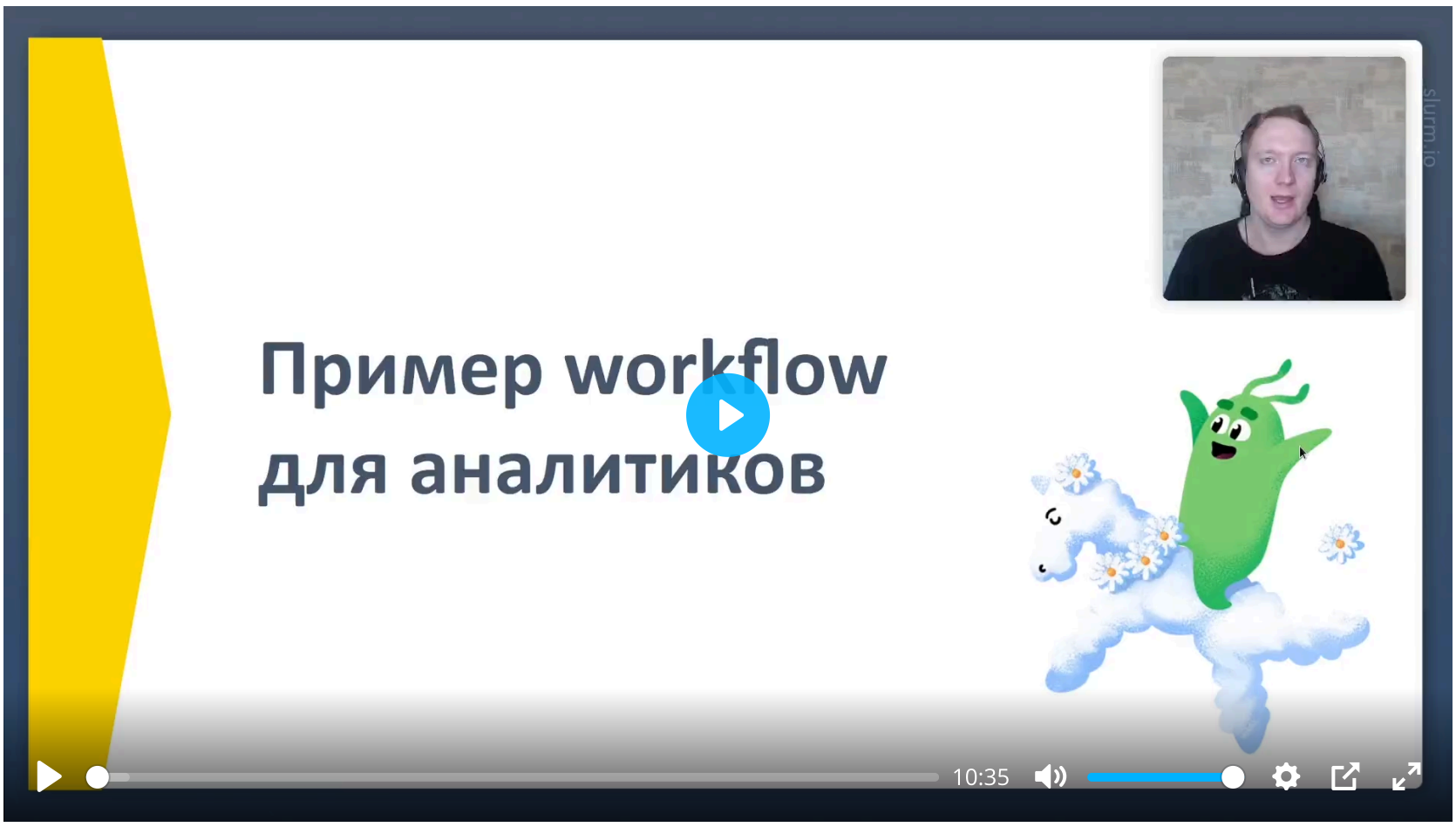


Текстовая расшифровка видео:

ПРИМЕР WORKFLOW ДЛЯ АНАЛИТИКОВ

План:

- Cookiecutter;
- Настраиваем без боли: Hydra и Dynaconf;
- Docker.

Cookiecutter

Вспомним урок, на котором мы обсуждали сборку Python-пакетов:

```
my_package <- это папка с нашим проектом
├── MANIFEST.in  <- до этого мы сейчас дойдем
├── my_package  <- имя для "import my_package"
│   ├── cli.py  <- это базовый файл с нашим кодом
│   ├── config.yaml  <- это файл конфигурации
│   └── __init__.py  <- это чаще всего пустой файл, который превращает папку в модуль питона
├── requirements.txt  <- это наш файл с зависимостями
└── setup.py  <- до этого мы сейчас дойдем
```

Возможно, на моменте объяснения данной структуры у вас возникли мысли о сложности в ее запоминании и создании.

