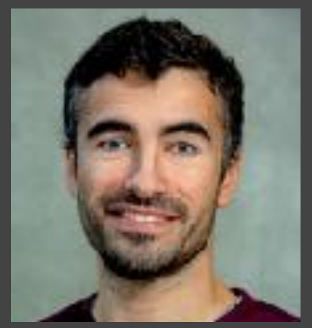
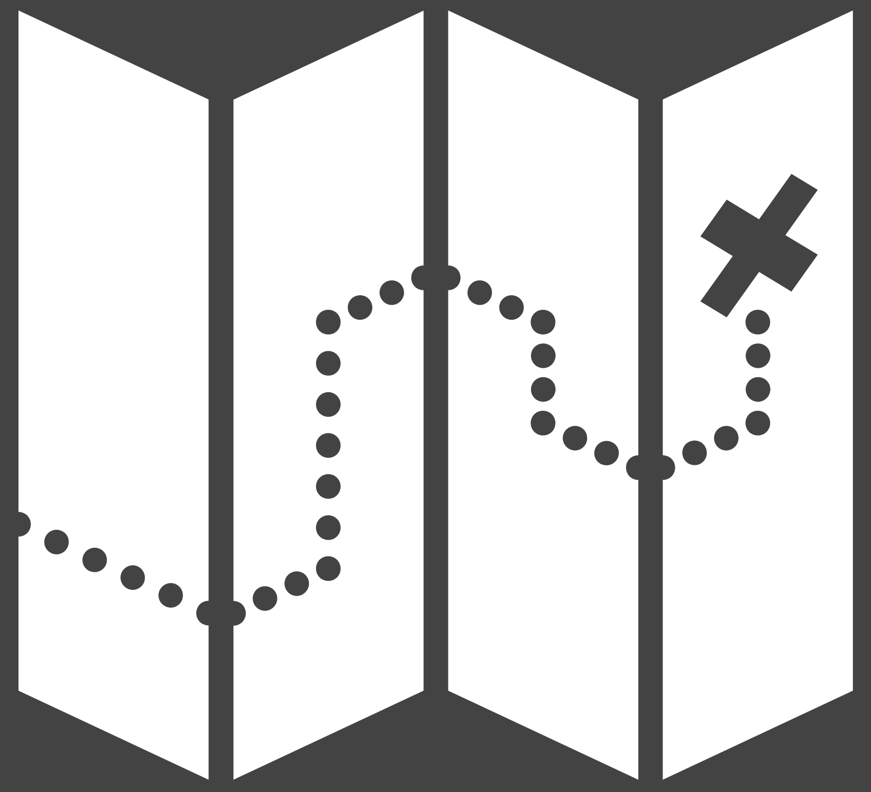
ML Pipelines & Code Quality

Release Engineering for Machine Learning Applications (REMLA, CS4295)


Luís Cruz
L.Cruz@tudelft.nl


Sebastian Proksch
S.Proksch@tudelft.nl

Outline

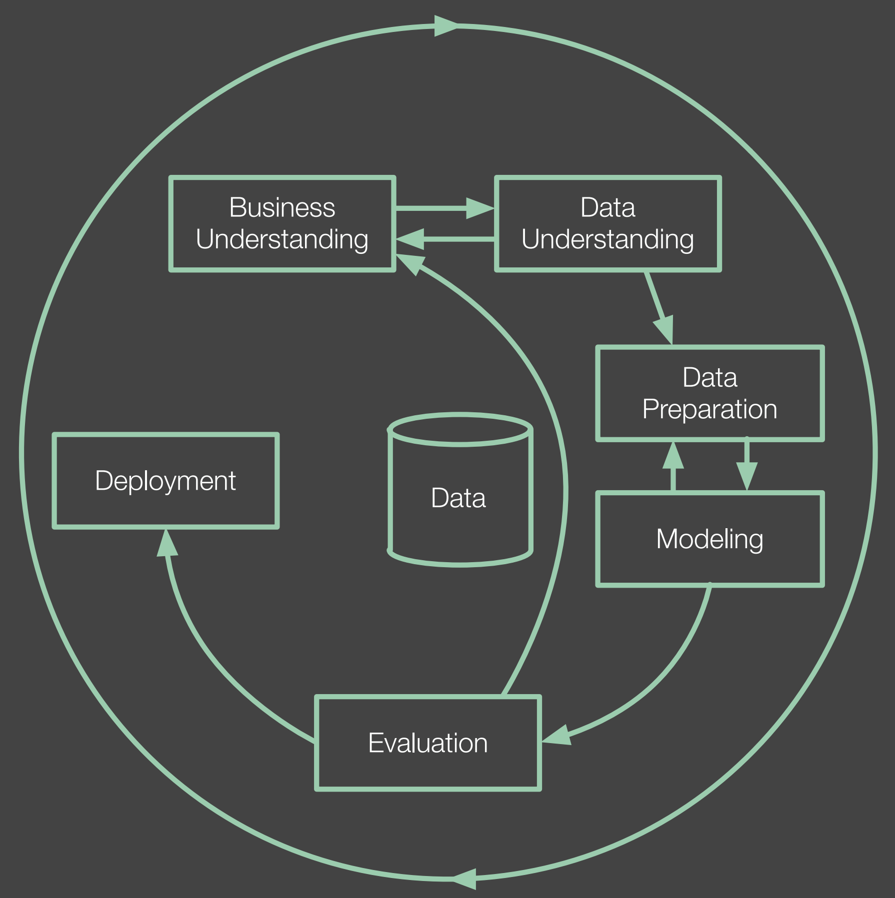


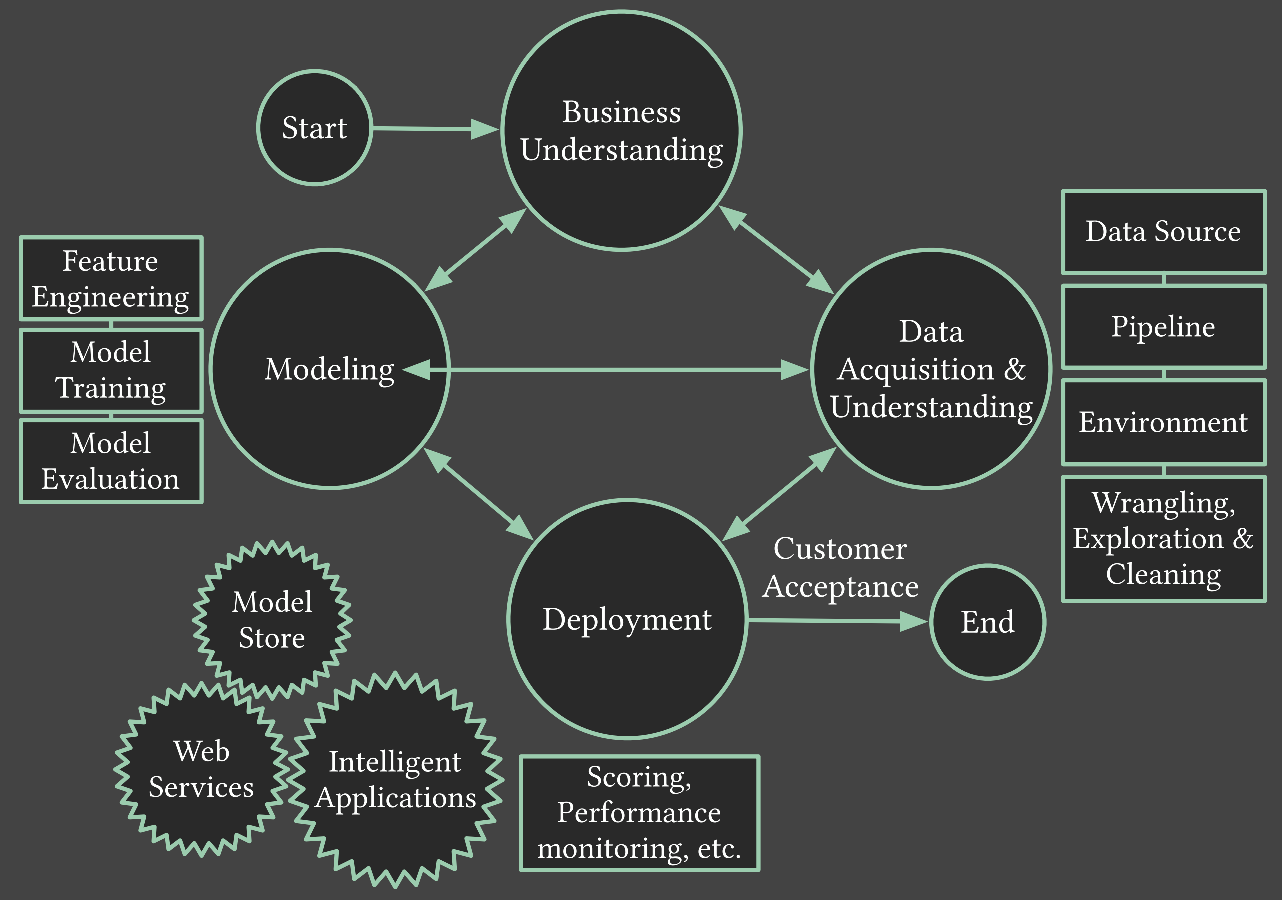
- AI lifecycle
- Pipeline Management
- ML version control
- Code smells in ML
- Code smells for ML
- ML Project boilerplate

