

Pipelines

Automation and Configuration

Lesson 3

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. Experiments Management and Collaboration

Lesson 7. Tools for Deep Learning Scenarios

Lesson 8. Review Advanced Topics and Use Cases



Lesson Outline

- ◆ Why automate ML pipelines?
- ◆ Requirements for pipeline automation
- ◆ Build automated pipelines: step by step guide
- ◆ Reproduce end-to-end ML pipelines



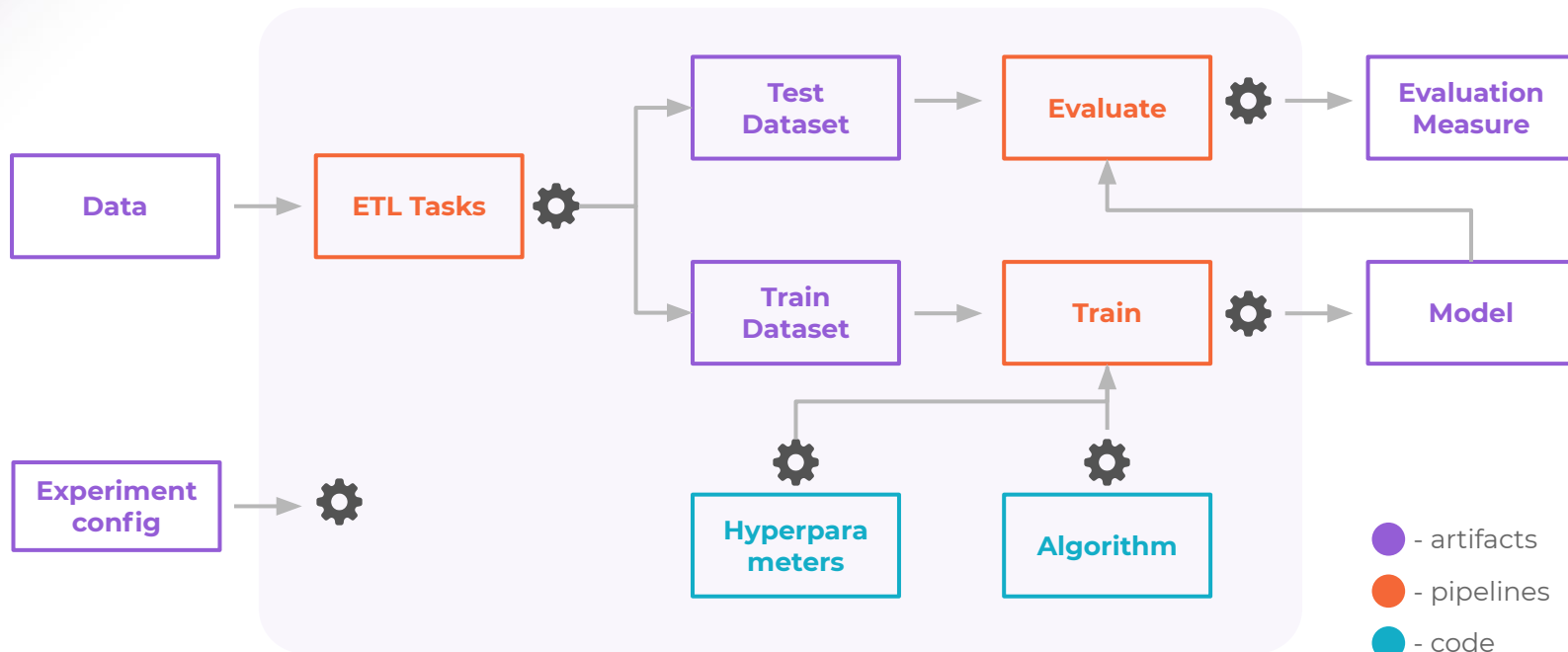


Why automate ML pipelines?

