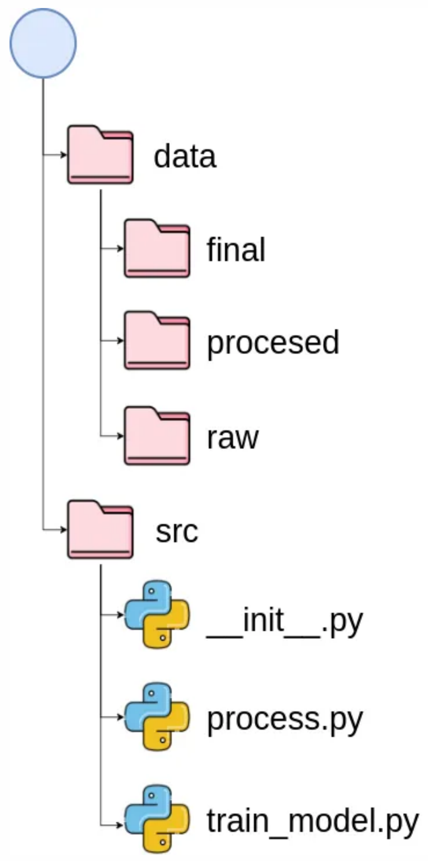
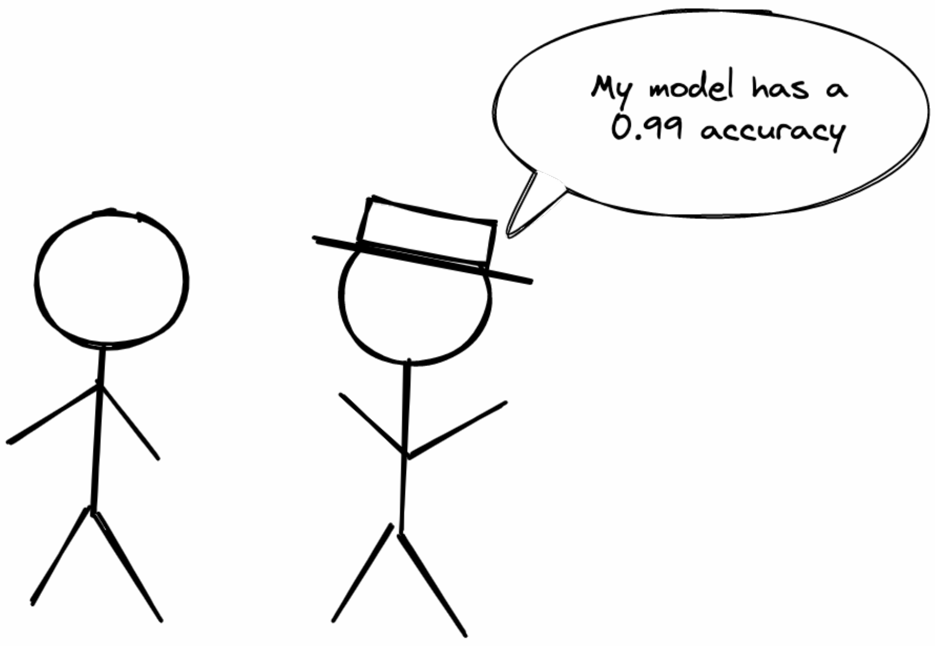
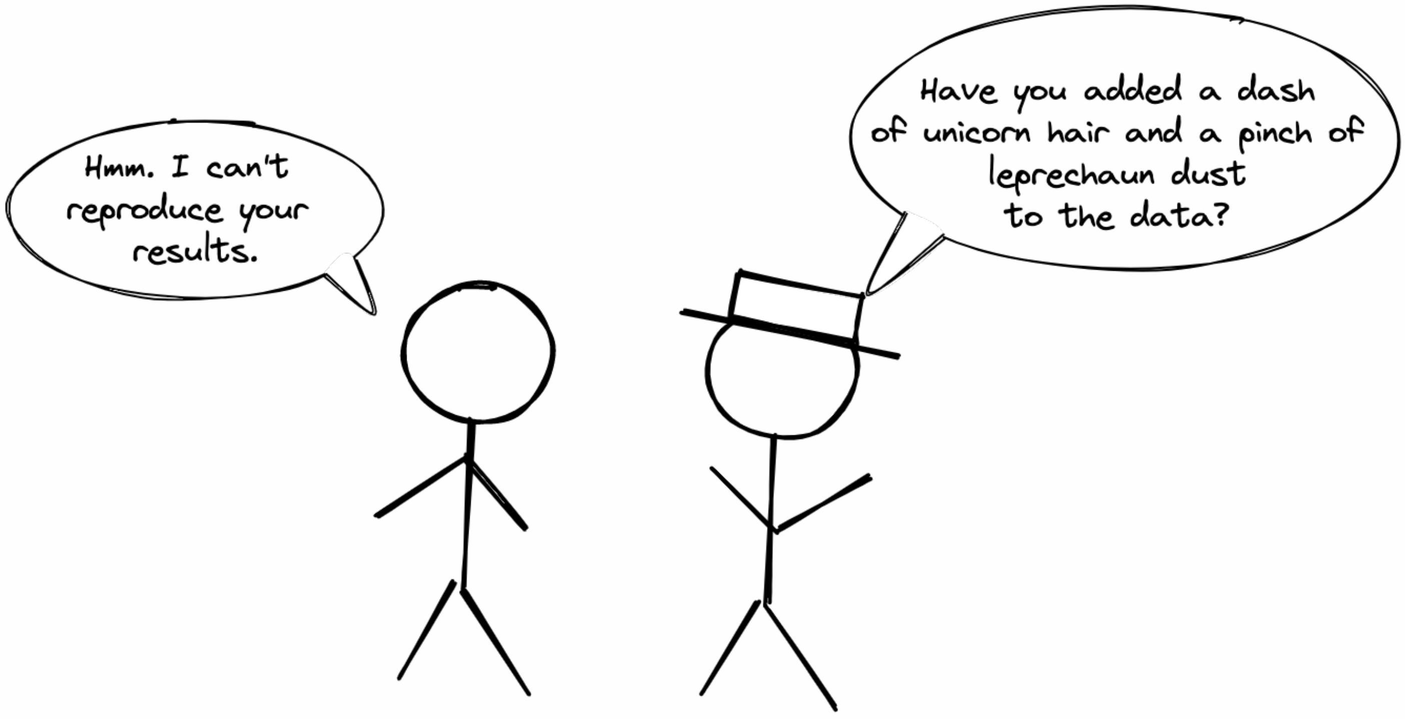
How to Structure an ML Project for Reproducibility and Maintainability

