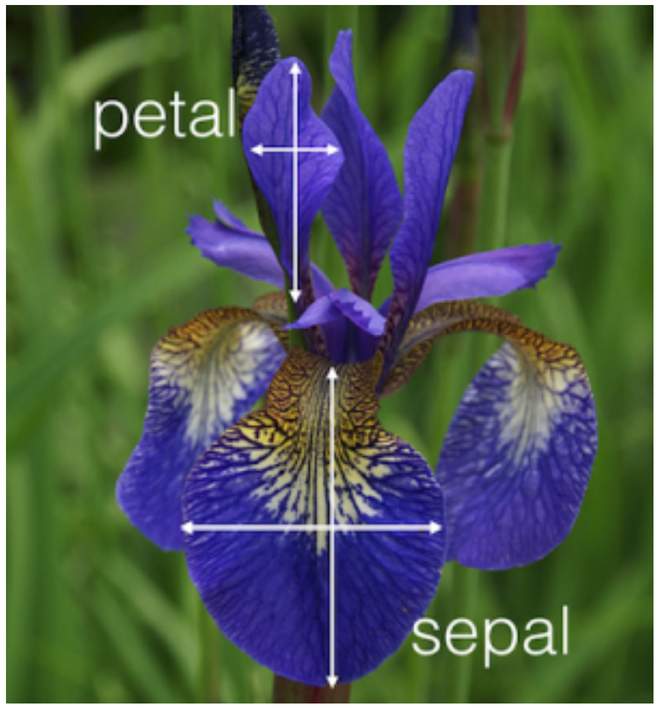
Example I: Classification

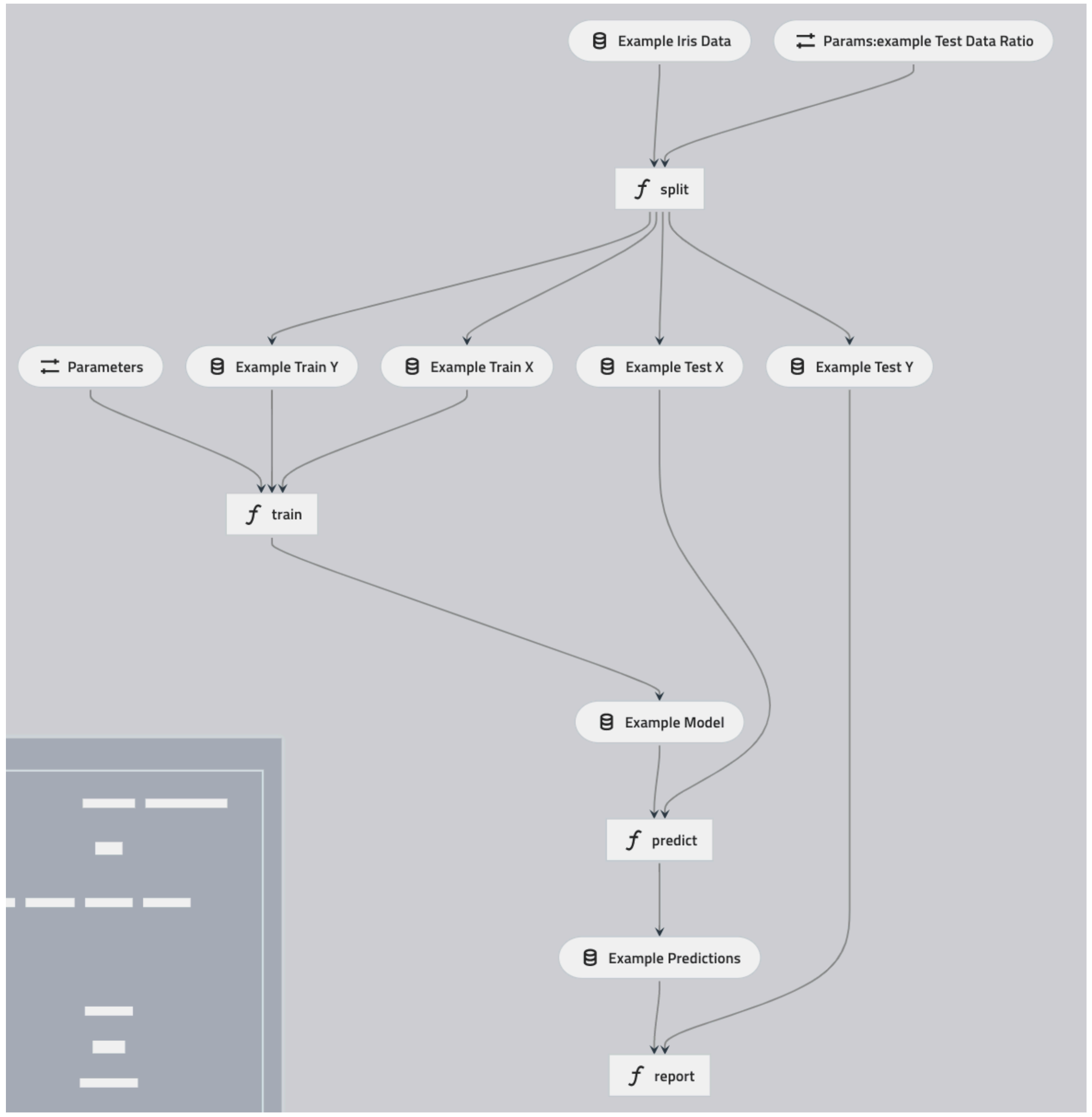
Demo

```
kipnisal@alonks-mbp% pip install Kedro
kipnisal@alonks-mbp% kedro new --starter=pandas-iris
```



