

Practices and tools for collaboration

in ML projects

Lesson 2

DVC Tools for Data Scientists &
Analysts

2021



Course lessons

Lesson 1. Course Introduction

Lesson 2. Practices and Tools for Efficient Collaboration in ML projects

Lesson 3. Pipelines Automation and Configuration Management

Lesson 4. Versioning Data and Models

Lesson 5. Visualize Metrics & Compare Experiments with DVC and Studio

Lesson 6. Experiment Management and Collaboration

Lesson 7. Tools for Deep Learning Scenarios

Lesson 8. Review Advanced Topics and Use Cases



Lesson Outline

- ◇ ML development workflow & collaboration
- ◇ Git
- ◇ Project repository structure & dev environment
- ◇ Coding (software development)
- ◇ Documentation & task tracking
- ◇ ML pipelines & experiments



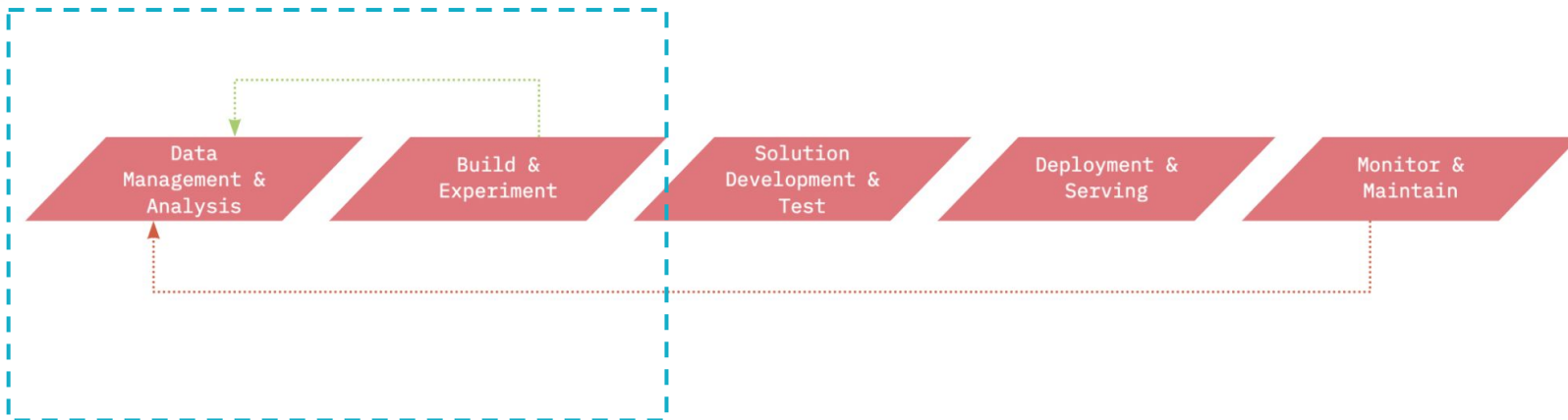


ML development workflow

Machine Learning Workflow



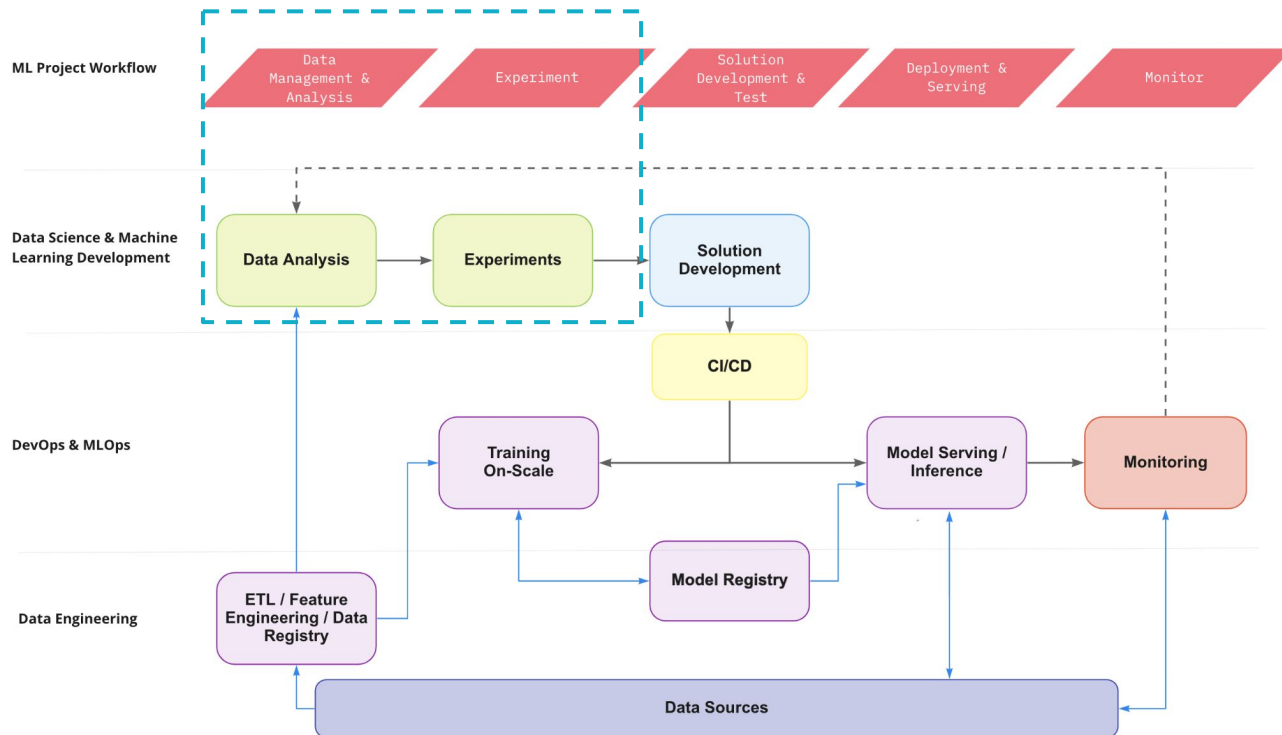
This course focuses on ML experiments and data management



Machine Learning Workflow



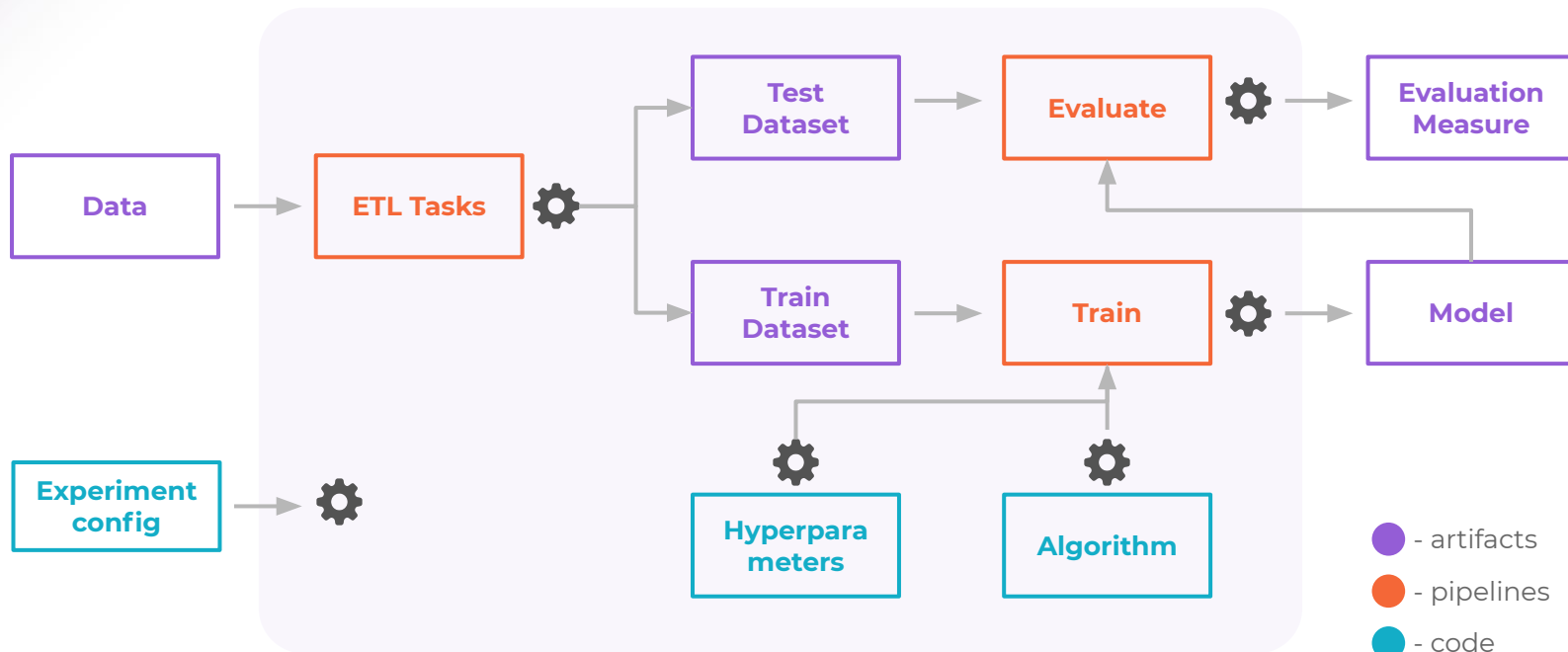
This course focuses on ML experiments and data management