ML Experiments



take long time and produce mess of metrics and artifacts

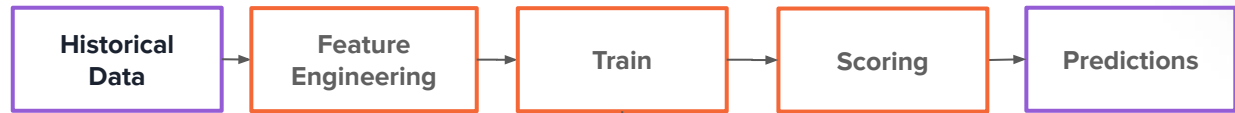


EXPERIMENT = CODE + DATASET + OUTPUTS

Priority 1: Reproducibility



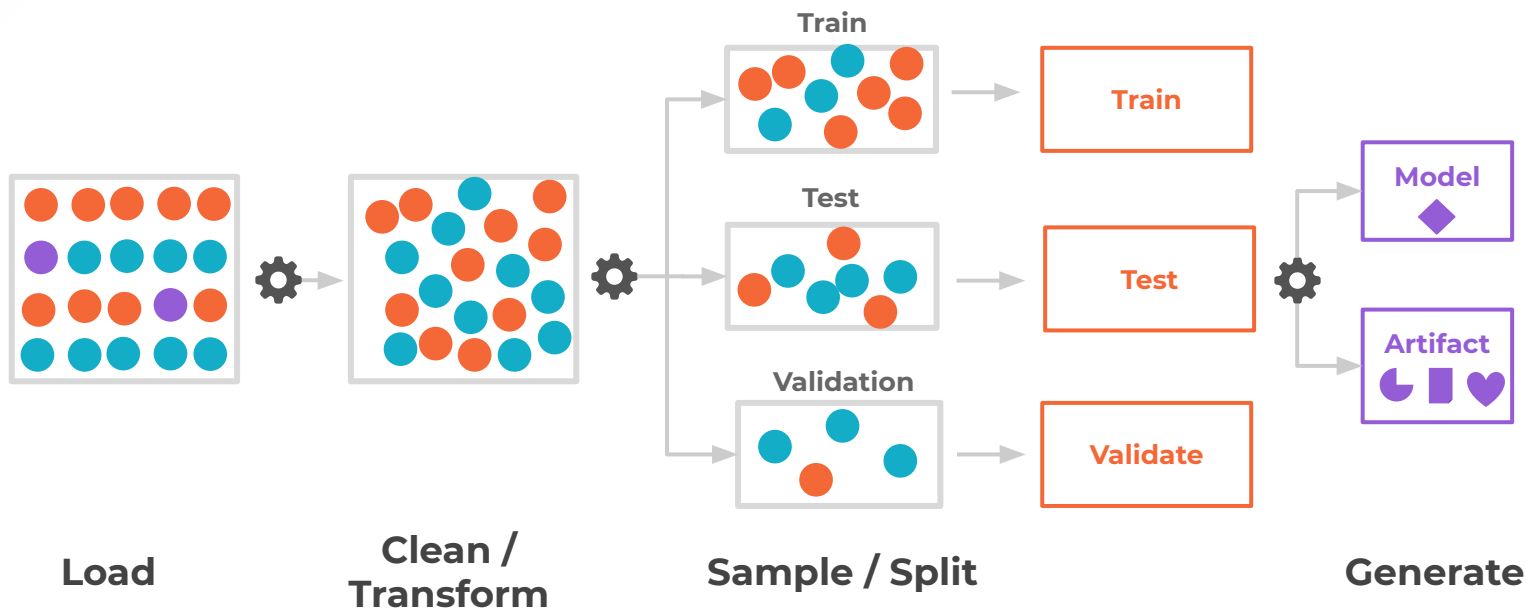
Research
Environment



Production
Environment



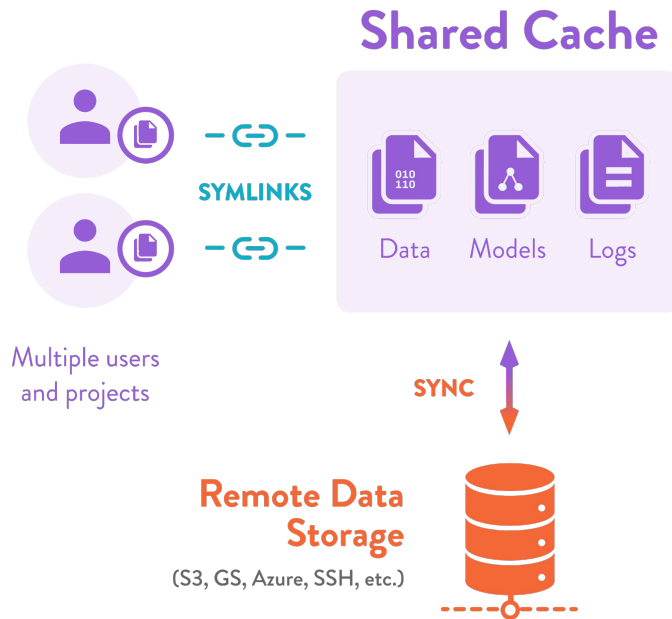
Optimized & deterministic ETL



Improve team collaboration



- ◇ Code versioning
- ◇ Experiment params control
- ◇ Data & artifacts control





Requirements for pipeline automation

From Jupyter Notebooks To .py



Jupyter Notebook

- ◇ interactive environment
- ◇ can run .py / bash scripts
- ◇ great for visualization

.py scripts

- ◇ versioning
- ◇ reproducible pipelines
- ◇ easy to automate

Manage configs

- ◇ no `hardcoded` paths & params
- ◇ move params to config file
 - ~ **params.yaml**



Reusable code



- ◇ clear pipeline workflow and stages
 - inputs
 - jobs
 - outputs
- ◇ organize code
- ◇ clean code
- ◇ reusable code



Build automated pipelines: **step by step guide**

Step 1: Organize repo & build a prototype



Guide

- ◇ Organize your project folder structure
- ◇ Use virtual environment or Docker
- ◇ Use tools and libraries you are comfortable with
- ◇ Keep a linear workflow to run cells in Jupyter Notebook
- ◇ Use table of content (TOC) widget and group cells

Tools

- ◇ Jupyter Notebooks



Live code example

Step 1:

**Build a Prototype with Jupyter
Notebook**

Step 2: Create a single configuration file



Guide

- ◇ Avoid hardcoded paths & params from code
- ◇ Move all params to a single config file: `params.yaml`
- ◇ Group parameters (e.g. data, train, test...)

Tools

- ◇ Jupyter Notebooks
- ◇ YAML

params.yaml: structure example



params.yaml

```
data_load:  
  dataset_csv: 'data/raw/iris.csv'
```

```
data_split:  
  test_size: 0.2  
  trainset_path: 'data/processed/train_iris.csv'  
  testset_path: 'data/processed/test_iris.csv'
```

```
train:  
  cv: 3  
  estimator_name: logreg      # sklearn.linear_model.LogisticRegression  
  C: 0.001  
  max_iter: 100  
  solver: 'lbfgs'  
  multi_class: 'multinomial'  
  model_path: models/model.joblib
```

**Set and save
configuration**

params.yaml: usage example



data_load.py

```
import yaml
```

```
def data_load(config_path: Text) -> None:
```

```
    config = yaml.safe_load(open(config_path))  
    raw_data_path = config['data_load']['raw_data_path']
```

```
    ...
```



Load and use
configuration



Live code example

Step 2:

Create a single configuration file

Step 3: Move reusable code to **.py** modules



Guide

- ◇ Move functions and classes into **.py** modules
 - import in Jupyter Notebooks
- ◇ Add **.py** modules to Git version control
- ◇ Organize code repo

Tools

- ◇ Jupyter Notebooks
- ◇ YAML
- ◇ Python scripts (.py)
- ◇ Git

Organize code repo (example)



- 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
- **report/** <- code for visualization and plots
- **train/** <- code for training and hyper-parameters tuning

Move **.py** code into
src/ directory

Organize code repo (example)



```
— README.md
— config/
— data/
— models/
— notebooks/
— reports/
— src/
  — data/
  — evaluate/
  — features/
  — report/
  — train/
```

