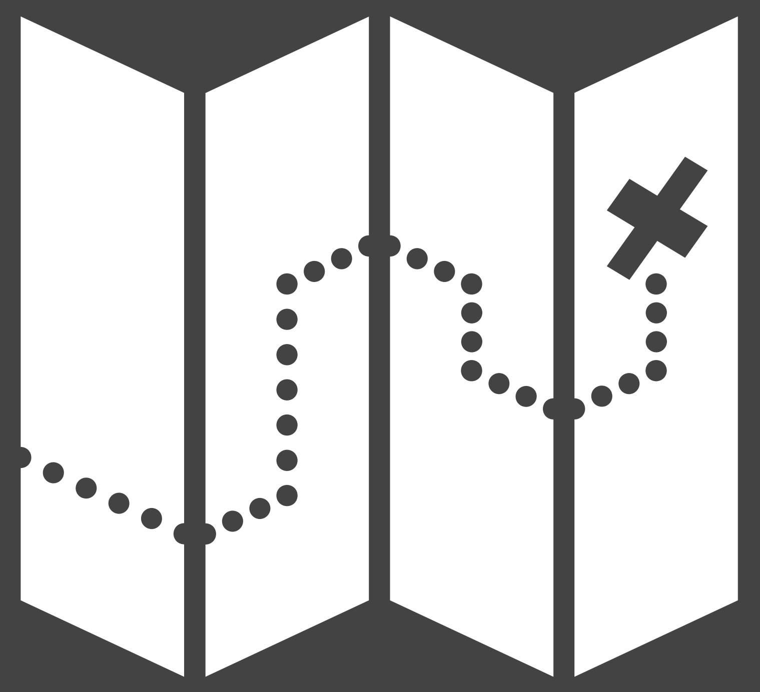
ML Pipeline Management

Release Engineering for Machine Learning (CS4295)

Luís Cruz <l.cruz@tudelft.nl> RE4ML 2021

Outline

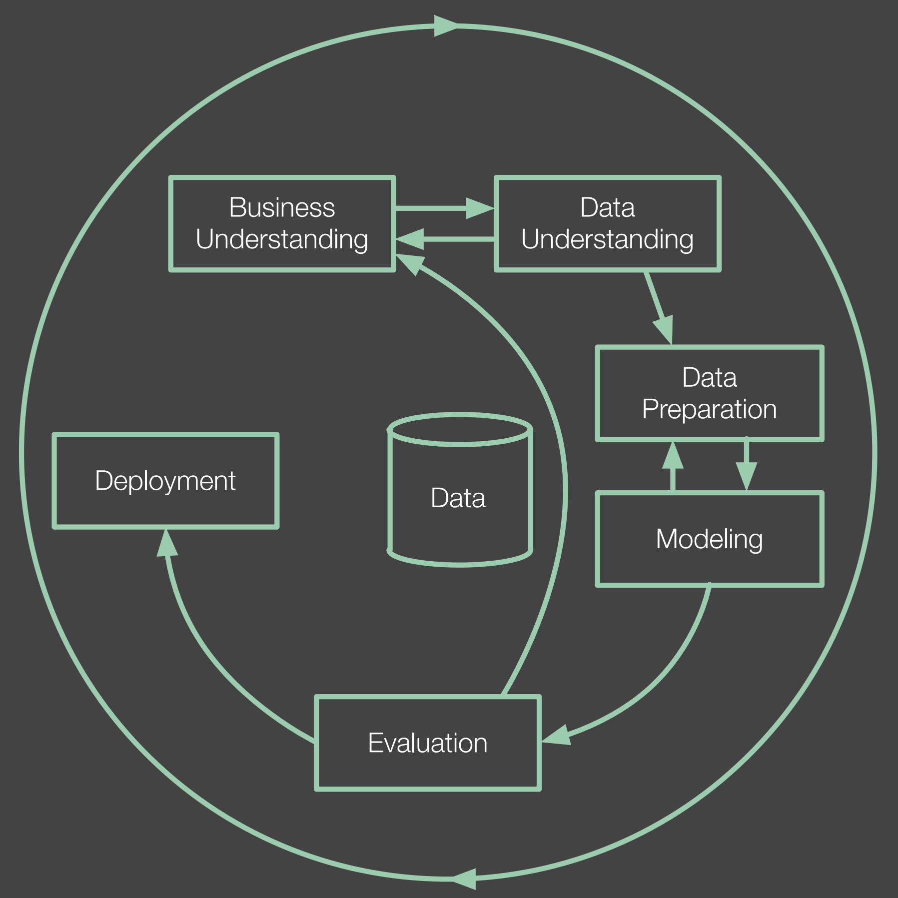


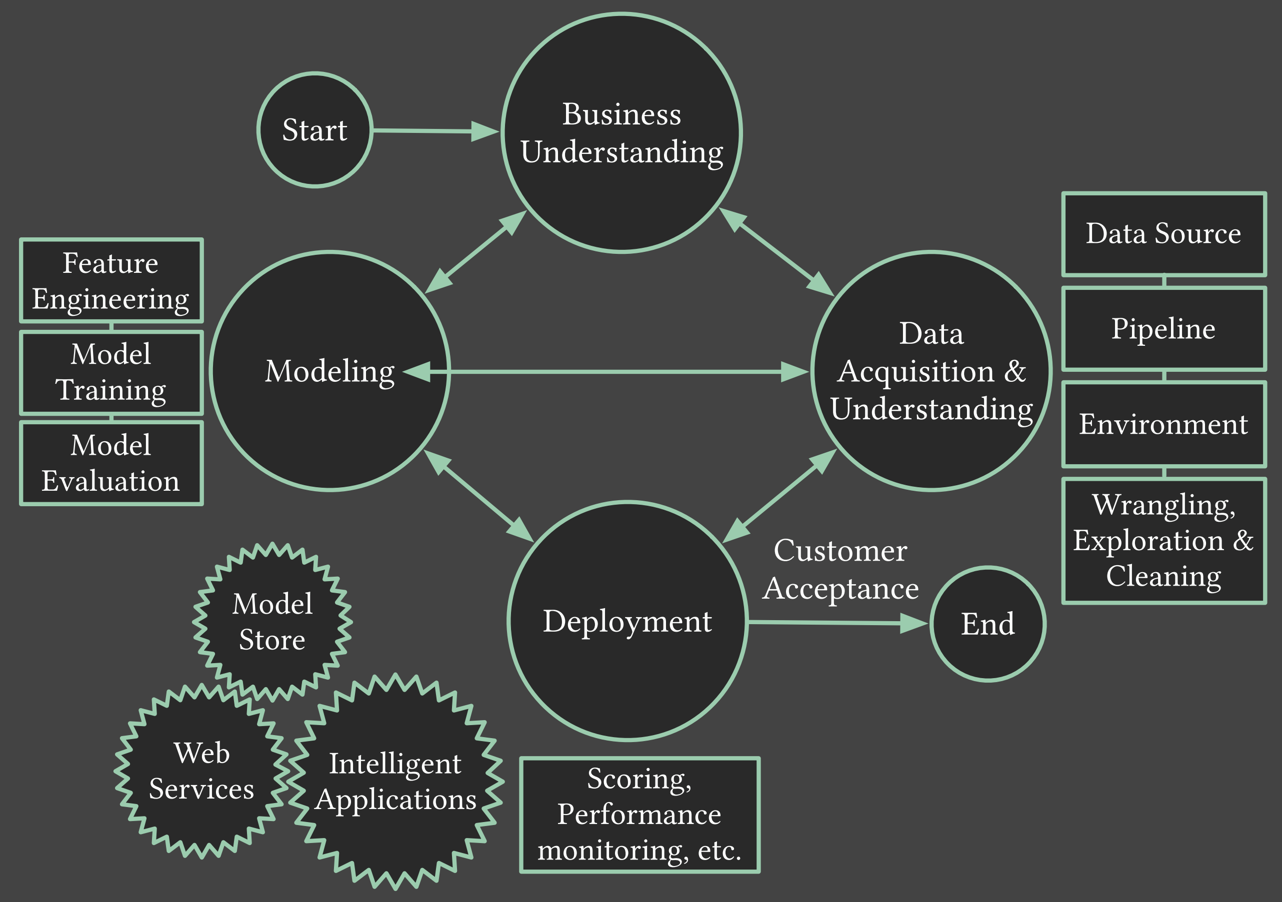
- AI lifecycle
- ML artefacts
- Pipeline Management
- ML version control
- Code smells in ML
- Code smells for ML