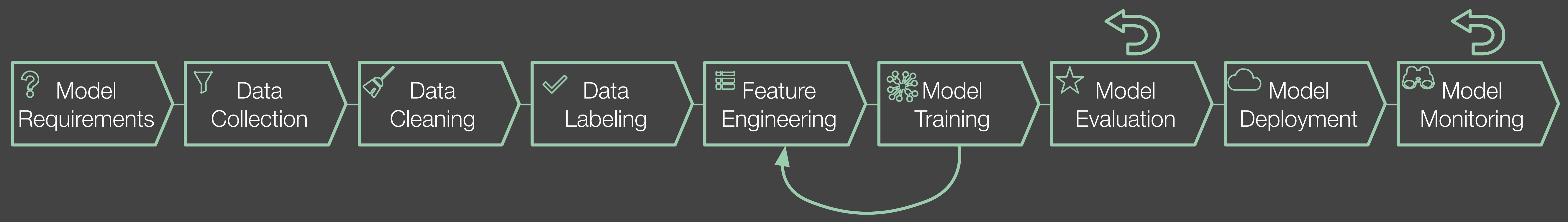
```python
import pandas as pd
from sklearn.linear_model import LogisticRegression
# ...

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)
```

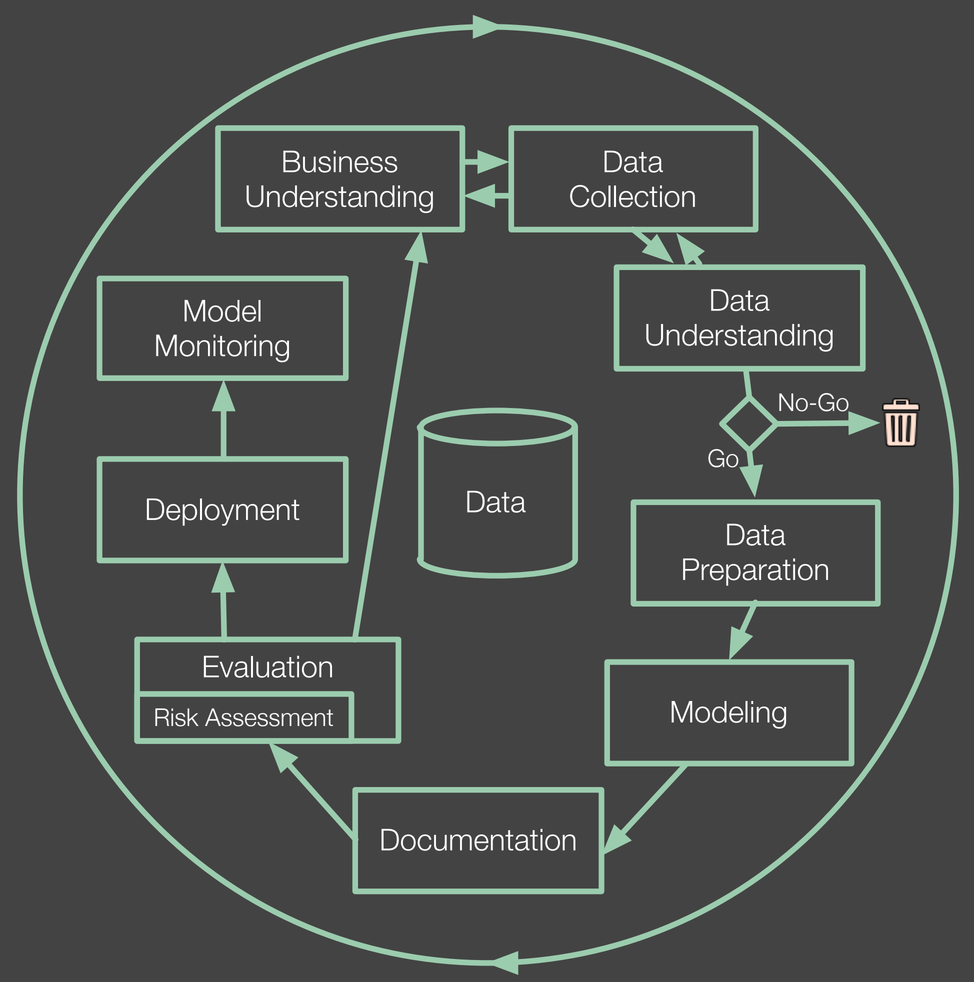
AI lifecycle

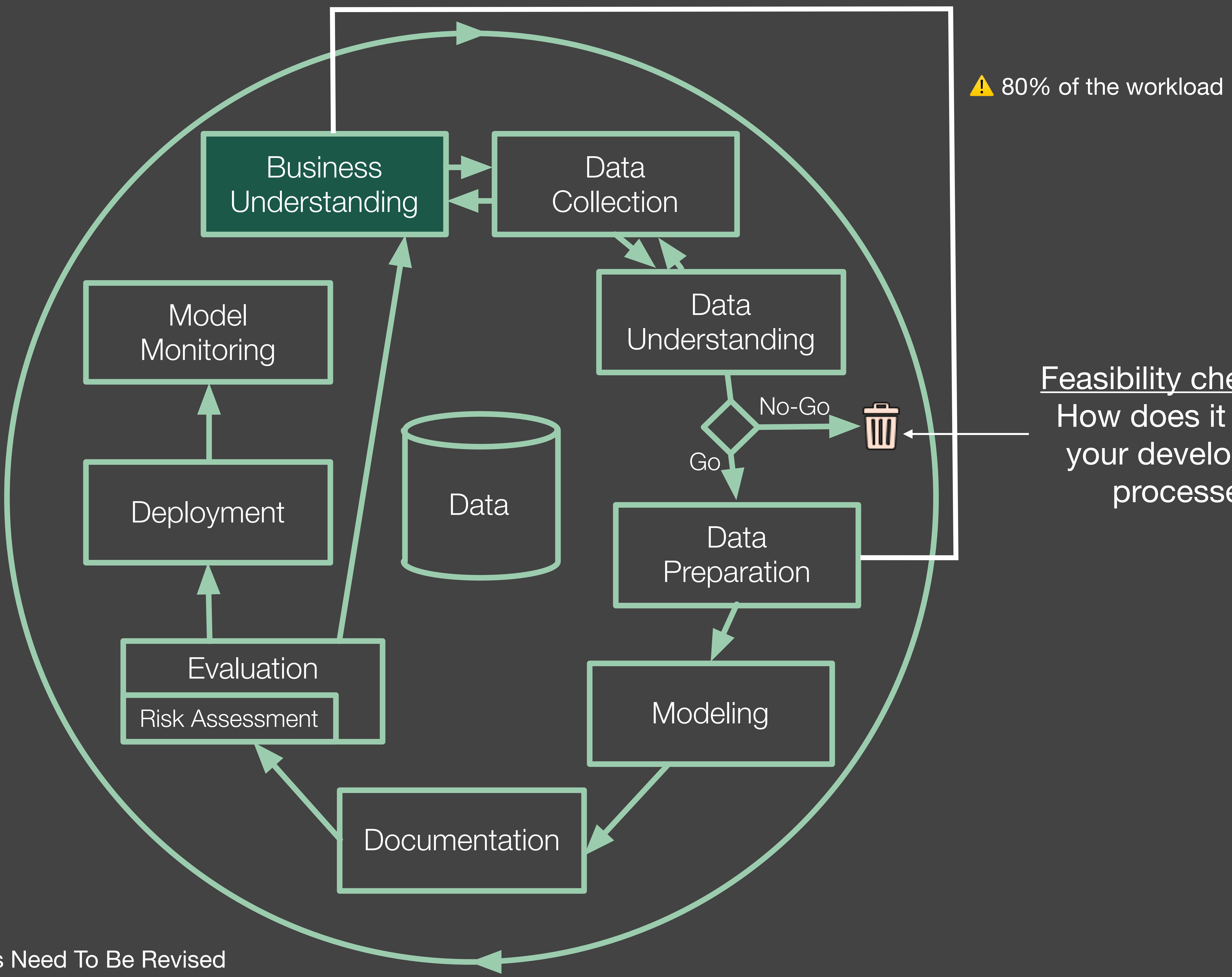
- CRISP-DM (2000)
- Microsoft TDSP (2017)
- Amershi et al. (2019)
- Haakman et al. (2021)
- …









