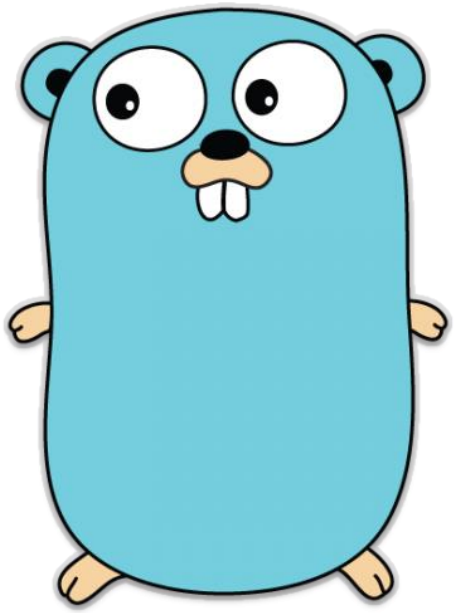
Маленькие инструменты для MLOps

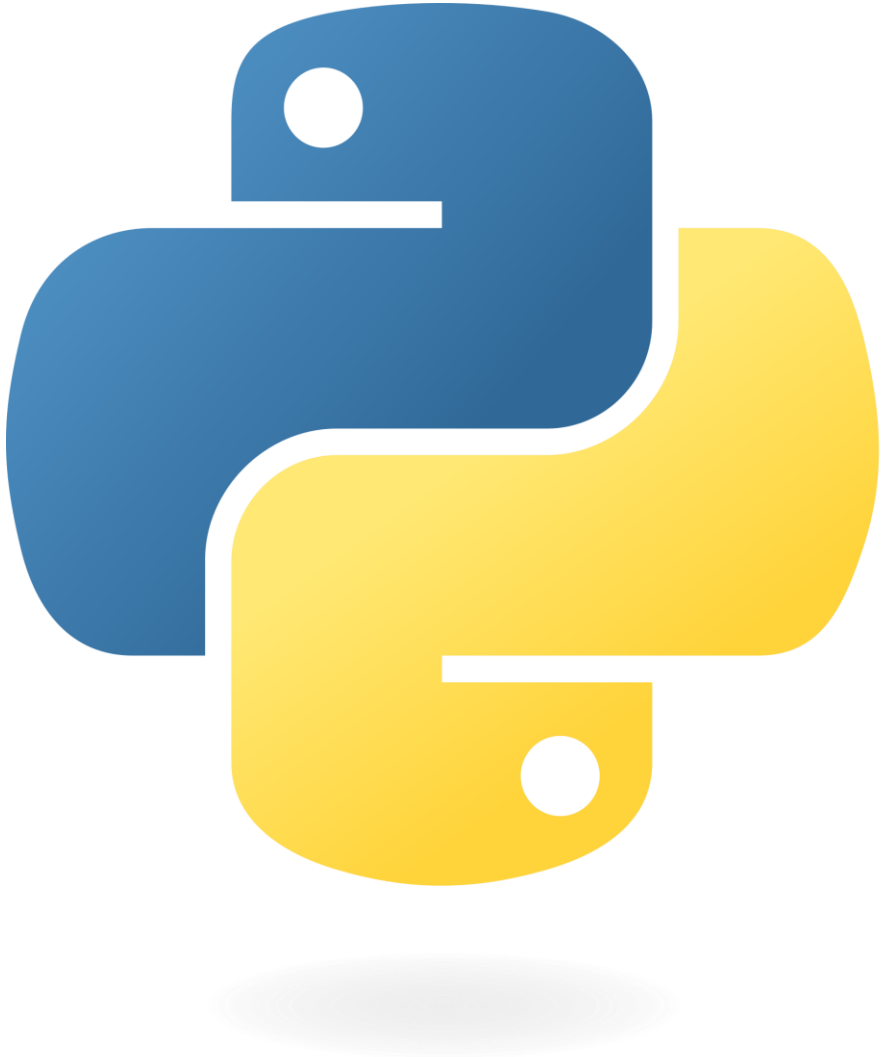
Лекция №3

Продакшн здорового человека



Продакшн в МЛ

- Динамическая типизация
- Интерпретируемый



Продакшн в МЛ

- Динамическая типизация
- Интерпретируемый
- Большое количество готовых пакетов
- Стандарт де факто в ML

Почему тяжело сразу делать продакшн код на Python?

