



ML Advanced

Production Code проекта на примере задачи
классификации/регрессии, Virtual environments,
dependency management, pip/gemfure

• REC

Проверить, идет ли запись

Меня хорошо видно
&& слышно?



Ставим "+", если все хорошо
"-", если есть проблемы

Тема вебинара

ML Advanced Production Code.

Игорь Стурейко



Teamlead, главный инженер проекта – НИИгазэкономика

Опыт:

Более 15 лет занимался прикладной математикой и математическим моделированием (Data Scientist) (Python, C++) в НИИ ПАО Газпром

Анализ временных рядов, эволюционное развитие сложных систем

+7 (916) 156-07-82 (whatsapp)

@stureiko (TG)

Правила вебинара



Активно
участвуем



Off-topic обсуждаем
в учебной группе



Задаем вопрос
в чат или голосом



Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения



Индивидуально



Время, необходимое
на активность



Пишем в чат



Говорим голосом

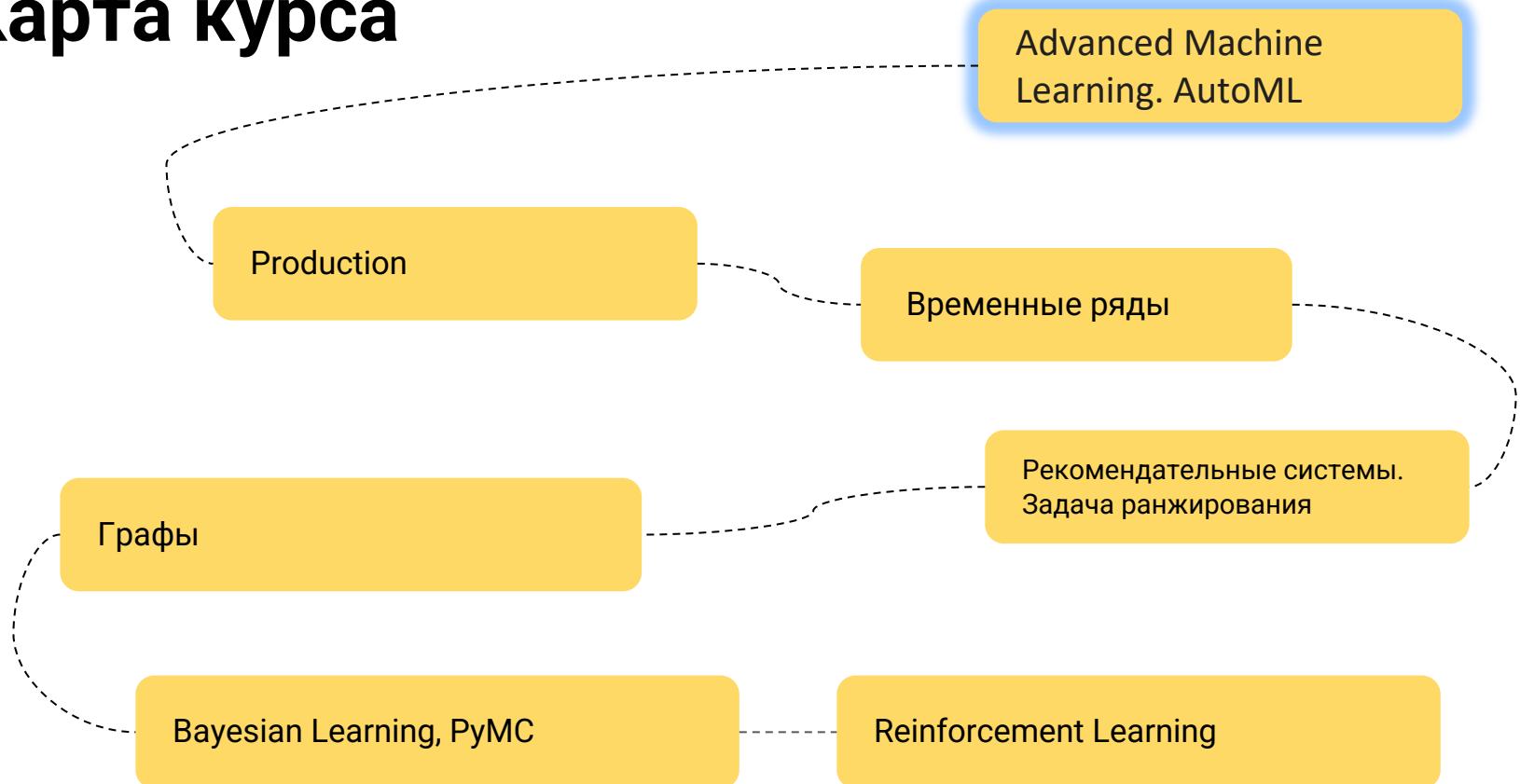


Документ



Ответьте себе или
задайте вопрос

Карта курса



Цели вебинара

К концу занятия вы сможете

1. Понять что такое качественный / «хороший» код
2. Понять почему это важно
3. Познакомиться с лучшими практиками

Маршрут вебинара

Проблемы обзор инструментов DS

Best practices

Полезные инструменты

Рефлексия

Что происходит в DS и в чем проблема?