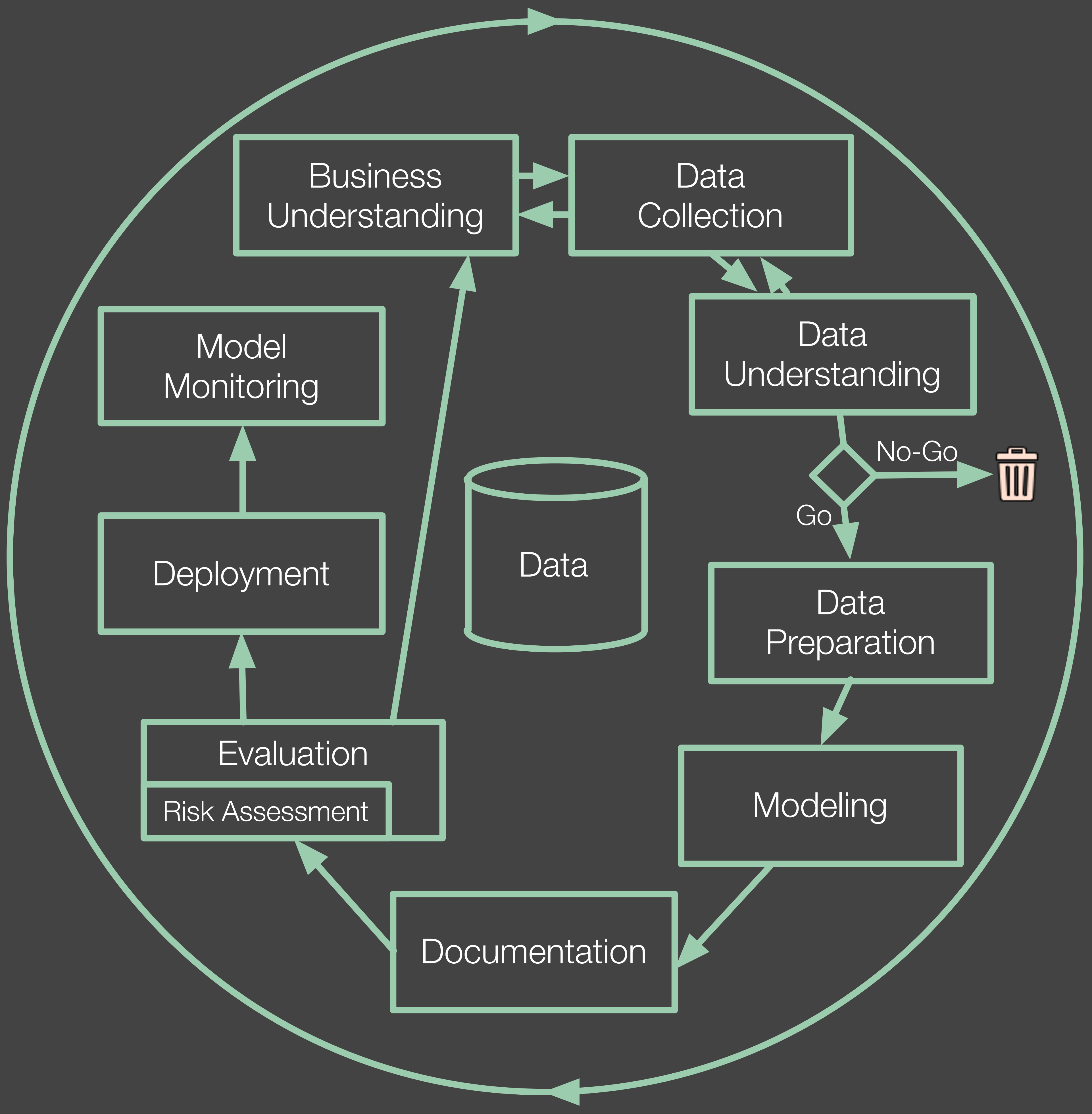


<u>Feasibility checkpoint</u>
How does it impact your development processes?

Haakman et al. (2021) – 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

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

Data Preparation

```python
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)
```

```python
# ...
```

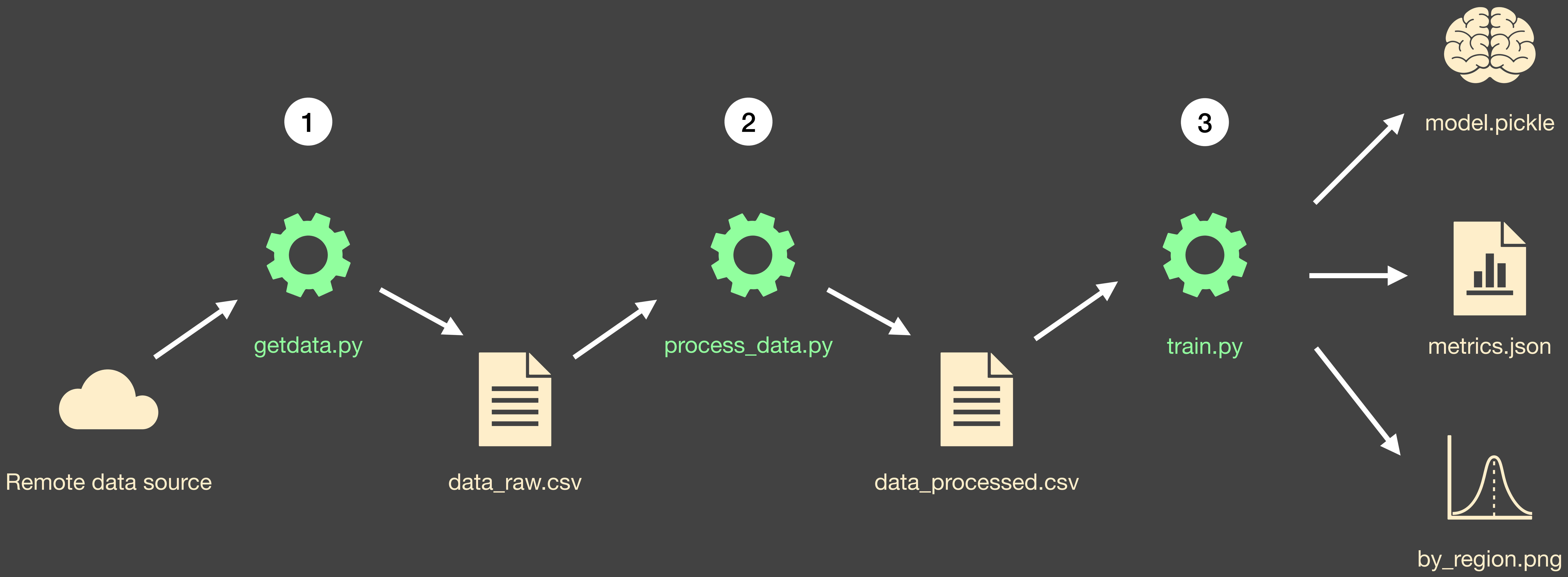
Model Training

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

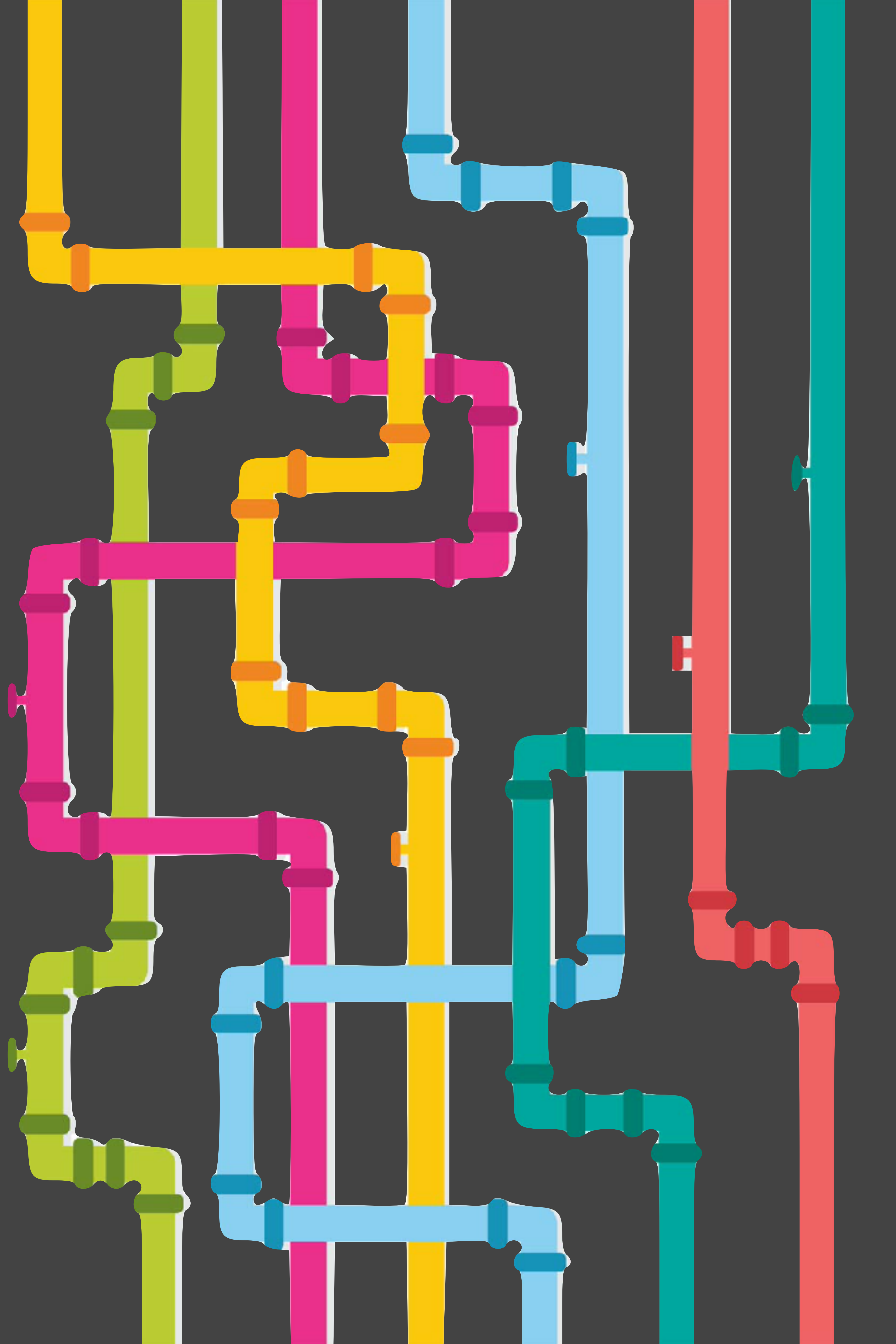
Model Validation

```python
# 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)
```

