# DVC in class exercise

## Learning Objectives

By the end of this lab, you should be able to:

- Explain the roles of Git and DVC in an ML workflow.
- Use DVC to version a real dataset from the internet.
- Build a two-stage DVC pipeline (`process_data` and `validate`).
- Use Git commits + DVC to restore older data and pipeline states.
- Handle a "new dataset version" scenario while keeping old experiments reproducible.
- Compare metrics across versions using `git diff`.

## Dataset & Scenario

You will work with a small classification dataset that is hosted online as a CSV file. You are acting as an ML engineer maintaining a small pipeline for a company. The data changes over time, and you must keep everything reproducible for your future self and collaborators.

## Repository Layout (Target)

After the lab, your repository should roughly look like this:

```
data/
      raw/
            customers_v1.csv
      processed/
          customers_clean.csv
reports/
      metrics.json
src/
      make_processed.py
      validate.py
params.yaml
dvc.yaml
dvc.lock
data/raw/customers_v1.csv.dvc
.gitignore (and data/.gitignore)
```

# 1 Project & Environment Setup

- Initialize a new Git repository
- Create and activate a Python virtual environment
- Install DVC
- Install any additional libraries you will need
- Initialize DVC
- Add and commit the initial Git state

# 2 Add Raw Data with DVC

- Create the raw data directory.
- Use DVC to download and track the dataset
- Check what Git wants to commit
- Add the new DVC file and ignore rules

# 3 Build the DVC Pipeline (Process & Validate)

In this part, you will create a two-stage pipeline:

1. `process_data`: cleans and preprocesses the raw data. I want only take out the following flower rows: Setosas for which Petal width is less than 0.2, Versicolor for which Sepal width is more than 3 and Virginica for which sepal length is more than 6.5. Make sure these rules are not magic numbers in your code.
2. `validate`: runs a simple statistics analysis on the processed data and writes metrics to `reports/metrics.json` that shows the mean, median, standard deviation, range, min and max of every attributes for every type of flower types (Setosa, Versicolor and Virginica).

### Create the Processing Script

- Create the directory structure
- Create a file `src/make_processed.py` that does he pre-processing and writes he cleaned data
- Create a `params.yaml` file to hold configuration values for pre-processing
- Your `make_processed.py` script should read these parameters

### Create the Validation Script

- Create a python file that reads cleaned data, computes simple statistics that I have asked for and writes a small JSON file containing those numbers/stats.

### Define the DVC Stages in `dvc.yaml`

- Add the process_data stage
- Add the validate stage
- Inspect the generated dvc.yaml file and make sure all looks good

### Run the Pipeline and Inspect `dvc.lock`

- Run the full pipeline
- Inspect dvc.lock
- Add and commit the updated files accordingly

## 4 New Dataset Version & Reproducibility

Now you will simulate a realistic situation: the business team uploads a new version of the raw dataset. Here's the link. Download this new data.

### Bring in the New Raw Dataset (v2)

- Use DVC to import the new dataset version, overwriting the old file
- Add a new metric in the validate pipeline, also calculate the variance
- Also change the preprocessing pipeline. Previously, it was taking out Setosas for which Petal width is less than 0.2. Now make it: take out Setosas for which Petal width is less than 0.1
- implement these changes and run the whole thing again
- Check what DVC thinks has changed

### Reproduce the Pipeline with the New Data

- Re-run the pipelines
- Inspect the new `reports/metrics.json`. Record the new metric
- Commit

### Compare Old vs New Metrics with Git

- Use git log to identify differences
- Use git diff to compare the metrics file between the two versions and show me the diff
- also do git diff on the yaml files to make sure you are also changing the parameters and piepelines accordingly

### Restore the Old Data & Metrics Exactly

Maybe this new version was found to be corrupted. You have to go back to he old version.

- Restore the repository state corresponding to the **old** data version
- Restore the exact data and outputs that correspond to that commit
- Confirm that this is indeed old data
- Finally, return to the latest commit when you are done inspecting the old state