**Add all Jupyter
Notebooks to a
separate folder**



Live code example

Step 3:

Move reusable code to **.py**

Step 4: Build ML experiment pipeline



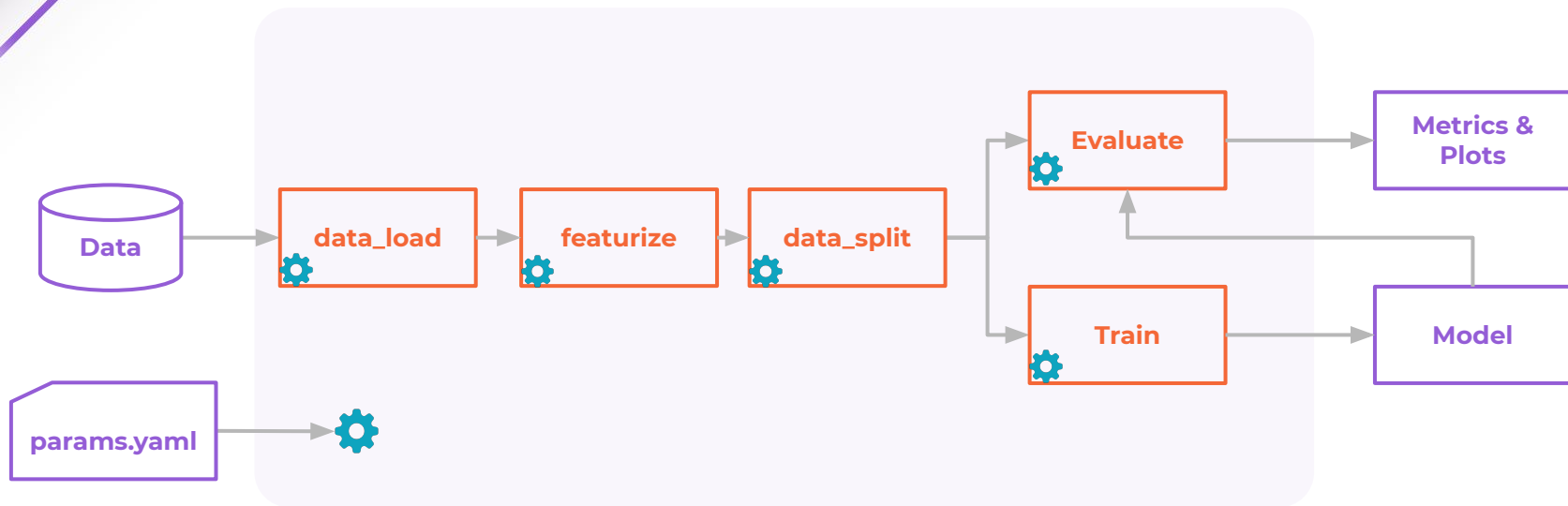
Guide

- ◇ Create .py module for each computation task (stage)
- ◇ Use **params.yaml** to manage configuration
- ◇ Structure **.py** modules to run in both mode
 - by importing in Jupyter Notebook
 - by running in terminal

Tools

- ◇ Jupyter Notebooks
- ◇ YAML
- ◇ Python scripts (.py)
- ◇ Git
- ◇ CLI

Example: Simple ML pipeline



EXPERIMENT = CODE + DATASET + OUTPUTS

- - artifacts & data
- - pipeline
- ⚙ - configs

Organize code repo (example)



- README.md
- config/
- data/
- models/
- notebooks/
- reports/
- src/
 - data/
 - evaluate/
 - features/
 - **stages/**
 - report/
 - train/

Add code for stages
got **stages/** folder

Make it simple to run & test with



data_load.py

```
import yaml
```

```
def data_load(config_path: Text) -> None:
```

```
    config = yaml.safe_load(open(config_path))
    raw_data_path = config['data_load']['raw_data_path']
    ...
    data.to_csv(config['dataset_processed_path'])
```

```
if __name__ == '__main__':
```

```
    args_parser = argparse.ArgumentParser()
    args_parser.add_argument('--config', dest='config', required=True)
    args = args_parser.parse_args()
```

```
    data_load(config_path=args.config)
```

**Import in Jupyter
Notebook or run via
CLI**

Run **data_load** stage in CLI



run from the project root folder

```
python src/stages/data_load.py --config=params.yaml
```



Live code example

Step 4:

Build ML experiment pipeline

Step 5: Automate pipelines with DVC



Guide

- ◇ Install DVC
- ◇ Organize stages into a DVC pipeline
- ◇ Setup dependencies and outputs
- ◇ Setup parameters

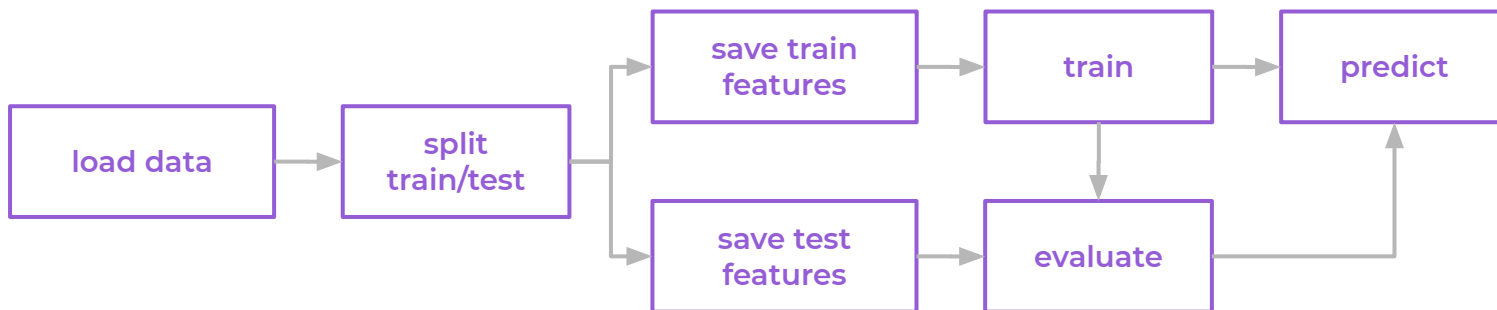
Tools

- ◇ YAML
- ◇ Python scripts (.py)
- ◇ Git
- ◇ DVC

Pipeline is a sequence of stages



- ◇ Each stage is a computation task that may produce data for a next stages
- ◇ DVC builds a dependency graph (DAG) of stages (pipeline)
- ◇ You decide what dependencies and outputs to control with DVC
- ◇ Reproduce stage or pipelines with **dvc repro**