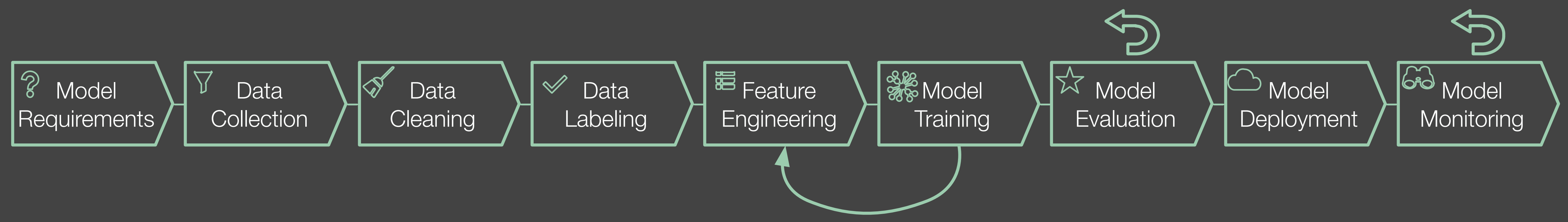
```python
import pandas as pd
from sklearn.linear_model import LogisticRegression
# ...

df = pd.read_csv("data_processed.csv")

# Get features ready to model!
y = df.pop("cons_general").to_numpy()
y[y < 4] = 0
y[y >= 4] = 1

X = df

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=SEED)

# ...

# Train model
clf = make_pipeline(
    preprocessing,
    LogisticRegression()
)
clf.fit(X_train, y_train)


# Verify model
yhat = clf.predict(X_test)

acc = np.mean(yhat == y_test)
tn, fp, fn, tp = confusion_matrix(y_test, yhat).ravel()
specificity = tn / (tn + fp)
```