[142]:
```
routes = routes[routes['gps_time'] > '2024-09-01']
```

[159]:
```
import geopy.distance

coords_1 = (52.2296756, 21.0122287)
coords_2 = (52.406374, 16.9251681)

print(geopy.distance.geodesic(prev, (r['lat'], r['lon'])).km * 1000 / )
```

```
0.0
```

[157]:
```
 [prev,
                      [r['lat'], r['lon']]
                     ]
```

[157]:
```
[[55.92463, 38.04195], [55.92463, 38.04195]]
```

[155]:
```
import geopy
```

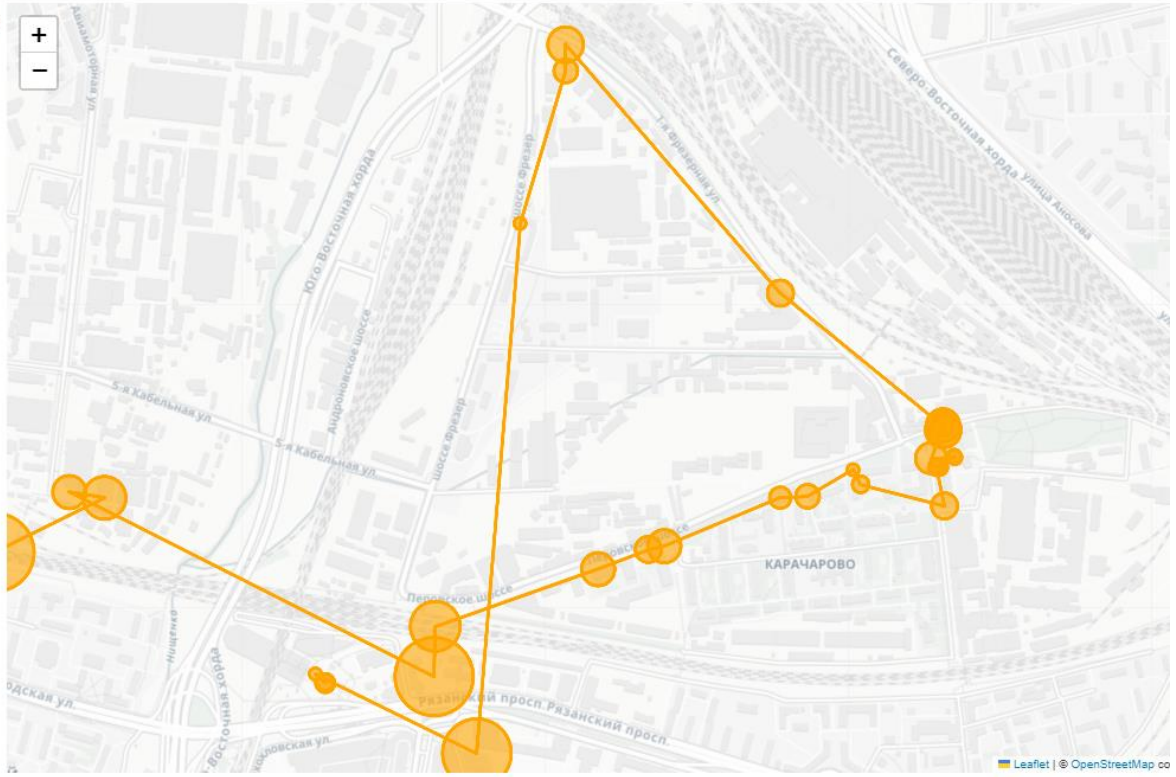
```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[155], line 1
----> 1 import geopy

ModuleNotFoundError: No module named 'geopy'
```

Jupyter – достоинства

- Удобно строить графики
- Можно делать в виде отчета
- Удобно изучать новые пакеты
- Просто устанавливать
- Легко развернуть на удаленном сервере для нескольких разработчиков (JupyterHub)

```
                ],
                color=color,
                tooltip=str(round(speed, 3)),
            ).add_to(map_)
        prev = [r['lat'], r['lon']]
        prev_time = r['gps_time']
        geo_j.add_to(map_)
map_
```

[170]:

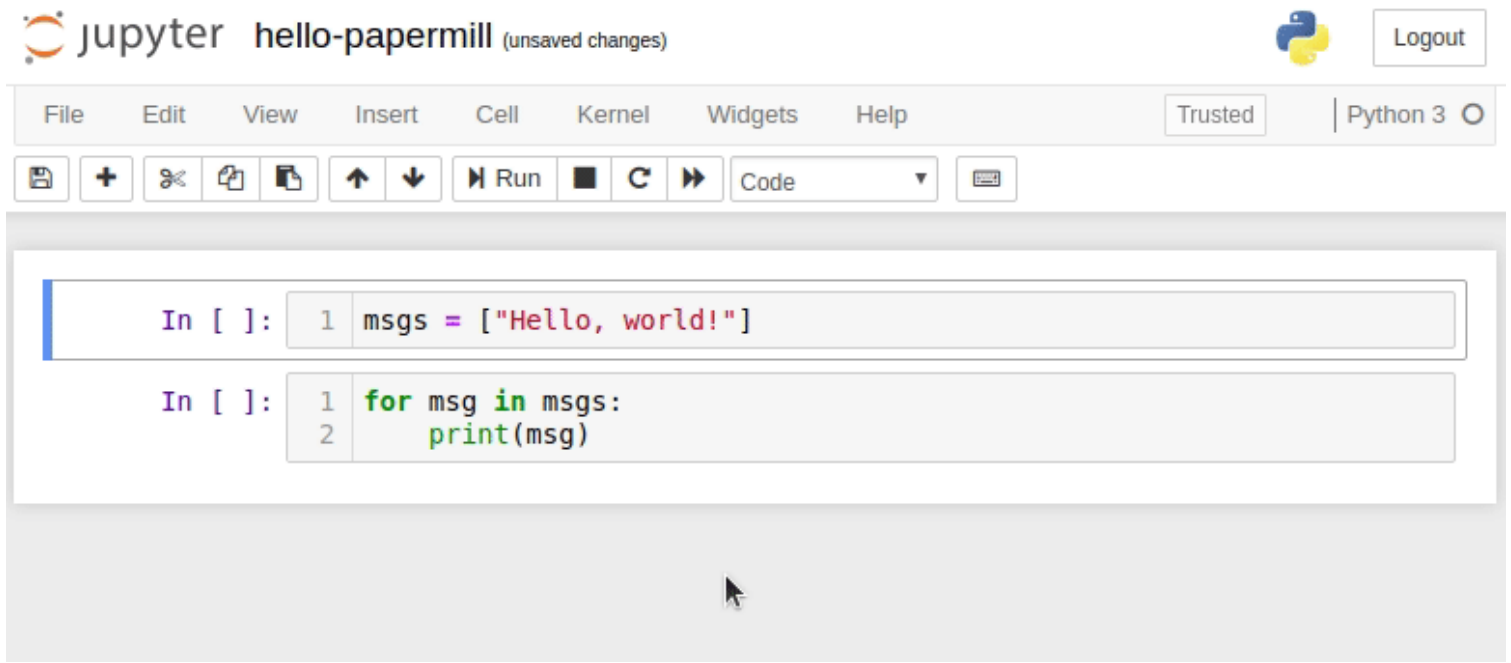


Jupyter – недостатки

- Проблемы с версионированием в гите
- Слабая IDE для .py файлов
- Сложно тестировать код
- Позволяет выполнять ячейки в произвольном порядке

Продакшн на Jupyter тоже возможен

```
import papermill as pm


pm.execute_notebook(
   'path/to/input.ipynb',
   'path/to/output.ipynb',
   parameters =
dict(alpha=0.6, ratio=0.1)
)
```



https://github.com/nteract/papermill

NETFLIX

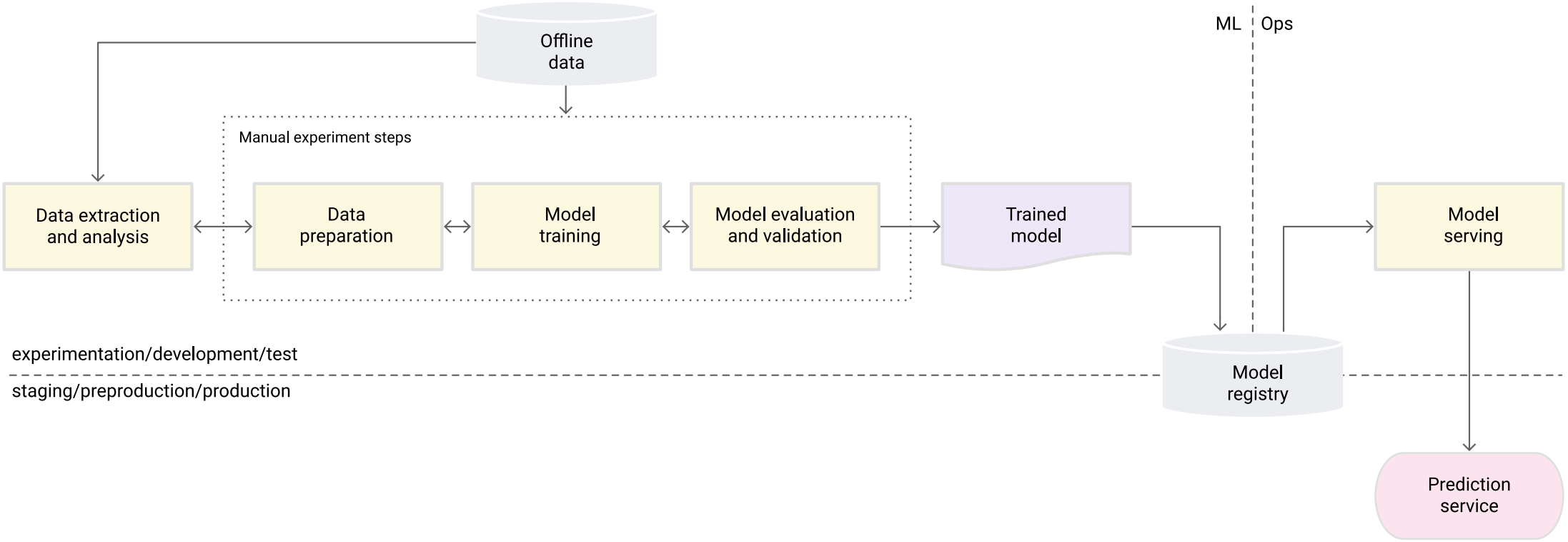
В чем писать код, если не в Jupyter?