Example of a basic modular pipeline

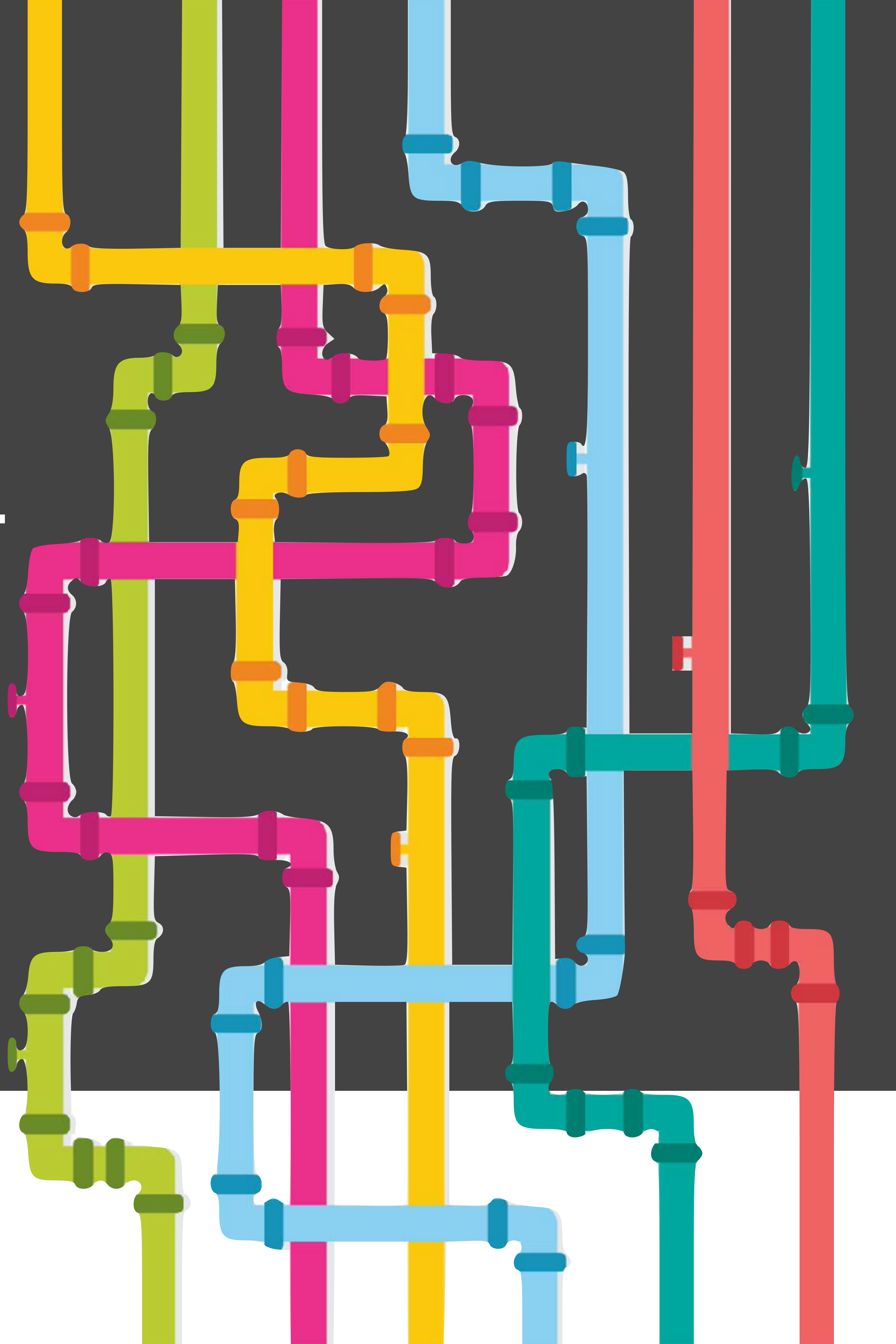


- Each stage tends to require their own code to process the data.
  - How to avoid running the whole processing pipeline every time you change something?
- Imagine that you are assynchronously working with other 3 ML engineers/ Data scientists.
  - How to guide collaborators to re-run the right scripts whenever something is changed?



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

There are solutions for Machine Learning as well.