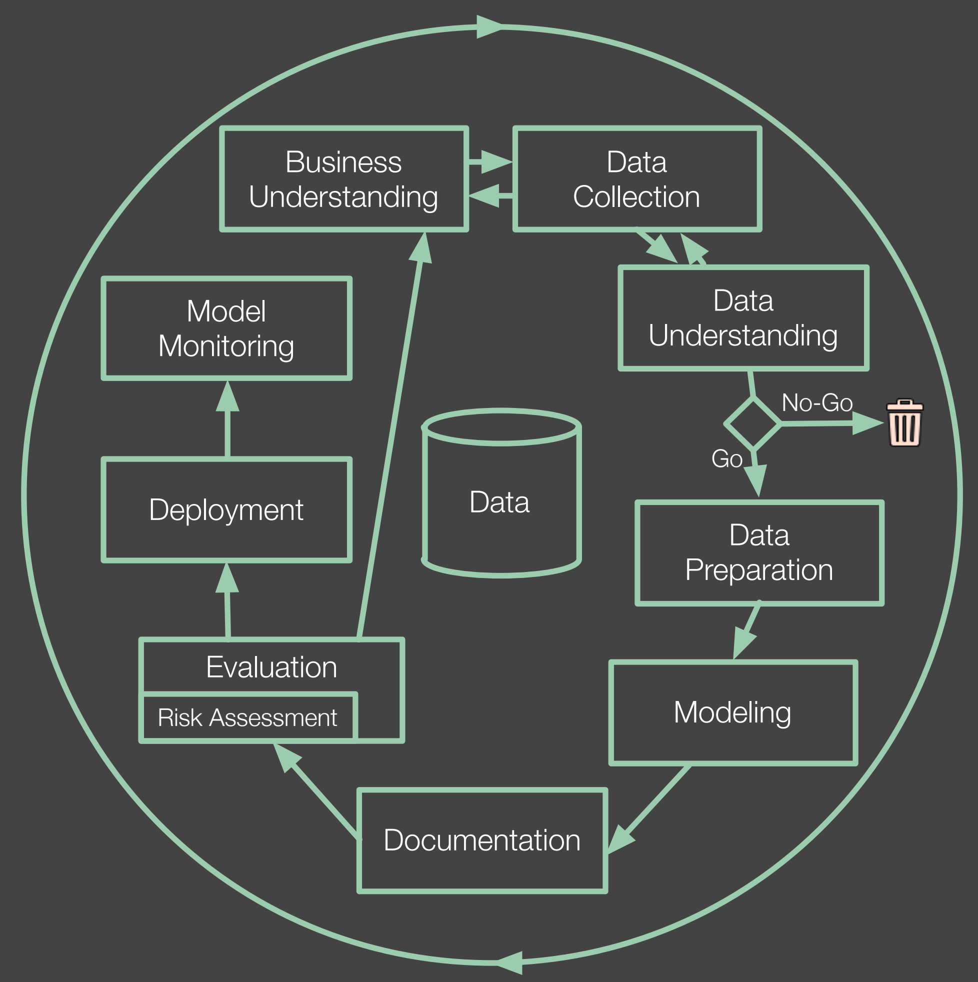
AI lifecycle

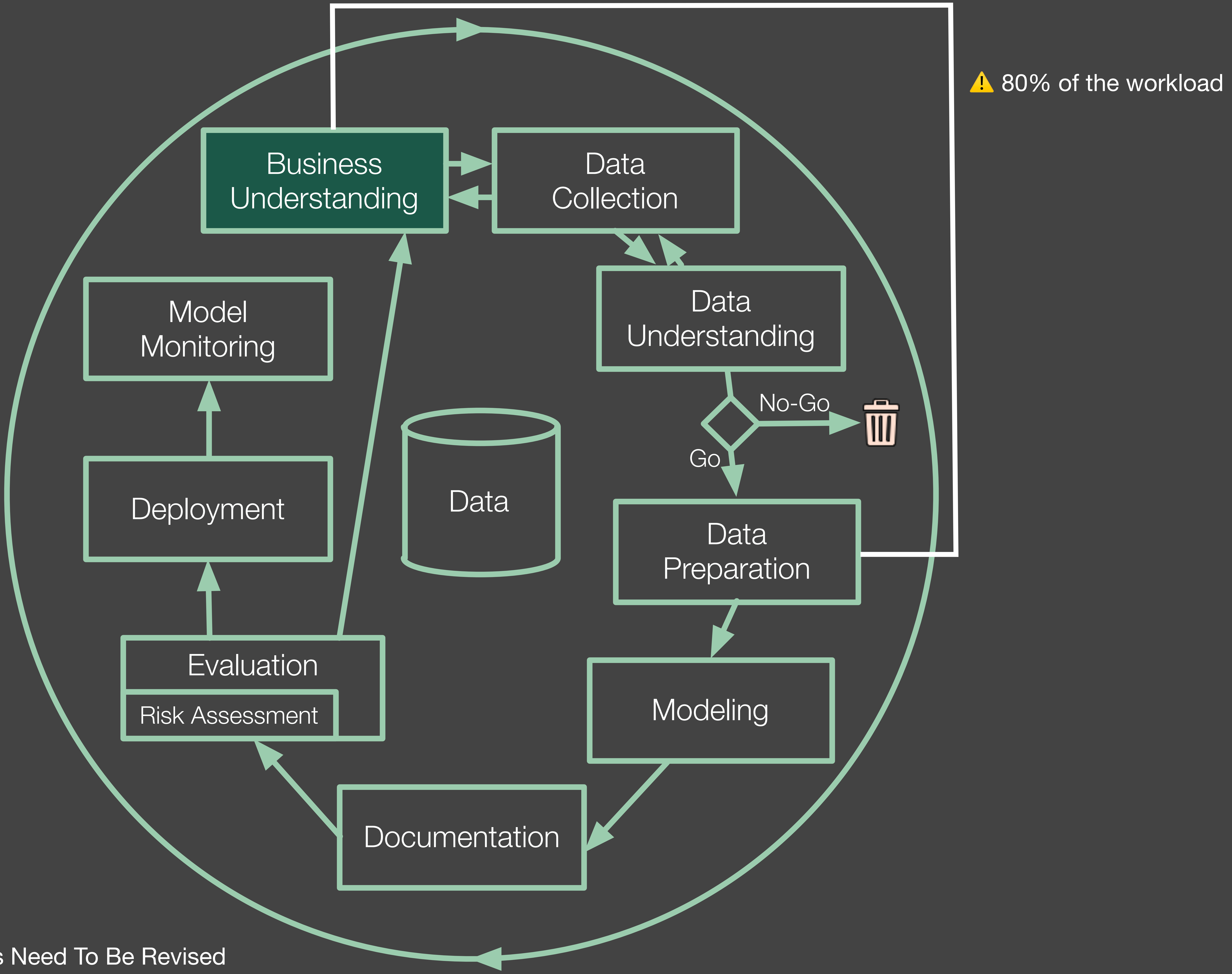
- CRISP-DM (2000)
- Microsoft TDSP (2017)
- Sculley et al. (2019)
- Haakman et al. (2021)
- …







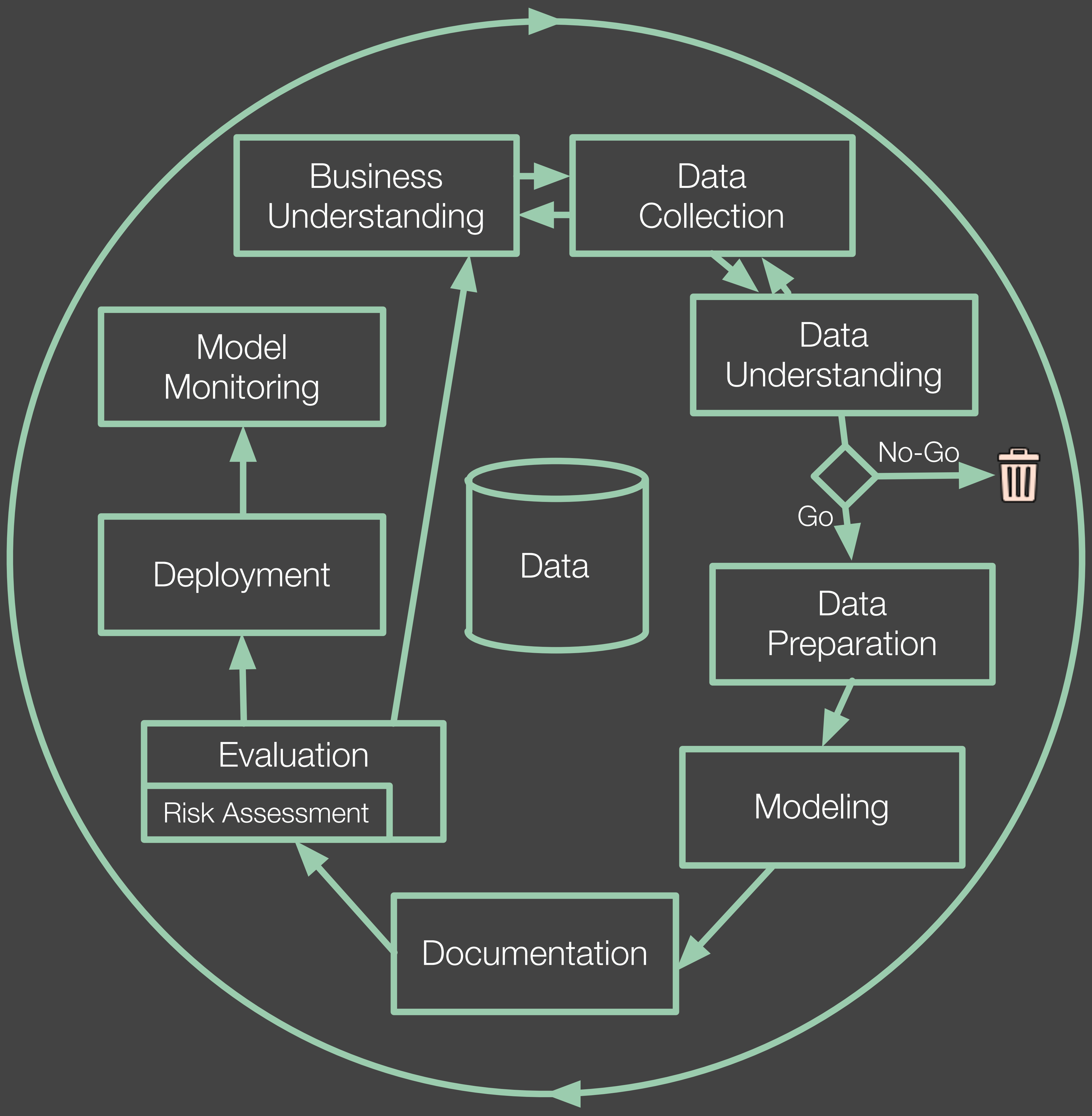




Haakman et al. (2020) – AI Lifecycle Models Need To Be Revised
https://arxiv.org/pdf/2010.02716.pdf

