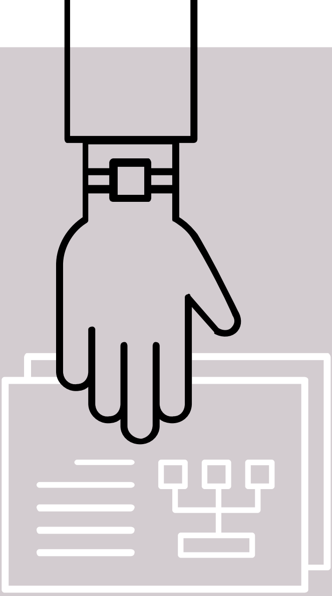
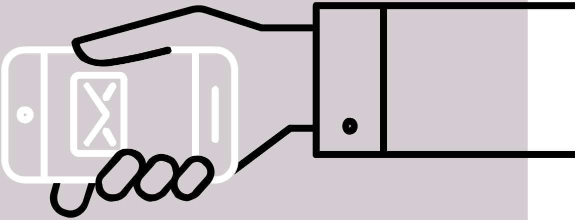
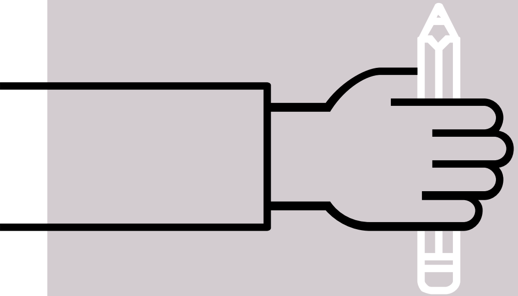
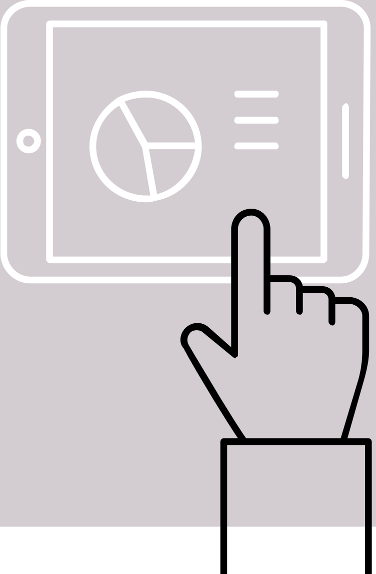




ML Pipeline





Kirill Vasin

2019

About me

- Kirill Vasin
- Data scientist at SEMrush

- Previously worked as a freelance python-dev and data scientist

About SEMrush

SEMrush - online Marketing Toolkit
for digital-marketing professionals

Founded
in **2008**

4,000,000+
users

700+
employees

Awards received in 2018-2019:

«Best SEO Suite»

«Best SEO Suite and
Best Search Software Tool»

«Best Digital Tool»

**6 offices
on two continents:**

Saint-Petersburg (Russia),

Prague (Czech Republic),

Limassol (Cyprus),

Philadelphia,

Boston,

Dallas (USA)

Structure

10x engineer

- Full-stack
- Converts "thought" into "code" in their mind
- Knows the entire production codebase
- Creates an ideal code from scratch

10x engineer

- Doesn’t use documentation or google things
- Laptop screen background color is black
- Keyboard keys such as **i, f, x** are usually worn out

10x engineer

- Hates meetings
- Attends the office irregularly
- Poor mentor

❌ Knowledge share
❌ Ideas share
❌ Code maintainability

Good software projects today

- Usually in one repo
- Usually well-structured
- Use VCS
- Have well-defined code style conventions
- Use testing

Most of ML Projects today

ML project surrounded by software projects

Andrew Ng
@AndrewYNg

Following

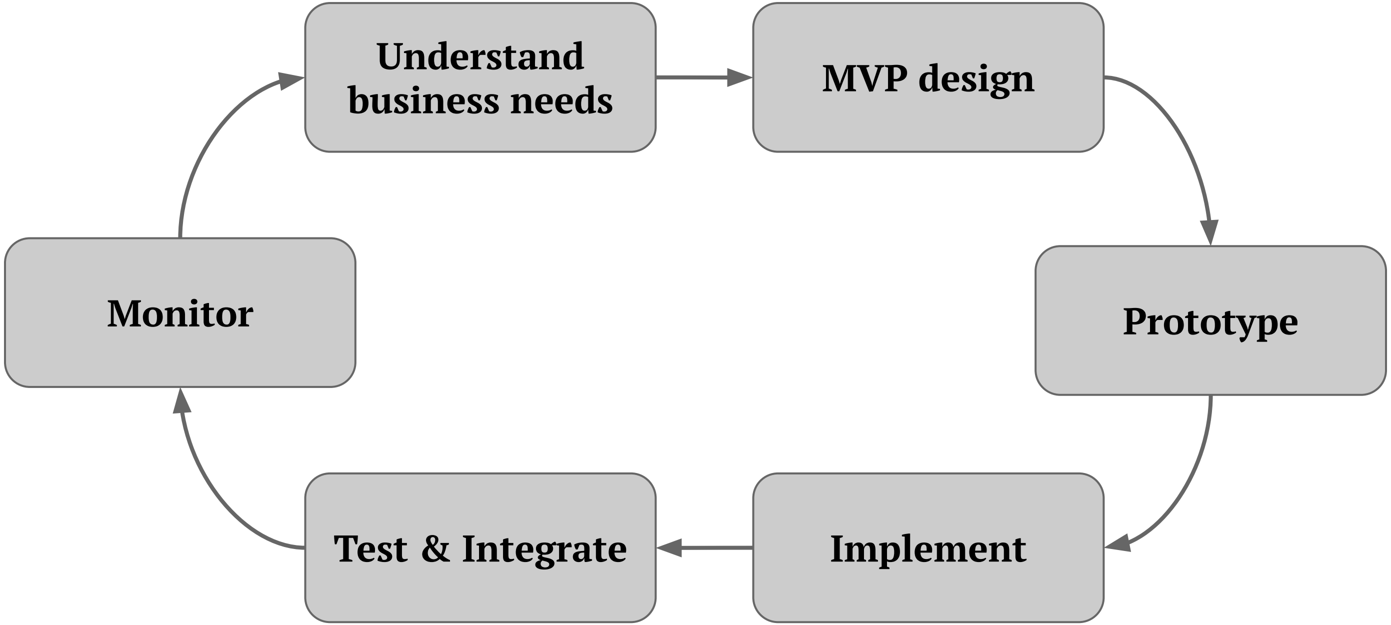
1/The rise of Software Engineering required inventing processes like version control, code review, agile, to help teams work effectively. The rise of AI & Machine Learning Engineering is now requiring new processes, like how we split train/dev/test, model zoos, etc.