Makefile for Machine Learning

Makefile

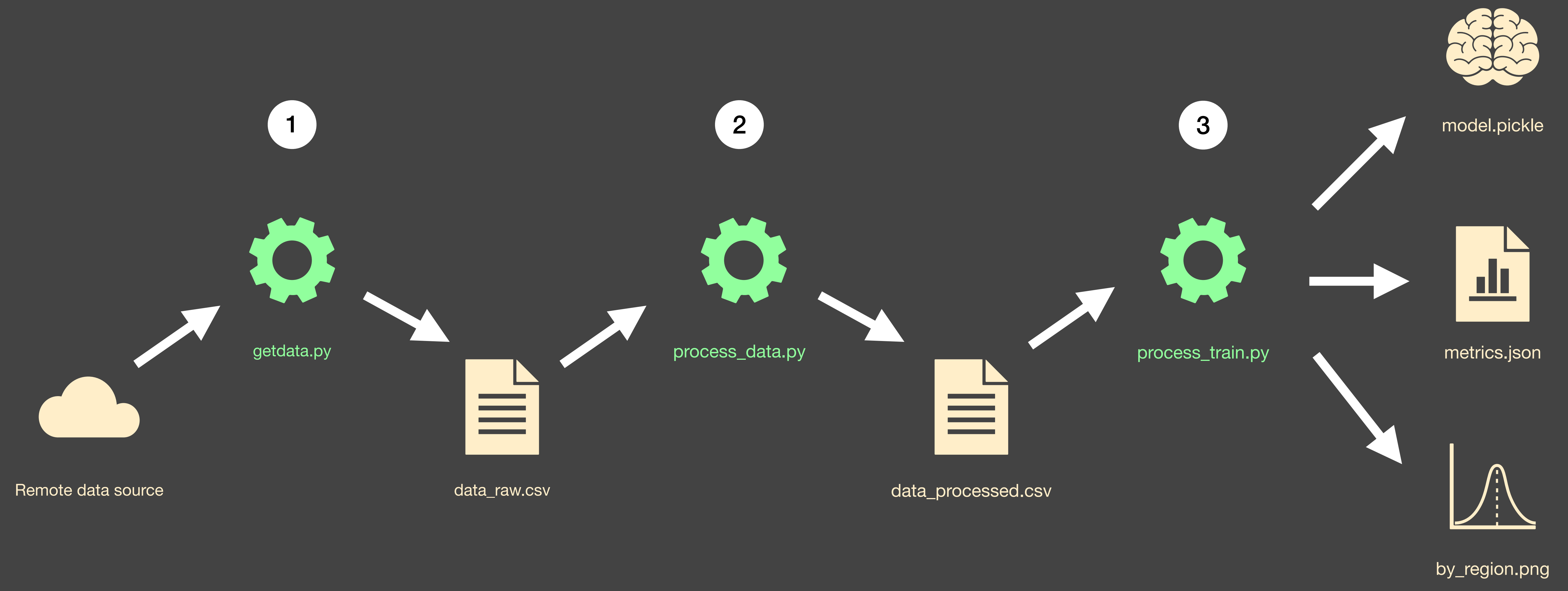
```
.PHONY: clean data lint requirements
## ...

## 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

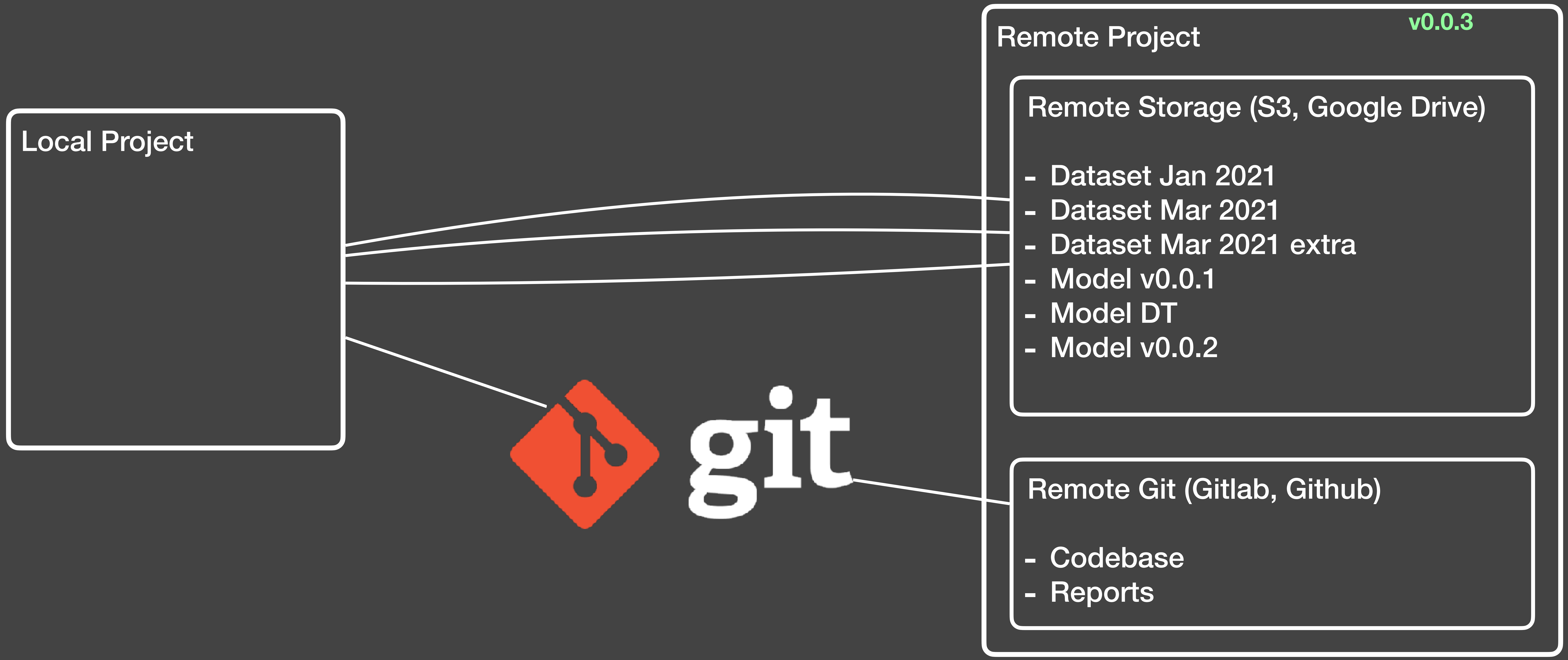
Example of a pipeline



dvc.yml

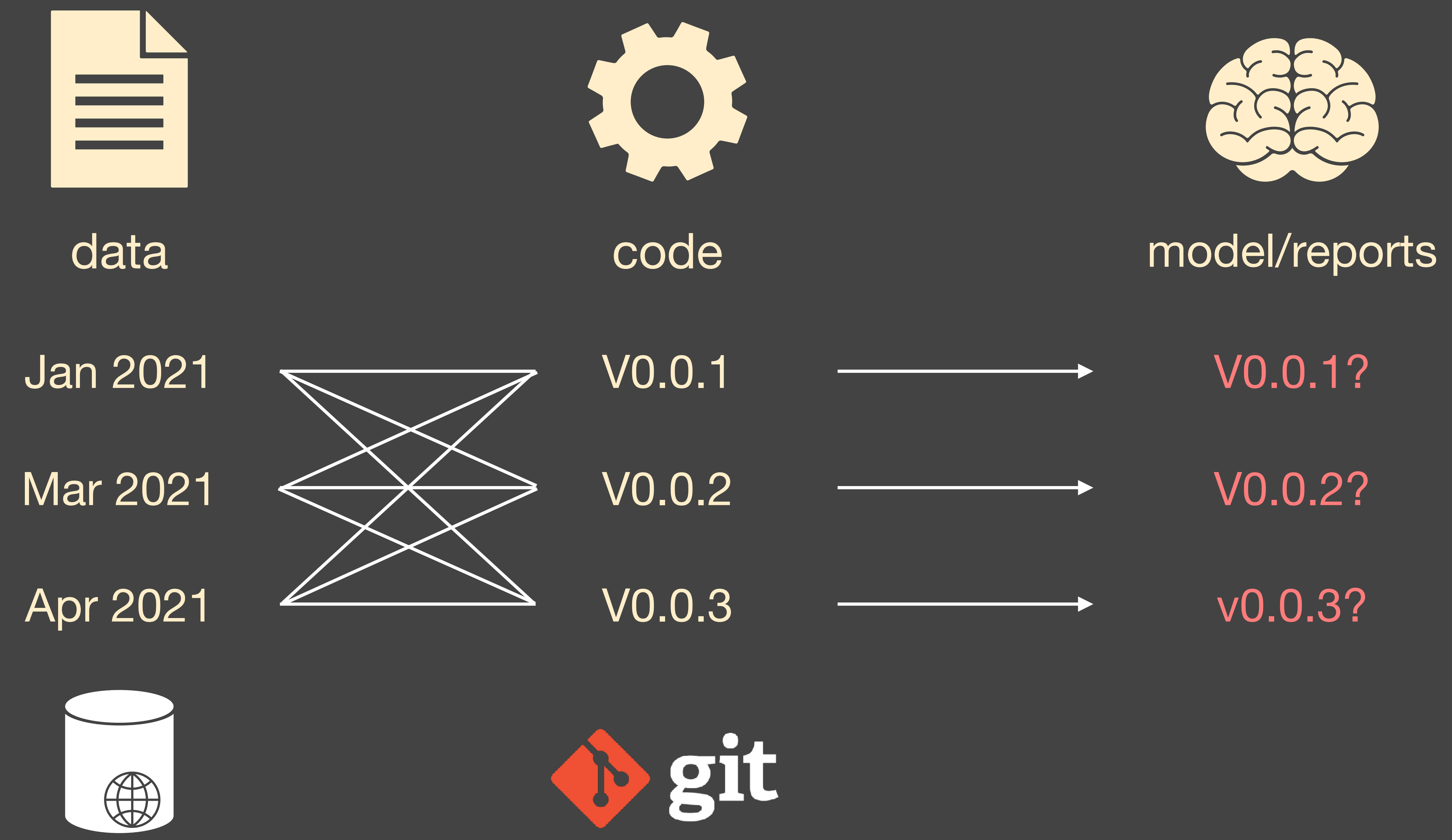
```yaml
stages:
  get_data:
    cmd: python get_data.py
    deps:
    - get_data.py
    outs:
    - data_raw.csv
  process:
    cmd: python process_data.py
    deps:
    - process_data.py
    - data_raw.csv
    outs:
    - data_processed.csv
  train:
    cmd: python train.py
    deps:
    - train.py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

Data Version Control
(and other artefacts)