Motivation

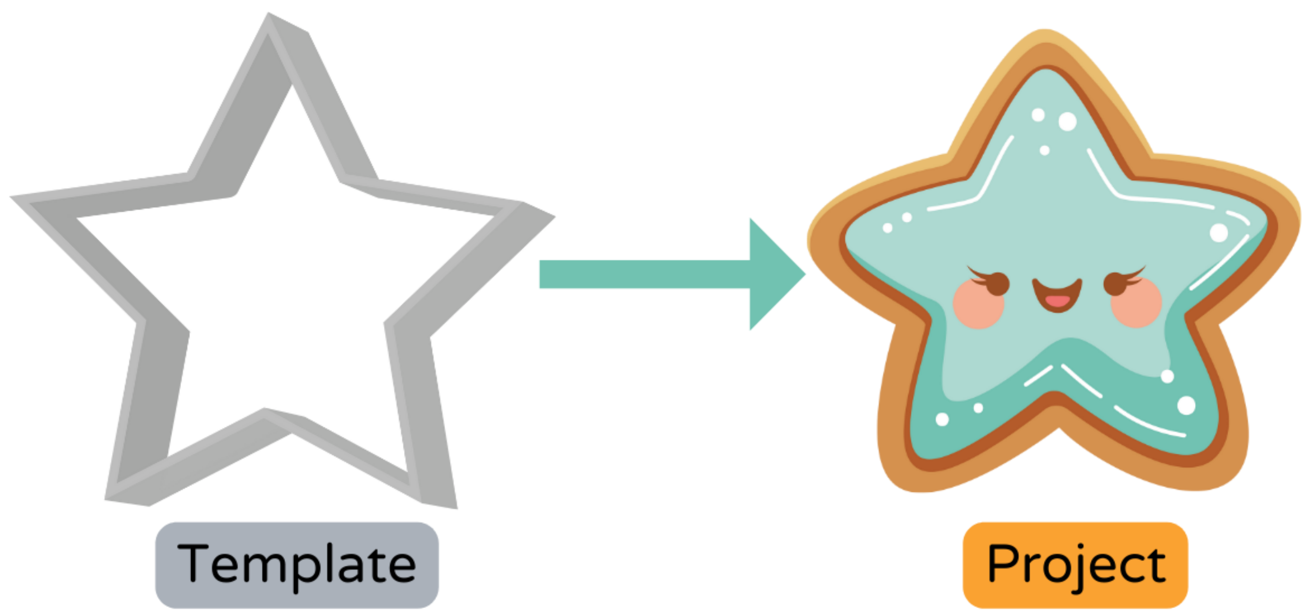
Getting started is often the most challenging part when building ML projects. How should you structure your repository? Which standards should you follow? Will your teammates be able to reproduce the results of your experimentations?




My model has a 0.99 accuracy


Hmm. I can't reproduce your results.
Have you added a dash of unicorn hair and a pinch of leprechaun dust to the data?

Instead of trying to find an ideal repository structure, wouldn't it be nice to have a template to get started?



That is why I created data-science-template, consolidating best practices I've learned over the years about structuring data science projects.

This template allows you to:

✅ Create a readable structure for your project

✅ Efficiently manage dependencies in your project

✅ Create short and readable commands for repeatable tasks

✅ Rerun only modified components of a pipeline

✅ Observe and automate your code

✅ Enforce type hints at runtime

✅ Check issues in your code before committing

✅ Automatically document your code

✅ Automatically run tests when committing your code

Tools Used in This Template

This template is lightweight and uses only tools that can generalize to various use cases. Those tools are:

- Poetry: manage Python dependencies
- Prefect: orchestrate and observe your data pipeline
- Pydantic: validate data using Python type annotations
- pre-commit plugins: ensure your code is well-formatted, tested, and documented, following best practices
- Makefile: automate repeatable tasks using short commands
- GitHub Actions: automate your CI/CD pipeline
- pdoc: automatically create API documentation for your project

Usage

To download the template, start by installing Cookiecutter:

```
pip install cookiecutter
```

Create a project based on the template:

```
cookiecutter https://github.com/khuyentran1401/data-
science-template
```

Try out the project by following these instructions.

In the following few sections, we will detail some valuable features of this template.

Create a Readable Structure

The structure of the project created from the template is standardized and easy to understand.