ML Experiments



take long time and produce mess of metrics and artifacts



EXPERIMENT = CODE + DATASET + OUTPUTS



Collaboration in ML projects

Common DS/ML Challenges

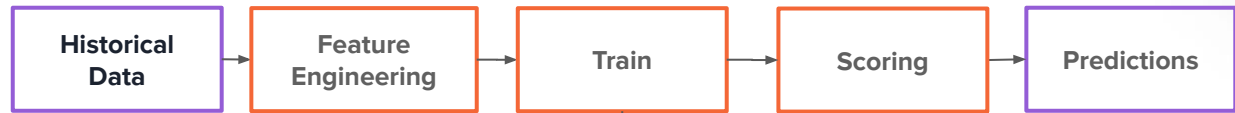
- ◇ Difficult sharing & collaboration
- ◇ Inefficiency & work duplication
- ◇ Slow updates
- ◇ Pipelines not reliable or not reproducible
- ◇ Data quality issues
- ◇ Model metrics tracking



Priority 1: Reproducibility



Research
Environment



Production
Environment

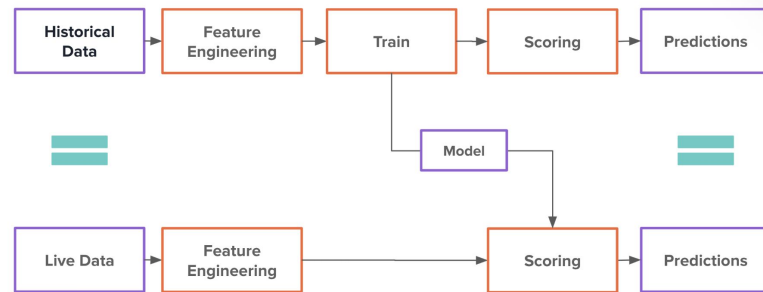


ML Reproducibility checklist



1. Environment dependency control
2. Code version control
3. Control run params
4. Automated pipelines
5. Artifact version control
6. Experiment results tracking
7. Automated CI/CD and MLOps

Priority 1: Reproducibility



Good practices for ML projects



1. Coding (Software Development)

- ◇ Clean Code
- ◇ Code version control (Git)
- ◇ Testing

2. Project structure & dev environment

- ◇ Organize a project repository
- ◇ Environment dependencies control

3. Documentation & task tracking

- ◇ Document your code, experiments and findings
- ◇ Task tracking

4. ML pipelines development & experiments

- ◇ Automated pipelines
- ◇ Control run params
- ◇ Model and artifact version control
- ◇ Experiment results tracking
- ◇ Reproducible experiments

Coding (software development)



Coding (Software Development)



- ◇ Organize code into clean reusable units (functions, classes, modules)
- ◇ Use Git for code version control
- ◇ Follow style-guides (i.t. PEP8 in case of Python)
 - a. Write comments, docstrings and type annotations
 - b. Give functions and variables meaningful names
- ◇ Make dependencies and requirements explicit
 - a. Add requirements.txt and Dockerfile to a project repository
- ◇ Testing

```
def headline(text: str, align: bool = True):  
    if align:  
        return f"{text.title()}\n{'-' * len(text)}"  
    else:  
        return f" {text.title()} ".center(50, "o")  
  
print(headline("python type checking"))  
print(headline("use pycharm", "center"))
```

Expected type 'bool', got 'str' instead more... (Ctrl+F1)