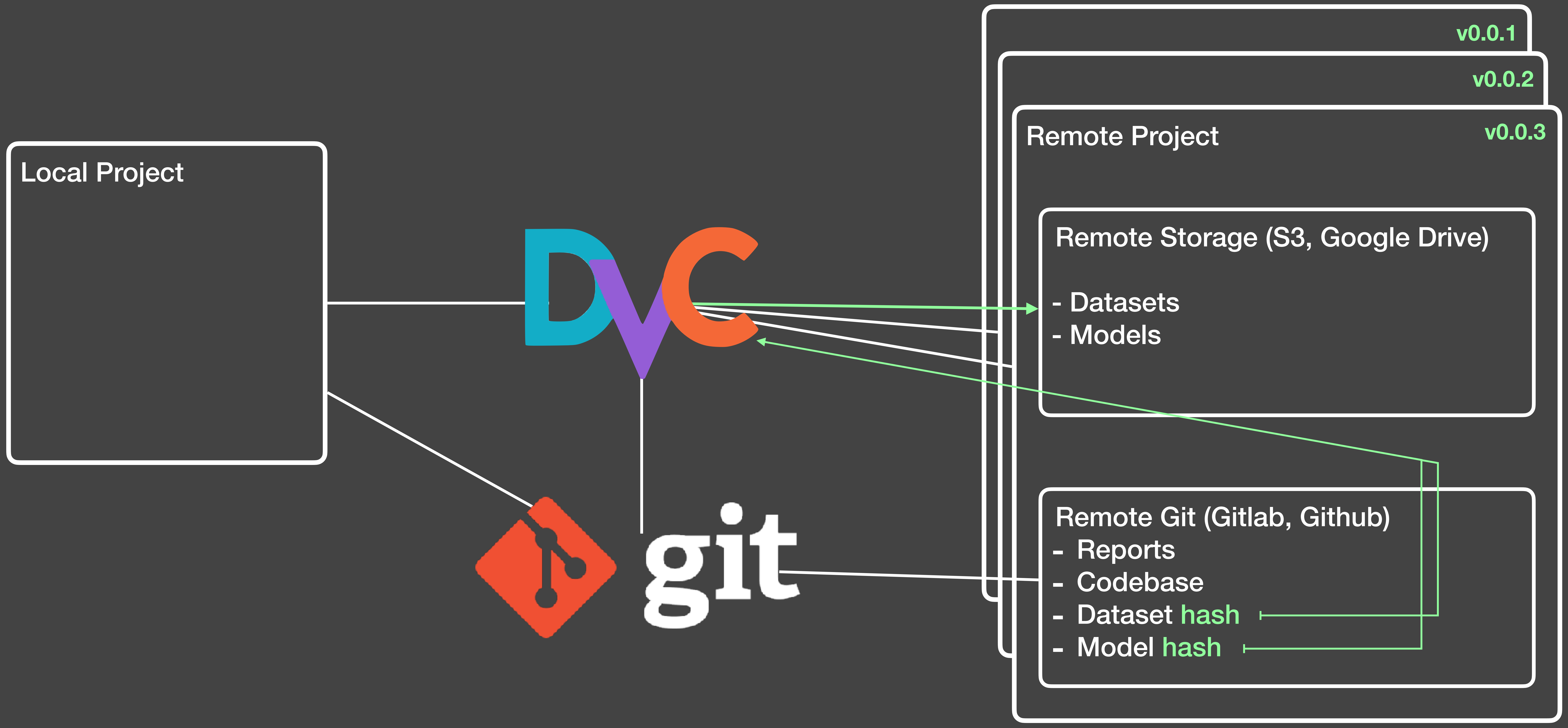


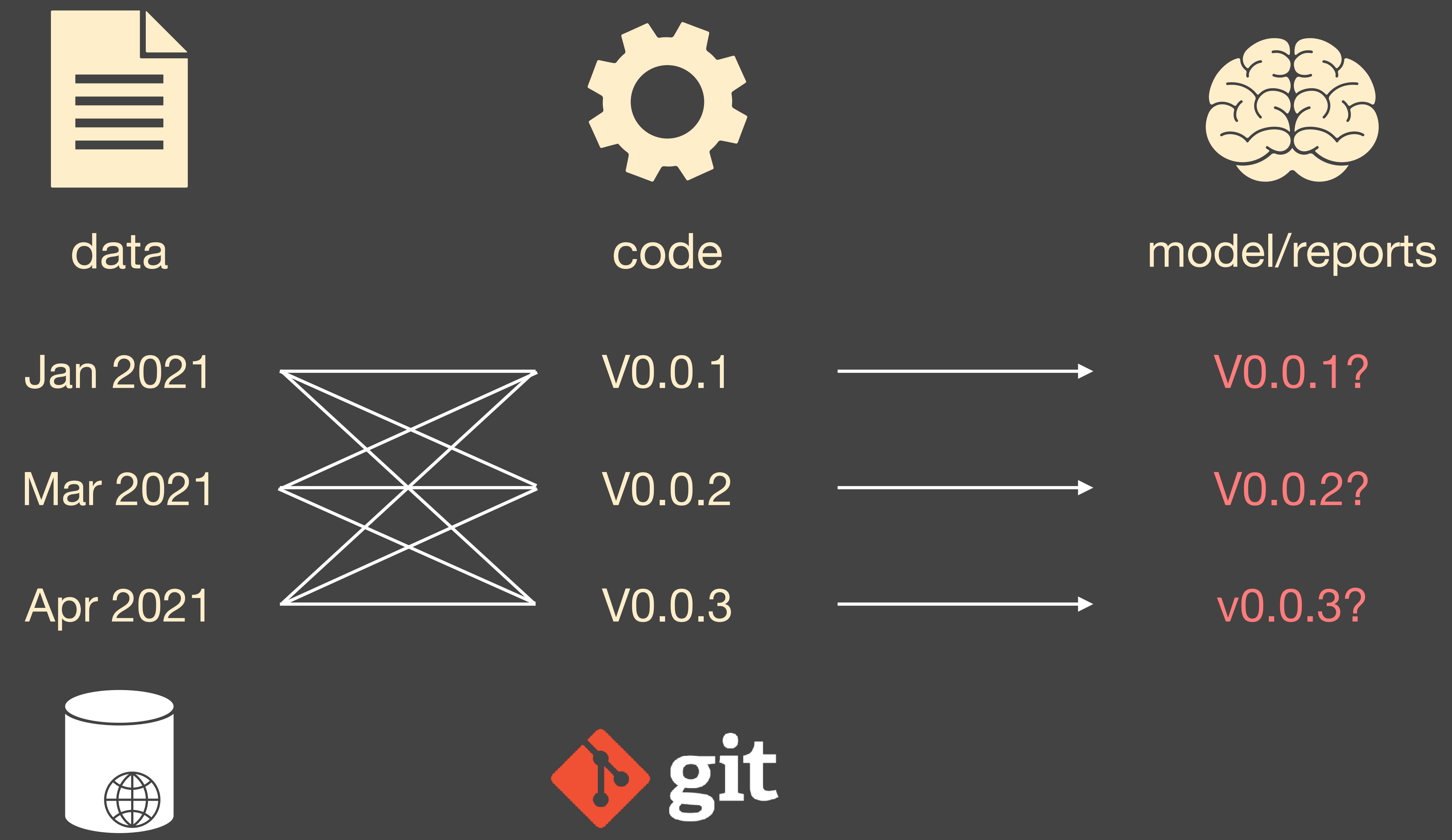
data		code		model/reports
Jan 2021	—	V0.0.1	→	V0.0.1
Mar 2021	—	V0.0.2	→	V0.0.2
Apr 2021	—	V0.0.3	→	v0.0.3

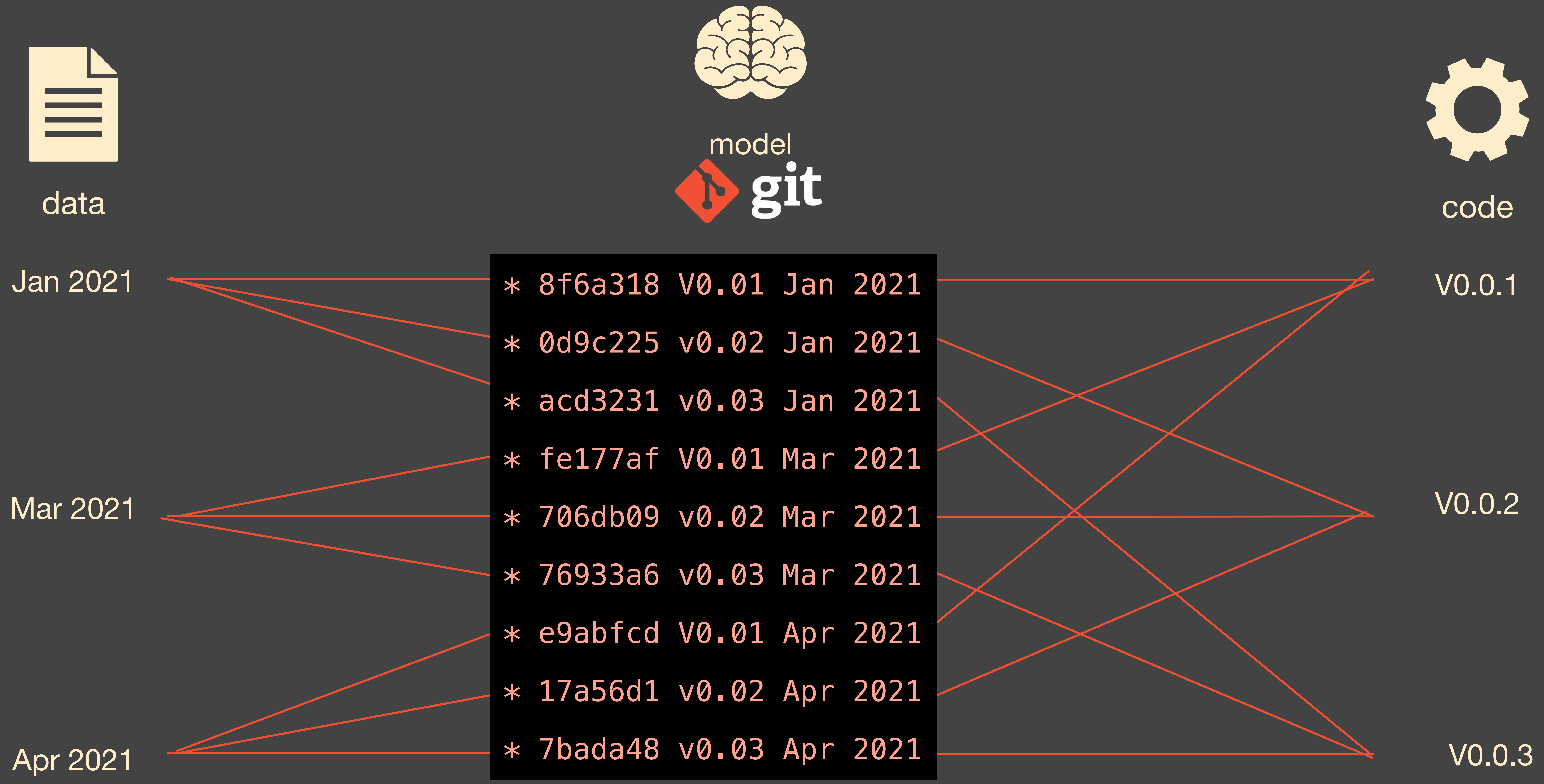
git


data
code
model/reports
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)