Here is the summary of the roles of these files:

```
.
├── data
│   ├── final                     # data after training the
model
│   ├── processed                 # data after processing
│   ├── raw                       # raw data
├── docs                          # documentation for your
project
├── .flake8                       # configuration for code
formatter
├── .gitignore                    # ignore files that cannot
commit to Git
├── Makefile                      # store commands to set up
the environment
├── models                        # store models
├── notebooks                     # store notebooks
├── .pre-commit-config.yaml       # configurations for pre-
commit
├── pyproject.toml                # dependencies for poetry
├── README.md                     # describe your project
├── src                           # store source code
│   ├── __init__.py               # make src a Python module
│   ├── config.py                   # store configs
│   ├── process.py                  # process data before
training model
│   ├── run_notebook.py             # run notebook
│   └── train_model.py              # train model
└── tests                           # store tests
    ├── __init__.py                 # make tests a Python
module
    ├── test_process.py             # test functions for
process.py
    └── test_train_model.py         # test functions for
train_model.py
```

Efficiently Manage Dependencies

Poetry is a Python dependency management tool and is an alternative to pip.

With Poetry, you can:

- Separate the main dependencies and the sub-dependencies into two separate files (instead of storing all dependencies in requirements.txt)
- Remove all unused sub-dependencies when removing a library
- Avoid installing new packages that conflict with the existing packages
- Package your project in several lines of code

and more.

Find the instruction on how to install Poetry here.

Create Short Commands for Repeatable Tasks

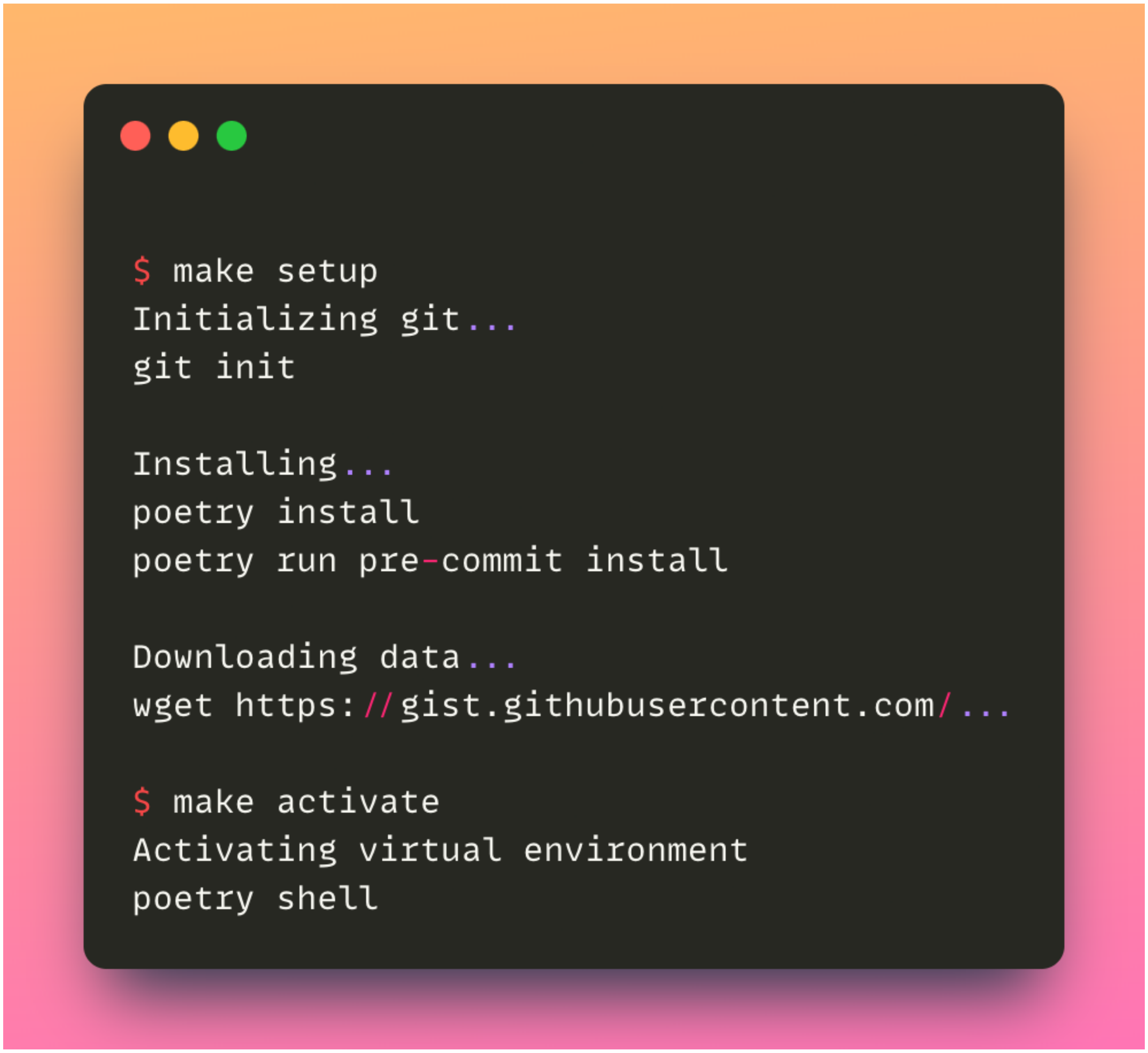
Makefile allows you to create short and readable commands for tasks. You can use Makefile to automate tasks such as setting up the environment:

```
initialize_git:
 @echo "Initializing git..."
 git init

install:
 @echo "Installing..."
 poetry install
 poetry run pre-commit install

activate:
 @echo "Activating virtual environment"
 poetry shell

download_data:
 @echo "Downloading data..."
 wget
https://gist.githubusercontent.com/khuyentran1401/a1abde0a
7d27d31c7dd08f34a2c29d8f/raw/da2b0f2c9743e102b9dfa6cd75e94
708d01640c9/Iris.csv -O data/raw/iris.csv

setup: initialize_git install download_data
```