Install DVC



Install with pip

```
$ pip install dvc
```

Install with support for Amazon S3 storage

```
$ pip install "dvc[s3]" # [all] to support all remote storages
```

Install with conda

```
$ conda install -c conda-forge dvc
```

Installation docs link: <https://dvc.org/doc/install>

Initialize DVC



run command

```
$ dvc init
```

output

```
Adding '.dvc/state' to '.dvc/.gitignore'.
Adding '.dvc/lock' to '.dvc/.gitignore'.
Adding '.dvc/config.local' to '.dvc/.gitignore'.
Adding '.dvc/updater' to '.dvc/.gitignore'.
Adding '.dvc/updater.lock' to '.dvc/.gitignore'.
Adding '.dvc/state-journal' to '.dvc/.gitignore'.
Adding '.dvc/state-wal' to '.dvc/.gitignore'.
Adding '.dvc/cache' to '.dvc/.gitignore'.
```

You can now commit the changes to Git.

**We will learn
more about
Internal files
and
directories in
during the
Course**

Commit changes



run command

```
$ git add .  
$ git commit -m "Initialize DVC"
```

output

```
Initialize DVC  
2 files changed, 8 insertions(+)  
create mode 100644 .dvc/.gitignore  
create mode 100644 .dvc/config
```

Add pipeline stages with **dvc run**



-n specifies name
for DVC stage

```
dvc run -n data_load \
```

```
[ python src/data_load.py --config=params.yaml
```

command to run
for the stage

Notes

- ◇ `\` symbol at the end of each line helps to continue command at the next line

Add pipeline stages with **dvc run**



Specify
-d dependencies
-o outputs

```
dvc run -n data_load \
```

```
-d src/data_load.py \
```

```
-o data/iris.csv \
```

```
-p data_load \
```

```
python src/data_load.py --config=params.yaml
```

-p specifies stage
parameters

Add pipeline stages with **dvc run**



Specify

-d dependencies

-o outputs

-n specifies name
for DVC stage

```
dvc run -n data_load \
```

```
-d src/data_load.py \
```

```
-o data/iris.csv \
```

```
-p data_load \
```

```
python src/data_load.py --config=params.yaml
```

command to run
for the stage

-p specifies stage
parameters

Add pipeline stages with **dvc run**



run command

```
$ dvc run -n data_load \  
-d src/data_load.py \  
-o data/iris.csv \  
-p base,data_load \  
python src/data_load.py --config=params.yaml
```

output

Running stage 'data_load' with command:

```
python src/data_load.py --config=params.yaml
```

Creating 'dvc.yaml'

Adding stage 'data_load' in 'dvc.yaml'

Generating lock file 'dvc.lock'

To track the changes with git, run:

```
git add dvc.yaml dvc.lock
```

Create **dvc.yaml** file

dvc.yaml file content

stages:

data_load:

cmd: python src/stages/data_load.py --config=params.yaml

deps:

- src/stages/data_load.py

params:

- base

- data_load

outs:

- data/raw/iris.csv



Stage name

dvc.yaml file content

stages:

data_load:

cmd: python src/stages/data_load.py --config=params.yaml

deps:

- src/stages/data_load.py

params:

- base

- data_load

outs:

- data/raw/iris.csv

**A command to run this
stage**

dvc.yaml file content

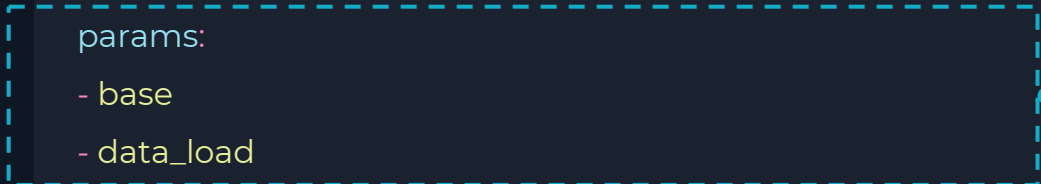
```
stages:  
  data_load:  
    cmd: python src/stages/data_load.py --config=params.yaml  
    deps:  
      - src/stages/data_load.py  
    params:  
      - base  
      - data_load  
    outs:  
      - data/raw/iris.csv
```

List of dependencies

dvc.yaml file content

```
stages:  
  data_load:  
    cmd: python src/stages/data_load.py --config=params.yaml  
    deps:  
      - src/stages/data_load.py  
    params:  
      - base  
      - data_load  
    outs:  
      - data/raw/iris.csv
```

Parameters from
params.yaml
(by default)



dvc.yaml file content

```
stages:  
  data_load:  
    cmd: python src/stages/data_load.py --config=params.yaml  
    deps:  
      - src/stages/data_load.py  
    params:  
      - base  
      - data_load  
    outs:  
      - data/raw/iris.csv
```

Stage outputs



Manual update **dvc.yaml**

```
stages:  
  data_load:  
    cmd: python src/stages/data_load.py --config=params.yaml  
    deps:  
      - src/stages/data_load.py  
    params:  
      - base  
      - data_load  
    outs:  
      - data/raw/iris.csv
```

```
  featurize:  
    cmd: python src/stages/featurize.py --config=params.yaml  
    deps:  
      - data/raw/iris.csv  
      - src/stages/featurize.py  
    params:  
      - base  
      - data_load  
      - featurize  
    outs:  
      - data/processed/featured_iris.csv
```

**Add a stage
configuration manually**

A light blue rounded rectangular callout box with the text "Add a stage configuration manually" in bold. A line extends from the bottom of the box, turning left to point at the 'featurize' stage in the code block below. The 'featurize' stage is enclosed in a dashed light blue border.