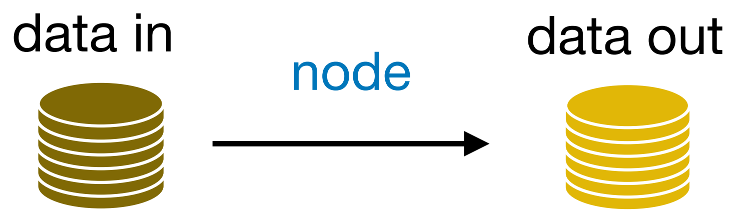
```
Name
▼ conf
    ▼ base
        catalog.yml
        logging.yml
        parameters.yml
    ▶ local
      README.md
▶ data
▶ docs
▶ logs
▶ notebooks
▶ src
  main.py
  README.md
  info.log
  pyproject.toml
  setup.cfg
```





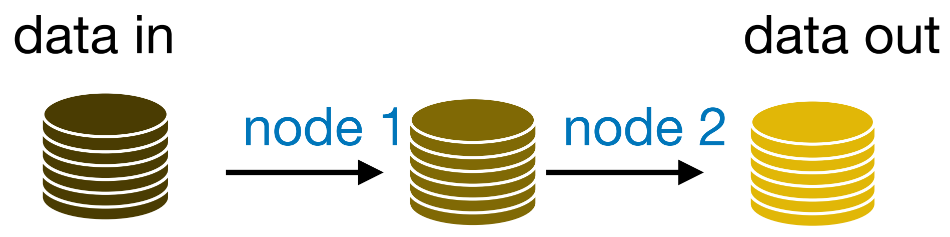
Useful Convention in Kedro

- **Node** - a pure Python function that has inputs and outputs



- **Dataset** - an impure Python function allowing reading/writing to storage; all datasets are registered in the Catalog file `catalog.yml`

- **Pipeline** - a collection of nodes with defined relationships and dependencies



- **Parameter** - hard-coded variable; all parameters are specified in `parameters.yml`