Языки



Прелести Python

- Динамическая типизация
- Огромная экосистема -готовые пакеты
- Удобство связи делегирования вычислений
- Популярность, в частности в ML

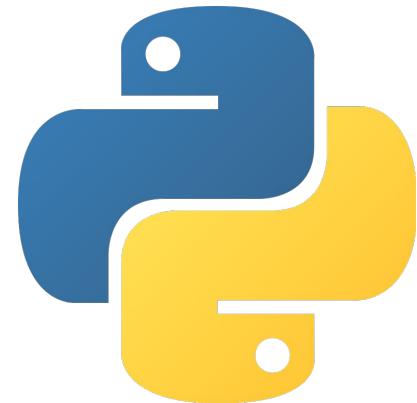


PYTORCH

XGBoost



TensorFlow



Плюсы и минусы Jupyter Notebook

Плюсы:

- Интерактивность
- Удобная визуализация
- Сохраняются промежуточные результаты
- Разворачивается на удаленном сервере

Минусы:

- Глобальный скоп перменных
- Порождают копипасту
- Очень слабая IDE
- Не дружит с GIT
- Никаких тестов

5 reasons why jupyter notebooks suck



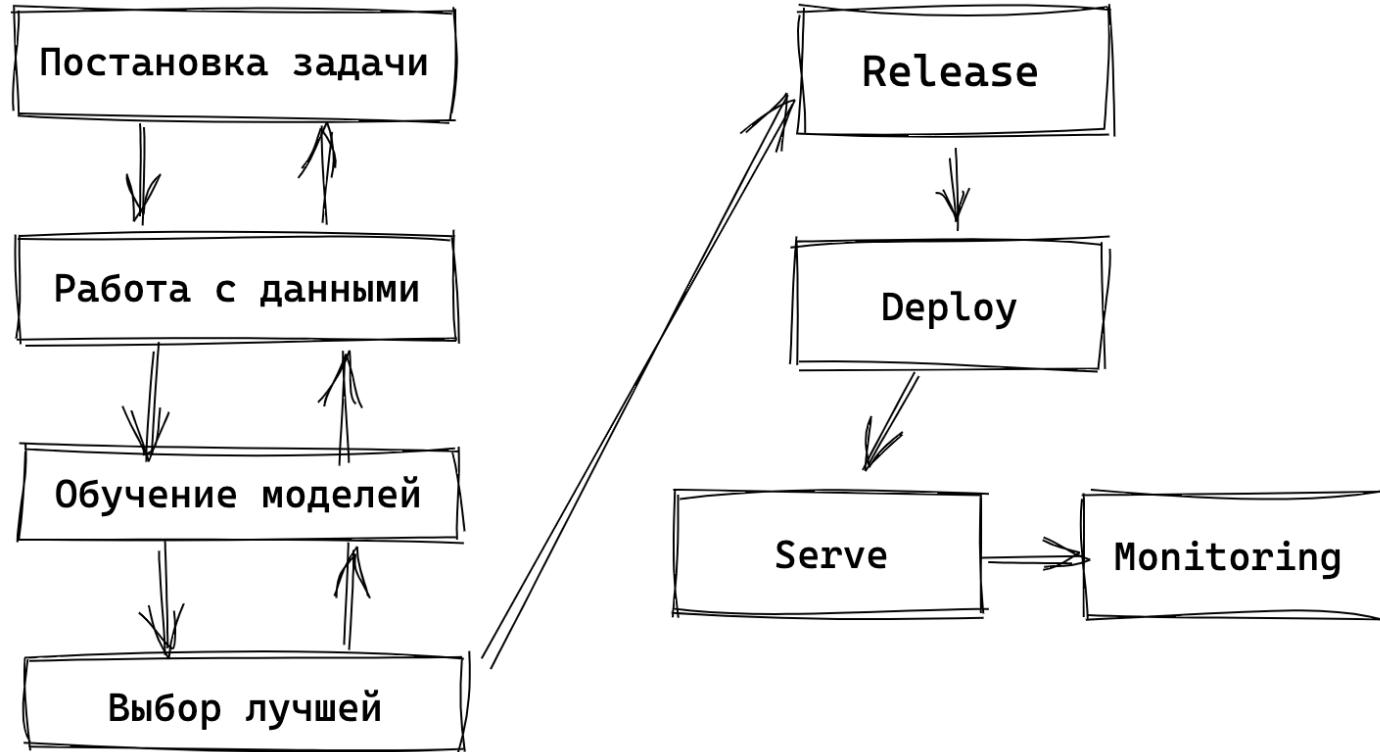
Alexander Mueller Mar 24, 2018 · 3 min read ★



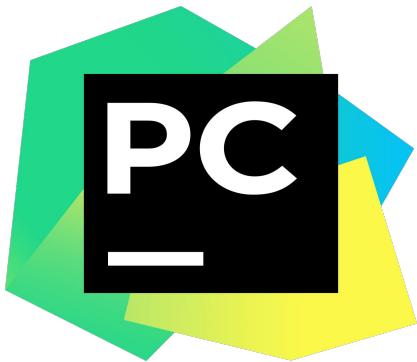
How it feels like managing jupyter notebooks (Complexity ©
<https://www.flickr.com/photos/bitterjung/7670055210>)