Live code example

Step 5:
Automate pipelines with DVC
Build end-to-end pipeline



Reproduce end-to-end ML pipelines

Reproduce **end-to-end** ML pipelines



Guide

- ◇ Update code & params.yaml
- ◇ Run **dvc repro** / **dvc exp run**
- ◇ Commit changes to Git
- ◇ Store artifacts with DVC

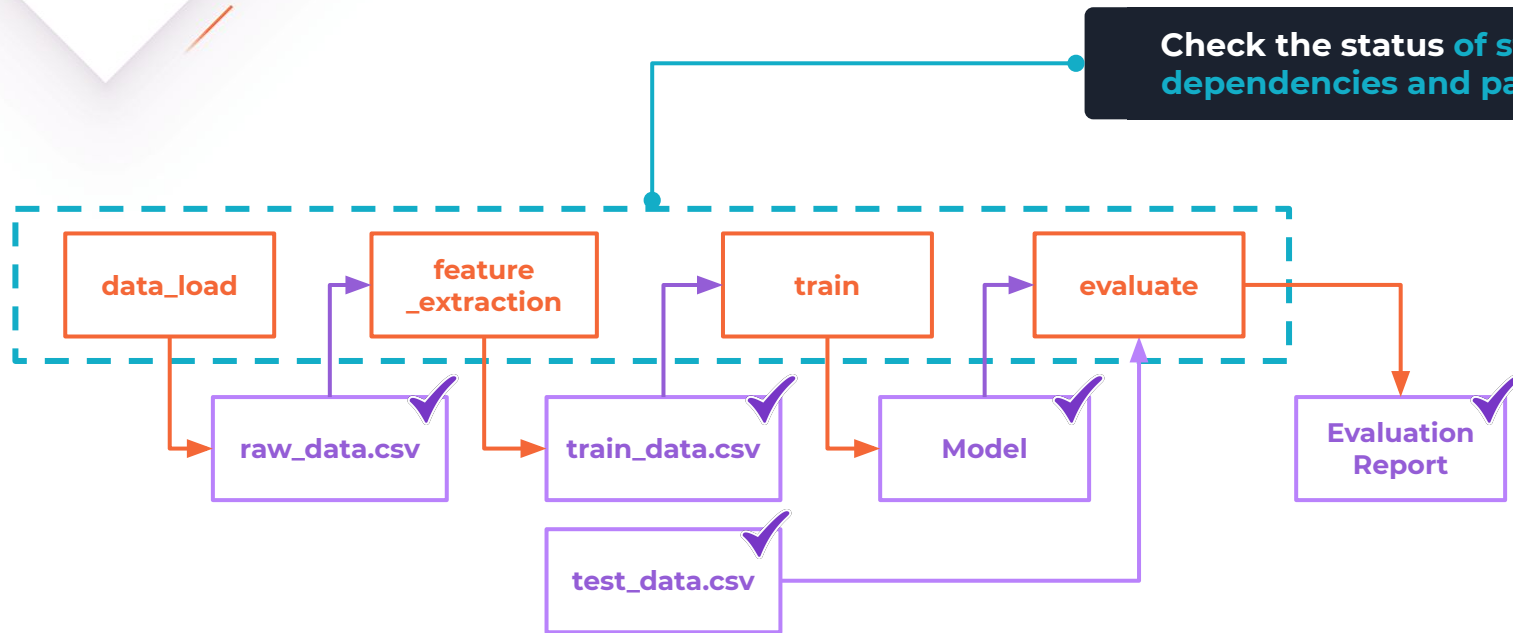
Tools

- ◇ DVC
- ◇ Git

How **dvc repro** works?



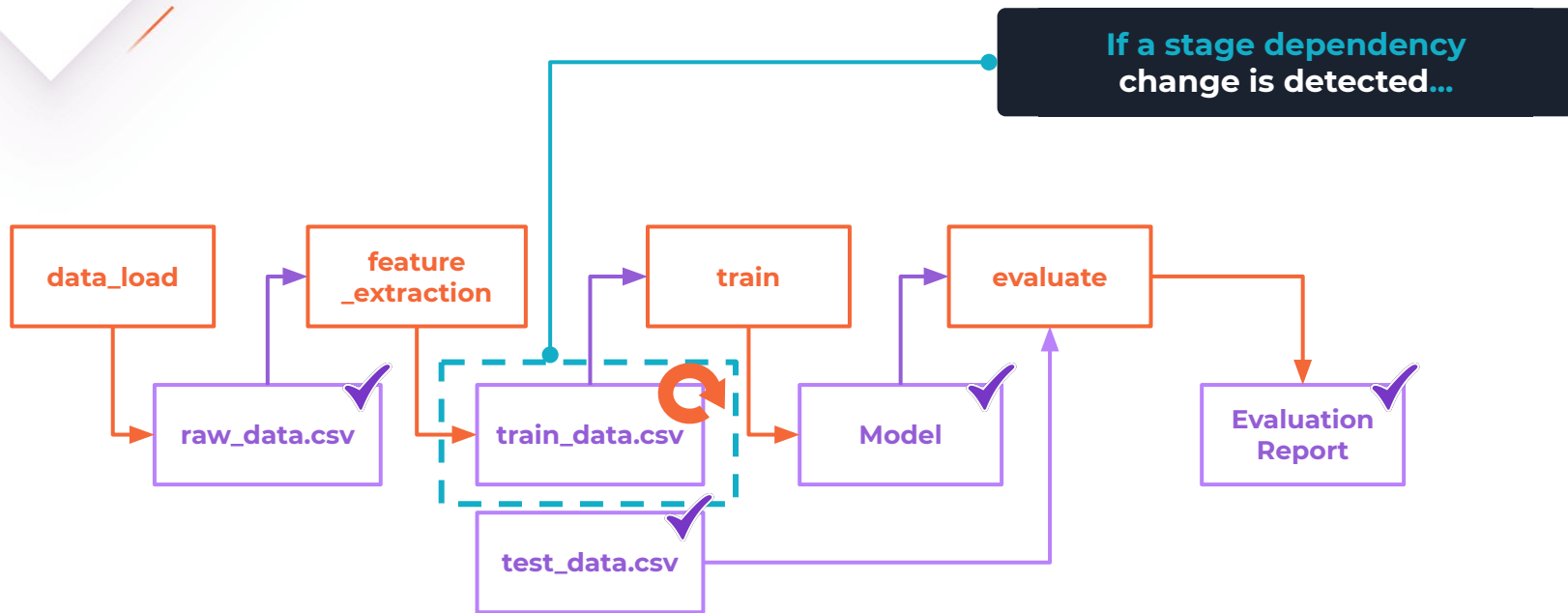
Check the status of stages dependencies and params