ML Artefacts

- Code
- Data
- Model



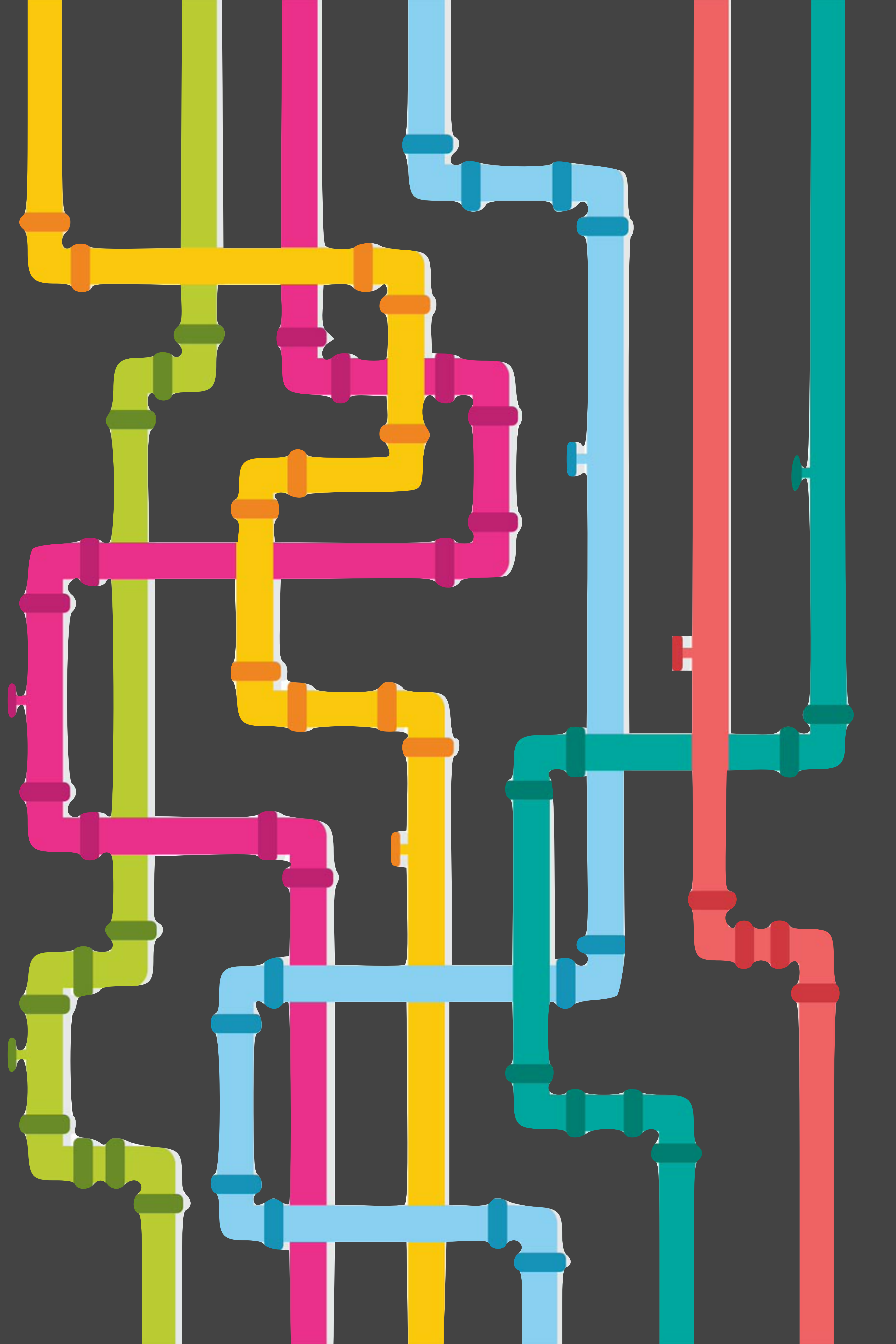
- Code
- Exploratory Data Analysis Reports (e.g., Jupiter notebooks)
- Data
- Clean Data
- Feature Engineered
- Model
- Performance Report
- Docs
- Container

- How to version large-scale data?

- How to avoid processing large-scale data every time you change something?

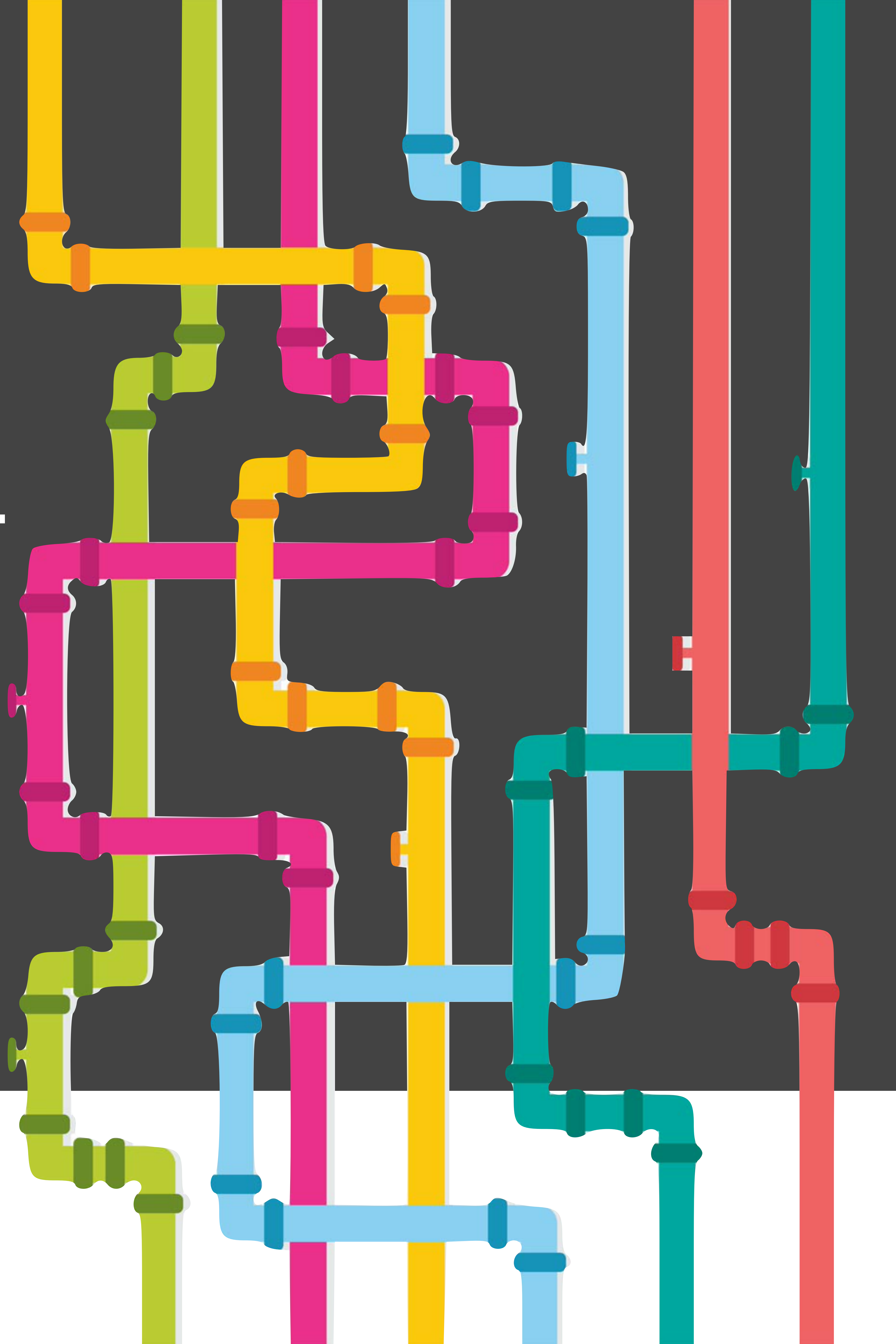
- How to guide collaborators to re-run the right scripts whenever something changed?