9:59 AM - 3 Jan 2019

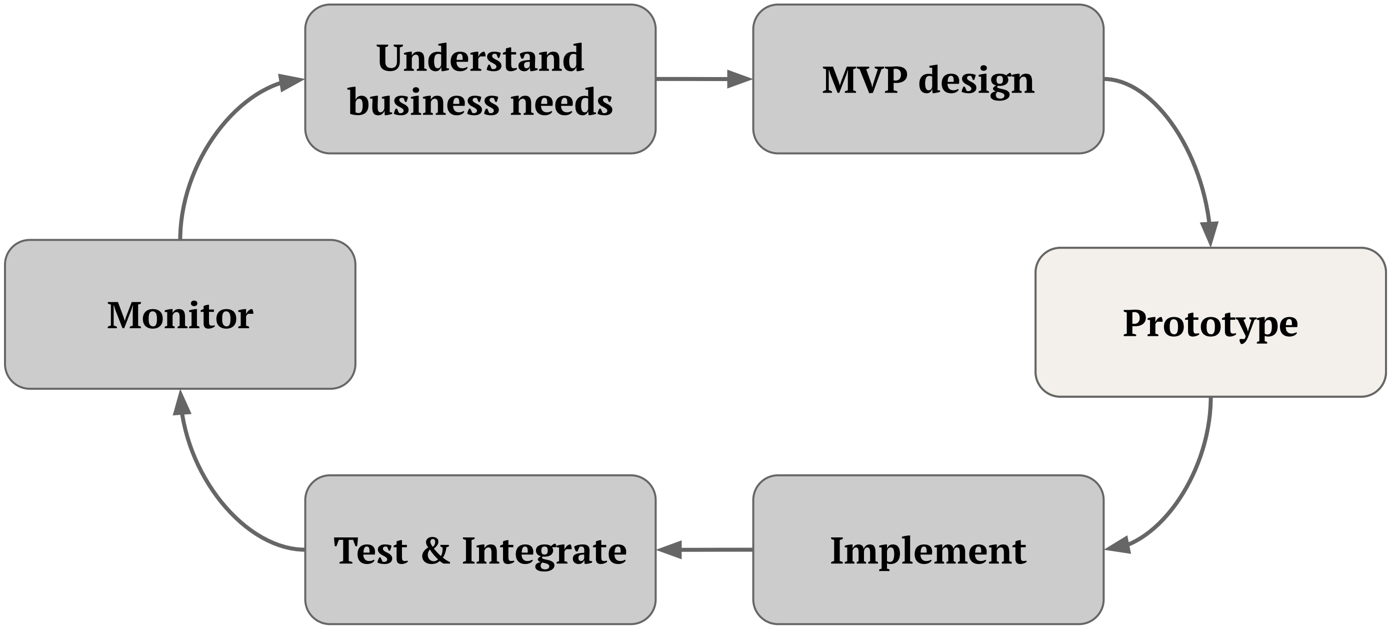
1,069 Retweets **3,462** Likes

50 1.1K 3.5K

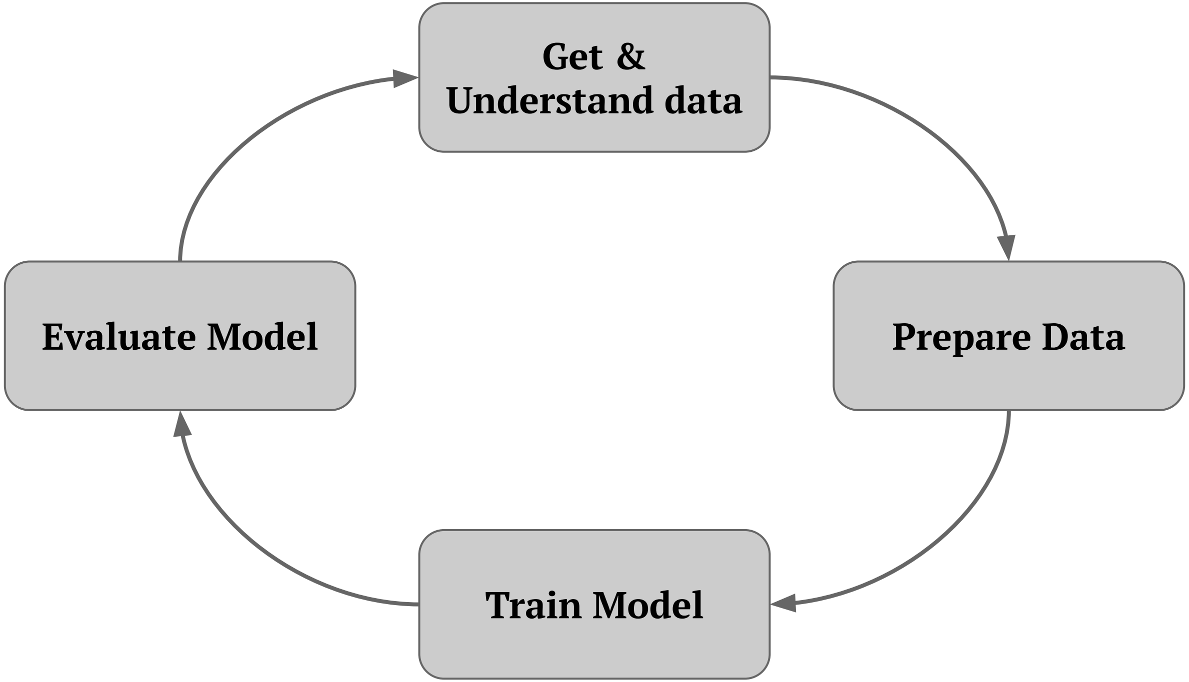
Software Dev Process



ML Process

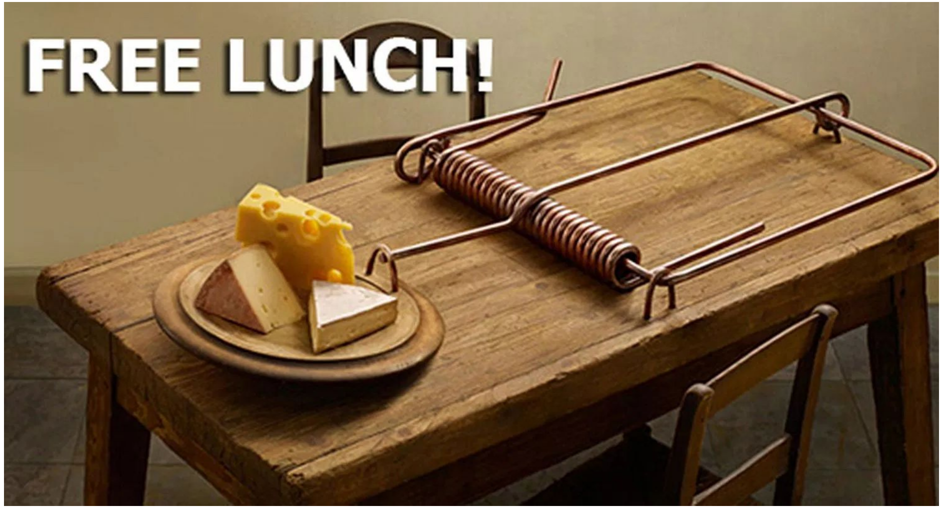

Understand business needs
MVP design
Prototype
Implement
Test & Integrate
Monitor