Характеристики прод кода

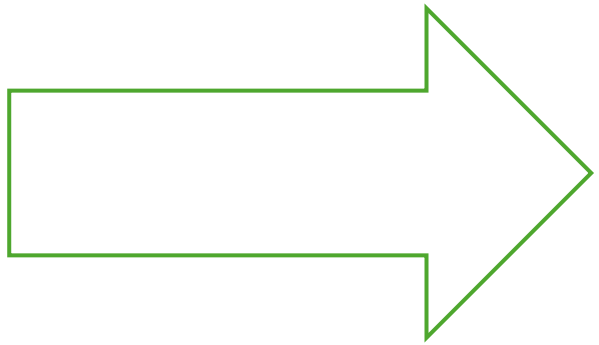
- Читаемый
- Эффективный
- Модульный
- Содержащий тесты и прописанные exceptions
- Задокументированный

Проект:
- Версионируемый
- Удобная структура
- Явные зависимости

MLOps Level 0

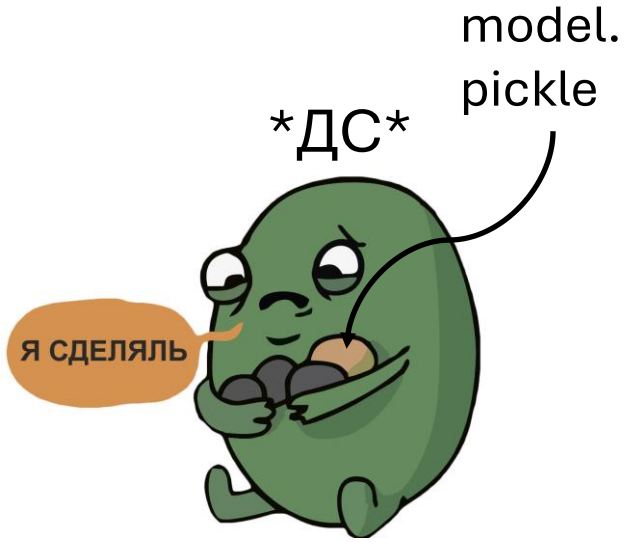


Типичная картина



Код построения модели:

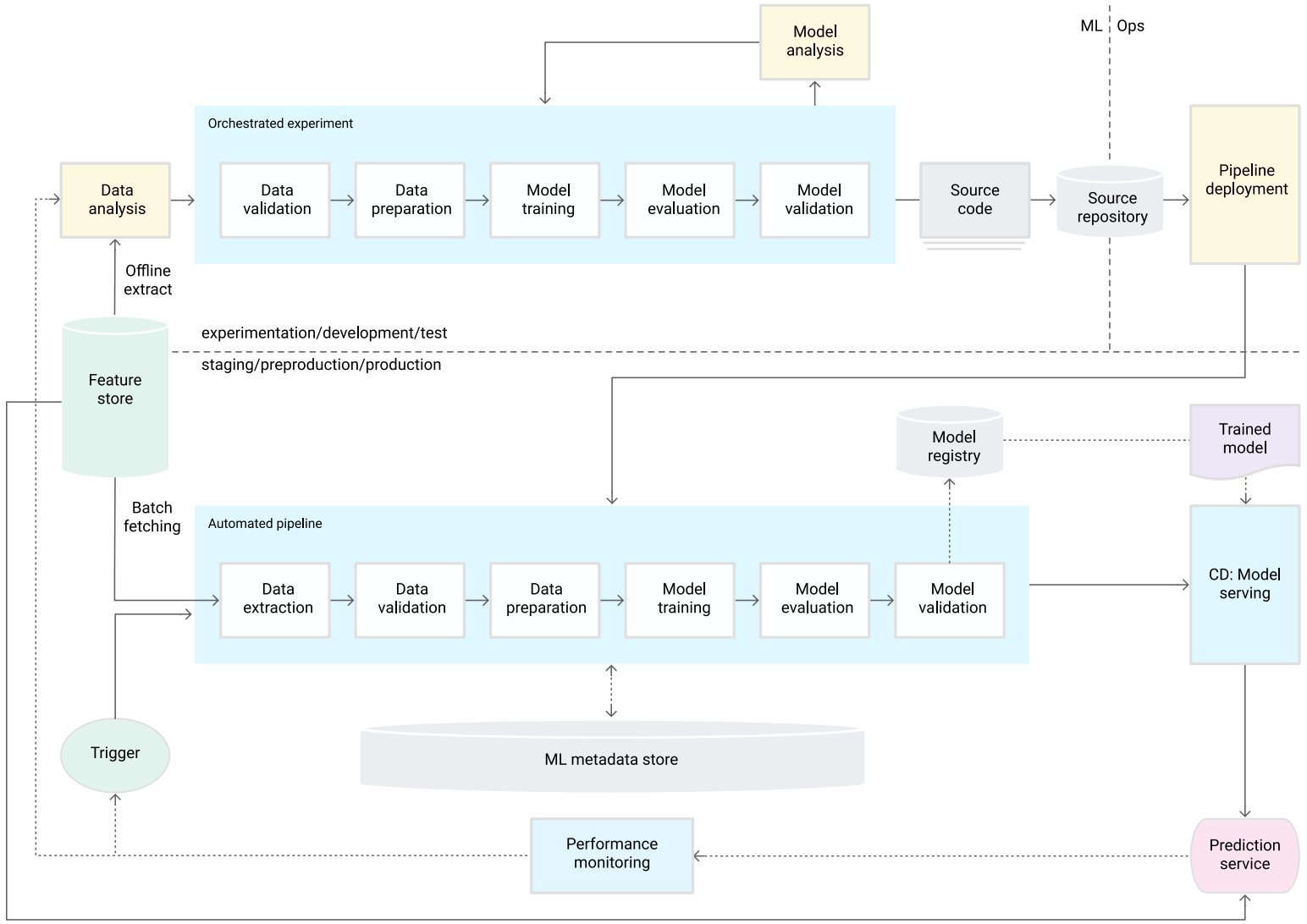
- не воспроизводим
- не обобщаемый
- не универсальный



Код использования модели:

- хороший
- воспроизводимый

MLOps Level 1



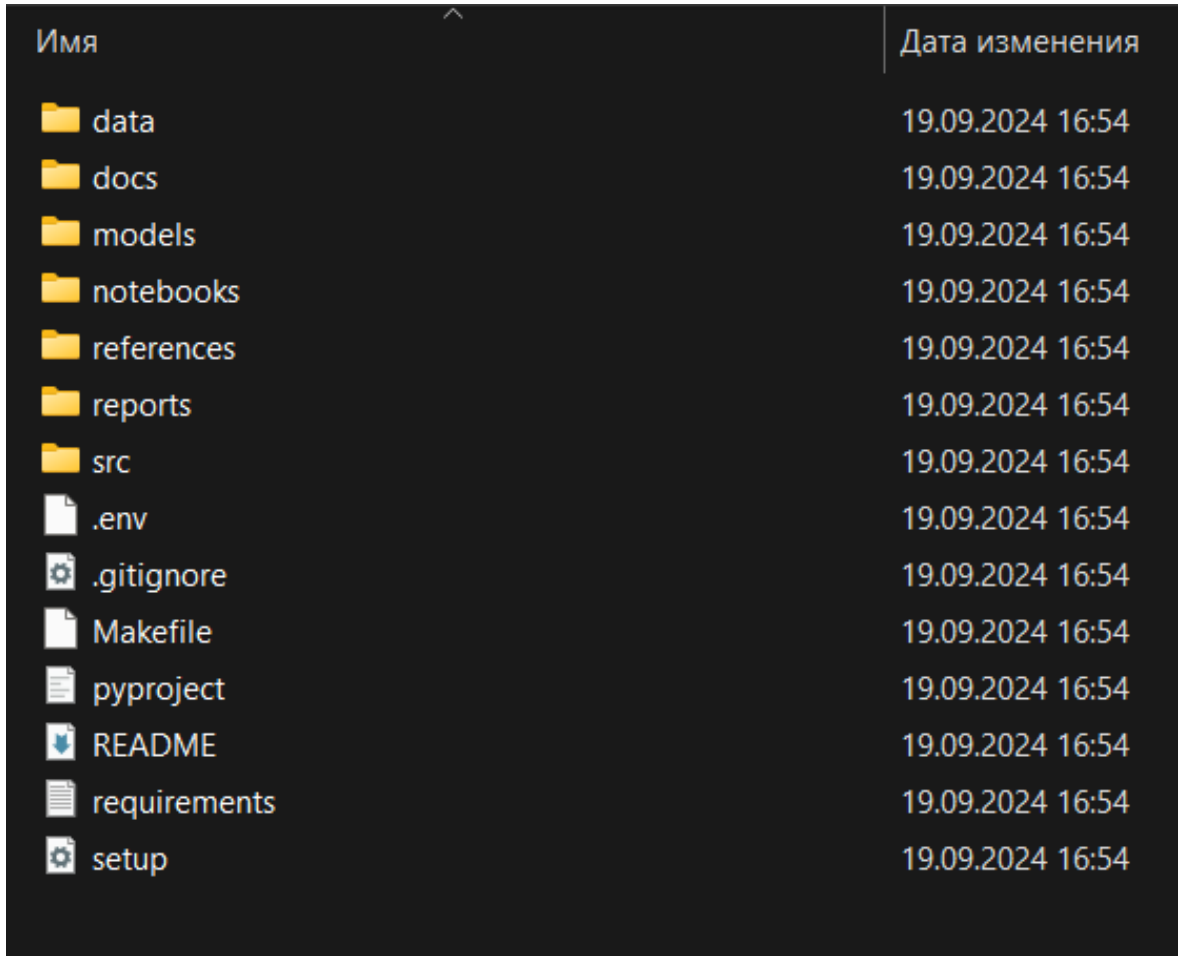
Инструменты

Cookiecutter

Инструмент для шаблонизации проектов

```
pip install cookiecutter-data-science

ccds
```

https://github.com/drivendataorg/cookiecutter-data-science



Cookiecutter

```
├── LICENSE            <- Open-source license if one is chosen
├── Makefile           <- Makefile with convenience commands like `make data` or `make train`
├── README.md          <- The top-level README for developers using this project.
├── data
│   ├── external       <- Data from third party sources.
│   ├── interim        <- Intermediate data that has been transformed.
│   ├── processed      <- The final, canonical data sets for modeling.
│   └── raw            <- The original, immutable data dump.
│
├── docs               <- A default mkdocs project; see www.mkdocs.org for details
│
├── models             <- Trained and serialized models, model predictions, or model summaries
│
├── notebooks          <- Jupyter notebooks. Naming convention is a number (for ordering),
│                         the creator's initials, and a short `-` delimited description, e.g.
│                         `1.0-jqp-initial-data-exploration`.
│
├── pyproject.toml     <- Project configuration file with package metadata for
│                         {{ cookiecutter.module_name }} and configuration for tools like black
│
├── references         <- Data dictionaries, manuals, and all other explanatory materials.
│
├── reports            <- Generated analysis as HTML, PDF, LaTeX, etc.
│   └── figures        <- Generated graphics and figures to be used in reporting
│
├── requirements.txt   <- The requirements file for reproducing the analysis environment, e.g.
│                         generated with `pip freeze > requirements.txt`
│
├── setup.cfg          <- Configuration file for flake8
│
└── {{ cookiecutter.module_name }}   <- Source code for use in this project.
    │
    ├── __init__.py             <- Makes {{ cookiecutter.module_name }} a Python module
    │
    ├── config.py               <- Store useful variables and configuration
    │
    ├── dataset.py              <- Scripts to download or generate data
    │
    ├── features.py             <- Code to create features for modeling
    │
    ├── modeling
    │   ├── __init__.py
    │   ├── predict.py          <- Code to run model inference with trained models
    │   └── train.py            <- Code to train models
    │
    └── plots.py                <- Code to create visualizations
```

Виртуальные окружения

Варианты создания виртуальных окружений:

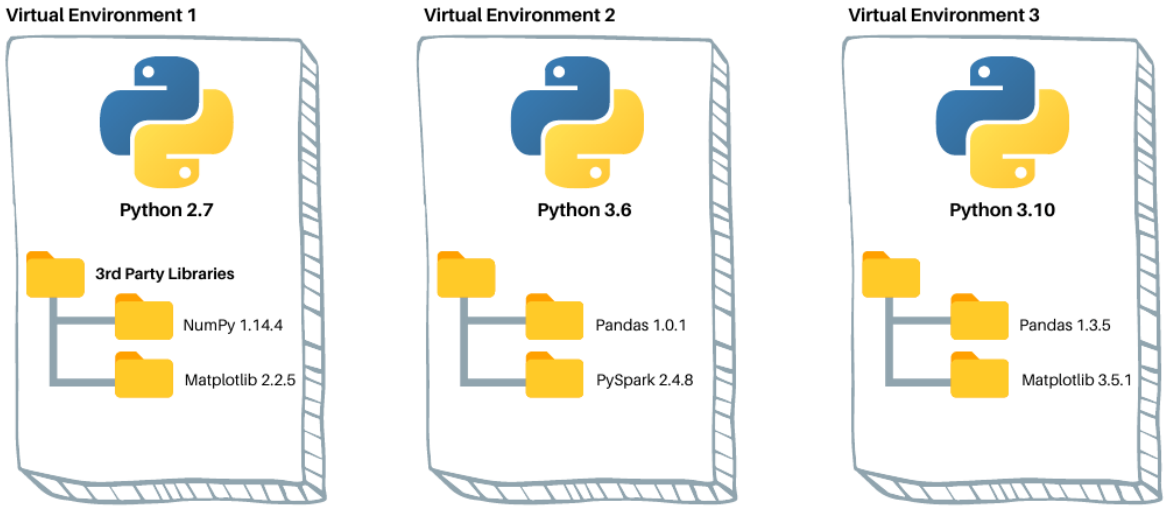
venv

virtualenv

pipenv

conda

poetry



dataquest.io

PEP

https://peps.python.org/pep-0008/

```
# Correct:

def foo(x):
    if x >= 0:
        return math.sqrt(x)
    else:
        return None

def bar(x):
    if x < 0:
        return None
    return math.sqrt(x)
```

```
# Wrong:

def foo(x):
    if x >= 0:
        return math.sqrt(x)

def bar(x):
    if x < 0:
        return
    return math.sqrt(x)
```

А если PEP соблюдать лениво?

https://github.com/psf/black

Установка:

```
pip install black
pip install black[jupyter]
```

Использование:

```
black some_py.py
```

Black в действии

```python
# in

def my_some_function(arg1: str = 'bla', arg2: int = 55, arg3: float = 0.5, arg4: int = 5, arg5: int = 7, arg6: float = 0.3) -> str:
    res = int((arg2*2*float //3 + arg4 -arg5) *5 + 1 -3   ) *arg1
    return res


# out

def my_some_function(
    arg1: str = "bla",
    arg2: int = 55,
    arg3: float = 0.5,
    arg4: int = 5,
    arg5: int = 7,
    arg6: float = 0.3,
) -> str:
    res = int((arg2 * 2 * float // 3 + arg4 - arg5) * 5 + 1 - 3) * arg1
    return res
```

```
$ black src/dataset.py
reformatted src\dataset.py

All done! ✨ 🍰 ✨
1 file reformatted.
```

Линтеры

Большое описание линтеров -- ссылка

Из основного:

- pylint
- Flake8
- mypy
- isort
- ...

уже не жива ☹

---

cemsbr/**yala**

Yet Another Linter Aggregator

4 Contributors | 49 Used by | 15 Stars | 5 Forks

Линтеры

```
$ yala ./src/examples/01_check_pylint.py > yala_report.txt
INFO: Finished pycodestyle
INFO: Finished isort
INFO: Finished pyflakes
INFO: Finished pydocstyle
INFO: Finished flake8
INFO: Finished radon cc
INFO: Finished radon mi
INFO: Finished mypy
INFO: Finished pylint
```