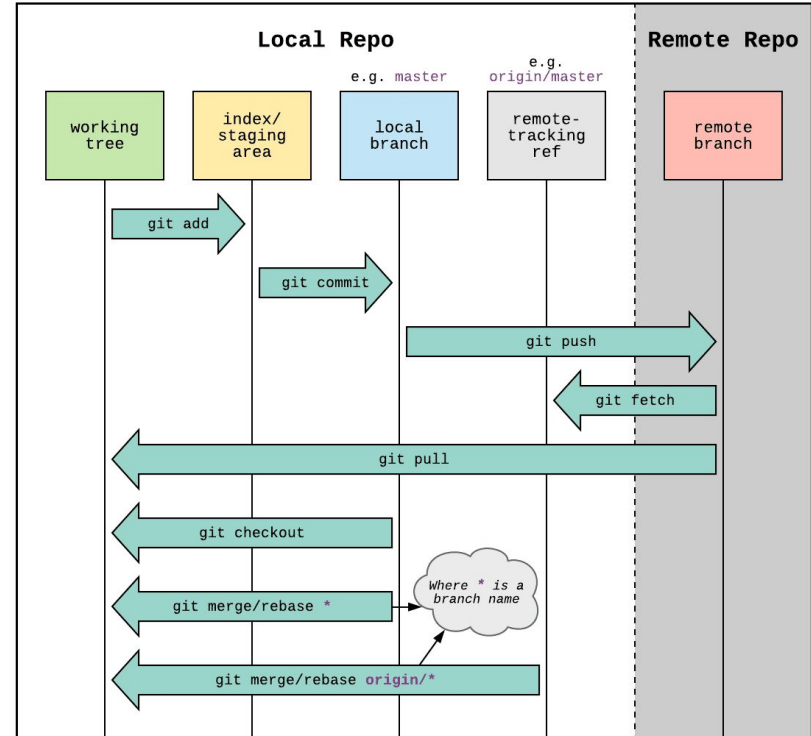


Git basics for machine learning development

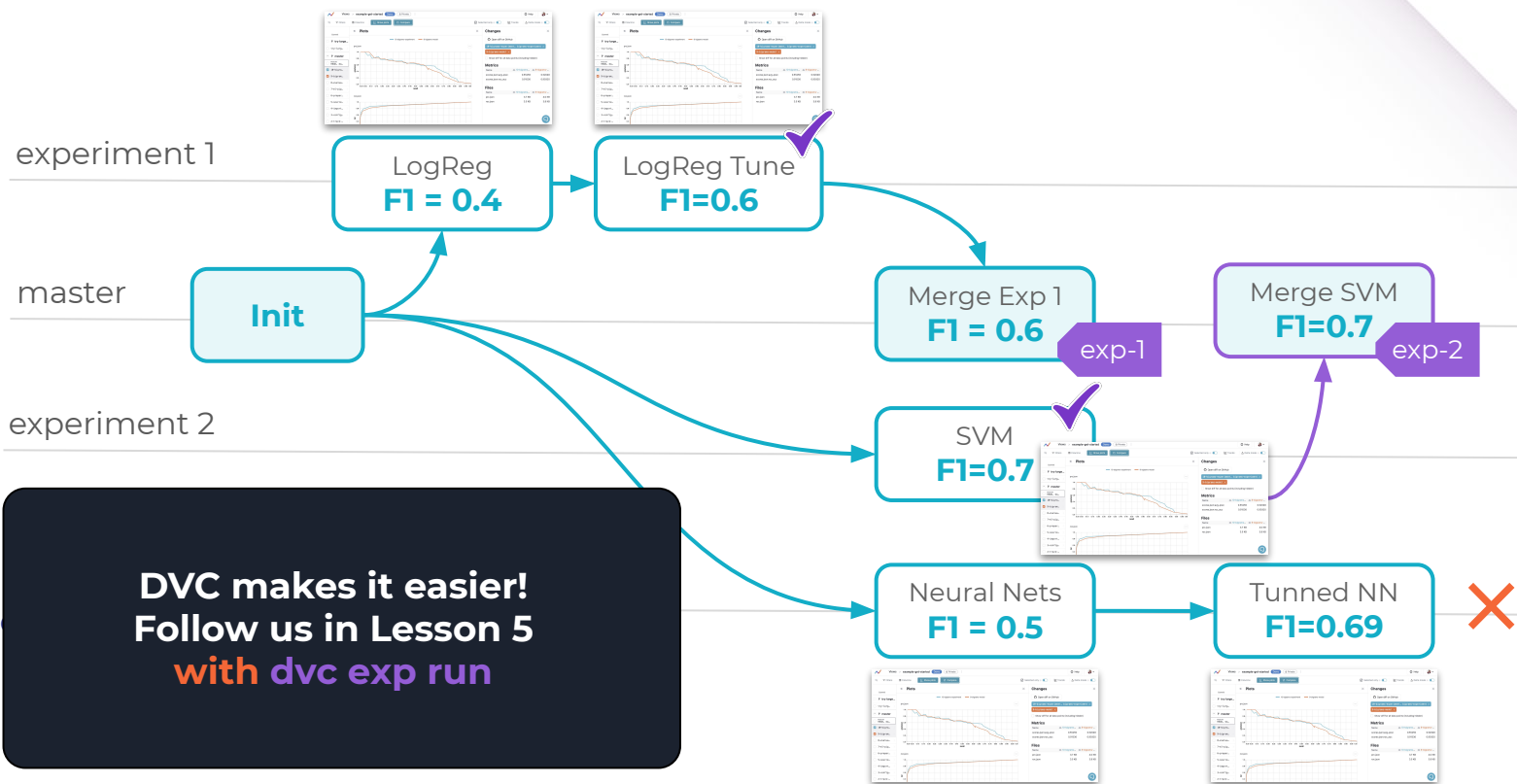
Git workflow



- ◇ git add
- ◇ git commit
- ◇ git push
- ◇ git fetch
- ◇ git pull
- ◇ git checkout
- ◇ git merge / rebase



Apply Git based workflows to ML development





Live code example

Git basics



Project repository structure

Cookiecutter DS Project structure

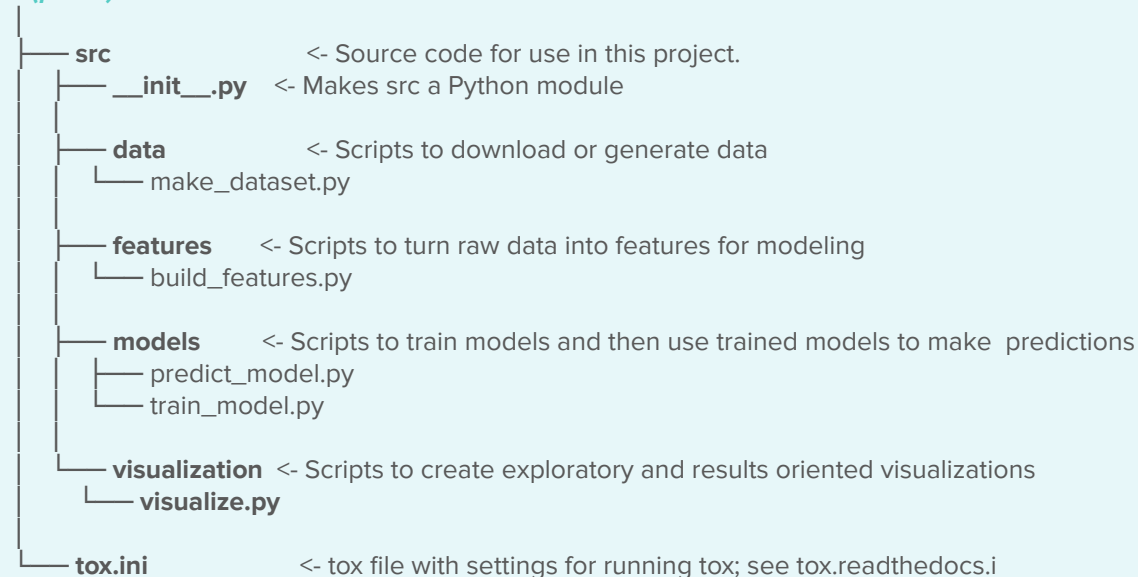


...	
— README.md	<- The top-level README for developers using this project.
— data	
— external	<- Data from third party sources.
— interim	<- Intermediate data that has been transformed.
— processed	<- The final, canonical data sets for modeling.
— raw	<- The original, immutable data dump.
— docs	<- A default Sphinx project (sphinx-doc.org) (optional : Text documents associated with the project)
— 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. <code>`1.0-jqp-initial-data-exploration`</code> .
...	
— 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 <code>`pip freeze > requirements.txt`</code>
... (part 2)	

Cookiecutter DS Project structure



...(part 1)



Custom `template_repo`



```
— README.md
— config/
— data/
— models/
— notebooks/
— reports/
— src/
  — data/      <- data prepare and/or preprocess
  — evaluate/  <- code for model quality evaluation and metrics
  — features/  <- code to compute features
  — stages/    <- DVC stages code
  — report/    <- code for visualization and plots
  — train/     <- code for training and hyper-parameters tuning
```