Now, whenever others want to set up the environment for your projects, they just need to run the following:

```
make setup
make activate
```

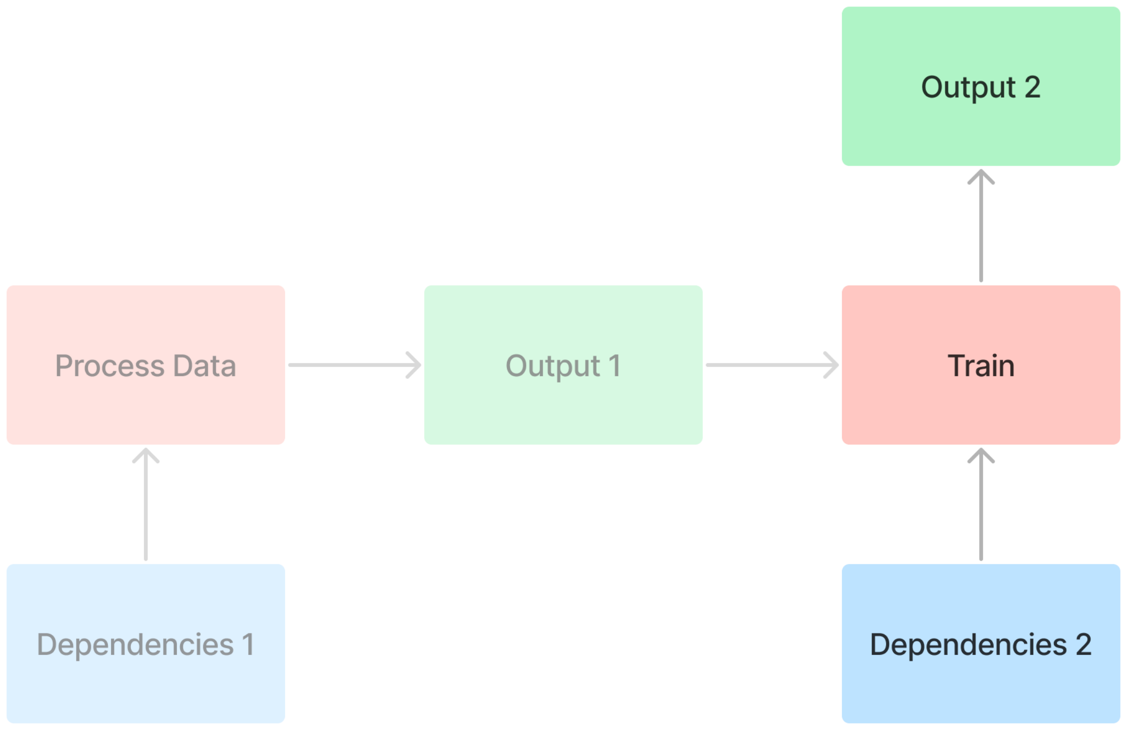
And a series of commands will be run! View the full Makefile here.



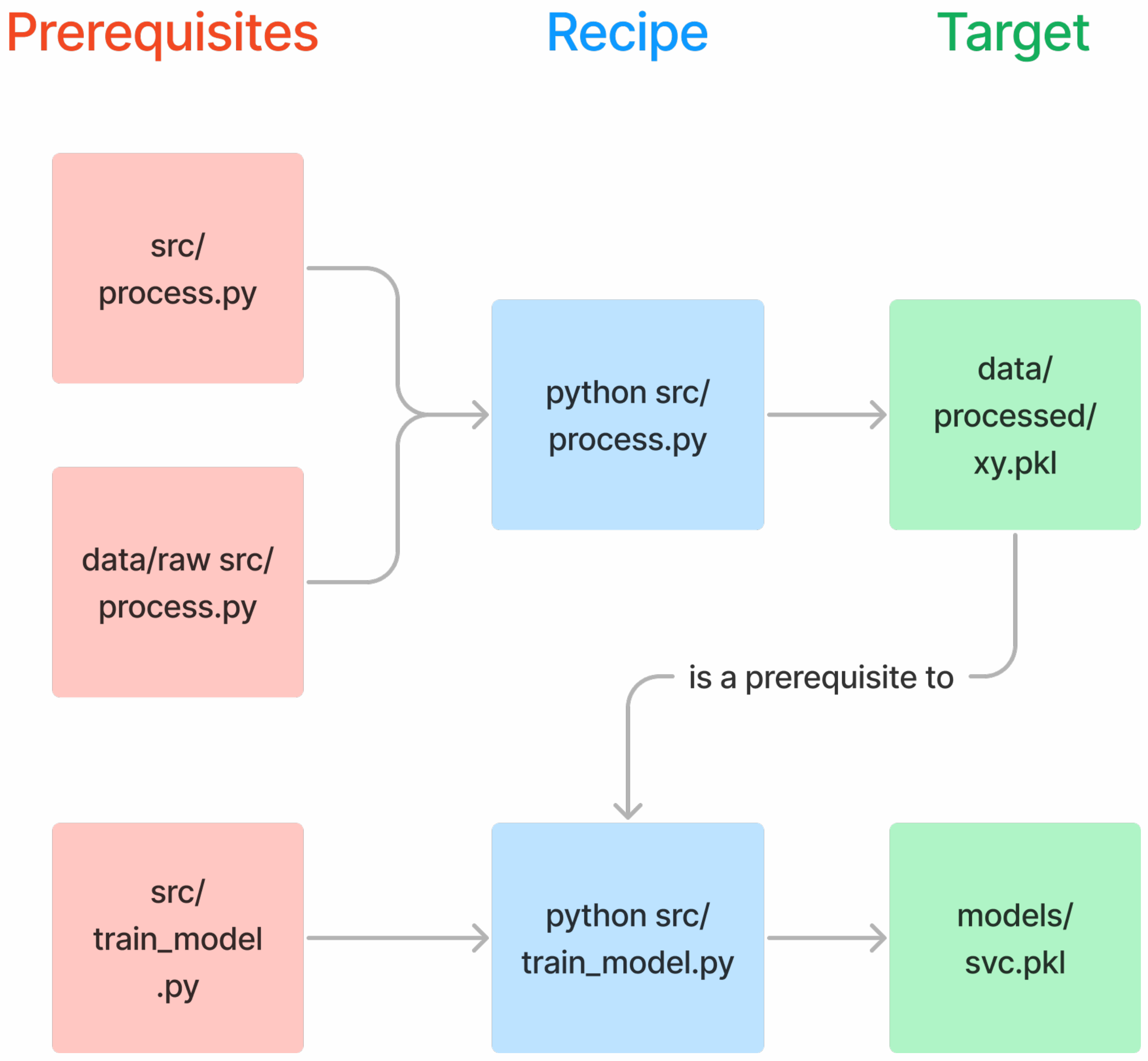
```
$ make setup
Initializing git...
git init

Installing...
poetry install
poetry run pre-commit install

Downloading data...
wget https://gist.githubusercontent.com/...

$ make activate
Activating virtual environment
poetry shell
```