● - artifacts

● - pipelines

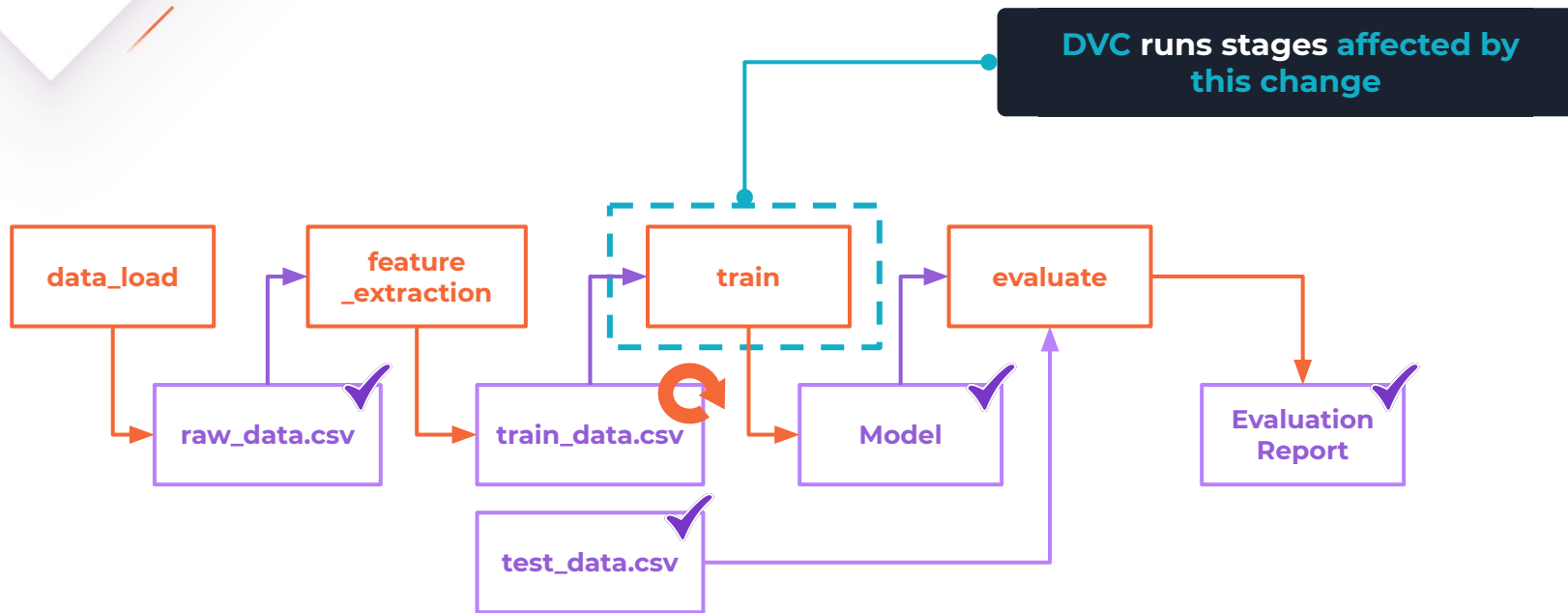
How **dvc repro** works?



● - artifacts

● - pipelines

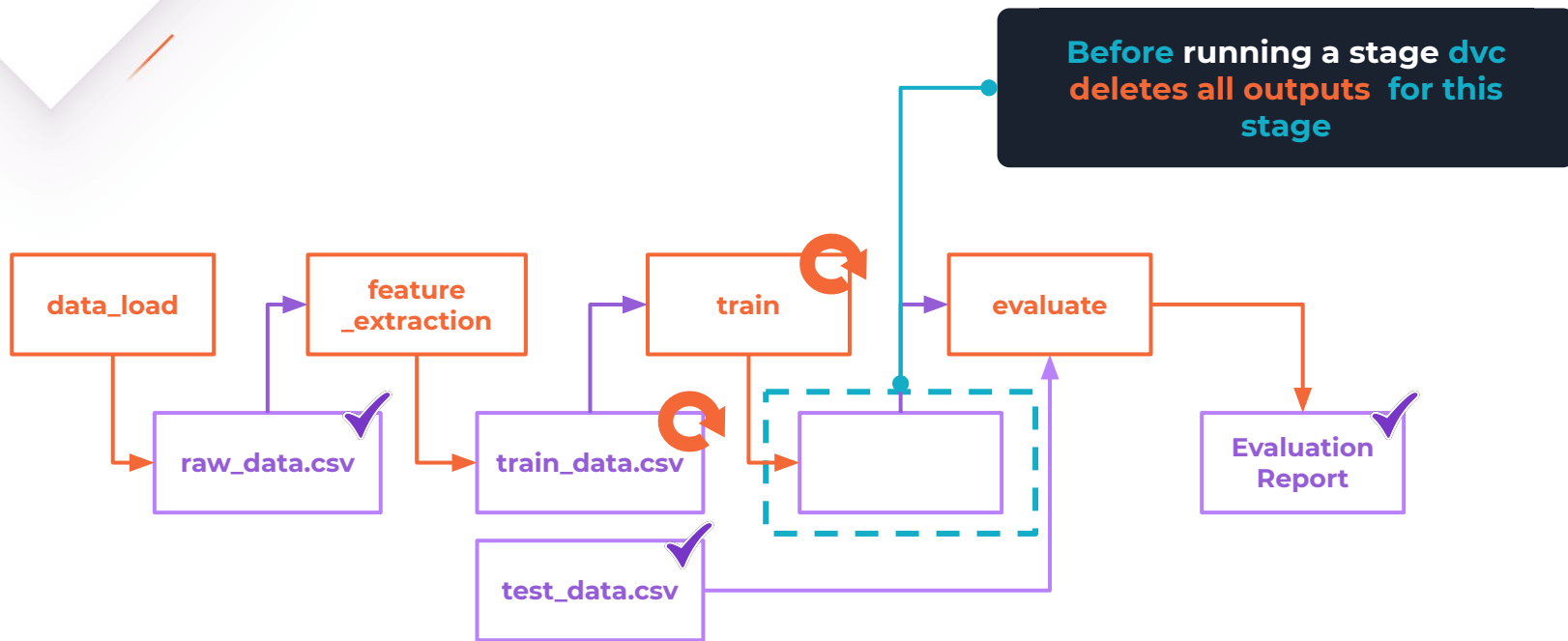
How **dvc repro** works?