ML Process: Prototyping

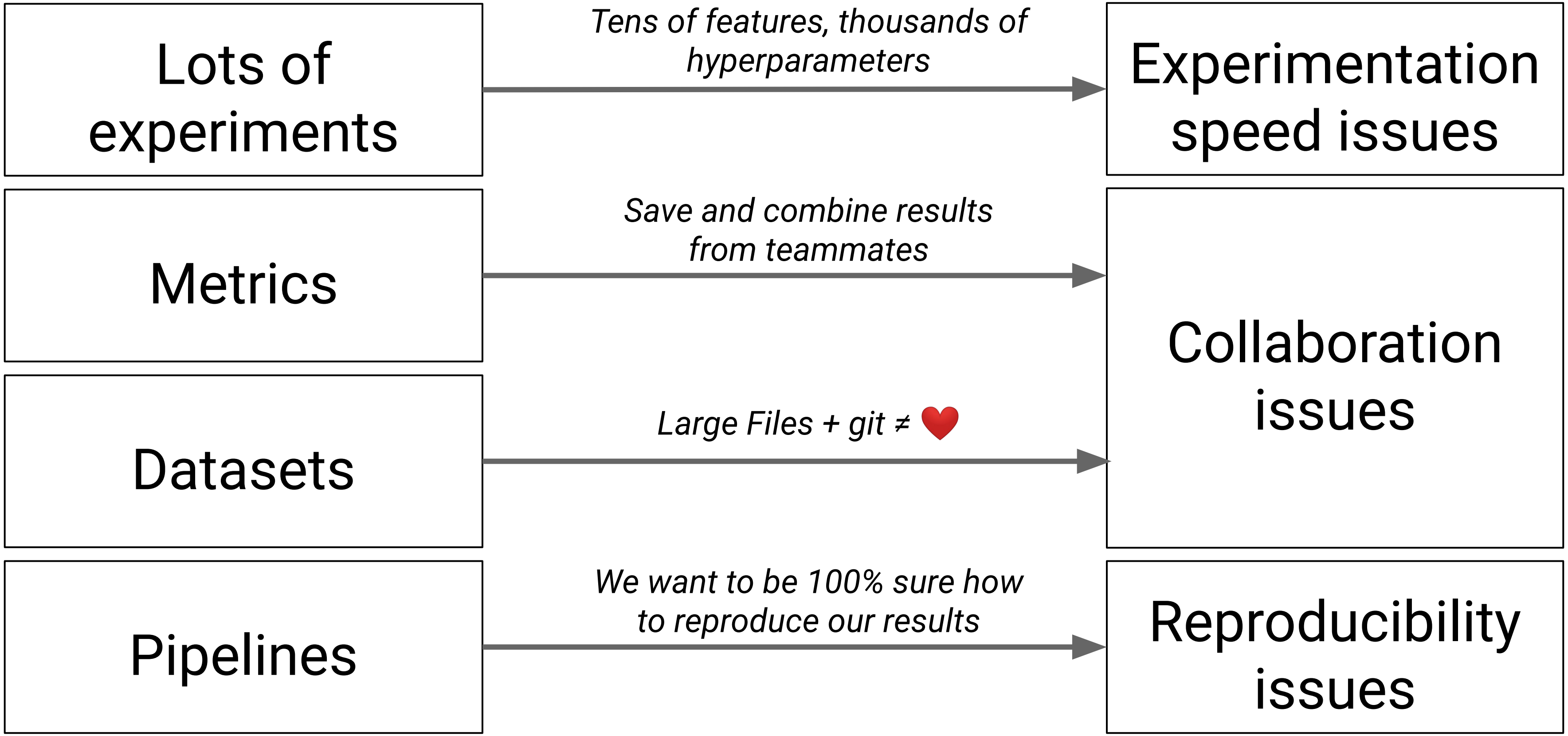


Experiments in ML

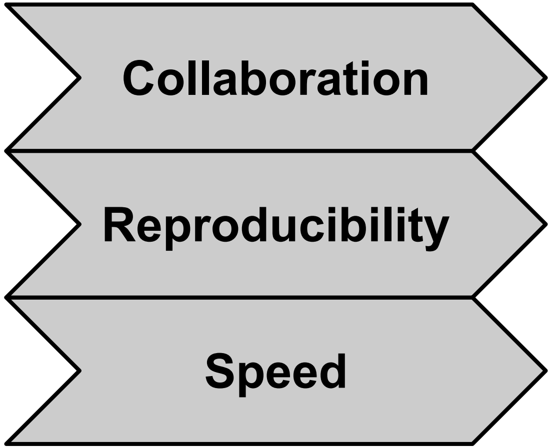
Experiment = (*Dataset * Data Pipeline * Model*) -> <u>Metric</u>



That creates some issues



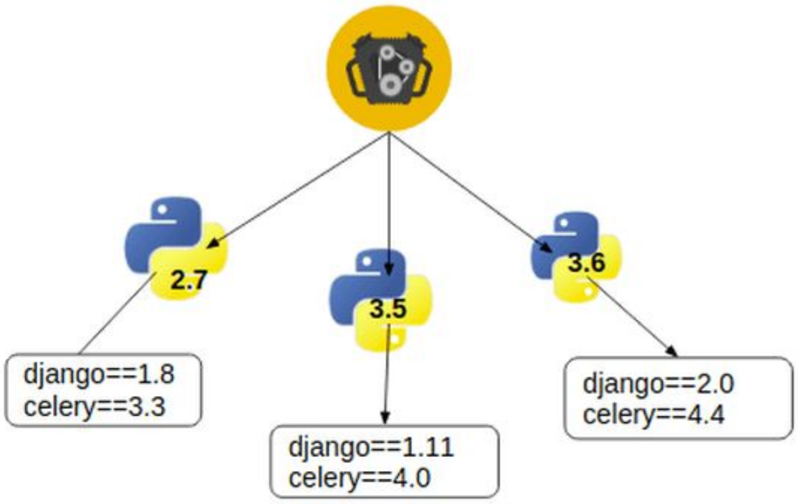
Let’s try to solve them



Virtualenv and/or Docker

Reproducibility

- Create isolated environments



<u>As a result</u>:

- Portability
- Easy project dependency tracking
- Update python packages without risk of breaking old projects

Pre-commit hooks

- Runs scripts before each commit

As a result, you and your teammates follow the same code style agreement

Usually I use the following hooks:

- pylint
- flake8
- check-added-large-files

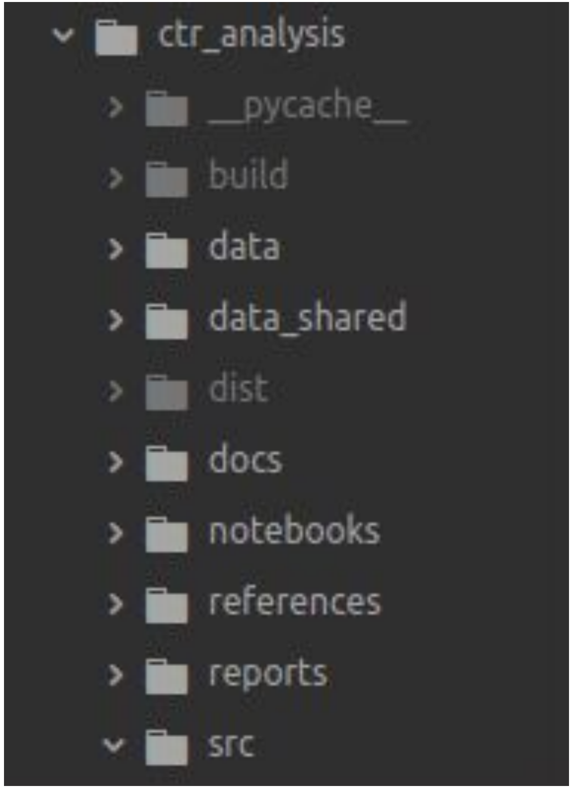
Cookiecutter

$ django-admin startproject mysite

```
mysite/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

Cookiecutter

$ cookiecutter my_awesome_template_folder



Cookiecutter

Collaboration | Speed

Creates custom project templates.

<u>As a result</u>:

- Navigation inside your projects become easier
- New projects are created with one command
- Simple onboarding

Cookiecutter-data-science

- **Makes a library out of your project code** which you can use inside your jupyter notebooks
- Automatically creates a documentation for your projects

Good starting point for almost any ML project

Jupyter notebooks

```
In [2]: print(a)

        hello world!

In [1]: a = "hello world!"
```

Jupyter notebooks

❌ No version control support
❌ Non-linear workflow
❌ No modularity
❌ Hard to test, read and reuse

❤️ Perfect for fast prototyping
❤️ Ideal for EDA and visualizations

NETFLIX

jupyter

Speed

Jupyter notebooks

- Fast prototyping
- EDA and visualisations

How to use:

- Name conventions (1.0-author_name-eda.ipynb)
- Move good code to the .py files

As a result:

- notebooks are easier to read
- code becomes shareable
- reports creation is faster

Collaboration → Speed

Optuna

- Bayesian hyperparameter optimization
- Pruning of unpromising trials

As a result:

- Hyperparameter tuning takes less time