Rerun Only Modified Components of a Pipeline

Make is also useful when you want to run a task whenever its dependencies are modified.



As an example, let's capture the connection between files in the following diagram through a Makefile:



```
data/processed/xy.pkl: data/raw src/process.py
 @echo "Processing data..."
 python src/process.py

models/svc.pkl: data/processed/xy.pkl src/train_model.py
 @echo "Training model..."
 python src/train_model.py

pipeline: data/processed/xy.pkl models/svc.pk
```

To create the file models/svc.pkl , you can run:

```
make models/svc.pkl
```

Since data/processed/xy.pkl and src/train_model.py are the prerequisites of the models/svc.pkl target, make runs the recipes to create both data/processed/xy.pkl and models/svc.pkl .

```
Processing data...
python src/process.py

Training model...
python src/train_model.py
```

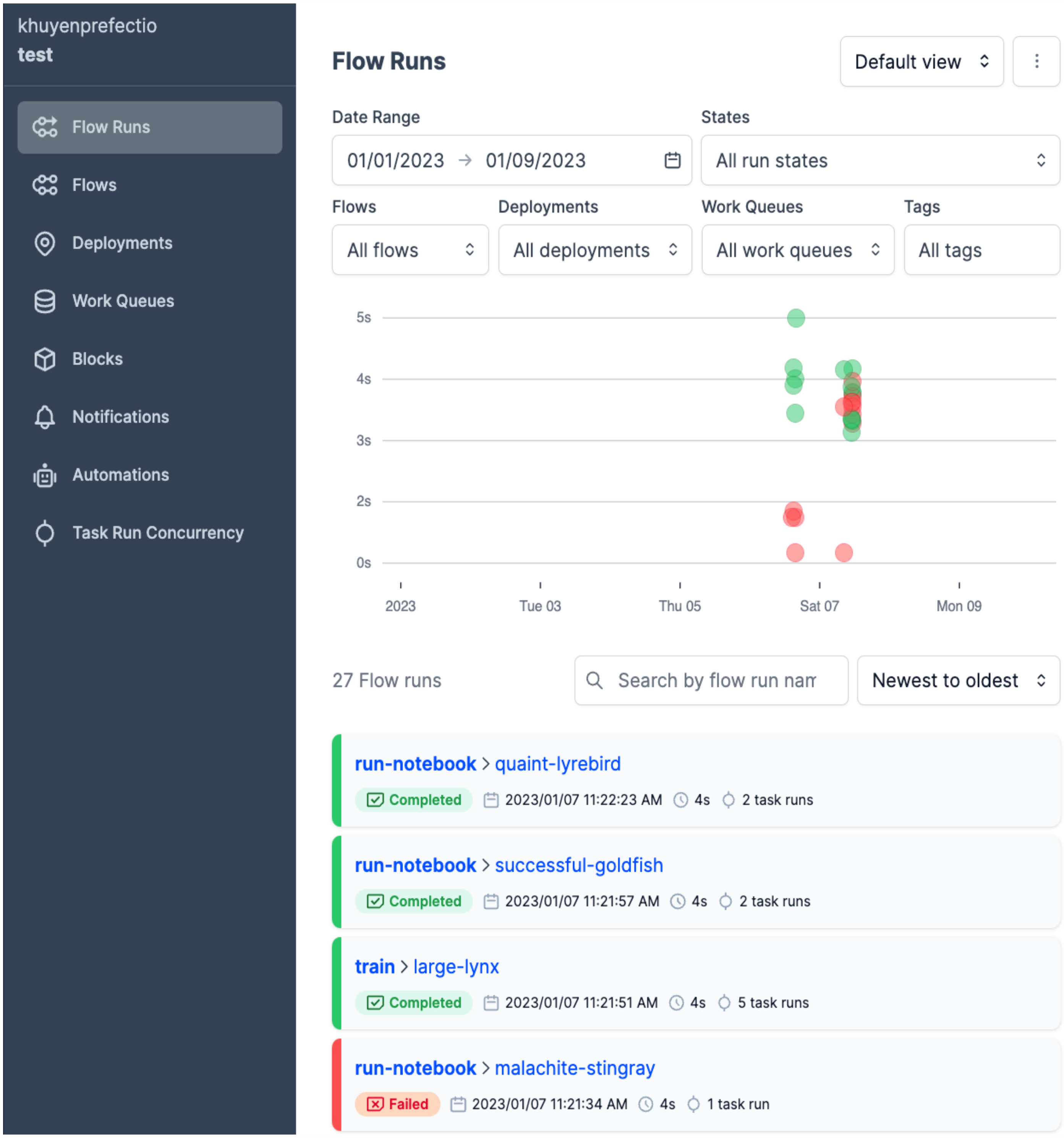
If there are no changes in the prerequisite of models/svc.pkl, make will skip updating models/svc.pkl .

```
$ make models/svc.pkl
make: `models/svc.pkl' is up to date.
```