Example II: Authorship of Biblical Texts

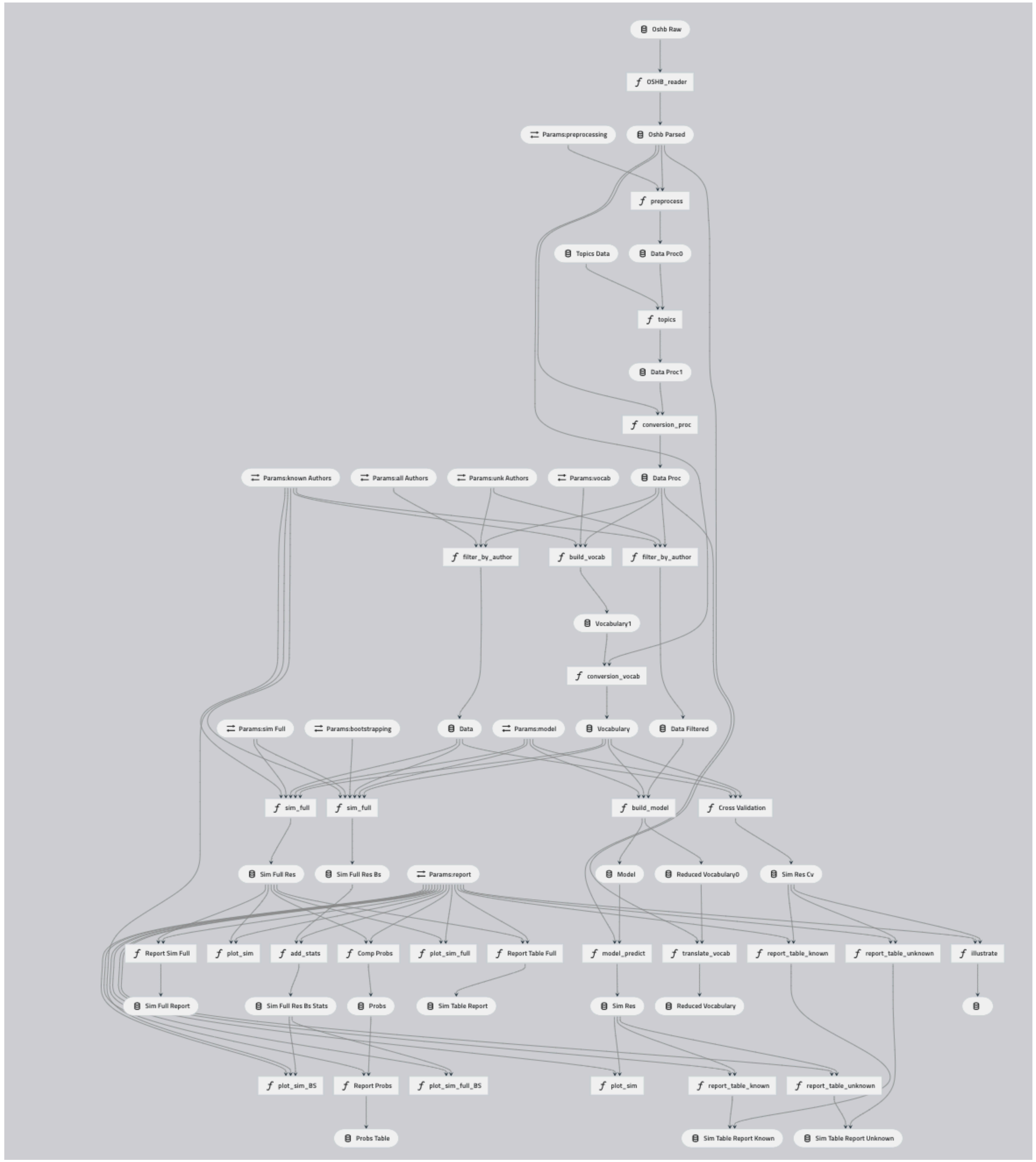
- **Goal:** test an algorithm for classifying texts from the Bible in terms of authorship
- **Data:** list of word-lemma-morphcode by book-chapter-verse (https://github.com/openscriptures/morphhb)
  - Select relevant parts by book-chapter-verse
  - Remove prefixes and suffixes
  - Remove some words according to their POS