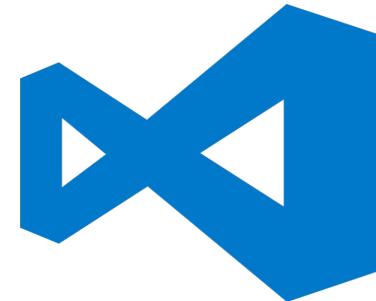
Чего бы хотелось? Жизненный цикл в ML



Инструменты



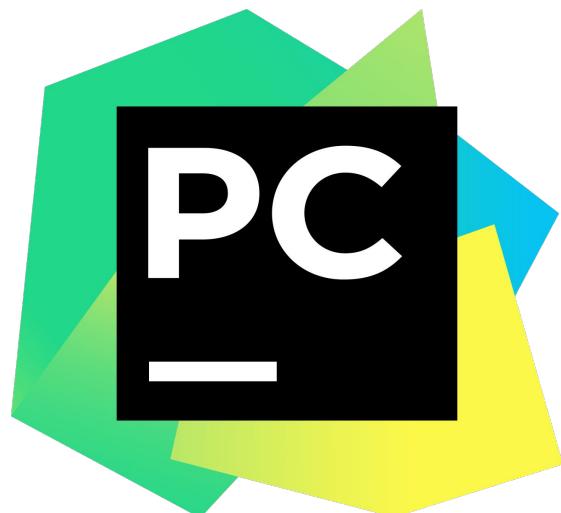
PyCharm



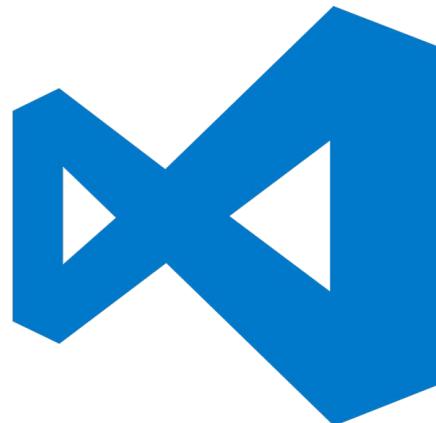
H2O.ai



Рекомендации для курса



PyCharm



VS CODE

Центральная проблема мировоззрений

Удобство как в Jupyter

Стабильность как в Pycharm



Характеристики продакшн кода

- Читабельный
- Эффективный
- Покрытый тестами
- Модульный
- Задокументированный

+ Характеристики продакшн проекта:

- Версионирование
- Удобная структура
- Явные зависимости

“Always code as if the guy who ends up maintaining your code
will be a violent psychopath who knows where you live”

— John Woods



Вопросы?



Ставим “+”,
если вопросы есть



Ставим “-”,
если вопросов нет

Читабельный код

Читабельный код: переменные

```
for i in range(n):
    for j in range(m):
        for k in range(l):
            temp_value = X[i][j][k] * 12.5
            new_array[i][j][k] = temp_value + 150
```

<https://towardsdatascience.com/data-scientists-your-variable-names-are-awful-heres-how-to-fix-them-89053d2855be>



Читабельный код: переменные

```
PIXEL_NORMALIZATION_FACTOR = 12.5
```

```
PIXEL_OFFSET_FACTOR = 150
```

```
for row_index in range(row_count):
    for column_index in range(column_count):
        for color_channel_index in range(color_channel_count):
            normalized_pixel_value = (
                original_pixel_array[row_index][column_index][color_channel_index]
                * PIXEL_NORMALIZATION_FACTOR
            )
            transformed_pixel_array[row_index][column_index][color_channel_index] = (
                normalized_pixel_value + PIXEL_OFFSET_FACTOR
            )
```

<https://towardsdatascience.com/data-scientists-your-variable-names-are-awful-heres-how-to-fix-them-89053d2855be>

PEP8 is all you need

Naming Styles

The table below outlines some of the common naming styles in Python code and when you should use them:

Type	Naming Convention	Examples
Function	Use a lowercase word or words. Separate words by underscores to improve readability.	function, my_function
Variable	Use a lowercase single letter, word, or words. Separate words with underscores to improve readability.	x, var, my_variable
Class	Start each word with a capital letter. Do not separate words with underscores. This style is called camel case.	Model, MyClass
Method	Use a lowercase word or words. Separate words with underscores to improve readability.	class_method, method
Constant	Use an uppercase single letter, word, or words. Separate words with underscores to improve readability.	CONSTANT, MY_CONSTANT, MY_LONG_CONSTANT
Module	Use a short, lowercase word or words. Separate words with underscores to improve readability.	module.py, my_module.py
Package	Use a short, lowercase word or words. Do not separate words with underscores.	package, mypackage



Нет портнякам

jupyter lgb-and-cat (автосохранение)

File Edit View Insert Cell Kernel Widgets Help Не доверять Python 3

Out[4]:

	card_id	target	delivery_type	addr_region_reg	addr_region_fact	channel_name	channel_name_2	channel_name_modified_2018	sas_limit_after_003_amt	s
0	cid_10620	1	cat_1	107	107	cat_0	cat_3	cat_0	1	
1	cid_105724	0	cat_1	9	9	cat_2	cat_5	cat_2	3	
2	cid_101410	1	cat_1	109	109	cat_0	cat_3	cat_0	1	
3	cid_38961	0	cat_1	66	66	cat_0	cat_3	cat_0	3	
4	cid_57462	0	cat_1	16	16	cat_0	cat_3	cat_0	0	