Observe and Automate Your Code

This template leverages Prefect to:

- Observe all your runs from the Prefect UI.



Flow Runs

27 Flow runs

run-notebook > quaint-lyrebird
Completed 2023/01/07 11:22:23 AM 4s 2 task runs

run-notebook > successful-goldfish
Completed 2023/01/07 11:21:57 AM 4s 2 task runs

train > large-lynx
Completed 2023/01/07 11:21:51 AM 4s 5 task runs

run-notebook > malachite-stingray
Failed 2023/01/07 11:21:34 AM 4s 1 task run

Among others, Prefect can help you:

- Retry when your code fails
- Schedule your code run
- Send notifications when your flow fails

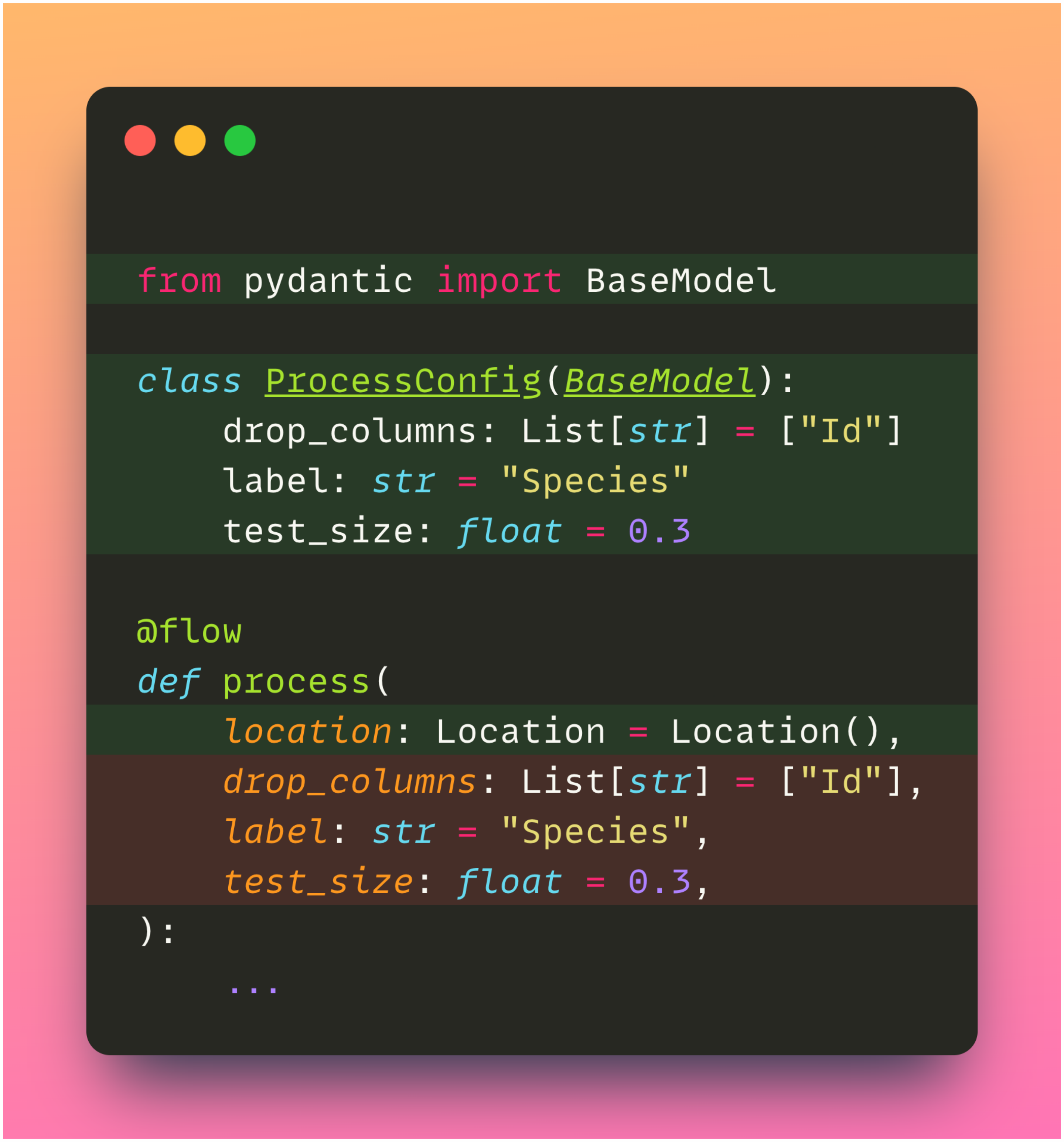
You can access these features by simply turning your function into a Prefect flow.

```
from prefect import flow

@flow
def process(
    location: Location = Location(),
    config: ProcessConfig = ProcessConfig(),
):
    ...
```

Enforce Type Hints At Runtime

Pydantic is a Python library for data validation by leveraging type annotations.