data
code
model/reports
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
model
git
code
Jan 2021
Mar 2021
Apr 2021
* 8f6a318 V0.01 Jan 2021
* 0d9c225 v0.02 Jan 2021
* acd3231 v0.03 Jan 2021
* fe177af V0.01 Mar 2021
* 706db09 v0.02 Mar 2021
* 76933a6 v0.03 Mar 2021
* e9abfcd V0.01 Apr 2021
* 17a56d1 v0.02 Apr 2021
* 7bada48 v0.03 Apr 2021
V0.0.1
V0.0.2
V0.0.3

more in the next class…

Code quality in ML projects

- Pair-programming
- Manual code review
- Guidelines/Checklists
- …
- Static analysis

Code smells in ML projects

- What is a code smell?
  - Any code pattern that **may** indicate a deeper problem in the project.
- We already have a long list of code smells for software projects.
- Can you name a few tools that help you detect code smells?
  - For python: pylint, flake8, Bandit, etc.
- How do traditional code smells fit the realm of ML projects?

Code smells in ML

The Prevalence of Code Smells in Machine Learning projects

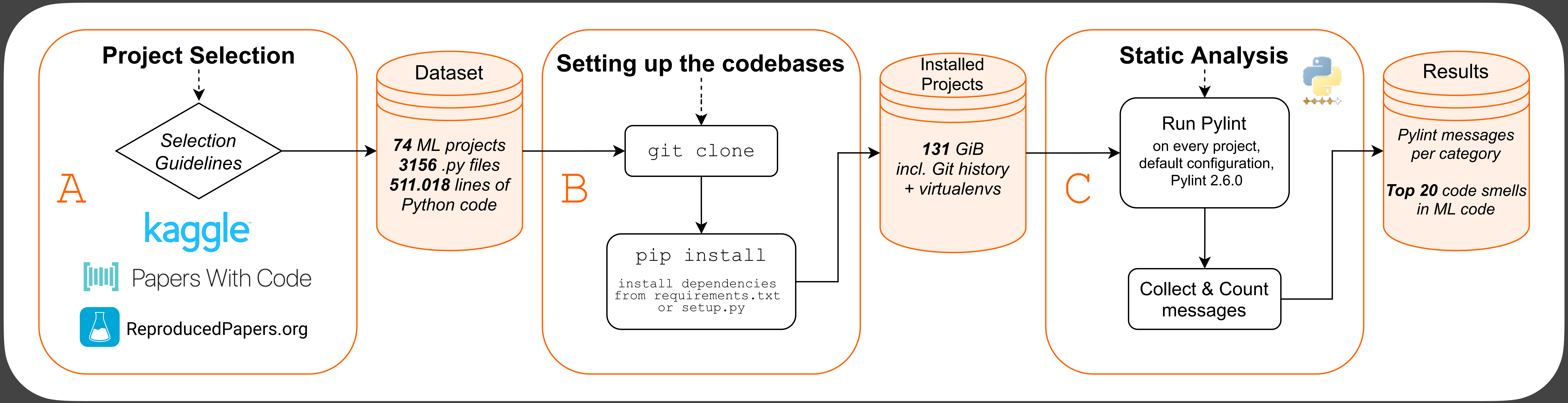
Bart van Oort[1,2], Luís Cruz[2], Maurício Aniche[2], Arie van Deursen[2]
Delft University of Technology
[1] *AI for Fintech Research, ING*
[2] Delft, Netherlands
bart.van.oort@ing.com, {l.cruz, m.f.aniche, arie.vandeursen}@tudelft.nl

***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
git clone
B
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.
- Bottom line: you configure your linter so that it fits your project/conventions.

Pandas snipet

```
import pandas as pd
df = pd.DataFrame([-1])
df.abs()
print(df)
```

```
>     0
   0 -1
```

Also a problem with other libraries.

Numpy snipet

```
import numpy as np
zhats = [2, 3, 1, 0]
np.clip(zhats, -1, 1)
```

1+ months to be fixed here:
https://github.com/bamos/dcgan-completion.tensorflow/commits/e8b930501dffe01db423b6ca1c65d3ac54f27223/model.py

Code smells ~~in~~for ML

Code Smells for Machine Learning Applications

Haiyin Zhang
haiyin.zhang@ing.com
AI for Fintech Research, ING
Amsterdam, Netherlands

Luís Cruz
L.Cruz@tudelft.nl
Delft University of Technology
Delft, Netherlands

Arie van Deursen
Arie.vanDeursen@tudelft.nl
Delft University of Technology
Delft, Netherlands

ABSTRACT

The popularity of machine learning has wildly expanded in recent years. Machine learning techniques have been heatedly studied in academia and applied in the industry to create business value. However, there is a lack of guidelines for code quality in machine learning applications. In particular, code smells have rarely been studied in this domain. Although machine learning code is usually

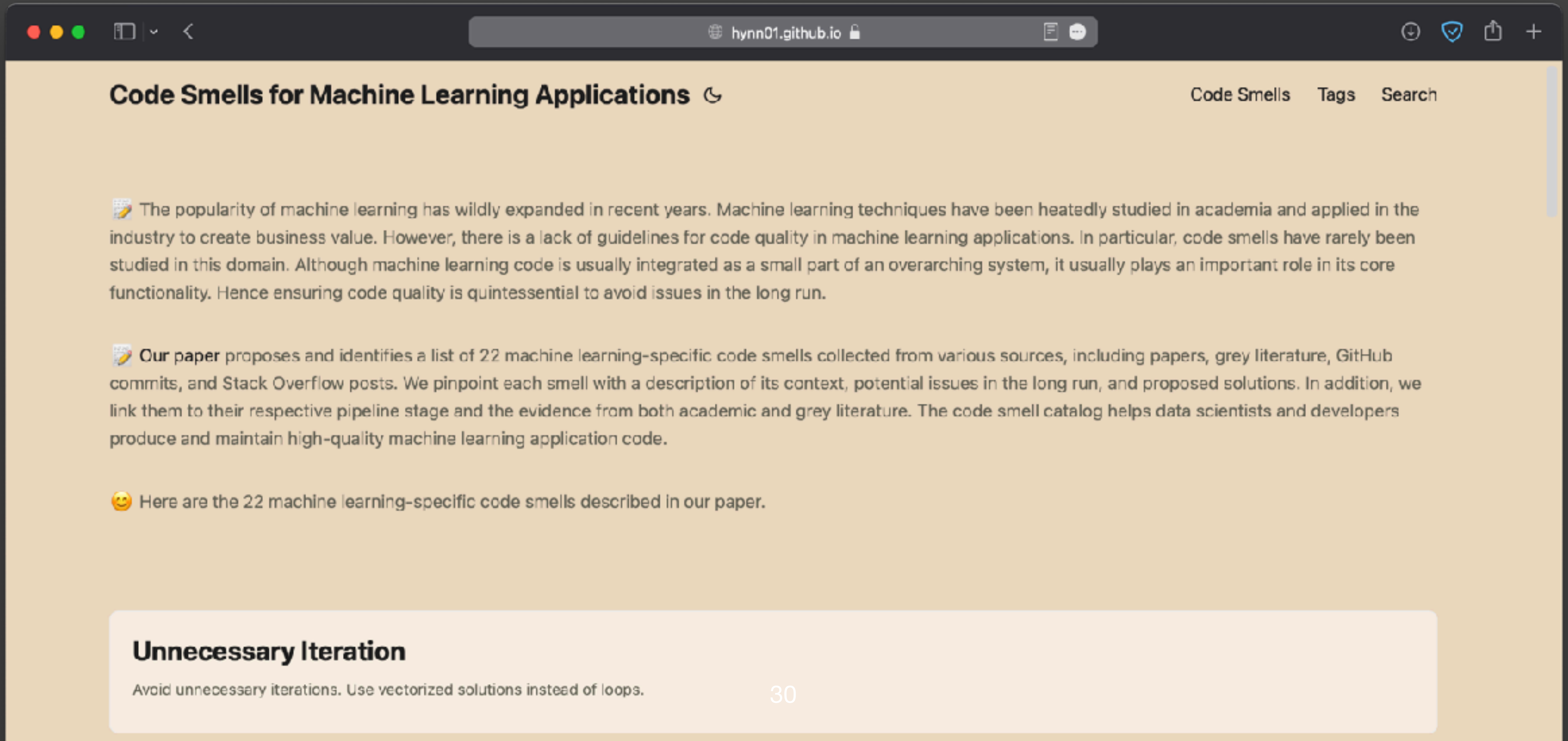
that practitioners are eager to learn more about engineering best practices for their machine learning applications [5].