- Unified approach to hyperparameter tuning

Collaboration | Speed

Optuna

- SQL Backend

<u>As a result:</u>

- Parallel experiments on different machines
- Results kept in a database in a unified fashion

Reproducibility → Collaboration → Speed

DVC

- Provides version control after data, pipelines and models
- Supports S3, Azure, GCP, SSH as a data storage
- Caches results of the pipeline stages
- Language agnostic

<u>As a result:</u>

- Easy rollbacks and branching
- Single storage for all data
- No recalculations
- Incremental development.


DVC REPRO
risovach.ru

Pipeline tools overview

- **Reproducibility**
  - Portable environment (virtualenv)
  - VCSs for data, pipelines and models (dvc)
- **Collaboration**
  - VCSs for code, data, pipelines and models (git and dvc)
  - Standardized templates (cookiecutter)
  - Code quality standards (pre-commit)
  - Shared RDB backend(optuna)
- **Experimentation speed**
  - Cache pipeline steps (dvc)
  - Smart hyperparameter optimization (optuna)
  - Fast prototyping (jupyter notebooks).

Good software and ML projects today

- Usually in one repo
- Usually well-structured
- Use VCS
- Have well-defined code style conventions
- Use testing

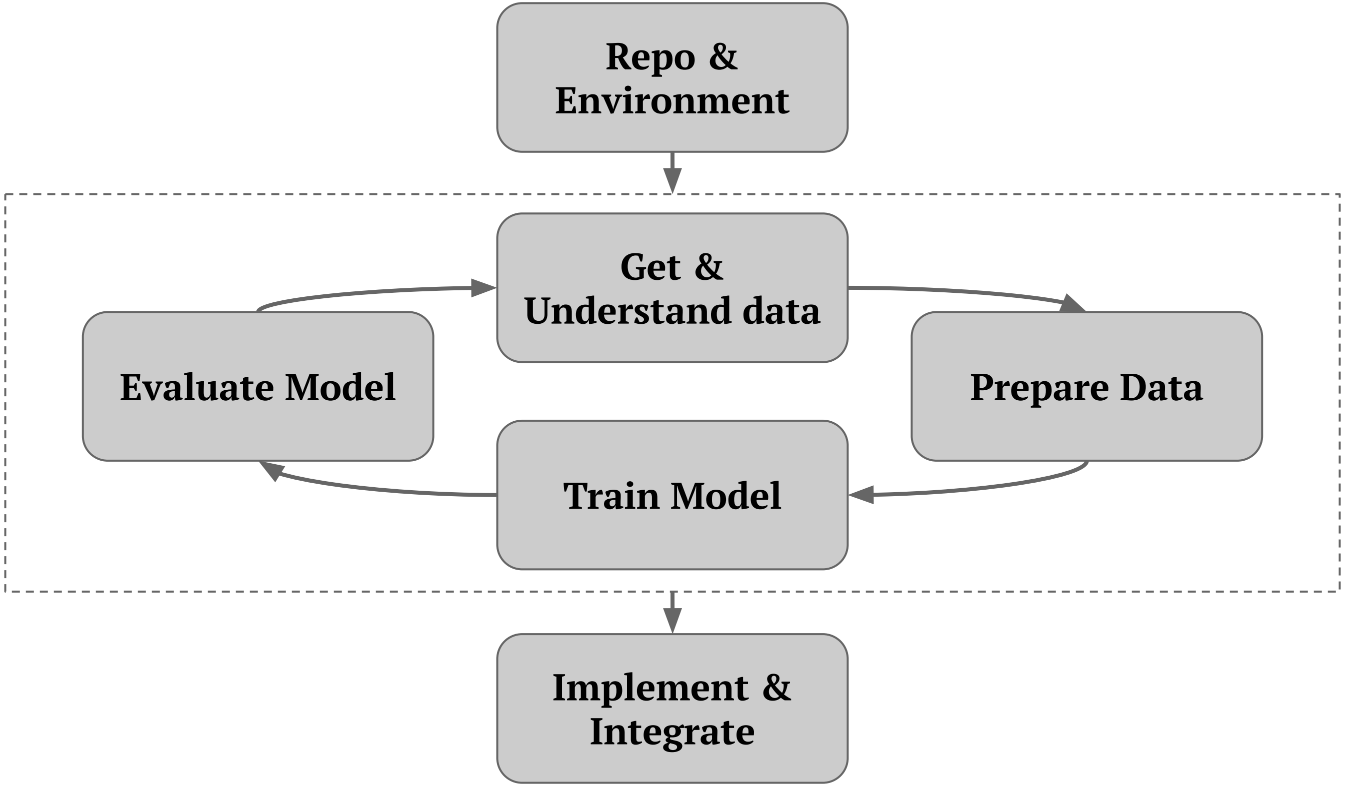
Pause

10 minutes after learning about this pipeline

10 hours after learning about this pipeline

My working process

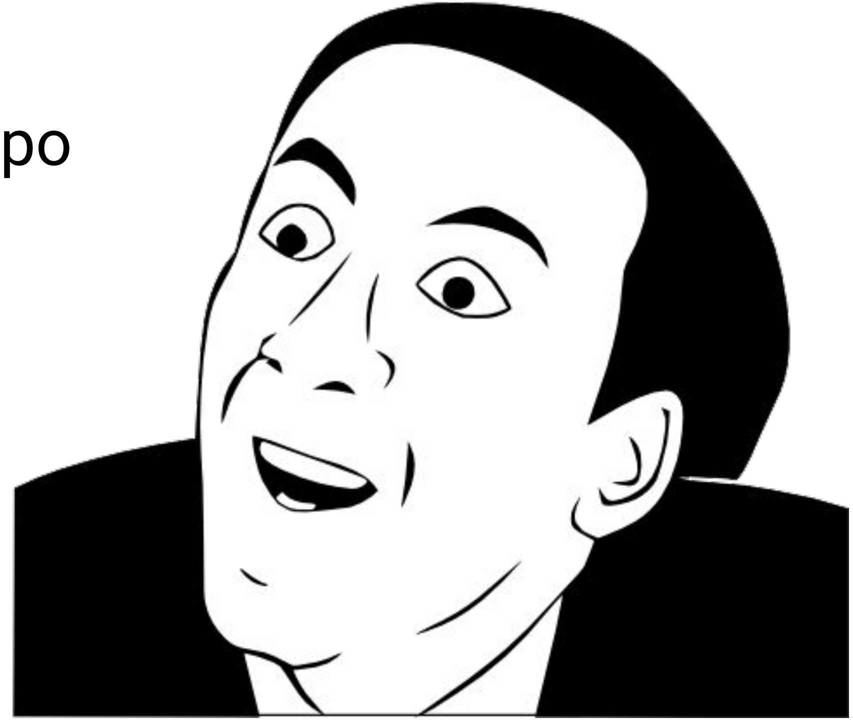
My working process



Repo & environment