Pydantic models enforce data types on flow parameters and validate their values when a flow run is executed.



```python
from pydantic import BaseModel

class ProcessConfig(BaseModel):
    drop_columns: List[str] = ["Id"]
    label: str = "Species"
    test_size: float = 0.3

@flow
def process(
    location: Location = Location(),
    drop_columns: List[str] = ["Id"],
    label: str = "Species",
    test_size: float = 0.3,
):
    ...
```

If the value of a field doesn’t match the type annotation, you will get an error at runtime:

```
process(config=ProcessConfig(test_size='a'))

pydantic.error_wrappers.ValidationError: 1 validation
error for ProcessConfig
test_size
  value is not a valid float (type=type_error.float)
```

All Pydantic models are in the src/config.py file.

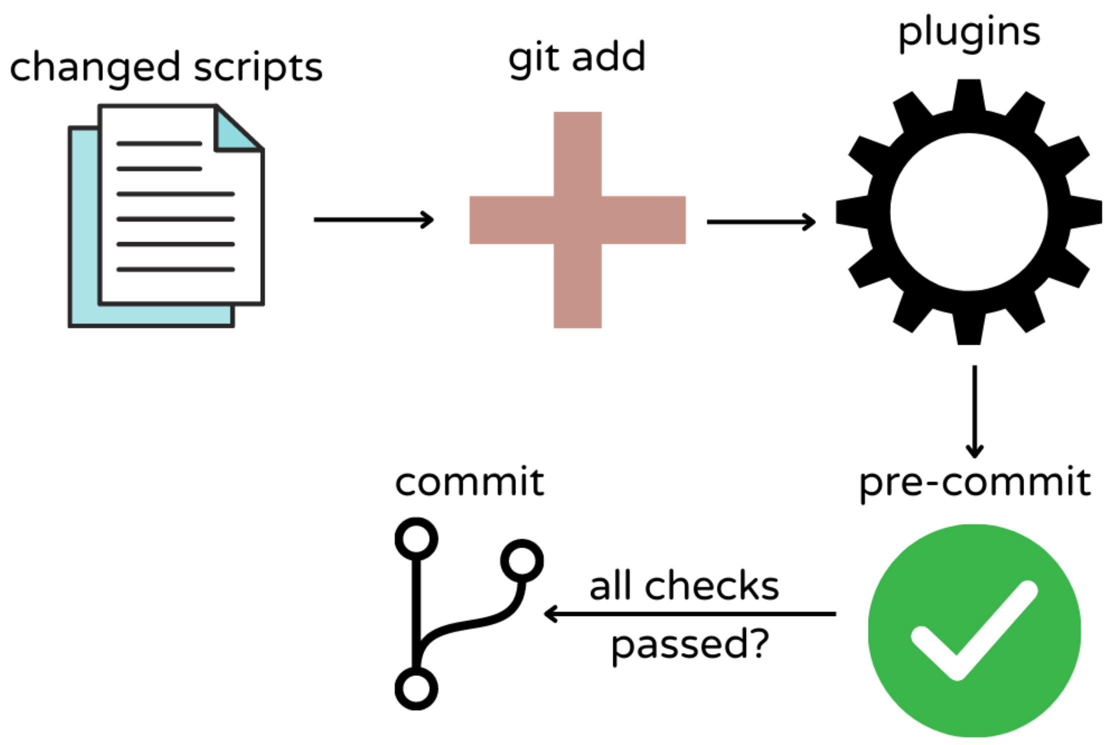
Detect Issues in Your Code Before Committing

Before committing your Python code to Git, you need to make sure your code:

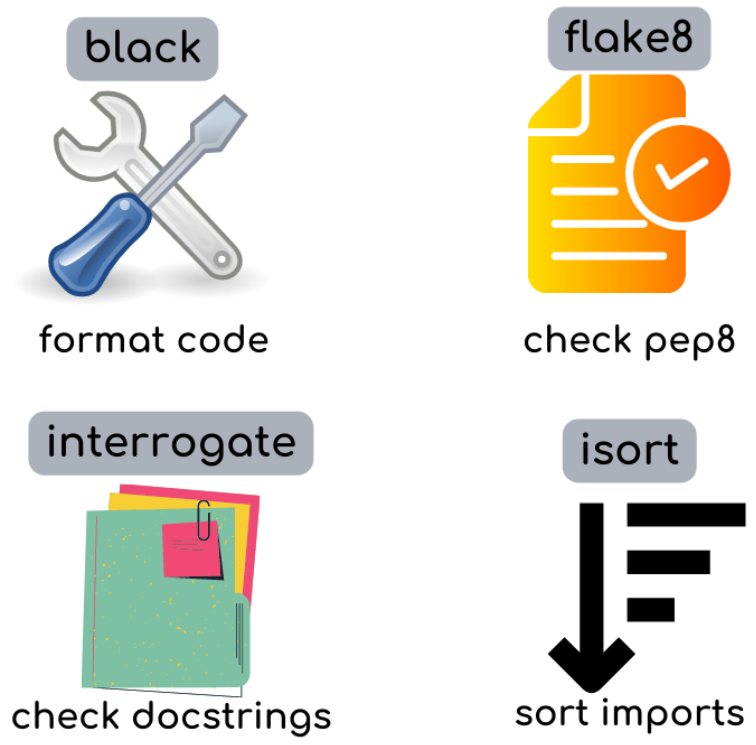
- passes unit tests
- is organized
- conforms to best practices and style guides
- is documented

However, manually checking these criteria before committing your code can be tedious. pre-commit is a framework that allows you to identify issues in your code before committing it.