- **Inference:**
  - Train model
  - Test model
  - Predict

- **Reporting:**
  - Accuracy per text
  - Figures

Authorship of Biblical Texts (cont’d)

Demo



Kedro's Data Catalog

API for datasets

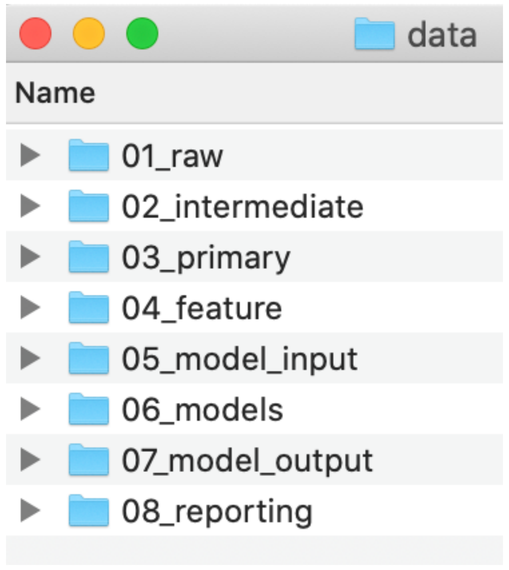
- Manages the loadings and savings of your data:
  - Standardized I/O operatins
  - Integrates with pandas, spark, SQLAlchemy, Cloud…
  - Versioning capabilities

no read/write/databse access/authentication command in your main code

when in doubt, write it out

Demo data catalog

Data Engineering Convention



Raw	Initial start of the pipeline, containing the sourced data model(s) that should never be changed, it forms your single source of truth to work from. These data models are typically un-typed in most cases e.g. csv, but this will vary from case to case.
Intermediate	Optional data model(s), which are introduced to type your `raw` data model(s), e.g. converting string based values into their current typed representation.
Primary	Domain specific data model(s) containing cleansed, transformed and wrangled data from either `raw` or `intermediate`, which forms your layer that you input into your feature engineering.
Feature	Analytics specific data model(s) containing a set of features defined against the `primary` data, which are grouped by feature area of analysis and stored against a common dimension.
Model input	Analytics specific data model(s) containing all `feature` data against a common dimension and in the case of live projects against an analytics run date to ensure that you track the historical changes of the features over time.
Models	Stored, serialised pre-trained machine learning models.
Model output	Analytics specific data model(s) containing the results generated by the model based on the `model input` data.
Reporting	Reporting data model(s) that are used to combine a set of `primary`, `feature`, `model input` and `model output` data used to drive the dashboard and the views constructed. It encapsulates and removes the need to define any blending or joining of data, improve performance and replacement of presentation layer without having to redefine the data models.

https://kedro.readthedocs.io/en/0.14.3/06_resources/01_faq.html#what-is-data-engineering-convention

Kedro and CJ

- Versionize all output dataset

- At each iteration:

    1. modify `parameters.yml`

    2. run pipeline

Kedro and CJ (cont’d)

```python
import KedroSession

for x in X :
    for y in Y :
        for z in Z :
            with open(parameters.yaml) as f :
                f.write(parms_file_str(x,y,z))
            with KedroSession.create('atomic_xpr') as session:
                session.run()
```