5 rows x 92 columns

B []:

```
# d = data.groupby("addr_region_reg")['inquiry_1_week'].mean().to_dict()
# data['inquiry_1_week_mean'] = data.addr_region_reg.apply(lambda x: d[x])

# data["inquiry_old_count"] = data.ttl_inquiries - data.inquiry_12_month
data["ttl_inquiries_with_current"] = data.ttl_inquiries.fillna(0) + data.inquiry_recent_period.fillna(0)
data["inquiry_recent_period_perc"] = data.inquiry_recent_period.fillna(0) / (data.ttl_inquiries_with_current.fillna(0)*
data["loans_period"] = data.last_loan_date.fillna(0)-data.first_loan_date.fillna(0)
# data["loans_close_percent"] = (data.loans_main_borrower+data.loans_active)/(data.loans_active+0.001)
data["loans_close_times"] = data.loans_main.borrower.fillna(0) - data.loans_active.fillna(0)
# data["sas_limit_last_amt_delta"] = data.sas_limit_after_003_amt.fillna(0)==data.sas_limit_last_amt.fillna(0)

data["region_same"] = data.addr_region_reg.fillna(0)==data.addr_region_fact.fillna(0)
# data["region_same2"] = data.addr_region_reg==data.app_addr_region_reg
# data["region_same3"] = data.addr_region_fact==data.app_addr_region_fact

data['age'] = data.cltnt_birth_year.fillna(0)*365-data.first_loan_date.fillna(0)
data['age2'] = data.cltnt_birth_year.fillna(0)*365-data.last_loan_date.fillna(0)
data['f1'] = data.cltnt_income_month_avg_net_amt.fillna(0)-data.sas_limit_after_003_amt.fillna(0)
data['f2'] = data.loans_main.borrower.fillna(0)/(data.loans.period.fillna(0)*0.001)
data['f3'] = data.loans_active.fillna(0)/(data.loans_main.borrower.fillna(0)*0.001)
data["feature_20_21_27"] = data.feature_20.fillna(0)*data.feature_21.fillna(0)+data.feature_27.fillna(0)
data["feature_22_23_24_25_28_12_10_13"] = data.feature_13.fillna(0)*data.feature_10.fillna(0)+ data.feature_12.fillna(0)
```

Нет портянкам: разбиение на функции

```
def run_training() → NoReturn:  
    data = load_dataset(file_name=config.TRAINING_DATA_FILE)  
  
    X_train, X_test, y_train, y_test = train_test_split(  
        data[config.FEATURES], data[config.TARGET], test_size=0.1, random_state=0  
    )  
  
    pipeline.fit(X_train[config.FEATURES], y_train)  
  
    _logger.info(f"saving model version: {_version}")  
    save_pipeline(pipeline_to_persist=pipeline.price_pipe)
```

Если вы копируете код - пора создать функцию.

Если в функции больше 50 строчек, пора создать две.

SOLID: делаем хорошие блоки кода

S - Single Responsibility Principle - принцип единственной ответственности.

Каждый класс должен иметь только одну зону ответственности.

O - Open closed Principle - принцип открытости-закрытости.

Классы должны быть открыты для расширения, но закрыты для изменения.

L - Liskov substitution Principle - принцип подстановки Барбары Лисков.

Должна быть возможность вместо базового (родительского) типа (класса) подставить любой его подтип (класс-наследник), при этом работа программы не должна измениться.

I - Interface Segregation Principle - принцип разделения интерфейсов.

Данный принцип обозначает, что не нужно заставлять клиента (класс) реализовывать интерфейс, который не имеет к нему отношения.

D - Dependency Inversion Principle - принцип инверсии зависимостей.

Модули верхнего уровня не должны зависеть от модулей нижнего уровня. И те, и другие должны зависеть от абстракции. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.



Zen of Python

- 1.Красивое лучше, чем уродливое.
- 2.Явное лучше, чем неявное.
- 3.Простое лучше, чем сложное.
- 4.Сложное лучше, чем запутанное.
- 5.Плоское лучше, чем вложенное.
- 6.Разреженное лучше, чем плотное.
- 7.Читаемость имеет значение.
- 8.Особые случаи не настолько особые, чтобы нарушать правила.
- 9.Практичность важнее безупречности.
- 10.Ошибки никогда не должны замалчиваться, если они не замалчиваются явно.
- 11.Встретив двусмысленность, отбрось искушение угадать.
- 12.Должен существовать один и, желательно, только один очевидный способ сделать это.
- 13.Сейчас лучше, чем никогда.
- 14.Если реализацию сложно объяснить — идея плоха.
- 15.Если реализацию легко объяснить — идея, возможно, хороша.
- 16.Пространства имён — отличная штука! Будем делать их больше!



Как сделать код читабельным в вашем проекте

1. Ввести практику ревью кода
2. Автоматически форматировать код: flake8, black, mypy

```
def very_important_function(template: str, *variables, file: os.PathLike,
                            engine: str, header: bool = True, debug: bool = False):
    """Applies `variables` to the `template` and writes to `file`."""
    with open(file, 'w') as f:
        ...

# out:

def very_important_function(
    template: str,
    *variables,
    file: os.PathLike,
    engine: str,
    header: bool = True,
    debug: bool = False,
):
    """Applies `variables` to the `template` and writes to `file`."""
    with open(file, "w") as f:
        ...
```



Логирование. logging вместо print

- Возможности настраивать уровни
- Включать / отключать вывод
- Печатать не только на экран, но и в файл
- Гибкие настройки форматирования

```
import logging

logger = logging.getLogger()
logger.setLevel(logging.ERROR)
console_handler = logging.StreamHandler()
log_format = '%(asctime)s | %(levelname)s: %(message)s'
console_handler.setFormatter(logging.Formatter(log_format))

logger.addHandler(console_handler)
logger.debug('Here you have some information for debugging.')
logger.info('Everything is normal. Chill!')
logger.warning('Something unexpected but not important.')
logger.error('Something unexpected and important happened.')
logger.critical('OMG!!!!')
```

Вопросы?