● - artifacts

● - pipelines

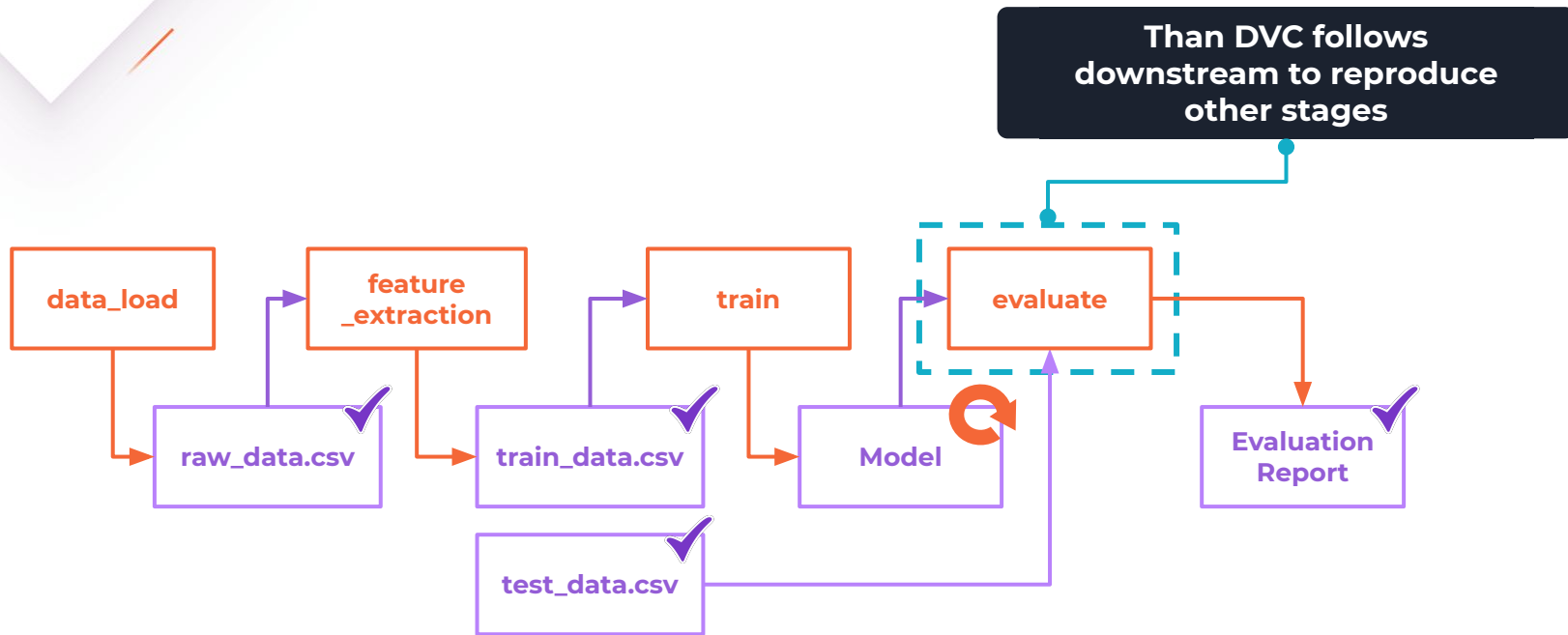
How **dvc repro** works?



● - artifacts

● - pipelines

How **dvc repro** works?