```
$ sudo apt install cookiecutter
$ cookiecutter cookiecutter_repo

$ make initial_setup
```



What "$ make initial_setup" does

- Sets up git
- Creates a fresh virtualenv for a project
- Installs and sets up dvc
- Installs pre-commit
- Installs requirements.txt

Automate setup with Makefile 💡 😸

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

EDA

Get &
Understand data

Prepare Data

Model input generation

```
$   dvc run my_study/pipelines/split_data.dvc
    -d my_study/data/train_test_val_split.py
    -d my_study/data/train_test_val_split.ini
    -d data/raw/raw_data.tsv
    -o data/interim/dataset.pkl
    python my_study/data/train_test_val_split.py
    my_study/data/train_test_val_split.ini
    data/raw/raw_data.tsv  data/interim/dataset.pkl
```

Inside split_data.dvc

- deps
- outs
- cmd
- …

- A lot of repeats
- No deterministic order

Prepare Data

DVC tip #1

- Automate dvc-stage creation 💡 😺

f(stage_name, py_file, inps, outs, ...) -> “dvc_command”

f2(dvc_command) -> $ cd base_dir; dvc_command

As a result:

- Less typos
- Simpler maintenance

DVC's dirty secret

Prepare Data

- No recalculations
- Incremental development

Writes results of each stage on disk and calculates hash

<u>Problem:</u>

- Not suitable for production
  - Build manually?

DVC tip #2

- Use one featurize.py with different config files 💡 😺

DVC tip #2

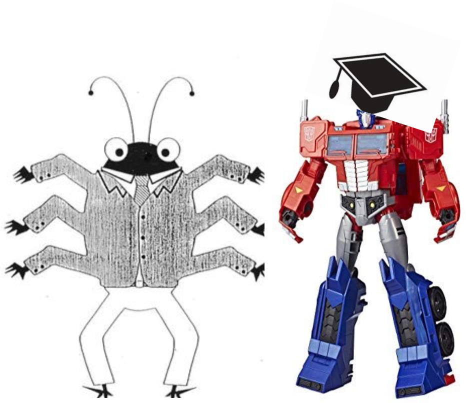
- Use one featurize.py with different config files 💡 😸

Transformer name&params

Input file name

Columns list

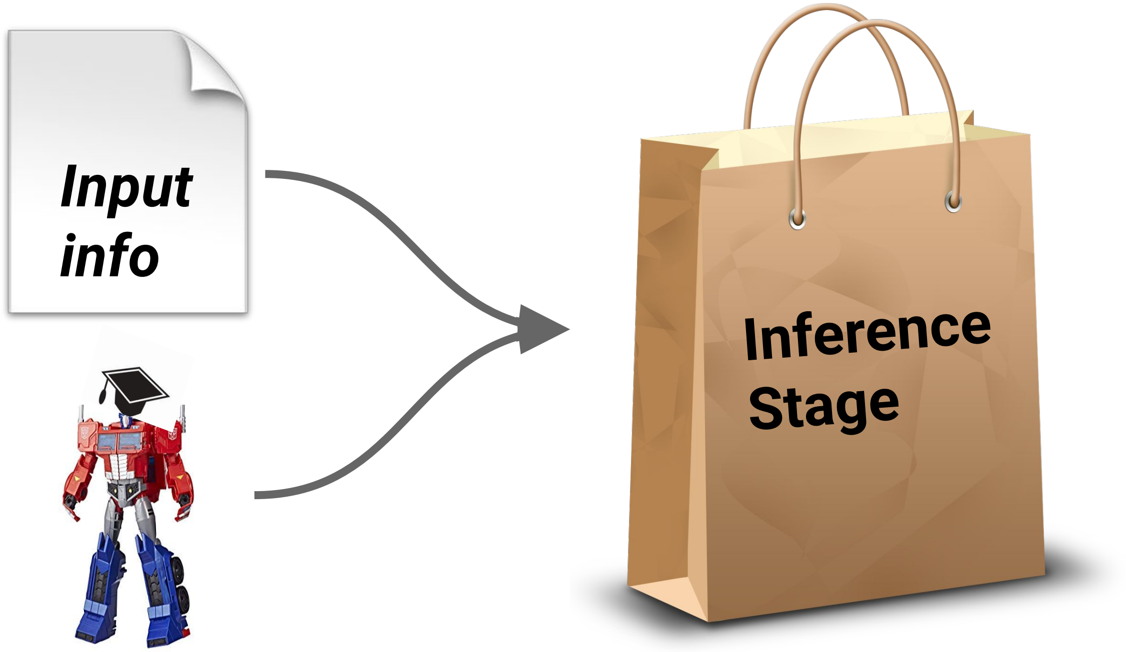
→ **featurize.py** →



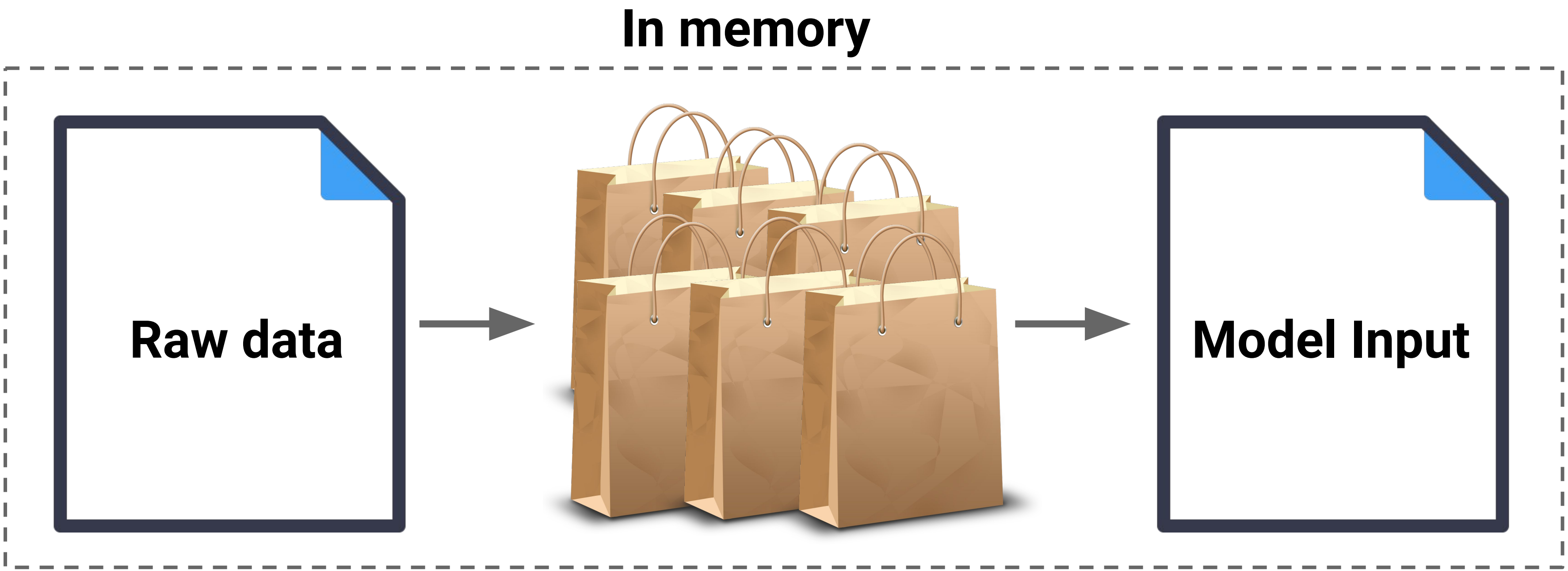
Prepare Data

DVC tip #3

- Create InferenceStage objects as an extra output 💡 😺



Rationale



Train Model

Train a family of models

- Study
  - Has a name
  - Has user-defined attributes

study_id	study name	best trial id	important notes
1	AAAA	252	Felt hungry. Ordered a pizza

Train Model

Situation #1

- You want to stop your machine after 50% of the training

Training tip #1

- Catch SigInt in your train.py 💡😺