Ставим “+”,
если вопросы есть



Ставим “-”,
если вопросов нет

Тестирование

Зачем тесты?

```
# test_capitalize.py

def capitalize(x):
    return x.capitalize()

def test_capitalize():
    assert capitalize('semaphore') == 'Semaphore'
```

1. Код без тестов это сломанный код.
2. Даже если всё работает сейчас, вы сломаете это завтра. Без тестов вы не узнаете об этом.
3. Заставляет вас писать код лучше.



Важно знать

1. Покрыть всё тестами нельзя, но нужно покрыть достаточно.
2. Обязательно тестировать, что вещи которые **не должны** работать действительно **не работают**.
3. Тесты должны быть изолированы и атомарны.
4. Тестируя связанные с ML вещи можно двумя способами:
 - a. Зафиксировать random.seed.
 - b. Исполнять множество раз, проверять агрегированные значения.

Тестирование кода - Pytest

Как проверить, что ничего не сломалось?

- CodeReview
- Модульное тестирование
 - Pytest
 - Unittest
- Покрытие кода тестами
 - Pytest-cov (pytest --cov=src tests)
- Continuous Integration



Тестирование кода

- Минимум, что нужно для тестов - функция с префиксом **test_**
- Проверка с помощью ключевого слова **assert**
- Наборы тестов и отчеты
- Более сложные варианты шаблона Arrange-Act-Assert (Given-When-Then) с помощью **fixtures**



pytest

Тестирование ML - Hypothesis

Инструмент для тестирования ML кода

```
@given(st.lists(st.integers()))
def test_reversing_twice_gives_same_list(xs):
    # This will generate lists of arbitrary length (usually between 0 and
    # 100 elements) whose elements are integers.
    ys = list(xs)
    ys.reverse()
    ys.reverse()
    assert xs == ys

@given(st.tuples(st.booleans(), st.text()))
def test_look_tuples_work_too(t):
    # A tuple is generated as the one you provided, with the corresponding
    # types in those positions.
    assert len(t) == 2
    assert isinstance(t[0], bool)
    assert isinstance(t[1], str)
```



Безопасность кода



- [Bandit](#) - это инструмент, предназначенный для поиска типичных проблем безопасности в коде на языке Python.



[Safety](#)



[Snyk](#)

Вопросы?



Ставим “+”,
если вопросы есть



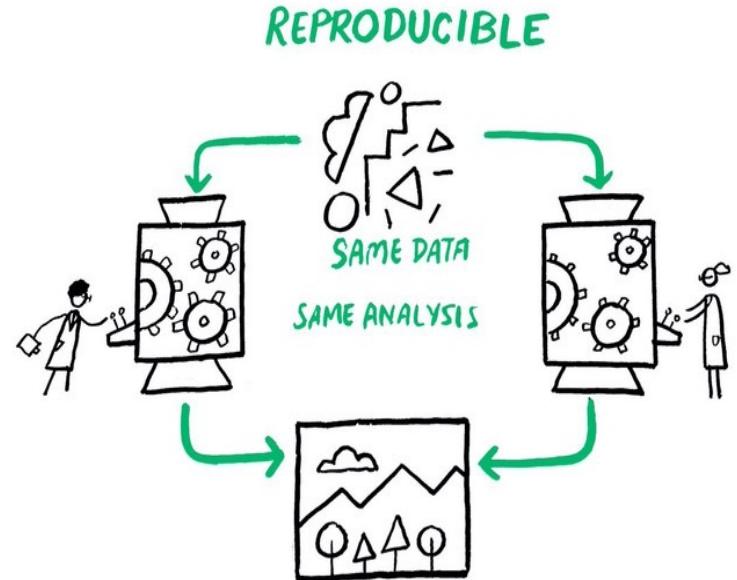
Ставим “-”,
если вопросов нет

Зависимости

Воспроизводимый код

Уровни изоляции кода:

- виртуальные окружения (venv)
- conda containers
- docker containers



Зависимости

```
pip install -r requirements.txt
```

```
sklearn==1.6.0
```

```
dataclasses=0.8
```

```
pyyaml==3.11
```

```
marshmallow-dataclass==8.3.0
```

```
torch>=1.5
```

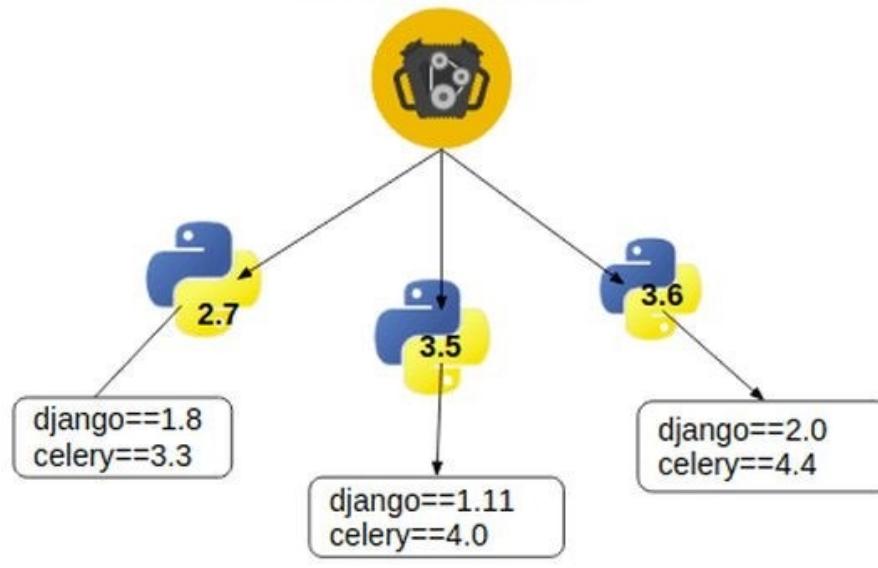
pip не проверяет конфликты зависимостей!