- How to keep track of different versions of the pipeline?



The traditional way of automating the build pipeline is through Makefile, Maven, Gradle, etc.

There are solutions for Machine Learning as well.



Maven™

Gradle

Makefile for Machine Learning

Makefile

```
.PHONY: clean data lint requirements
## ...

## Install Python Dependencies
requirements: test_environment
	$(PYTHON_INTERPRETER) -m pip install -U pip setuptools wheel
	$(PYTHON_INTERPRETER) -m pip install -r requirements.txt

## Make Dataset
data: requirements
	$(PYTHON_INTERPRETER) src/data/make_dataset.py data/raw data/processed

## Delete all compiled Python files
clean:
	find . -type f -name "*.py[co]" -delete
	find . -type d -name "__pycache__" -delete

## Lint using flake8
lint:
	flake8 src

## ...
```

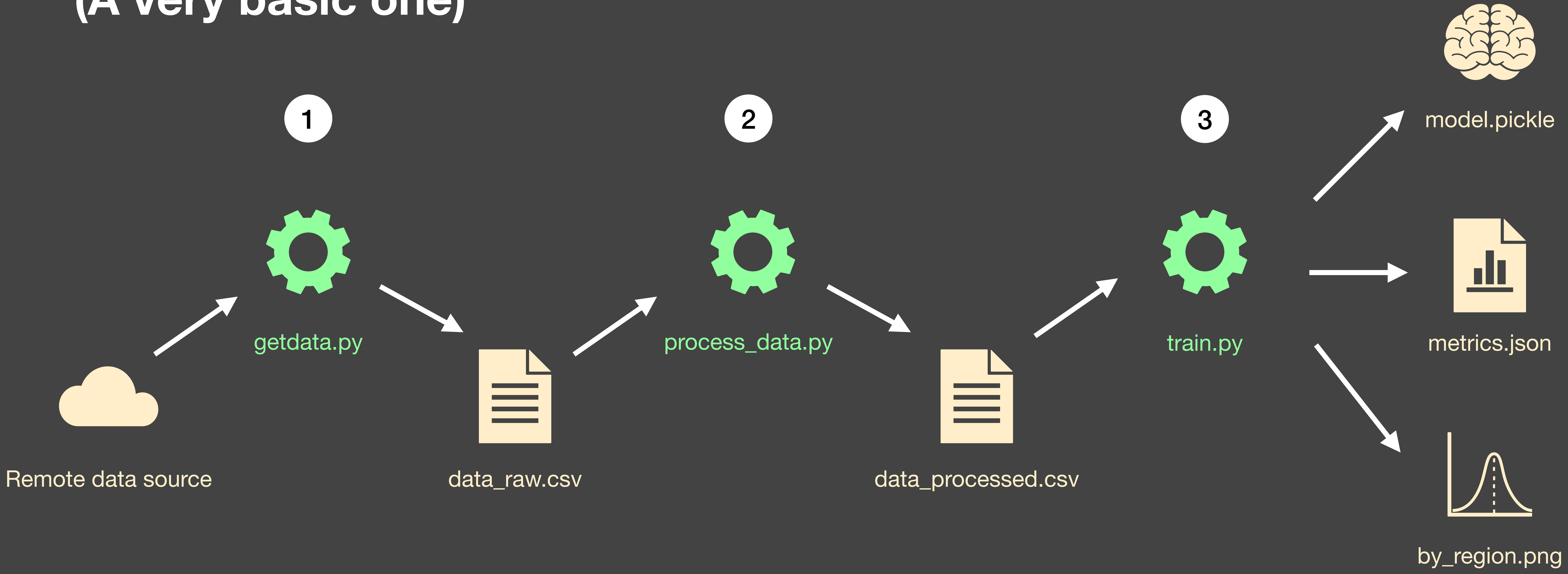
Suggested Read: “Make My Day…ta Science Easier” by David Stevens. URL: https://edu.nl/eaxag
Makefile Example: https://edu.nl/a78xy

DVC

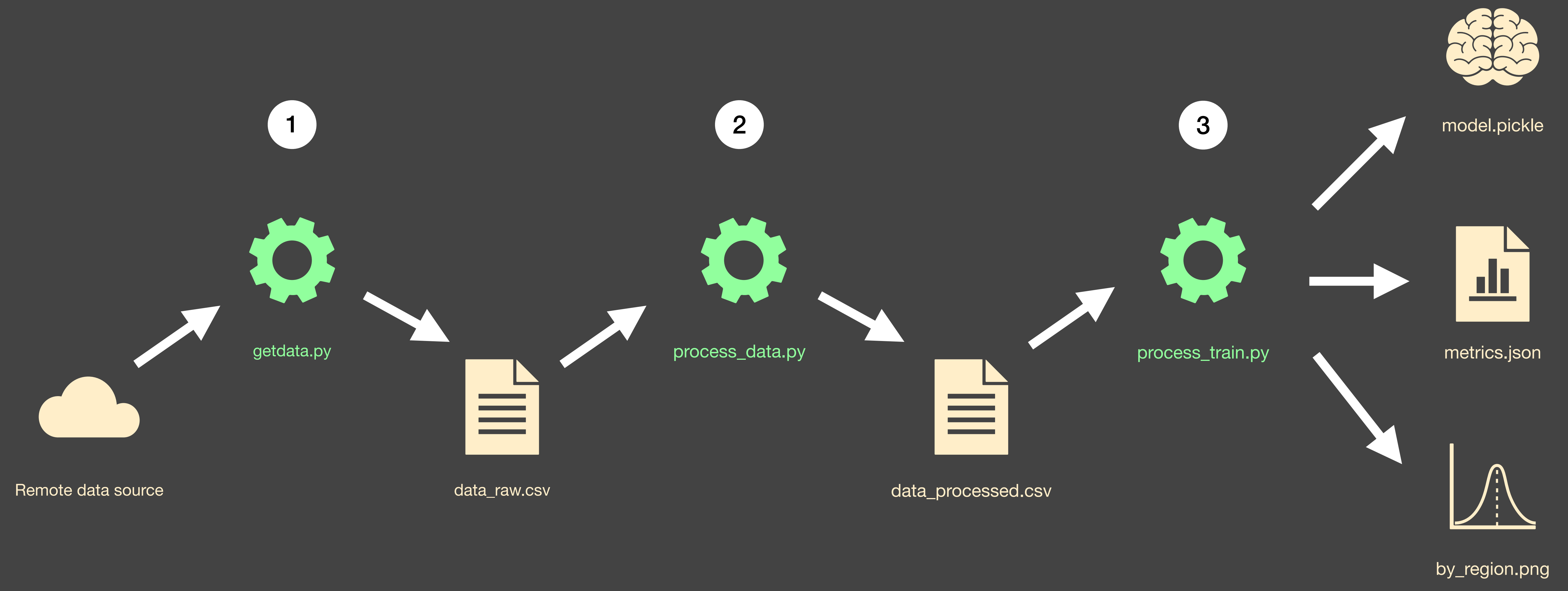
- Open-source tool.
- Automate pipelines.
- Remote storage setup.
- Version control for data, models (and other intermediate artefacts).
- Experiment management.
- Website: https://dvc.org

