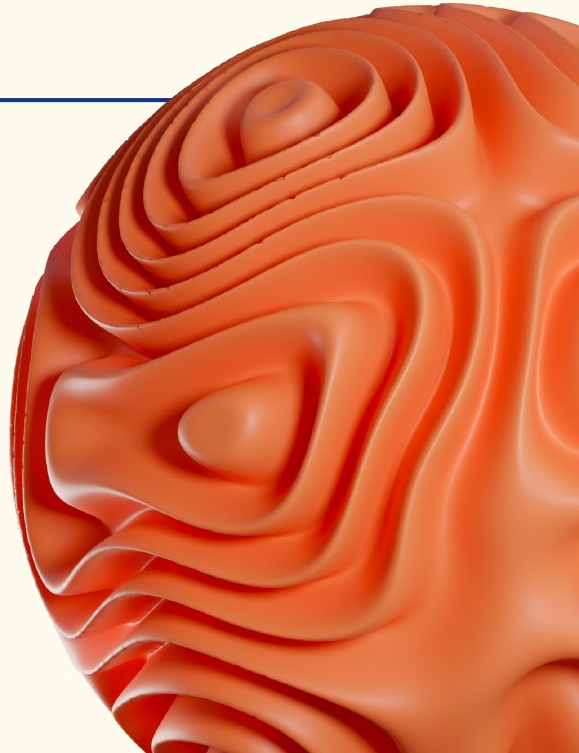
Модели ML в production

МИФИ × SKILLFACTORY

Лекция № 5

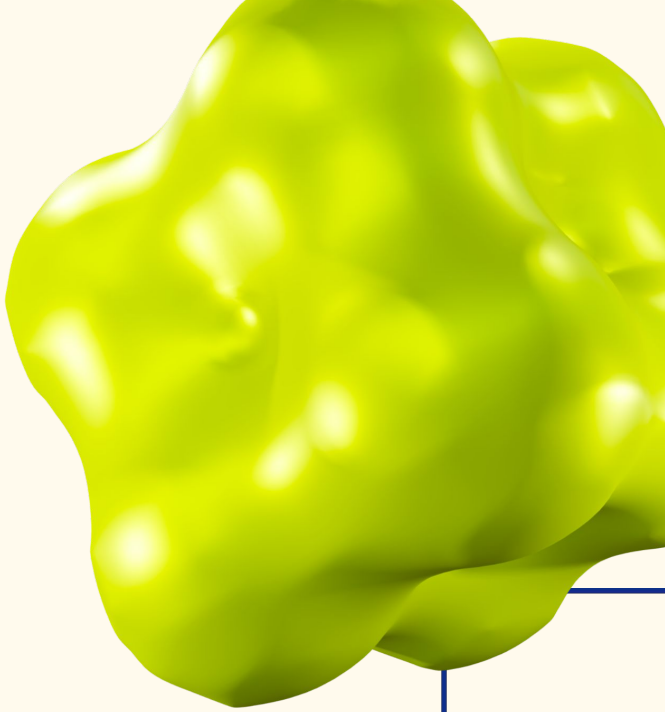
“Виртуальные окружения”



Жарова Мария Александровна
DS WB-tech, Math&Python&DS lecturer
t.me/data_easy

План занятия

1. Проблема изолированности и практика с ВО:
    - virtual env
    - conda
    - poetry



Проблема изолированности

Какие могут быть проблемы?

МИФИ × SKILLFACTORY

Наш сервис (и сама модель) протестирован только локально на нашем компьютере.

Что, если его запустит кто-то другой у себя?

- у него может не быть нужных библиотек
- у него может не быть нужных *версий* библиотек
- у него может быть другая, несовместимая ОС...

Это проблема **воспроизводимости**!

Реальный проект - большая команда

МИФИ × SKILLFACTORY

Что поможет отслеживать ход выполнения проекта:

- системы **контроля версий для кода** (git: GitHub, GitLab, Bitbucket)
- версионирование **артефактов** (DVC, Nexus)
- управление **экспериментами** (MLFlow, Comet.ML)
- **виртуализация и контейнеризация** (Docker)
- отслеживание запусков **расчётов на проде** (AirFlow)

Виртуализация

Виртуализация — это технология изоляции, сохранения состояния и воспроизведения *окружения*.

Сможем воссоздать на другом компьютере точную копию вашего *окружения* и тем самым обеспечить **воспроизводимость**.

1. Первые решения: *удалённое управление конфигурациями* (на основе большого конфигурационного файла, со всеми настройками, зависимостями и библиотеками, настраивались все удалённые машины).