Виртуальные окружения

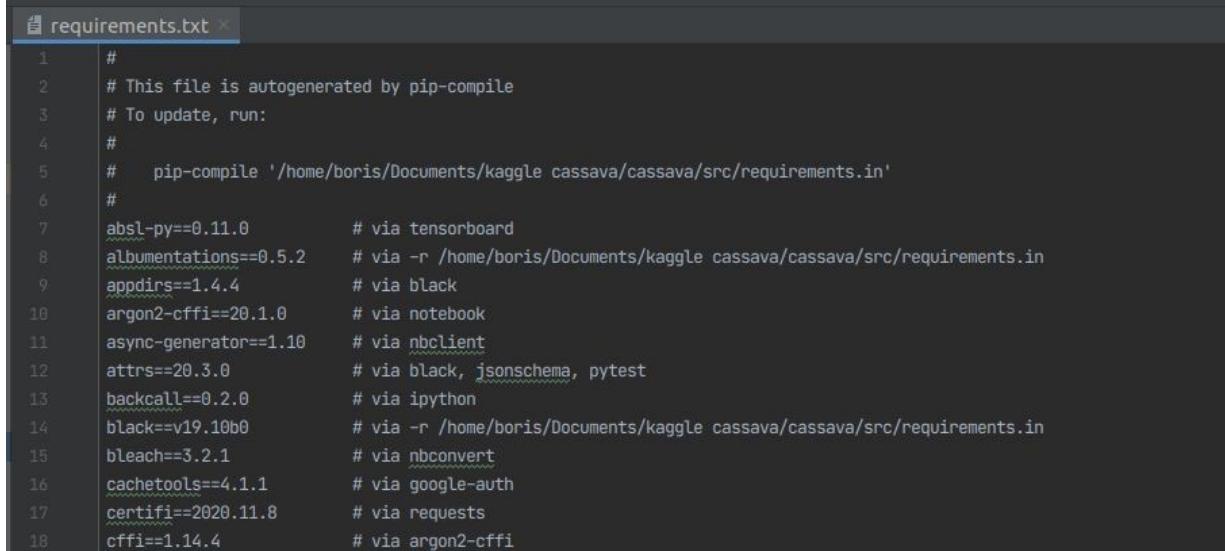
`python3.6 -m venv venv36`

Virtualenv



Инструменты

- `venv`
- `pipenv`
- `Conda`



A screenshot of a code editor window titled "requirements.txt". The file contains a list of Python package dependencies with their versions and source locations. The code is color-coded, with package names in blue and version numbers in green. The text is as follows:

```
1  #
2  # This file is autogenerated by pip-compile
3  # To update, run:
4  #
5  #     pip-compile '/home/boris/Documents/kaggle_cassava/cassava/src/requirements.in'
6  #
7  absl-py==0.11.0      # via tensorboard
8  albumentations==0.5.2 # via -r /home/boris/Documents/kaggle_cassava/cassava/src/requirements.in
9  appdirs==1.4.4        # via black
10 argon2-cffi==20.1.0   # via notebook
11 async-generator==1.10 # via nbclient
12 attrs==20.3.0         # via black, jsonschema, pytest
13 backcall==0.2.0       # via ipython
14 black==v19.10b0        # via -r /home/boris/Documents/kaggle_cassava/cassava/src/requirements.in
15 bleach==3.2.1          # via nbconvert
16 cachetools==4.1.1     # via google-auth
17 certifi==2020.11.8    # via requests
18 cffi==1.14.4          # via argon2-cffi
```

Инструмент Docker

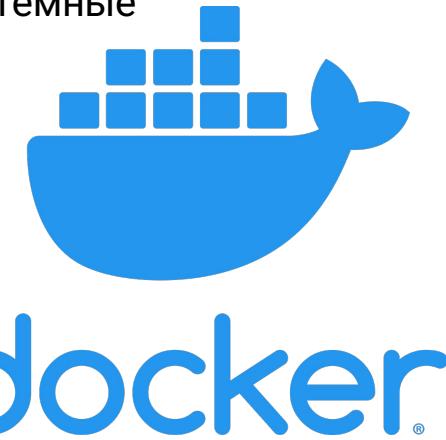
Главная идея: пусть окружение на компьютере будет таким же, как на продакшене.

Пример: https://github.com/btseytlin/celery_dao

Докер позволяет упаковать сервис в контейнер.

Контейнер ведет себя как маленькая виртуальная машина -своя OS.

Это позволяет контролировать не только Python зависимости, но и системные зависимости, и даже внешние зависимости.



Вопросы?



