MLOPS

Step 1 — ML Lifecycle & Versioning (DVC + MLflow)

Goal (1 slide)

* Track the whole baseline pipeline (data → features → model) so results are reproducible & auditable.
* Tools: Git (code history), DVC (data/artifact versioning + pipeline), MLflow (experiments, params, metrics, artifacts).

What we added (1 slide)

* Params: params.yaml (hyperparams & feature flags)
* Code:
  + src/ingest/ingest.py (validates + normalizes raw CSV)
  + src/features/build\_features.py (feature engineering with flags from params)
  + src/models/train\_baseline.py (Ridge baseline + MLflow logging)
* Pipeline: dvc.yaml with 3 stages: ingest → features → train
* Tracking: MLflow local tracking store at ./mlruns

Repository wiring (explain each piece)

1) Parameters (single source of truth)

params.yaml

train:

alpha: 2.0

target: units

features:

use\_one\_hot: true

include\_price: true

include\_promo: true

include\_holiday: true

include\_competitor\_diff: true

* Flip these values → DVC will rebuild the right stages; MLflow will log new runs.

2) Ingest (data quality gate)

src/ingest/ingest.py (you have the improved version)

* Validates required columns, coerces types, enforces non-negative/positive checks.
* Normalizes date, sorts rows, writes data/interim/ingested.csv.

Command run by DVC:

python src/ingest/ingest.py data/raw/sales.csv --out data/interim/ingested.csv

3) Feature engineering (param-driven)

src/features/build\_features.py

* Builds 7/14-day lags & rolling means over the chosen target (units).
* Optional features toggled via params: price, on\_promo, is\_holiday, competitor\_diff.
* One-hot encoding toggle: use\_one\_hot.

Command (DVC fills in from params.yaml):

python src/features/build\_features.py \

--in data/interim/ingested.csv \

--out data/processed/features.csv \

--target ${train.target} \

--use-one-hot ${features.use\_one\_hot} \

--include-price ${features.include\_price} \

--include-promo ${features.include\_promo} \

--include-holiday ${features.include\_holiday} \

--include-competitor-diff ${features.include\_competitor\_diff}

4) Baseline training + MLflow logging

src/models/train\_baseline.py

* Trains Ridge(alpha) on engineered features, drops date from X.
* Logs params (alpha, target), metrics (MAE, RMSE), and artifacts to MLflow.
* Saves repo artifacts: models/baseline\_linear.json, models/metrics\_baseline.json.

Command:

python src/models/train\_baseline.py \

--in data/processed/features.csv \

--model models/baseline\_linear.json \

--metrics models/metrics\_baseline.json \

--alpha ${train.alpha} \

--target ${train.target}

The pipeline (show this snippet)

dvc.yaml

params:

- params.yaml

stages:

ingest:

cmd: python src/ingest/ingest.py data/raw/sales.csv --out data/interim/ingested.csv

deps:

- src/ingest/ingest.py

- data/raw/sales.csv

outs:

- data/interim/ingested.csv

features:

cmd: >

python src/features/build\_features.py

--in data/interim/ingested.csv

--out data/processed/features.csv

--target ${train.target}

--use-one-hot ${features.use\_one\_hot}

--include-price ${features.include\_price}

--include-promo ${features.include\_promo}

--include-holiday ${features.include\_holiday}

--include-competitor-diff ${features.include\_competitor\_diff}

deps:

- src/features/build\_features.py

- data/interim/ingested.csv

- params.yaml

outs:

- data/processed/features.csv

params:

- train.target

- features.use\_one\_hot

- features.include\_price

- features.include\_promo

- features.include\_holiday

- features.include\_competitor\_diff

train:

cmd: >

python src/models/train\_baseline.py

--in data/processed/features.csv

--model models/baseline\_linear.json

--metrics models/metrics\_baseline.json

--alpha ${train.alpha}

--target ${train.target}

deps:

- src/models/train\_baseline.py

- data/processed/features.csv

- params.yaml

outs:

- models/baseline\_linear.json

- models/metrics\_baseline.json

params:

- train.alpha

- train.target

- features.use\_one\_hot

- features.include\_price

- features.include\_promo

- features.include\_holiday

- features.include\_competitor\_diff

Key idea: DVC caches and versions the outs (interim/ingested.csv, processed/features.csv, and models/\*.json) and remembers exactly which deps and params produced them. That’s your data + model lineage.