Example of a pipeline

(A very basic one)



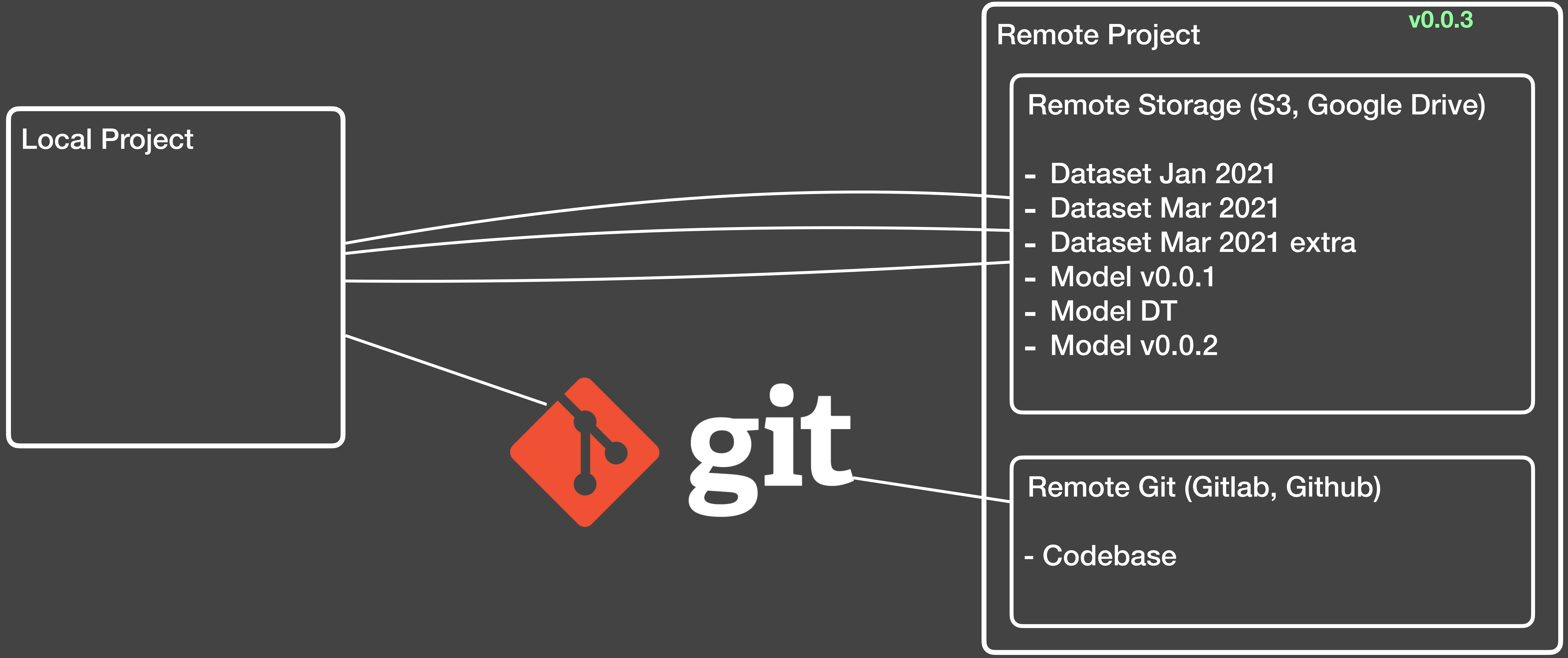
Example of a pipeline



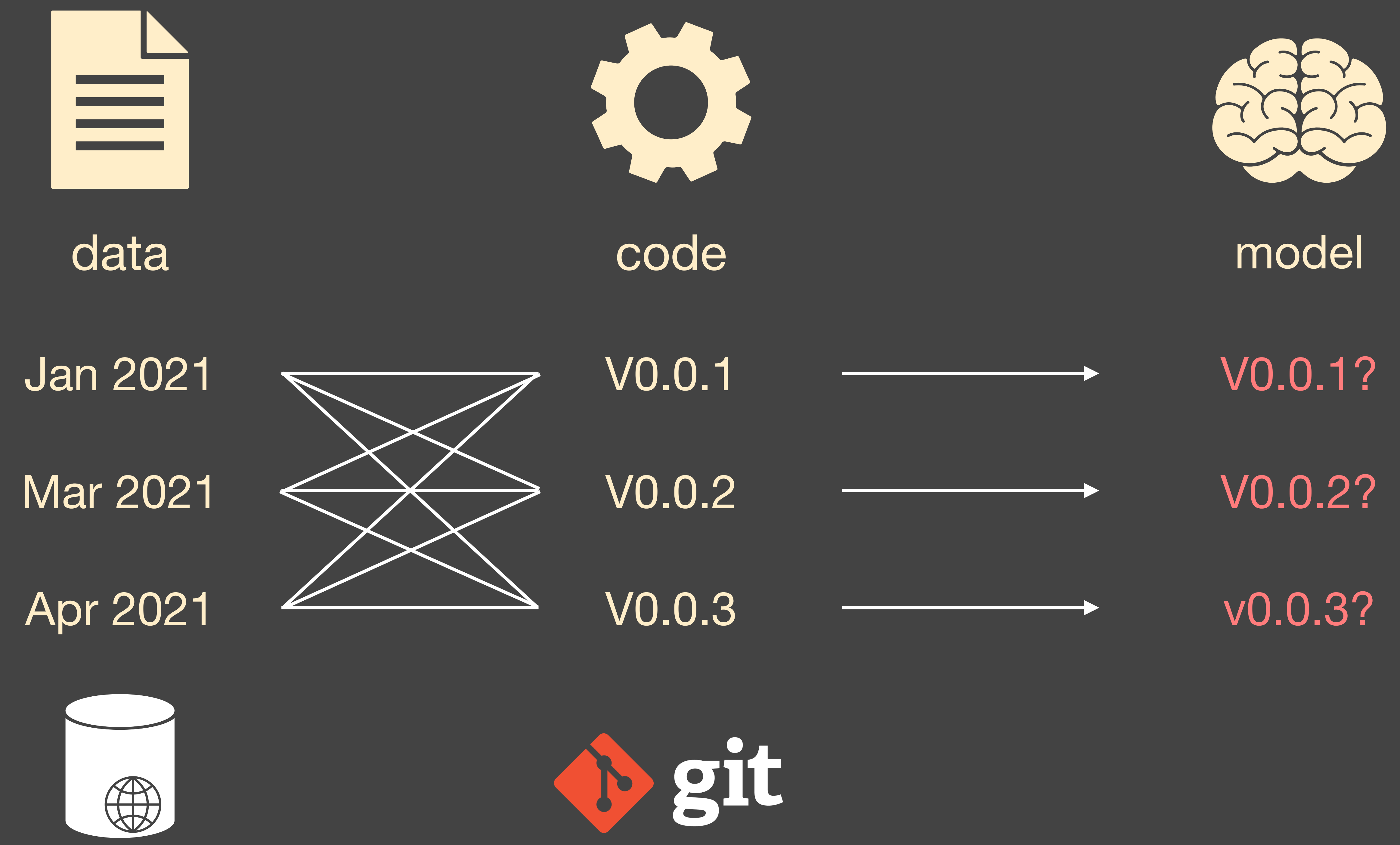
dvc.yml

```yaml
stages:
  get_data:
    cmd: python get_data.py
    deps:
    - get_data.py
    outs:
    - data_raw.csv
  process:
    cmd: python process_data.py
    deps:
    - process_data.py
    - data_raw.csv
    outs:
    - data_processed.csv
  train:
    cmd: python train.py
    deps:
    - train.py
    - data_processed.csv
    outs:
    - by_region.png
    - model.pickle
    metrics:
    - metrics.json:
        cache: false
```