You can add different plugins to your pre-commit pipeline. Once your files are committed, they will be validated against these plugins. Unless all checks pass, no code will be committed.



You can find all plugins used in this template in this .pre-commit-config.yaml file.



Automatically Document Your Code

Data scientists often collaborate with other team members on a project. Thus, it is essential to create good documentation for the project.

To create API documentation based on docstrings of your Python files and objects, run:

```
make docs_view
```

Output:

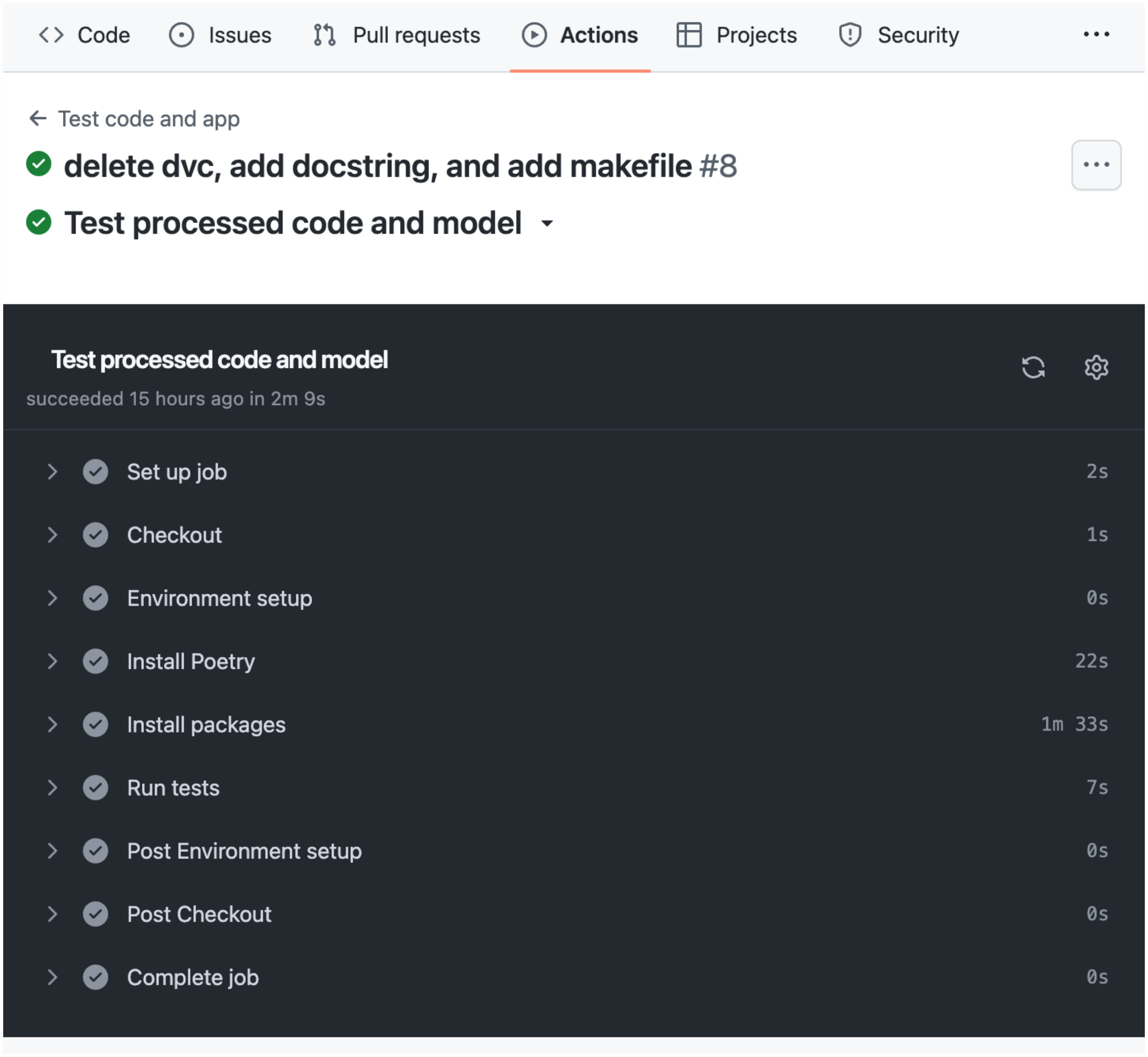
```
Save the output to docs...
pdoc src --http localhost:8080
Starting pdoc server on localhost:8080
pdoc server ready at http://localhost:8080
```

Now you can view the documentation on http://localhost:8080.

Automatically Run Tests

GitHub Actions allows you to automate your CI/CD pipelines, making it faster to build, test, and deploy your code.

When creating a pull request on GitHub, the tests in your tests folder will automatically run.



← Test code and app

✓ delete dvc, add docstring, and add makefile #8

✓ Test processed code and model

Test processed code and model
succeeded 15 hours ago in 2m 9s

```
✓ Set up job                 2s
✓ Checkout                   1s
✓ Environment setup          0s
✓ Install Poetry             22s
✓ Install packages           1m 33s
✓ Run tests                  7s
✓ Post Environment setup     0s
✓ Post Checkout              0s
✓ Complete job               0s
```

> View the code for this workflow.

Conclusion

Congratulations! You have just learned how to use a template to create a reusable and maintainable ML project. This template is meant to be flexible. Feel free to adjust the project based on your applications.