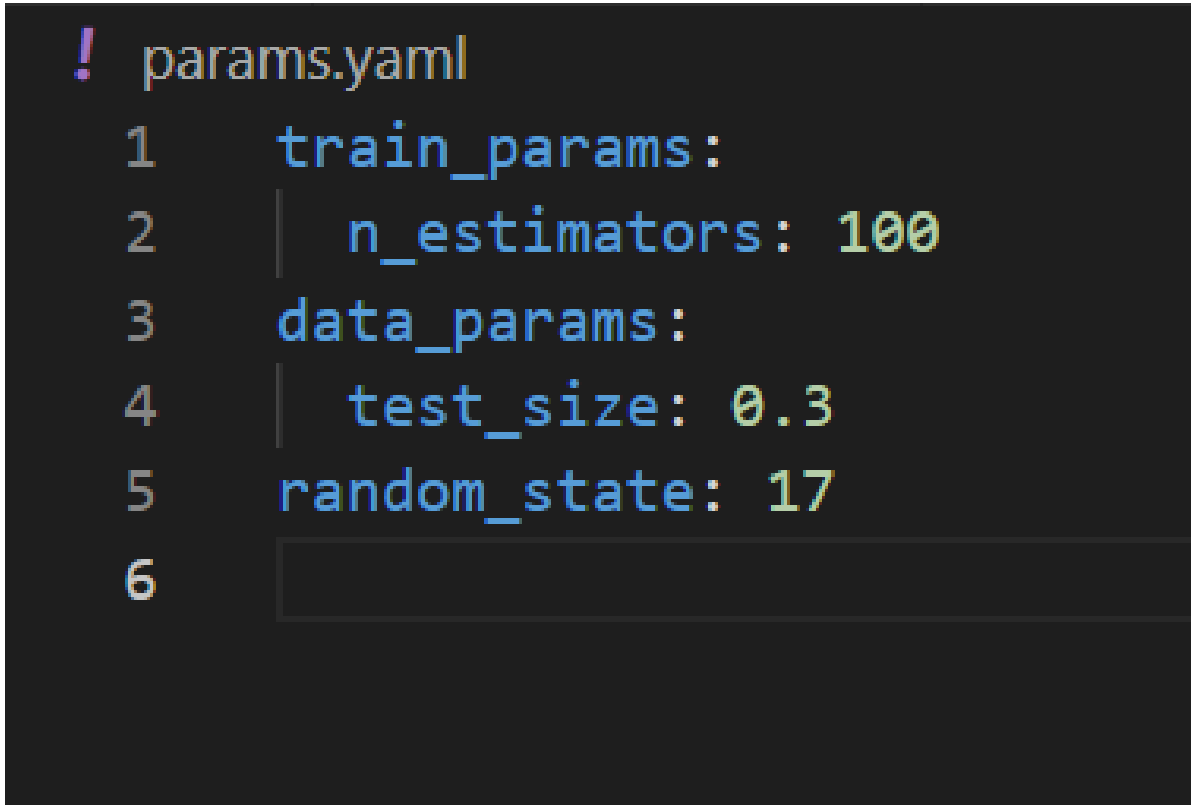
```
$ pylint src/examples/01_check_pylint.py
************* Module 01_check_pylint
src\examples\01_check_pylint.py:1:0: C0114: Missing module docstring (missing-module-docstring)
src\examples\01_check_pylint.py:1:0: C0103: Module name "01_check_pylint" doesn't conform to snake_case naming style (invalid-name)
src\examples\01_check_pylint.py:7:0: C0103: Constant name "shift" doesn't conform to UPPER_CASE naming style (invalid-name)
src\examples\01_check_pylint.py:10:0: C0103: Constant name "letters" doesn't conform to UPPER_CASE naming style (invalid-name)
src\examples\01_check_pylint.py:11:0: C0103: Constant name "encoded" doesn't conform to UPPER_CASE naming style (invalid-name)
src\examples\01_check_pylint.py:15:12: C0103: Constant name "encoded" doesn't conform to UPPER_CASE naming style (invalid-name)
src\examples\01_check_pylint.py:17:12: C0103: Constant name "x" doesn't conform to UPPER_CASE naming style (invalid-name)
src\examples\01_check_pylint.py:18:12: C0103: Constant name "encoded" doesn't conform to UPPER_CASE naming style (invalid-name)
src\examples\01_check_pylint.py:22:12: C0103: Constant name "encoded" doesn't conform to UPPER_CASE naming style (invalid-name)
src\examples\01_check_pylint.py:24:12: C0103: Constant name "x" doesn't conform to UPPER_CASE naming style (invalid-name)
src\examples\01_check_pylint.py:25:12: C0103: Constant name "encoded" doesn't conform to UPPER_CASE naming style (invalid-name)

-------------------------------------------------------------------
Your code has been rated at 4.21/10 (previous run: 4.21/10, +0.00)
```

Конфигурации

Можно создавать при помощи .yaml, .json, .py

Если оборачивать в dataclass, будет прекрасно

Оборачивать можно с помощью marshmallow , pydantic, hydra



Конфигурации

```python
@dataclass()
class PipelineParams:
    train_params: TrainParams
    data_params: DataParams
    random_state: int


PipelineParamsSchema = class_schema(PipelineParams)


def read_pipeline_params(path: str) -> PipelineParams:
    with open(path, "r") as input_stream:
        schema = PipelineParamsSchema()
        return schema.load(yaml.safe_load(input_stream))
```

Pytest

```
$ pytest tests/
================================== test session starts ==================================
platform win32 -- Python 3.9.6, pytest-8.3.3, pluggy-1.5.0
rootdir: C:\Users\anana\Desktop\projs\2024\mlops_course
configfile: pyproject.toml
plugins: anyio-4.2.0
collected 1 item

tests\test_params_loading.py .                                                    [100%]

=================================== 1 passed in 0.26s ===================================
```

Без тестирования:

1. Код сломанный
2. Если все работает сегодня, то может сломаться завтра
3. Сложно добавлять функциональность и проверять, что ничего не сломалось

Домашнее задание

1) Оформить свой репозиторий, согласно cookiecutter
2) Создать виртуальное окружение через venv, весь дальнеший код исполняется только через него. Не забудьте создать файлик requirements.txt.
3) Написать пайплайн обучения и валидации модели машинного обучения на сгенерированном датасете (для генерации используйте make_classification из sklearn)
   Необходимо реализовать все (почти все числа, которые будут использоваться в коде) в конфигах pydantic/marshmallow/hydra.
   Кроме того, пайплайн должен быть способным обучить логистическую регрессию, случайный лес и решающее дерево, в зависимости от того, как что сконфигурировано
4) Добиться оценки pylint >= 8.0
5) Написать тесты на реализованные функции, чтобы площадь покрытия тестами была не менее 70%