Data Version Control
(and other artefacts)

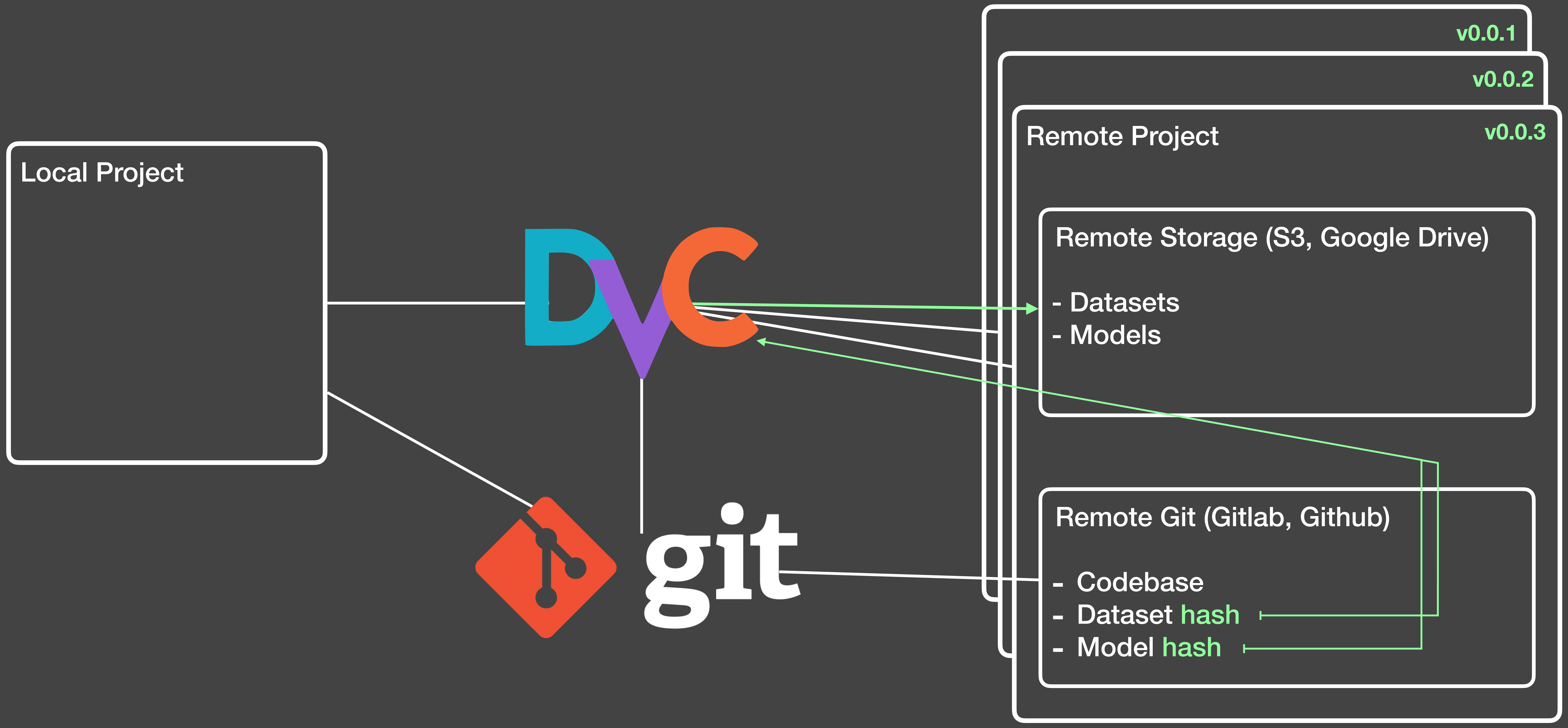


data
code
model
Jan 2021
V0.0.1
V0.0.1
Mar 2021
V0.0.2
V0.0.2
Apr 2021
V0.0.3
v0.0.3
git


data
code
model
Jan 2021
Mar 2021
Apr 2021
V0.0.1
V0.0.2
V0.0.3
V0.0.1?
V0.0.2?
v0.0.3?
git

Data Version Control

(and other artefacts)



Code smells in ML

The Prevalence of Code Smells in Machine Learning projects

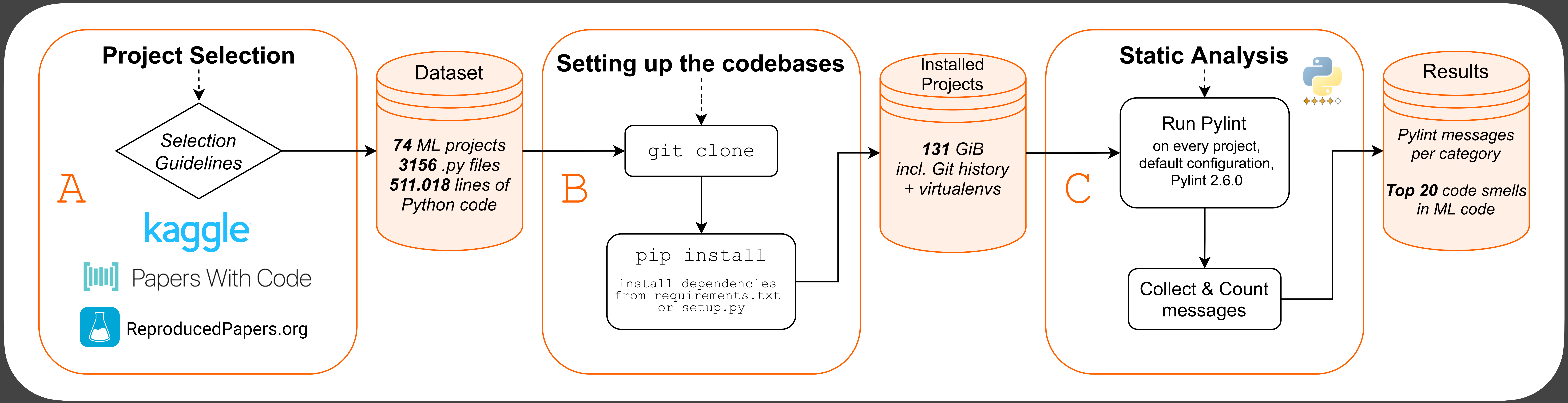
Bart van Oort[1,2], Luís Cruz[2], Maurício Aniche[2], Arie van Deursen[2]
Delft University of Technology
[1] *AI for Fintech Research, ING*
[2] Delft, Netherlands
bart.van.oort@ing.com, {l.cruz, m.f.aniche, arie.vandeursen}@tudelft.nl