Custom structure

- simple
- flexible
- easy to share & collaborate



Live code example

Project repository structure

Python Virtual Environments



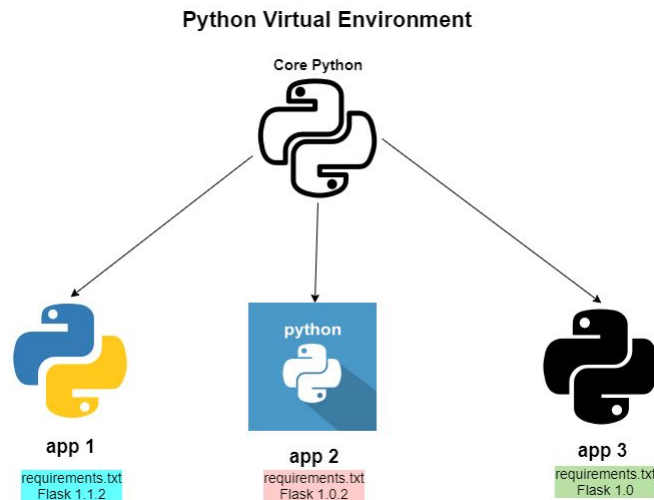
Virtual Environments



- ◇ a simple solution for reproduce development environment
- ◇ isolates the project-related libraries
- ◇ controls Python version

Example packages:

- ◇ venv & virtualenv
- ◇ conda
- ◇ pipenv
- ◇ poetry
- ◇ ...



Virtual Environments with **venv**



create virtual environment

python -m venv dvc-venv

activate a virtual environment

source dvc-venv /bin/activate

exit the virtual environment

deactivate

- ◇ **venv** a subset of **virtualenv** project, integrated into the standard library
- ◇ *Changed in version 3.5:* The use of **venv** is now recommended for creating virtual environments

Specify Python dependencies: **requirements.txt**



- ◇ Environment documentation
 - a. Add requirements.txt to the project repository
- ◇ Install dependencies from **requirements.txt**
 - # create virtual environment
 - pip install -r requirements.txt**