Однако создание этой структуры легко автоматизировать (можно частично использовать Poetry). Также существует программа «Cookiecutter».

Cookiecutter:

- Запустить программу;
- Дать «печеньку» (дословно «cookies»);
- Программа задает ряд вопросов (название пакета, версия, имя автора и т.д.).

Cookies – разные декларативные описания того, как реализовать ту или иную структуру. Внутри «печеньки» есть базовый скелетный проект (набор каталогов и файлов). Это генерирует вам желаемую структуру.

Например, вам нужно сделать готовую структуру для Python-пакета. Если вы хотите сделать поддерживаемый пайплайн для аналитиков, то не нужно будет каждого из них заставлять хранить код в подобной структуре (см. пример). С помощью программы можно сгенерировать костяк, поместив код в стандартный файл (например, в cli.py). После чего вы можете создать любую удобную вам иерархию каталогов.

Все это облегчает создание структуры. Однако в эту структуру можно включить не только описание каталогов, позволяющие в дальнейшем собирать Python-пакет, но и, например, dockerfile, инструкции для CI/CD и т.д.

Cookiecutter написан на Python и использует один из самых известных шаблонизаторов «Jinja».

```
1. {
2.  "package_name": "coolproject",
3.   "version": "0.0.1",
4.   "package_author": "Nikolay"
5. }

~$ cookiecutter https://some-repo.com/cookiecutters/MyCookieCutter
Cloning into 'MyCookieCutter'...
remote: Counting objects: 37, done.
Unpacking objects:  21% (8/37)
remote: Total 37 (delta 19), reused 21 (delta 3), pack-reused 0
Unpacking objects: 100% (37/37), done.
Checking connectivity... done.
package_name [coolproject]: my_cool_research
file_name [version]: 1.0.0
package_author [Nikolay]: Alevtina
```

Настраиваем без боли: Hydra и Dynaconf.

Hydra.

Основная идея:

Вы создаете конфиг (по умолчанию рекомендуется использовать yaml) и основную функцию main, которую запускает пайплайн, оборачиваете в декоратор с указанием на то, где находятся конфиги. Исходя из этого, Hydra автоматически парсит файл конфига, переопределяет параметры из конфига в виде аргументов командной строки и т.д:

conf/config.yaml

```yaml
db:
  driver: mysql
  user: omry
  pass: secret
```

Application:

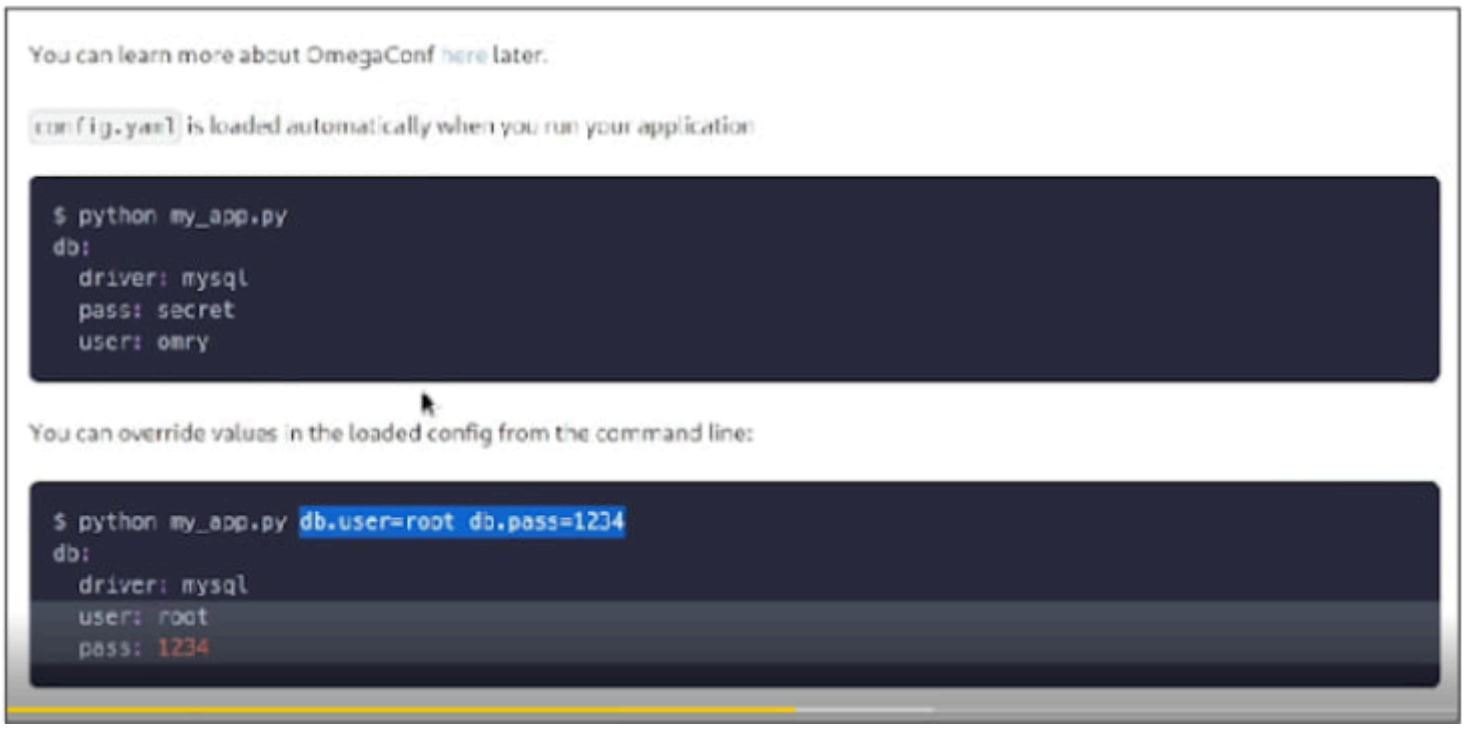
my_app.py

```python
import hydra
from omegaconf import DictConfig, OmegaConf

@hydra.main(version_base=None, config_path="conf", config_name="config")
def my_app(cfg : DictConfig) -> None:
    print(OmegaConf.to_yaml(cfg))

if __name__ == "__main__":
    my_app()
```