There has been a lot of interest in various machine learning system artifacts, including models and data. Researchers make efforts to improve machine learning model quality [10] and data quality [7]. However, the quality assurance of machine learning code has not been highlighted [12]. Recent work studied the code quality for

How did we create code smells?

Code Smells for ML

- In the end, we collected 22 ML-specific code smells.
- Available online: https://hynn01.github.io/ml-smells/



A few examples of code smells

hynn01.github.io

Code Smells for Machine Learning Applications

Code Smells Tags Search

Home » Posts » Code Smells

Dataframe Conversion API Misused

Description

Context

In Pandas, `df.to_numpy()` and `df.values()` both can turn a `DataFrame` to a NumPy array.

Problem

As noted in a Stack Overflow post, `df.values()` has an inconsistency problem. With `.values()` it is unclear whether the returned value would be the actual array, some transformation of it, or one of the Pandas custom arrays. However, the `.values()` API has not been not deprecated yet. Although the library developers note it as a warning in the documentation, it does not log a warning or error when compiling the code if we use `.value()` .

Solution

When converting `DataFrame` to NumPy array, it is better to use `df.to_numpy()` than `df.values()` .

Type

API-Specific

Display a menu

Existing Stage

hynn01.github.io

Code Smells for Machine Learning Applications

Code Smells Tags Search

Hyperparameter not Explicitly Set

Description

Context

Hyperparameters are usually set before the actual learning process begins and control the learning process. These parameters directly influence the behavior of the training algorithm and therefore have a significant impact on the model's performance.

Problem

The default parameters of learning algorithm APIs may not be optimal for a given data or problem, and may lead to local optima. In addition, while the default parameters of a machine learning library may be adequate for some time, these default parameters may change in new versions of the library. Furthermore, not setting the hyperparameters explicitly is inconvenient for replicating the model in a different programming language.

Solution

Hyperparameters should be set explicitly and tuned for improving the result's quality and reproducibility.

Type

Generic

Display a menu

Existing Stage

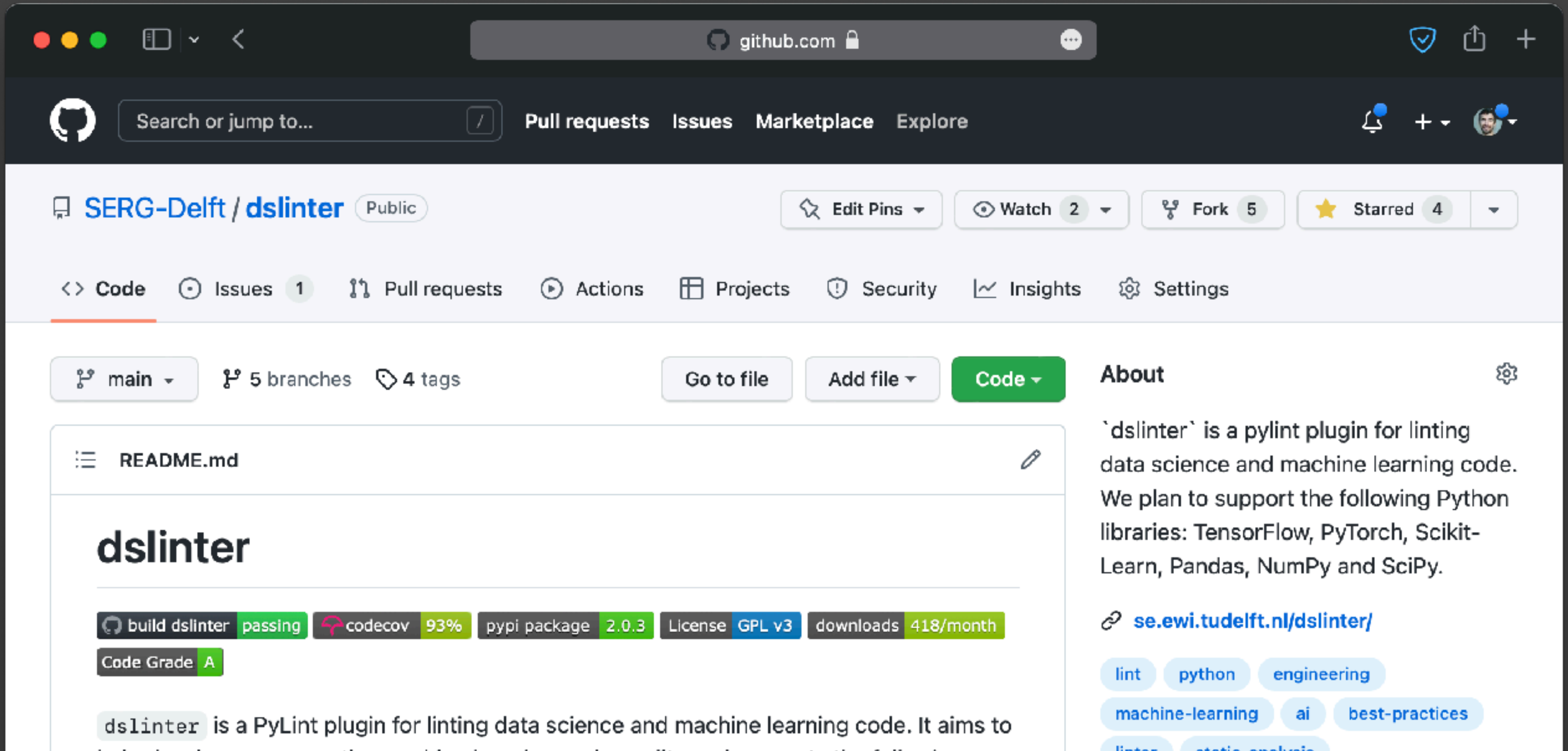
```
### Scikit-Learn
from sklearn.cluster import KMeans

- kmeans = KMeans()
+ kmeans = KMeans(n_clusters=8, random_state=0)
+ # Or, ideally:
+ kmeans = KMeans(n_clusters=8,
+ init='k-means++', n_init=10,
+ max_iter=300, tol=0.0001,
+ precompute_distances='auto',
+ verbose=0, random_state=0,
+ copy_x=True, n_jobs=1,
+ algorithm='auto')

### PyTorch
import torch
import numpy as np
from kmeans_pytorch import kmeans

# data
data_size, dims, num_clusters = 1000, 2, 3
x = np.random.randn(data_size, dims) / 6
x = torch.from_numpy(x)

# kmeans
- cluster_ids_x, cluster_centers = kmeans(X=x, num_clusters=num_clusters)
+ cluster_ids_x, cluster_centers = kmeans(
+     X=x, num_clusters=num_clusters, distance='euclidean', device=torch.device('cpu')
+ )
```

A few notes about smells

- Code smells can indicate issues, but not all of them have the same severity level.
- By definition, smells are not always a problem. They are just a warning that developers need to reflect and take action if needed.
- Some code smells can be automated.

dslinter – Code Smells for ML

https://github.com/SERG-Delft/dslinter



Give us a ⭐️
Contributions are welcome!

```
pip install dslinter
pylint —load-plugins=dslinter mysource.py
```

Structuring an ML project

- ML projects are very experimental.
- What’s the overhead of setting up dvc, removing all code smells, etc. for code that does not lead to anything?
- An ML project needs to allow both exploratory and production code to co-exist in the same repo.
  - (Still an open question)
  - Cookiecutter may help.

Cookiecutter

- Proposes a standard structure for ML projects.
- It is only a suggestion. Users can create their own boilerplate.
- Organisations should strive to create a standard project structure that fits their infrastructure/values.

```
├── LICENSE
├── Makefile
├── README.md
├── data
│   ├── external
│   ├── interim
│   ├── processed
│   └── raw
├── docs
├── models
├── notebooks
├── references
├── reports
│   └── figures
├── requirements.txt
├── setup.py
├── src
│   ├── __init__.py
│   ├── data
│   │   └── make_dataset.py
│   ├── features
│   │   └── build_features.py
│   ├── models
│   │   ├── predict_model.py
│   │   └── train_model.py
│   └── visualization
│       └── visualize.py
└── tox.ini
```

How today's lecture should impact your final project

- You should extract different stages to different python files
- You should have a structure that enables experimentation and production code
- Your pipeline should be managed by DVC (next class)
- Pylint + DSlinter should be properly configured and part of your continuous integration pipeline

Next lecture

- DVC tutorial