How you ran it (demo script)

# Reproduce whole pipeline

dvc repro

# (You saw:)

# INGESTED\_ROWS=...

# FEATURE\_ROWS=...

# MLflow: created experiment 'retail-sales' and logged run

# Launch MLflow UI

mlflow ui --host localhost --port 5000 --workers 1

# Open http://localhost:5000 → experiment: retail-sales

In the UI:

* Experiment: retail-sales
* Run: shows params (alpha, target), metrics (mae, rmse), artifacts (JSONs)

Show “versioning” live (2 quick demos)

A) Hyperparameter v2 (model v2)

# change alpha

notepad params.yaml # set train.alpha: 3.0

dvc repro

git add params.yaml dvc.lock models/\*.json

git commit -m "Model v2: alpha=3.0"

git tag model-v2

Explain: DVC saw train.alpha changed → re-ran train. MLflow shows a new run under retail-sales. You now have model v1 vs v2 with comparable metrics.

B) Data v2 (dataset change)

* Modify data/raw/sales.csv (e.g., add a few new rows or fix a value).

dvc repro

git add data/raw/sales.csv dvc.lock data/interim data/processed

git commit -m "Dataset v2 (ingest + features rebuilt)"

git tag data-v2

Explain: DVC detected raw data change → rebuilt ingest → features → train. You can now roll back to any point:

git checkout <old\_commit\_or\_tag>

dvc checkout # restores the exact ingested/features/models for that commit

If you want DVC to manage the raw CSV itself (instead of Git), you can dvc add data/raw/sales.csv and commit the .dvc file—useful for large files. (You avoided it earlier due to a Windows .gitignore conflict; remove any rule that ignores data/raw/sales.csv if you switch to dvc add.)

Talking points (close the loop)

* Reproducible: Anyone can git clone, dvc pull (when you add a remote), and dvc repro to rebuild the same artifacts.
* Auditable: dvc.lock + Git commit capture exactly which code, params, and data produced each artifact.
* Observable: MLflow holds every run with params, metrics, and artifacts for easy comparison.

Optional polish (if time permits)

* Tag baseline as model-v1:
* git tag model-v1
* git push --tags
* Add a short README section describing the 3 stages and how to run dvc repro + open MLflow.

**Step 2 — EDA & AutoML with PyCaret (Hands-on)**

**Goal (1 slide)**

* Objective: quickly explore models and pick a strong baseline automatically.
* Tools: PyCaret (AutoML), MLflow (tracking), DVC (reproducibility).
* Input: data/processed/features.csv (from Step 1).
* Output: leaderboard, sample predictions, top-model metrics, and a serialized model.

**What we added (1 slide)**

* New script: src/automl/pycaret\_regression.py
* New DVC stage: automl\_pycaret in dvc.yaml
* New artifacts:
  + reports/pycaret\_leaderboard.csv
  + reports/pycaret\_predictions\_sample.csv
  + reports/pycaret\_metrics.json
  + models/pycaret\_best.pkl
* New experiment in MLflow: pycaret-regression

How the script works (explain like this)

1. **Load features & clean columns**
   * Reads data/processed/features.csv.
   * Drops date (not useful for most regressors).
   * Why: keeps only predictive columns + the target.
2. **Configure PyCaret**

setup(

data=df,

target=args.target, # e.g., "units"

session\_id=42, # reproducible CV split

log\_experiment=False, # we log to MLflow manually (version-compat fix)

verbose=False

)

* What this does: infers types, sets up train/validation splits, applies PyCaret’s default preprocessing.

**Run AutoML**

best = compare\_models() # trains many algorithms via CV, ranks by RMSE

lb = pull() # fetch the leaderboard DataFrame

* What this does: tries models (e.g., Gradient Boosting, LightGBM, Ridge…), compares them on RMSE, returns the best.