Live code example

Python Virtual Environments

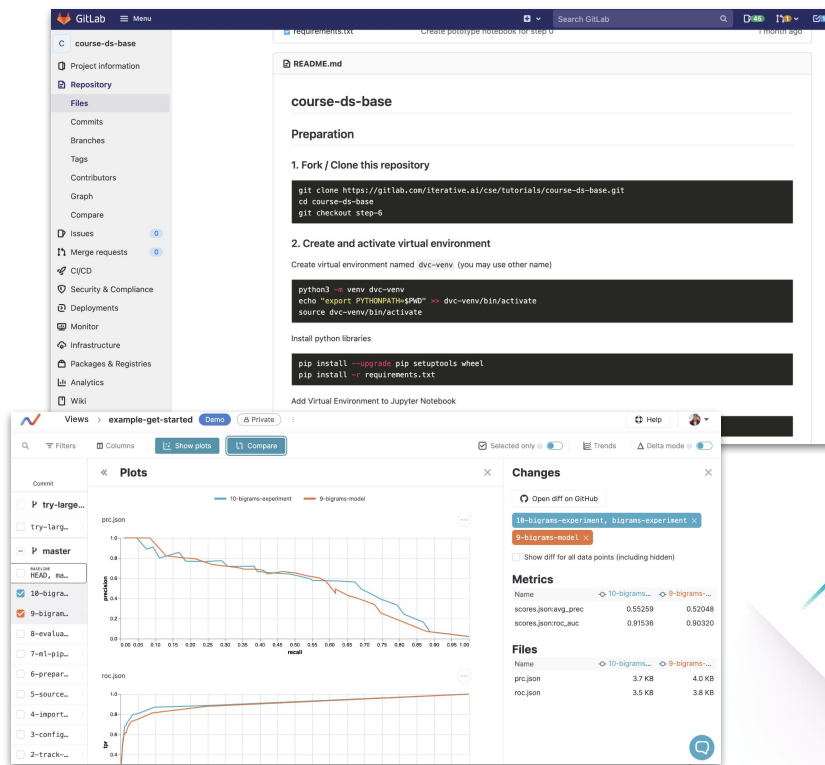
Documentation & task tracking



Good practices: Documentation



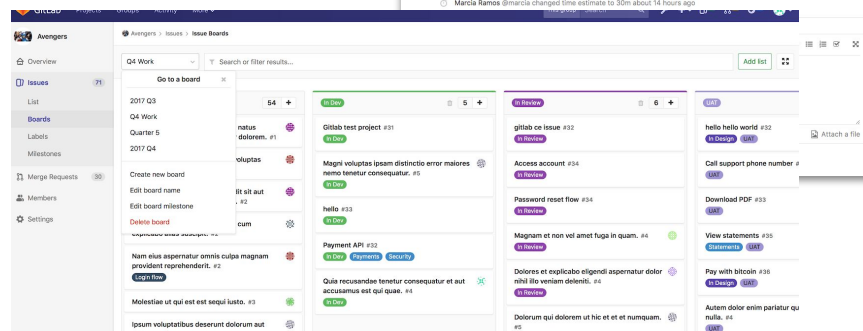
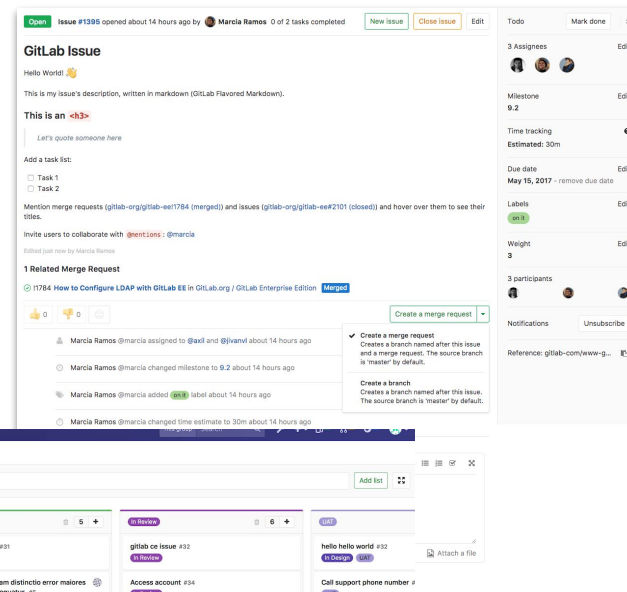
- ◇ Project repository documentation
 - a. README
 - info about the project
 - how to install and run instructions
 - contacts and author(s) details
 - b. docs/
 - c. License
- ◇ Project documentation (problem statement, methods, data, findings)
- ◇ Experiment metrics and reports



Good practices: Task Tracking



- ◇ Create a shared "to-do" list (task tracking)
- ◇ Keep changes small
- ◇ Share changes frequently
- ◇ Create tasks (issues) for each changes in task tracking systems (i.e. GitLab/GitHub/Bic)
- ◇ Link tasks to Git branches





Live code example

Documentation & task tracking



What have we learned?

What have we learned?



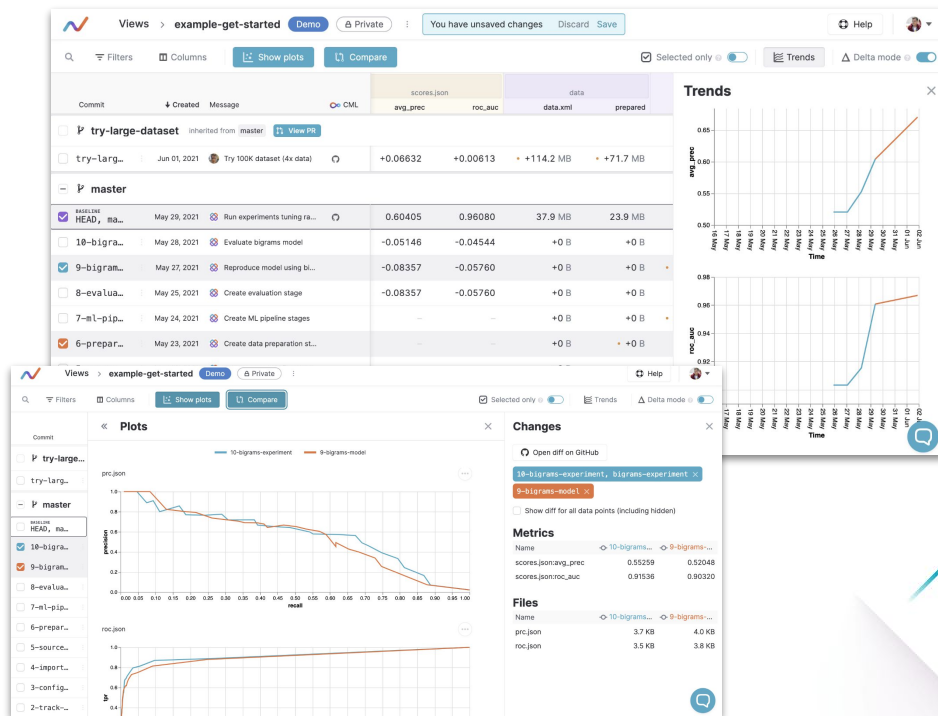
1. Requirements for successful collaboration
2. How to structure your repository
3. Good practices for coding and collaboration
4. Good practices for documentation and task tracking