- Тяжело следить за полнотой и правильностью конфигурационных файлов.
- ПО, напрямую влияющее на стабильность работы приложения.

virtualenv

Улучшенное решение: виртуальная среда, **изолированная** от основной системы ⇒ каждый проект имеет свои независимые настройки, в т. ч. разные библиотеки и их версии.

Python virtualenv:

```
$ python3 -m venv venv_name
$ source venv_name/bin/activate
(venv_name) $ pip install scikit-learn
deactivate

(venv_name)$ pip freeze
(venv_name) $ pip freeze > requirements.txt
pip install requirements.txt
```

virtualenv

Сравните:

```
$ which python3
```

и

```
(venv_name) $ which python3
```

При активации в.о. меняются специальные переменные среды ОС ⇒ при вызове интерпретатора **используется не глобальная версия интерпретатора Python**, который имеет доступ ко всем установленным пакетам, **а его копия**, которая лежит в папке с в.о.

Он имеет доступ только к “своей” папке, где хранятся библиотеки, установленные в это в.о.

conda

Conda – это менеджер пакетов и сред, ориентированный не только на Python, но и на C/C++, R, JavaScript и другие языки.

Управляет не только питоновскими пакетами, но и бинарными!

Лучше для ML и DS:)

conda

Создание новой среды:

```
conda create --name my_env python=3.9
```

Её активация:

```
conda activate my_env
```

Деактивация:

```
conda deactivate
```

Удаление среды:

```
conda env remove --name my_env
```

conda

Установка пакетов:

```
conda install numpy pandas scikit-learn
```

Обновить пакет:

```
conda update numpy
```

Удалить пакет:

```
conda remove pandas
```

Экспортировать:

```
conda env export > environment.yml
```

```
conda list --export > requirements.txt
```

conda-forge

conda-forge – альтернативный репозиторий (канал) для conda, который предоставляет свежие и хорошо протестированные и кроссплатформенные пакеты.

Как установить через conda-forge:

```
conda install -c conda-forge pandas
```

PS: по умолчанию используется канал defaults

poetry

Poetry — это более мощный инструмент управления зависимостями и упаковки Python-проектов.

В отличие от pip и virtualenv:

- предоставляет декларативный и удобный способ работы с зависимостями,
- автоматизирует управление версиями,
- связывает ВО с отдельными проектами.

Генерируются следующие файлы:

- pyproject.toml (декларативное описание зависимостей)
- poetry.lock (зафиксированные версии зависимостей:))

poetry

Установка:

```
pip install poetry
```

Проверка версии:

```
poetry --version
```

Создание нового проекта:

`poetry new my_project` *или* `poetry init`

(сформируется pyproject.toml – вместо setup.py и requirements.txt)

pyproject.toml

общая информация о проекте
(полезно менять версию!)

основные зависимости
(используем регулярные выражения и можно составить сразу список)
зависимости только для разработки

настройки сборки

```toml
[tool.poetry]
name = "my_project"
version = "0.1.0"
description = "Example project"
authors = ["Your Name <you@example.com>"]
license = "MIT"

[tool.poetry.dependencies]
python = "^3.10"
numpy = "^1.25"

[tool.poetry.dev-dependencies]
pytest = "^7.4"

[build-system]
requires = ["poetry-core"]
build-backend = "poetry.core.masonry.api"
```

poetry

Добавить новую (через терминал):

```
poetry add pandas
poetry add requests@^2.28
poetry add polars==’0.19.19’
```

Удалить существующую (через терминал):

```
poetry remove pandas
```

Обновить все:

```
poetry update
```

Обновить конкретную:

```
poetry update polars
```

PS: если изменили pyproject.toml вручную, нужно выполнить `poetry lock --no-update`

poetry

Активировать “оболочку”:

```
poetry shell
```

Запустить код внутри окружения (без активации!):

```
poetry run python script.py
```

Удалить окружение:

```
poetry env remove python
```

Сгенерировать requirements.txt:

```
poetry export -f requirements.txt --output requirements.txt
```

Итоги: что выбрать?

Функция	pip + virtualenv	conda	Poetry
Автоматическое создание виртуального окружения	❌	✅	✅
Декларативное управление зависимостями	❌	✅	✅
Локфайл для воспроизводимости	requirements.txt	environment.yml	poetry.lock
Поддержка pyproject.toml	❌	❌	✅
Встроенная публикация пакетов	❌	❌	✅
Поддержка Conda-зависимостей	❌	✅	❌

Спасибо
за внимание!

МИФИ × SKILLFACTORY