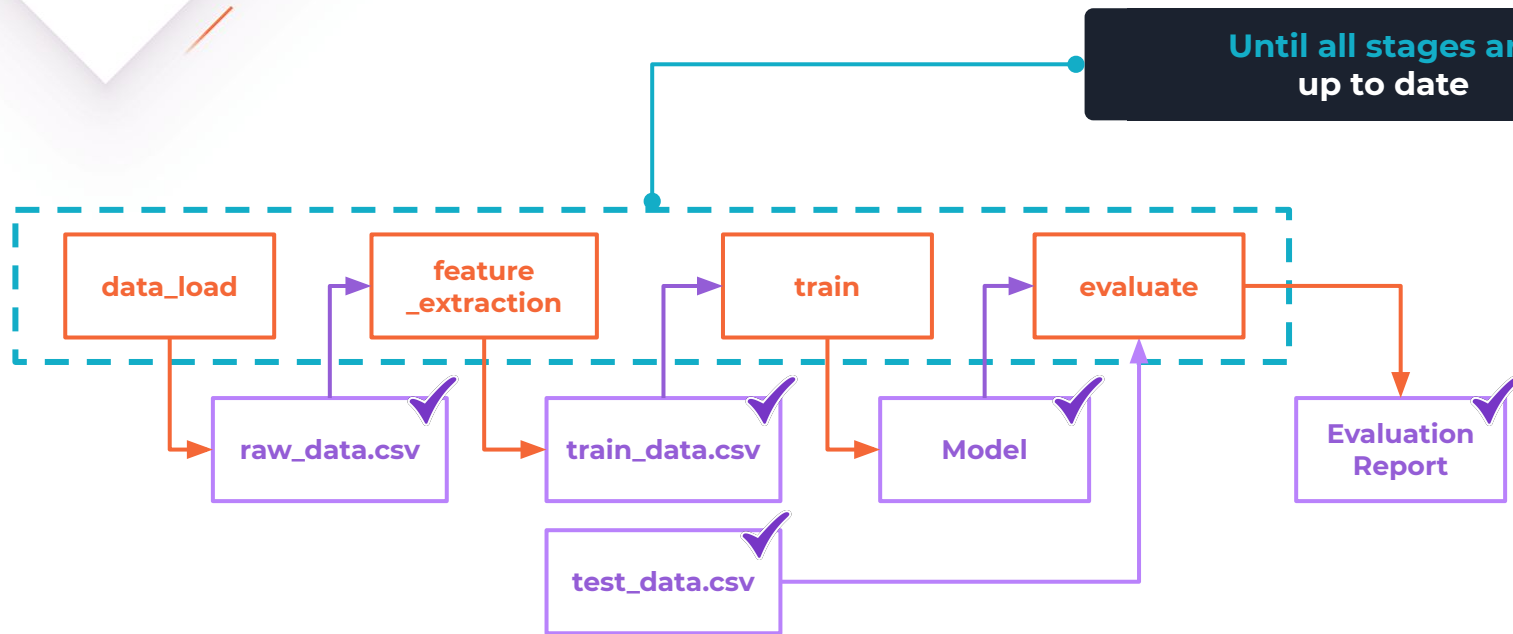
● - artifacts

● - pipelines

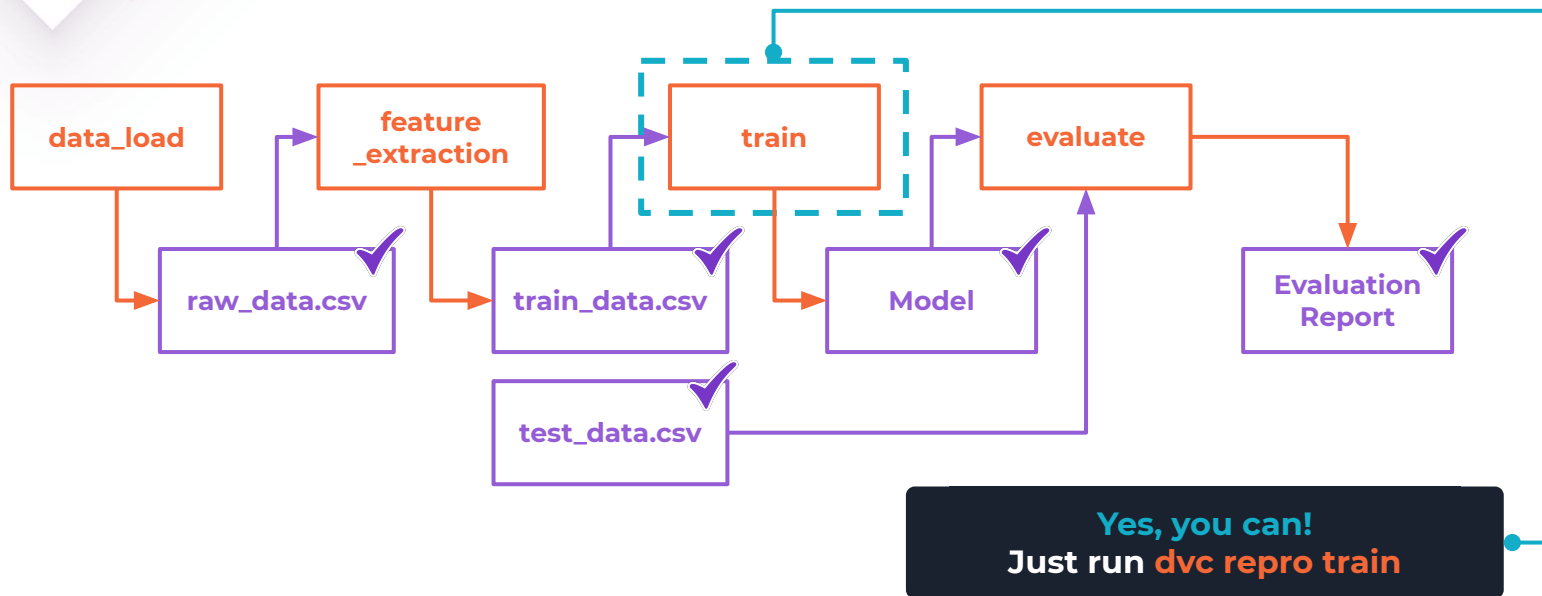
How **dvc repro** works?



Until all stages are
up to date



Can I reproduce only 'train' stage?



● - artifacts

● - pipelines

Tips to avoid unexpected behavior



- ❖ Read/write only from/to the specified dependencies and outputs (including parameters files, metrics, and plots)
- ❖ Completely rewrite outputs. Do not append or edit.
- ❖ Stop reading and writing files when the command exits
- ❖ Pipeline code should be deterministic (freeze random seeds, software and hardware dependencies, etc.)

References: <https://dvc.org/doc/command-reference/run#dependencies-and-outputs>



Live code example

Reproduce end-to-end ML pipelines



What have we learned?

What have we learned?



1. Simple way to upgrade code from Jupyter Notebook to automated pipelines
2. How to automated pipelines with DVC
3. How `dvc repro` works





Links

- ◆ DVC docs: **dvc repro** <https://dvc.org/doc/command-reference/repro#options>
- ◆ Deployment of Machine Learning Models (online course on Udemy created by Christopher Samiullah and Soledad Galli)
<https://www.udemy.com/course/deployment-of-machine-learning-models/>