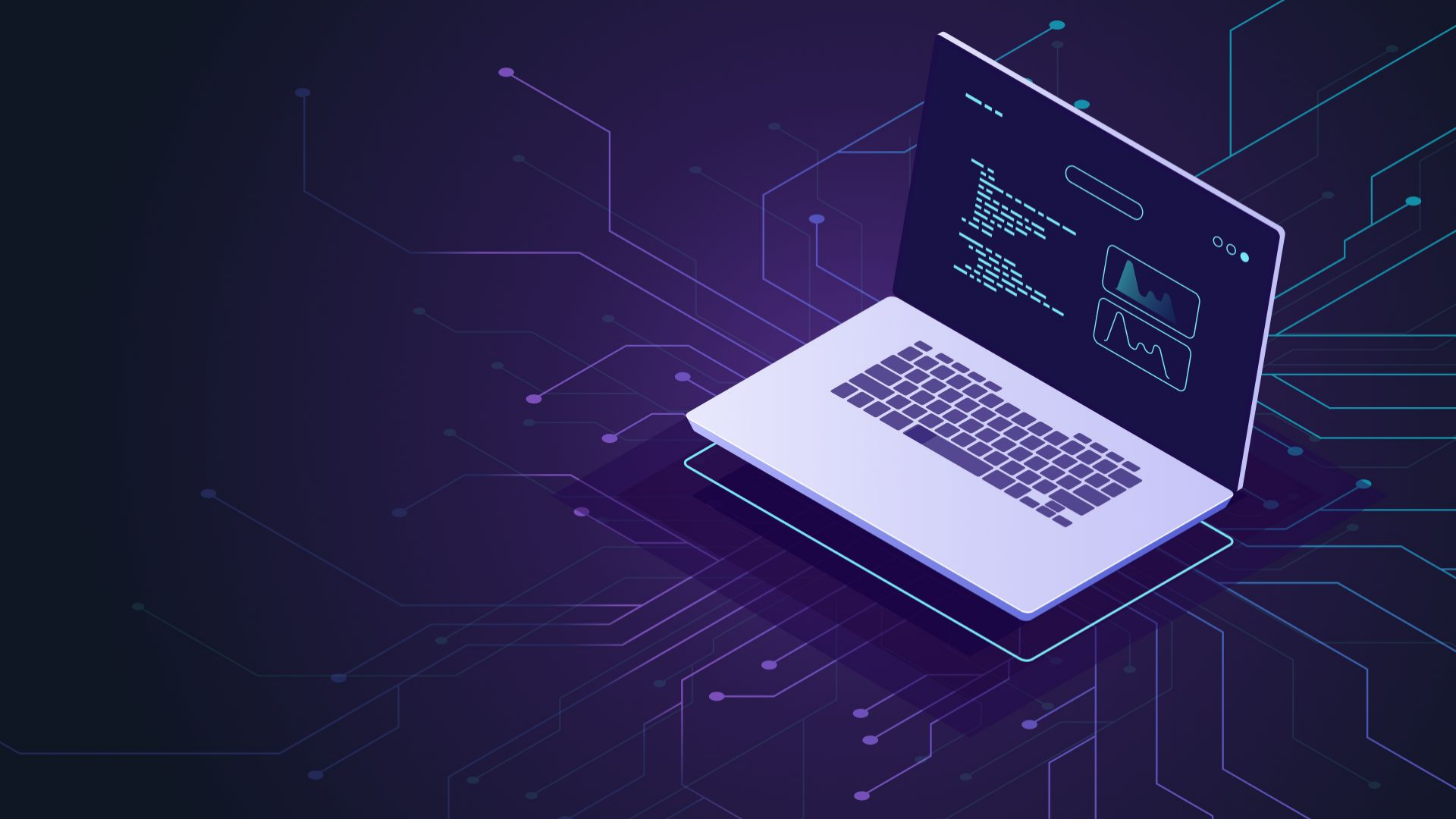
Good practices: ML pipelines & experiments



- ◇ Automated pipelines
- ◇ Control run params
- ◇ Models and artifacts version control
- ◇ Experiments results tracking
- ◇ Reproducible experiments

Follow the next lessons...







Links



Data Science blueprint

<https://data-science-blueprint.readthedocs.io/en/latest/presentation/schema.html>