Ставим “+”,
если вопросы есть



Ставим “-”,
если вопросов нет

Хранение кода

Где хранить исходный код



AWS CodeCommit



AWS
CodeCommit



AWS
CodeBuild



AWS
CodeDeploy



AWS
CodePipeline

Azure Repos



Изменения в исходном коде



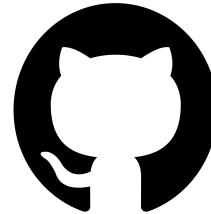
**Atlassian
Jira Software**



GitLab



Bugzilla



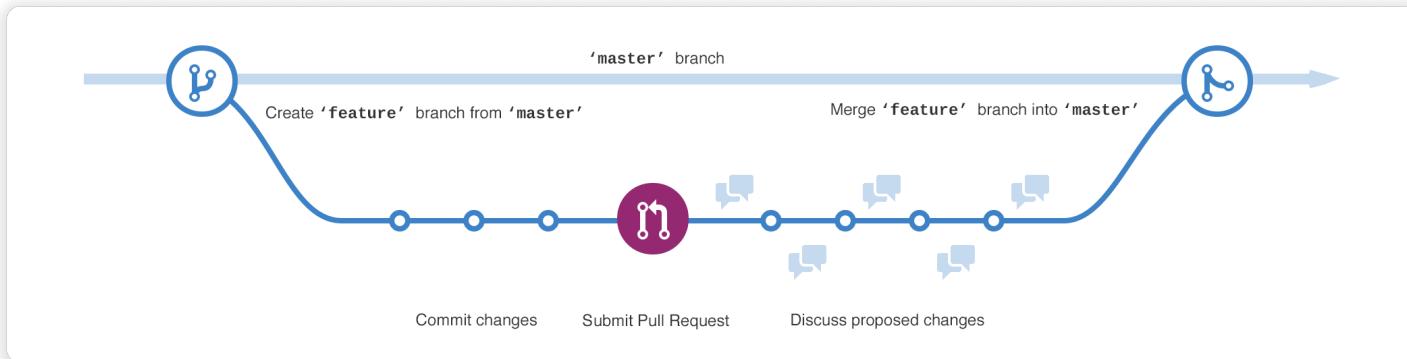
GitHub Issue



YouTrack

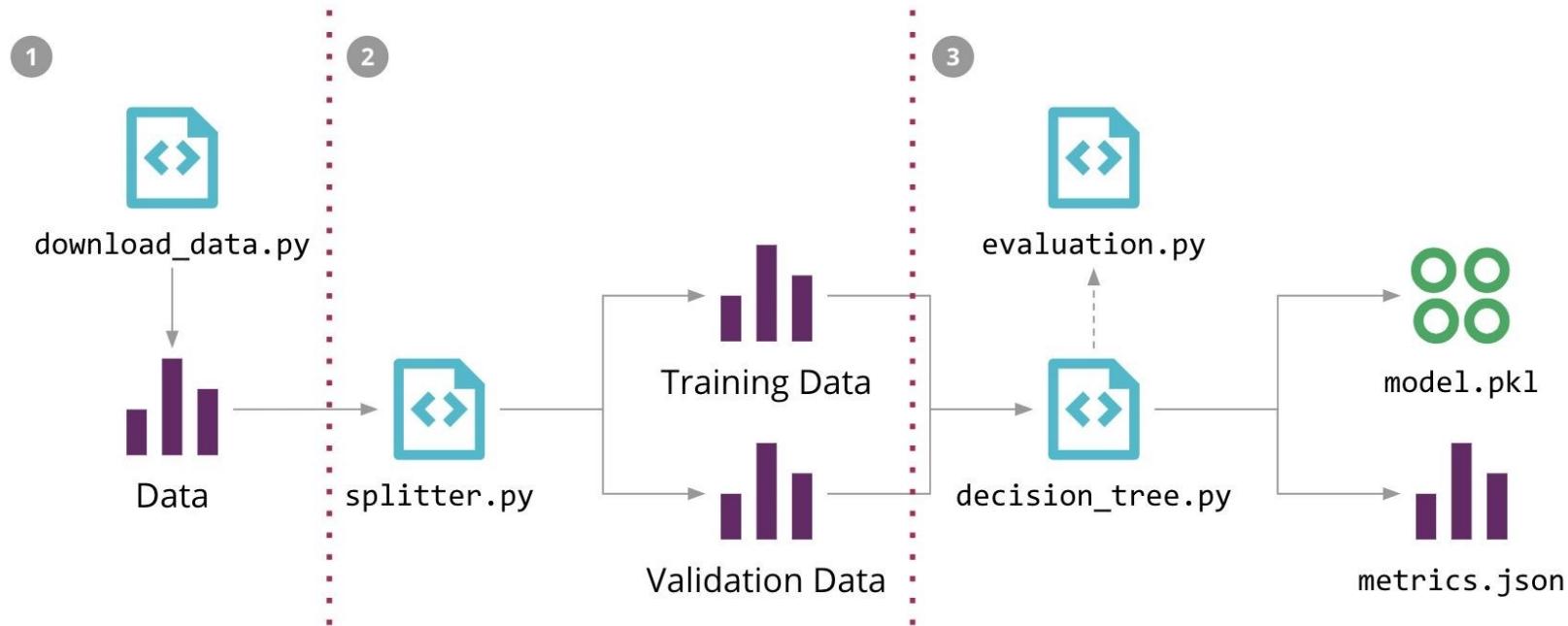
Как внести изменения

- Все изменения должны быть привязаны к тикетам
- Все изменения вносятся в ветках git
 - шаблон имени ветки - feature/<№ тикета>-описание
- В основные ветки код попадает через Pull Request
- PRы проходят Code Review