Помимо этого, можно иметь несколько конфиг-файлов в проекте с наследованием. Управление этим всем Hydra берет на себя:



Дата-сайентистам нравится в Hydra возможность использования плагина, который на основе конфигурационных параметров может делать базовый солвер. Например, если у вас есть гиперпараметры, и моделька быстро обучается, а вы не хотите писать логику по переборам параметров, но хотите решить оптимизационную задачу. Это решается при помощи плагина Hydr'ы. Так, вы запускаете оптимизатор, а он, в свою очередь многоразово запускает версии с кодом и ищет их оптимальную комбинацию:

Example 1: Single-Objective Optimization

We include an example in this directory. `example/sphere.py` implements a simple benchmark function to be minimized.

You can discover the Optuna sweeper parameters with:

```
python example/sphere.py hydra/sweeper=optuna --cfg hydra -p hydra.sweeper
```

Copy

```yaml
# @package hydra.sweeper
sampler:
  _target_: optuna.samplers.TPESampler
  seed: 123
  consider_prior: true
  prior_weight: 1.0
  consider_magic_clip: true
  consider_endpoints: false
  n_startup_trials: 10
  n_ei_candidates: 24
  multivariate: false
  warn_independent_sampling: true
_target_: hydra_plugins.hydra_optuna_sweeper.optuna_sweeper.OptunaSweeper
direction: minimize
storage: null
study_name: sphere
n_trials: 20
n_jobs: 1
max_failure_rate: 0.0
params:
  x: range(-5.5,5.5,step=0.5)
  y: choice(-5,0,5)
```

Dynaconf.

Основная идея:

В отличие от Hydra, данная программа работает с большим количеством конфигов, позволяет динамически импортировать в коде данные из конфигов и определяет их в качестве аргументов:

Dynaconf `init` command creates the following files

```
.
├── config.py       # Where you import your settings object (required)
├── .secrets.toml   # Sensitive data like passwords and tokens (optional)
└── settings.toml   # Application settings (optional)
```

your_program.py | config.py | settings.toml | .secrets.toml | env vars

On your own code you import and use **settings** object imported from your **config.py** file

```python
from config import settings

assert settings.key == "value"
assert settings.number == 789
assert settings.a_dict.nested.other_level == "nested value"
assert settings['a_boolean'] is False
assert settings.get("DONTEXIST", default=1) == 1
```

ℹ You can create the files yourself instead of using the `dynaconf init` command and it gives any name you want instead of the default `config.py` (the file must be in your importable python path)

Docker

С приходом Docker сильно изменился ландшафт работы с проектами. Теперь стадии в дата-инженеринговых пайплайнах мы можем реализовать в виде Python-проектов, собранных в Docker, а далее использовать удобные инструменты для запуска контейнеров, конфигурируя их снаружи конфигами или переменным окружением.

Данные контейнеры могут запускаться как инструментами CI/CD (например, Gitlab), так и более сложными инструментами для управления workflow (например, Дакстер).