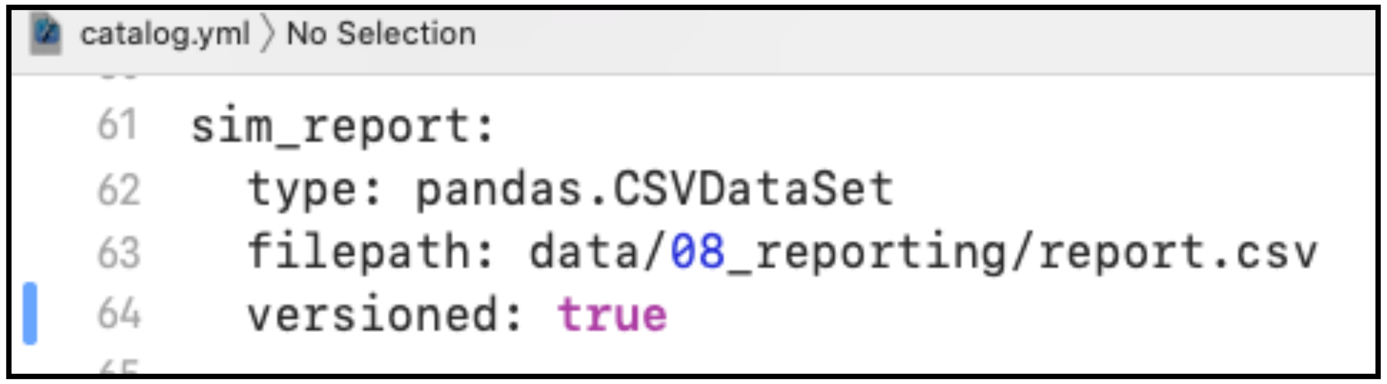
```python
base_file_path = 'conf/base/parameters.yml'
new_file_path = 'conf/local/parameters.yml'

def params_file_str(x,y,z) :

    with open(base_file_path) as f :
        params_str = f.read()

    params_str += '\n'
    params_str += f'x: {x}\n'
    params_str += f'y: {y}\n'
    params_str += f'z: {z}\n'


    with open(new_file_path, 'w') as fout :
        fout.write(params_str)
```



```yaml
sim_report:
  type: pandas.CSVDataSet
  filepath: data/08_reporting/report.csv
  versioned: true
```

Other Features

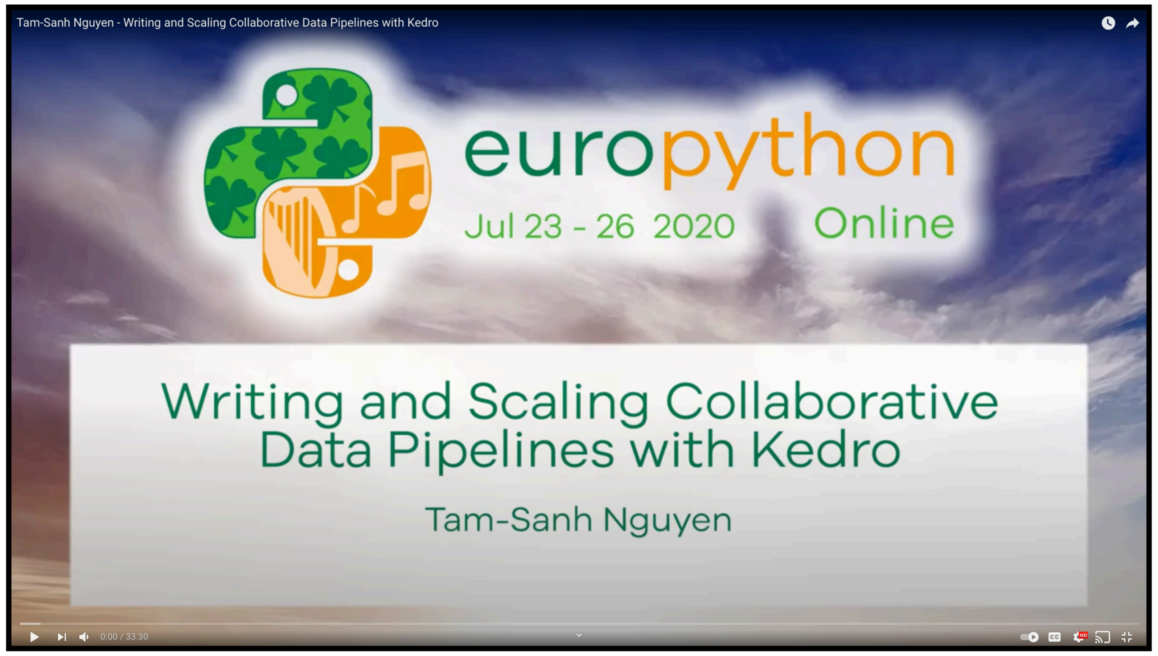
- Versatile **CLI**
- **Flexible Deployment**: supports .egg or .whl packaging
- **Kedro-Docker**: plugin to package Kedro projects in Docker containers
- **Documentation** (https://kedro.readthedocs.io/)