Структура проекта

Что-то такое



Организация проекта



<https://drivendata.github.io/cookiecutter-data-science>

<https://github.com/drivendata/cookiecutter-data-science>



<https://docs.kedro.org/en/stable/index.html>



```
. └── README.md          # основное описание проекта
    ├── data
    │   ├── external        # внешние данные
    │   ├── interim         # промежуточные данные
    │   ├── processed        # подготовленные данные
    │   └── raw              # исходные данные
    │       └── train.csv
    ├── models            # для сохранения моделей на диск
    ├── mypy.ini          # файл настроек mypy
    ├── notebook          # для jupyter блокнотов
    │   ├── Baseline.ipynb
    │   └── EDA.ipynb
    ├── reports           # для отчетов
    │   └── figures         # картинки для отчетов
    ├── requirements.txt  # зависимости
    └── setup.py          # описание пакета
```





```
src                         # исходные коды пакета
    titanic                   # пакет titanic
        __init__.py
        data                     # пакет titanic.data
            __init__.py
            make_dataset.py
            validation.py
        features                 # пакет titanic.features
            __init__.py
            extract.py
            fill.py
        models                   # пакет titanic.models
            __init__.py
            train.py
        visualization           # пакет titanic.visualization
    tests                      # директория с тестами
        test_data_validation.py
        test_features_extract.py
```





```
project-dir      # Parent directory of the template
├── .gitignore   # Hidden file that prevents staging of unnecessary files to `git`
├── conf         # Project configuration files
├── data         # Local project data (not committed to version control)
├── docs         # Project documentation
├── notebooks    # Project-related Jupyter notebooks (can be used for experimental code before moving the code to src)
├── pyproject.toml # Identifies the project root and contains configuration information
├── README.md    # Project README
└── .flake8       # Configuration options for `flake8` (linting)
└── src           # Project source code
```

Гибкая настройка конфигурации проекта на основе шаблонов

Вопросы?



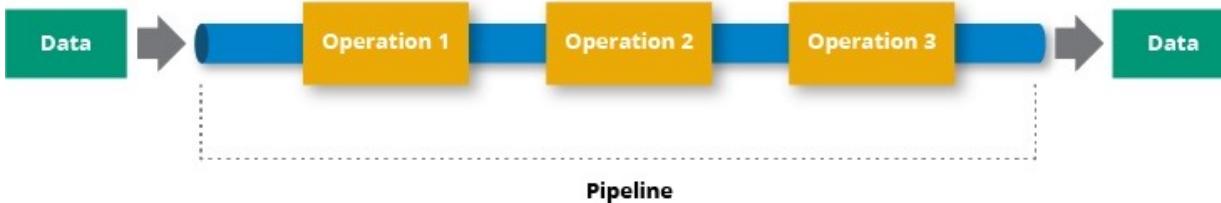
Ставим “+”,
если вопросы есть



Ставим “-”,
если вопросов нет

Что такое пайплайны и зачем они нужны

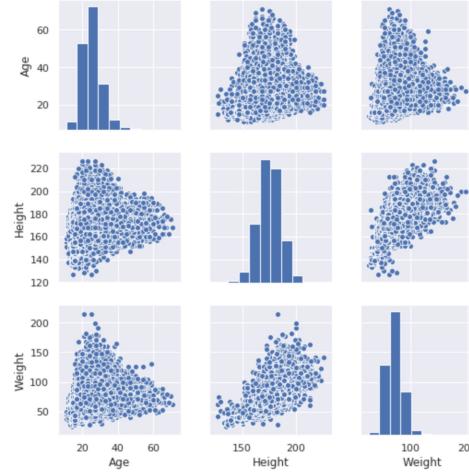
Pipeline



- Объединить несколько действий в один объект
- Убрать дублирование кода
- Работать с пайплайном как объектом sklearn вызывая стандартные функции .fit и .predict
- Сохранить пайплайн в виде целого объекта для дальнейшего переиспользования

Применение пайплайна

Да	Нет
EDA завершен и модель готова для фиксации и тюнинга	Стадия EDA
Модель готова для передачи в продакшн	Нет понимания итоговой структуры модели



Ключевые тезисы

1. Пайплайн помогает строить сложные модели быстрее и единообразным образом
2. Пайплайн следует применять на стадии перехода от EDA к finetuning
3. Пайплайн помогает стабилизировать модель и подготовить ее к переходу в production

Вопросы?



Ставим “+”,
если вопросы есть



Ставим “-”,
если вопросов нет

Рефлексия

Повторим основные мысли

1. Пишите читабельный код
2. Jupyter-ом всё не ограничивается
3. Дробите код на куски и пишите тесты
4. Фиксируйте зависимости
 - a. Lvl 1 crook: virtual-env
 - b. Lvl 100 boss: Docker
5. Определите структуру проекта
 - a. Lvl 1 crook: cookiecutter template
 - b. Level 100 boss: Kedro

Рефлексия



С какими впечатлениями уходите с вебинара?



Как будете применять на практике то,
что узнали на вебинаре?

**Заполните, пожалуйста,
опрос о занятии
по ссылке в чате**

Спасибо за внимание!

Следующие вебинары

03.11.23 – Оптимизация кода, parallelization, multiprocessing



Игорь Стурейко

Teamlead, главный инженер проекта – НИИгазэкономика

Опыт:

Более 15 лет занимался прикладной математикой и мат моделированием (Data Scientist) (Python, C++) в НИИ ПАО Газпром

Анализ временных рядов, эволюционное развитие сложных систем

+7 (916) 156-07-82 (whatsapp)

@stureiko (TG)