```
try:
    study.optimize(objective, **study_kwargs)
except KeyboardInterrupt:
    logger.info("Catched Ctrl+C. Stopping...")
```

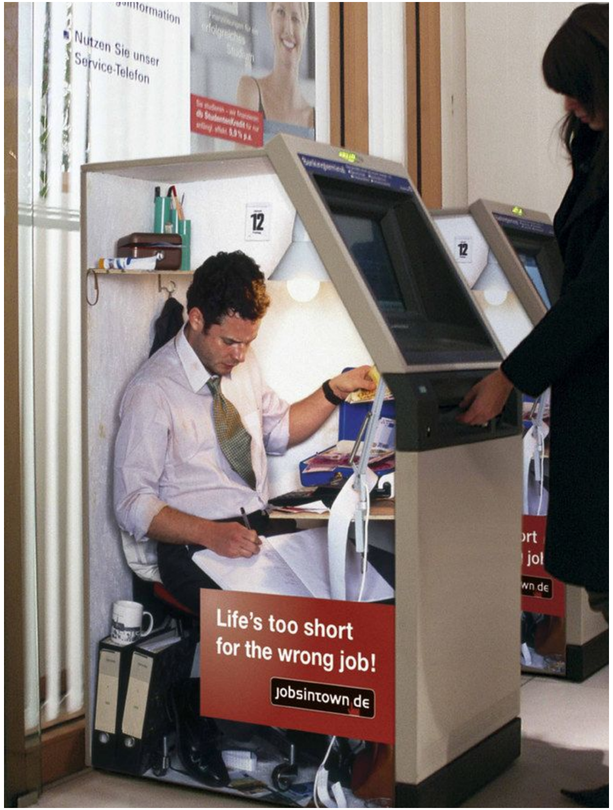
Train Model

Situation #2

- You want to resume a stopped training process

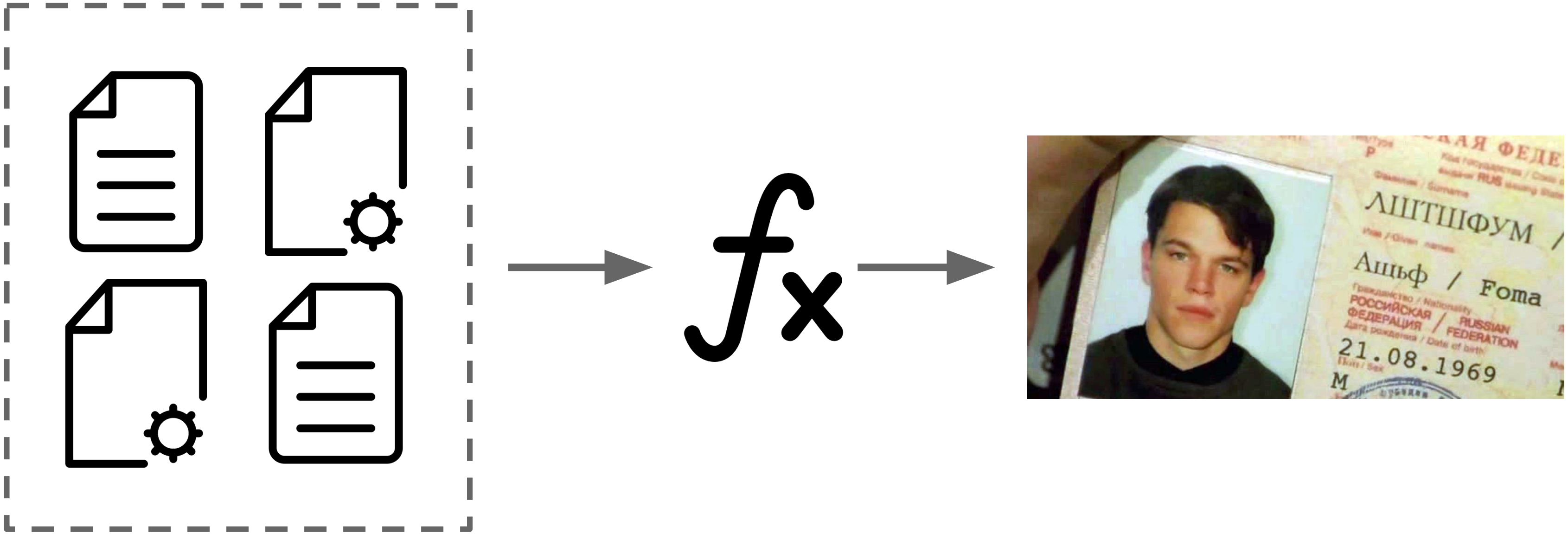
<u>Problem:</u>

- How to choose a study name?
  - Unique name for the each new experiment
  - Name in config?



Training tip #2

- Use a hash of all input files as an experiment name 💡 😺



Train Model

Situation #2.1

- You want to resume a stopped training process

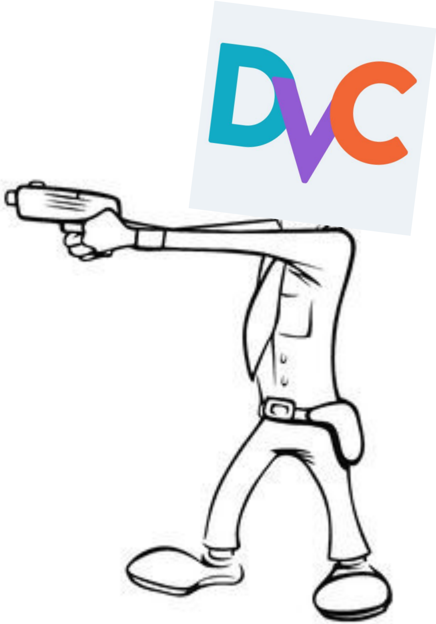
Problems:

- How to keep the best model?
  - DVC removes outputs before reproducing
  - Recalculate?

Train Model

Training tip #2.1

- Set **--outs-persist** as an output type for a model object 💡 😺



Train Model

Situation #3

- You want to checkout to the point of the most successful experiment

study name	**metric**
AAAA	0.99
BBBB	0.98

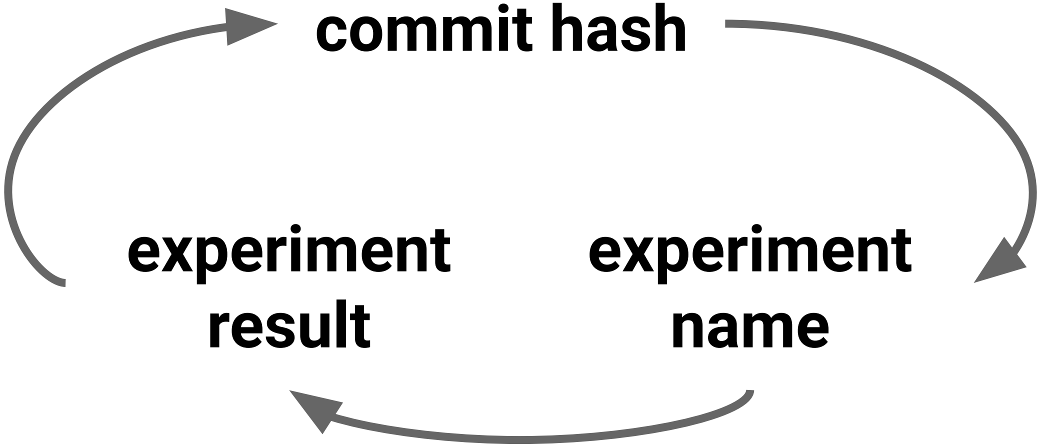
*$ git checkout **where to?***

$ dvc checkout

Train Model

Situation #3

- You want to checkout to the point of the most successful experiment



Training tip #3

- Tag completed experiments with a name of an experiment 💡 😺

  $ git tag -a exp_name -m “exp_name”