Recap

- Kedro
  - programing framework for data science projects
- Pipelining
  - data science project —> acyclic graph
- Data Catalog
  - API for datasets
  - no read/write commands
  - when in doubt, write it out
- Project Template
  - eases collaboration with others and your future self
- Python framework
  - extendible and modifiable

Resources

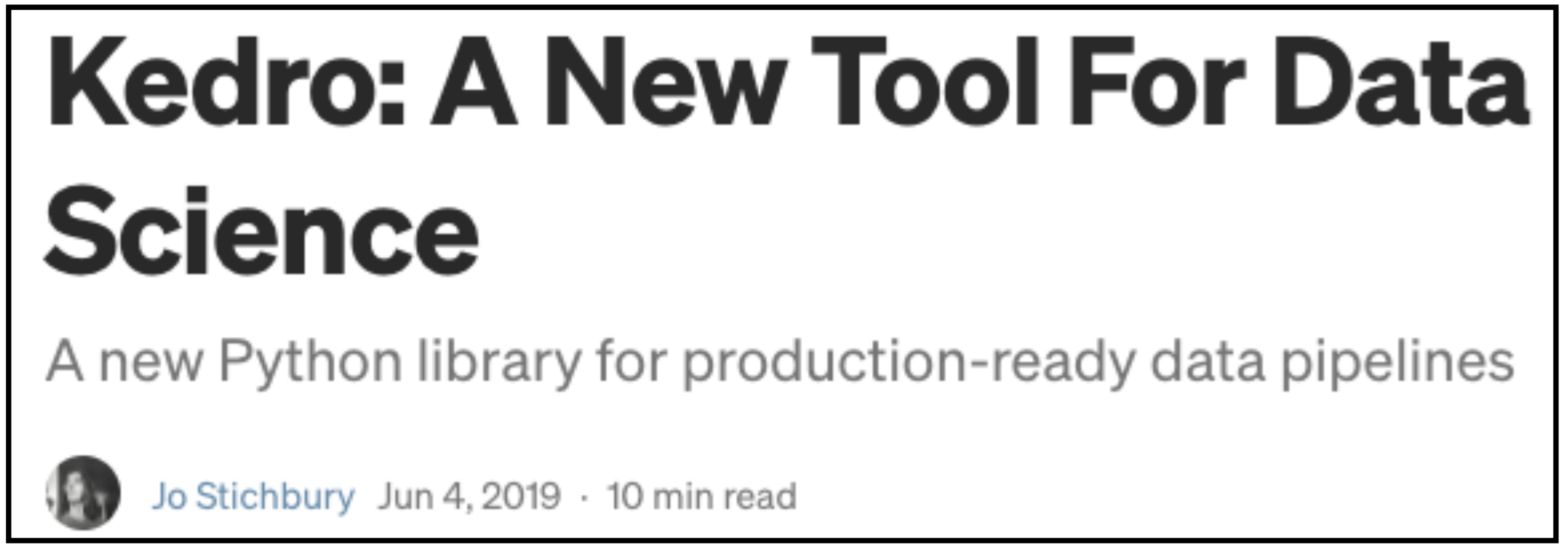
- Kedro's Documentation (kedro.readthedocs.io)


Tam-Sanh Nguyen - Writing and Scaling Collaborative Data Pipelines with Kedro

europython
Jul 23 - 26 2020 Online

Writing and Scaling Collaborative Data Pipelines with Kedro

Tam-Sanh Nguyen


Kedro: A New Tool For Data Science

A new Python library for production-ready data pipelines

Jo Stichbury Jun 4, 2019 · 10 min read