**Quick predictions & files**

preds = predict\_model(best) # get holdout predictions

lb.to\_csv(reports/pycaret\_leaderboard.csv)

preds.head(200).to\_csv(reports/pycaret\_predictions\_sample.csv)

* Why: you have something human-readable to show (which model won + sample outputs).

**Save the model**

base\_path = save\_model(best, "models/pycaret\_best") # PyCaret writes .pkl

* Result: models/pycaret\_best.pkl (plus any companion files).

**Log to MLflow (manually)**

mlflow.set\_tracking\_uri("file:./mlruns")

mlflow.set\_experiment("pycaret-regression")

with mlflow.start\_run():

mlflow.log\_param("best\_model\_name", top["Model"])

mlflow.log\_param("target", args.target)

mlflow.log\_metrics({"rmse": ..., "mae": ..., "r2": ...})

mlflow.log\_artifact(reports/pycaret\_leaderboard.csv)

mlflow.log\_artifact(reports/pycaret\_predictions\_sample.csv)

mlflow.log\_artifact(reports/pycaret\_metrics.json)

mlflow.log\_artifact(models/pycaret\_best.pkl)

* Why: keeps an auditable record of model, metrics, and artifacts in the same MLflow UI you used for Step 1.

**How we wired it into DVC (show dvc.yaml snippet)**

automl\_pycaret:

cmd: >

python src/automl/pycaret\_regression.py

--in data/processed/features.csv

--target ${train.target}

--leaderboard reports/pycaret\_leaderboard.csv

--pred-sample reports/pycaret\_predictions\_sample.csv

--model-dir models/pycaret\_best

--metrics reports/pycaret\_metrics.json

--experiment-name pycaret-regression

--tracking-uri file:./mlruns

deps:

- src/automl/pycaret\_regression.py

- data/processed/features.csv

- params.yaml

outs:

- reports/pycaret\_leaderboard.csv

- reports/pycaret\_predictions\_sample.csv

- reports/pycaret\_metrics.json

- models/pycaret\_best.pkl

params:

- train.target

* deps: what triggers reruns (code, features, params).
* outs: versioned outputs (PyCaret artifacts).
* params: if train.target changes, DVC rebuilds the stage.

**Commands you ran (demo script)**

# 1) Reproduce AutoML stage

dvc repro automl\_pycaret

# 2) Open MLflow UI

mlflow ui --host localhost --port 5000 --workers 1

# Visit http://localhost:5000 → experiment: pycaret-regression

# 3) Commit results

git add dvc.lock reports/\*.csv reports/\*.json models/pycaret\_best.pkl

git commit -m "Step 2: PyCaret AutoML run + artifacts"

**What to show in class (live/demo flow)**

1. DVC stage in dvc.yaml → point out deps, outs, and params.
2. Run dvc repro automl\_pycaret (or show it already ran).
3. Open MLflow → experiment pycaret-regression.
   * Open latest run → show:
     + Parameters: best\_model\_name, target.
     + Metrics: rmse, mae, r2.
     + Artifacts: pycaret\_leaderboard.csv, pycaret\_predictions\_sample.csv, pycaret\_metrics.json, pycaret\_best.pkl.
4. Open pycaret\_leaderboard.csv (artifact) → point to the ranking (e.g., Gradient Boosting Regressor won for you).
5. Optional: compare to Step 1 (Ridge) in the retail-sales experiment and mention the RMSE improvement.

Your run result (example you can quote):

* Best model: Gradient Boosting Regressor
* MAE ≈ 11.57, RMSE ≈ 16.49, R² ≈ 0.48

(Exact numbers can vary slightly across runs.)

Why this is valuable (talking points)

* Speed: AutoML evaluates many models with cross-validation in one go.
* Traceability: MLflow captures every run (params, metrics, artifacts).
* Reproducibility: DVC ties code + data + parameters → same results when rerun.
* Hand-off ready: You have a serialized model (.pkl) and a leaderboard for decision-making.

Optional extras (if time permits)

* Faster demos: compare\_models(n\_select=1) and setup(..., fold=3).
* Tuning: tune\_model(best) to squeeze extra performance.
* Feature toggles: change flags in params.yaml and show DVC rebuild + new MLflow run.