***Abstract*—Artificial Intelligence (AI) and Machine Learning (ML) are pervasive in the current computer science landscape. Yet, there still exists a lack of software engineering experience and best practices in this field. One such best practice, static code analysis, can be used to find code smells, i.e., (potential) defects in the source code, refactoring opportunities, and violations of common coding standards. Our research set out to discover the most prevalent code smells in ML projects. We gathered a dataset of 74 open-source ML projects, installed their dependencies and ran Pylint on them. This resulted in a top 20 of all detected code smells, per category. Manual analysis of these smells**

which we amalgamate into 'code smells' for the rest of this paper. Research has shown that the attributes of quality most affected by code smells are maintainability, understandability and complexity, and that early detection of code smells reduces the cost of maintenance [7].

With a focus on the maintainability and reproducibility of ML projects, the goal of our research is therefore to apply static code analysis to applications of ML, in an attempt to uncover the frequency of code smells in these projects and

https://arxiv.org/abs/2103.04146

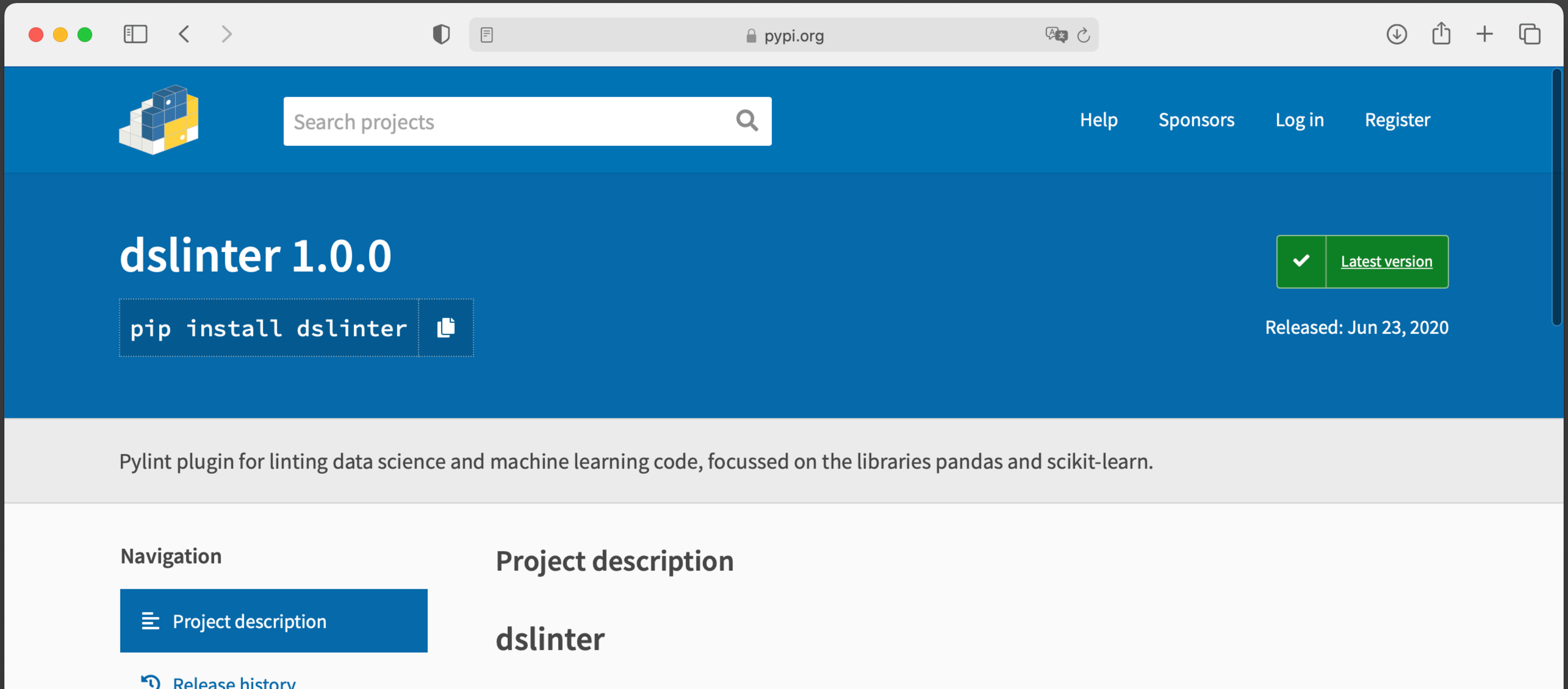

Project Selection
Selection Guidelines
A
kaggle
Papers With Code
ReproducedPapers.org
Dataset
74 ML projects
3156 .py files
511.018 lines of Python code
Setting up the codebases
B
git clone
pip install
install dependencies from requirements.txt or setup.py
Installed Projects
131 GiB incl. Git history + virtualenvs
Static Analysis
C
Run Pylint
on every project, default configuration, Pylint 2.6.0
Collect & Count messages
Results
Pylint messages per category
Top 20 code smells in ML code

Results

- Naming conventions do not apply for ML cases, due to its resemblance with mathematical notation.
- Code duplication is a common issue in ML applications
- There are several flaws when specifying dependencies. Many projects did not even have any written config.
- Pylint poses several incompatibilities with ML-specific libraries. Too many false positives.

Code Smells for ML

Code Smells for ML



Code Smells for ML

- **Unassigned DataFrame Checker:** Operations on DataFrames return new DataFrames. These DataFrames should be assigned to a variable.
- **DataFrame Iteration Checker:** Vectorized solutions are preferred over iterators for DataFrames.
- **Nan Equality Checker:** Values cannot be compared with np.nan, as np.nan != np.nan.
- **Hyperparameter Checker:** For (scikit-learn) learning algorithms, all hyperparameters should be set.
- **Import Checker:** Check whether data science modules are imported using the correct naming conventions.
- **Data Leakage Checker:** All scikit-learn estimators should be used inside Pipelines, to prevent data leakage between training and test data.