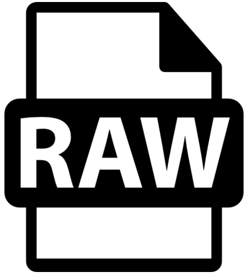
- <u>Bonus:</u>

  $ dvc gc --all-tags

Train Model

Situation #4

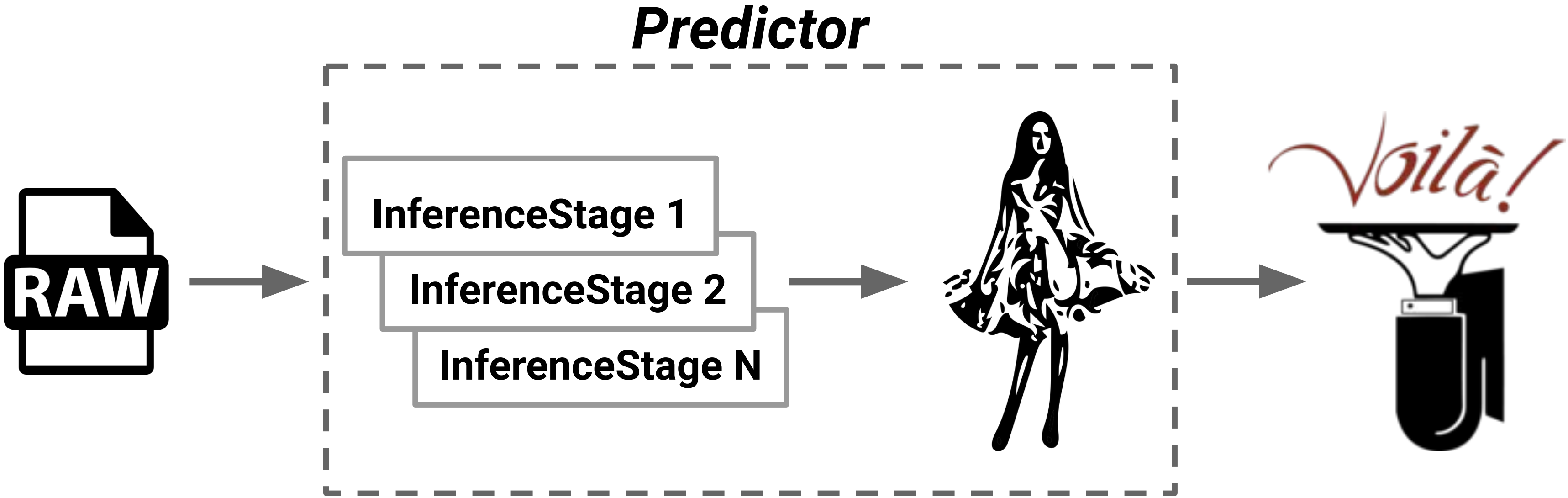
- You want to use the whole pipeline right after training



Train Model

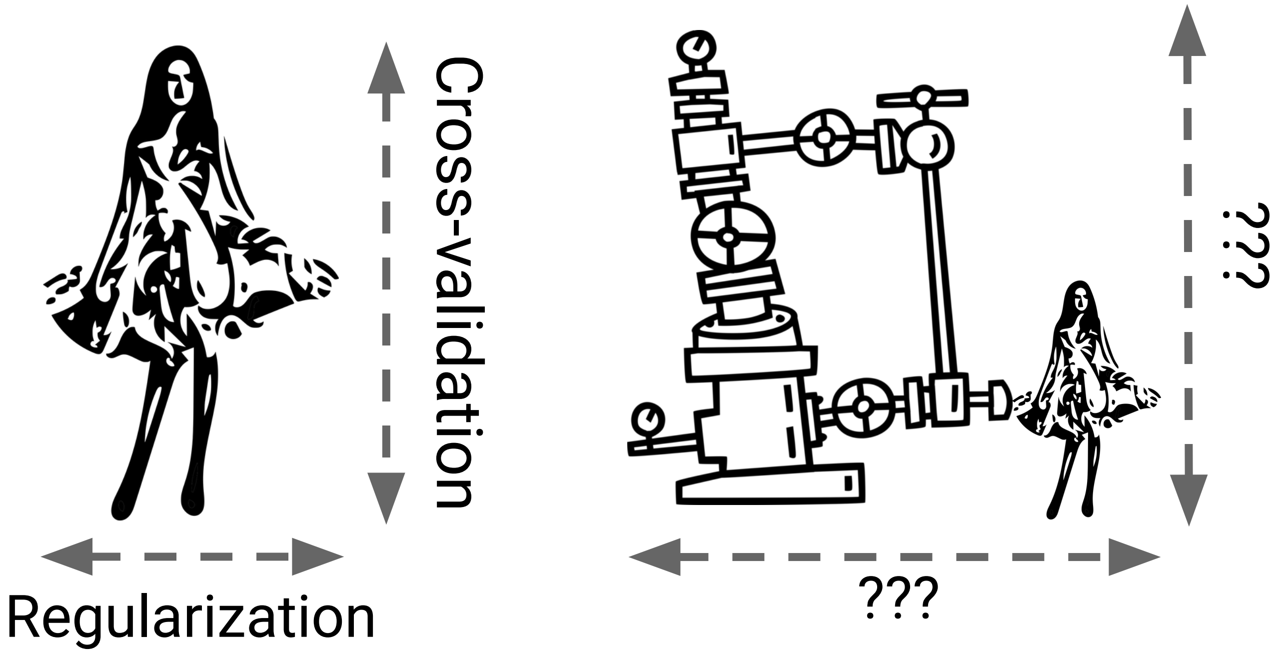
Training tip #4

- Create a predictor object using the best model and Inference stages 💡



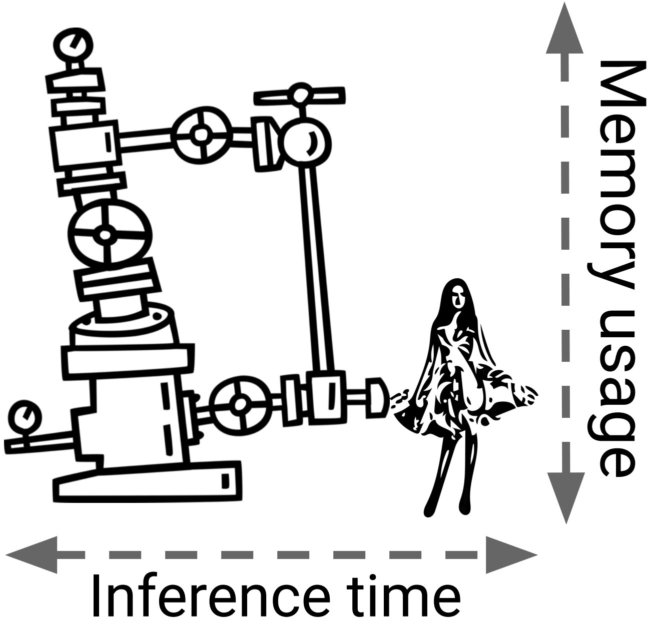
Situation #5

- You doubt how to choose the best pipeline



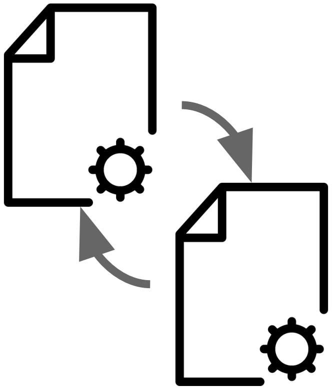
Training tip #5

- Measure inference time for a sample and store it as a study attribute 💡



Implement & Integrate

- Write a manager 💡



Share knowledge about this pipeline

Additional resources

- #ml_pipeline channel in ODS slack is a fountain of knowledge (ods.ai)
- Check out DVC promo video: you'll love it (dvc.org)
- Git LFS vs DVC (tiny.cc/git_lfs)
- "I don't like notebooks" - Joel Grus (youtube)

- Tweet from Shekhar Kirani :
  https://twitter.com/skirani/status/1149302828420067328
- Tweet from Andrew Ng :
  https://twitter.com/andrewyng/status/1080886439380869122

- My cookiecutter template:
  https://github.com/vasinkd/cookiecutter-data-science

semrush

Contacts

Kirill Vasin

vasin.kd@gmail.com

@vasinkd

@vasinkd (ODS Slack)