Day70_MLOps_1_Basics_with_MLflow

August 22, 2025

1 Introduction to MLOps & CI/CD Pipeline (with MLflow)

Today we are starting our journey into **MLOps (Machine Learning Operations)**.

1.1 What is MLOps?

Machine Learning Operations (MLOps) is a set of practices that **automate and simplify machine learning workflows and deployments**.
It unifies **ML development (Dev)** with **ML system deployment and operations (Ops)**.

1.2 Key idea:

- MLOps = DevOps + Machine Learning + Data Engineering
- Ensures **ML models are versioned, monitored, retrained, and deployed continuously**.

1.3 Why is MLOps required?

1. Data keeps changing → Model needs **retraining & new versions**.
2. Without MLOps:
   - Models become slow to update
   - High error-prone deployments
   - Hard to monitor performance
3. With MLOps:
   - Faster model development
   - Automated deployment pipelines (CI/CD)
   - Continuous monitoring & retraining

1.4 Principles of MLOps

1. **Version Control** → Track models, datasets, experiments.

2. **Reproducibility** → Same input should produce same output.

3. **Automation** → Automate data preprocessing, training, deployment.

4. **Continuous X** → CI (Integration), CD (Deployment), CT (Training), CM (Monitoring).

5. **Governance** → Security, fairness, compliance, ethical checks.

1.5 Benefits of MLOps

- Faster time to market
- Improved productivity
- Efficient deployment & monitoring
- Continuous model improvement
- Better governance & compliance

1.6 Levels of MLOps (Maturity)

- **Level 0 (Manual)**: Manual workflows, retraining only a few times a year.
- **Level 1 (Automation)**: Automates retraining with new data.
- **Level 2 (Advanced)**: Full automation, multiple pipelines, continuous retraining & deployment.

1.7 CI/CD in MLOps

- **Continuous Integration (CI):** Every code/data/model change is tested automatically.
- **Continuous Deployment (CD):** Model updates are automatically deployed into production.
- **Continuous Monitoring (CM):** Model performance is tracked in production.
- **Continuous Training (CT):** Model is retrained when performance drops or new data arrives.

1.8 MLOps Workflow (Simplified)

1. Data → Preprocessing → Model Training
2. Save Model → Store Metadata (MLflow)
3. CI/CD Pipeline → Automated retraining

4. Deploy to API/Cloud (AWS, GCP, Azure, Streamlit, FastAPI)

5. Monitor Performance → Retrain with new data

2 Running MLflow UI for Experiment Tracking

Before accessing the MLflow interface in the browser, we need to **start the MLflow tracking server**.

2.1 Run MLflow UI from Terminal

```
mlflow ui
```

What happens:

- This command launches a local MLflow tracking server.
- By default, it runs at: http://127.0.0.1:5000
- The notebook code logs experiments, parameters, metrics, and models into this server.

Important:

- Keep this terminal open while using MLflow.
- If you close it, the UI will stop, and your browser links will not work.

2.2 Why do we need MLflow UI?

- To organize experiments into projects (Experiments).
- To log runs (each training execution = 1 run).
- To compare models based on metrics (accuracy, F1-score, recall, etc.).
- To store artifacts (saved models, plots, requirements).
- In short: MLflow UI is your control panel for all experiments.

2.3 How links work

After running your experiment, MLflow prints two links in the notebook:

Link 1 – MLflow Home (http://127.0.0.1:5000): Opens the dashboard with all Experiments.

Link 2 – Direct Run URL: Opens the specific run you just executed.

Both links open in your browser (as shown in screenshots below).rom Terminal

3 Import Libraries

We start by importing required Python libraries.

- **numpy** → numerical operations

- **make_classification** → generates synthetic datasets for ML experiments

- **train_test_split** → splits data into training/testing sets
- **LogisticRegression** → classification algorithm
- **classification_report** → evaluate model performance
- **warnings** → suppress warnings for cleaner output
- **mlflow** → track experiments, parameters, metrics, and models

[1]:
```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
import warnings
import mlflow
import mlflow.sklearn

warnings.filterwarnings('ignore')
```

4 Dataset Creation

We will create a **synthetic imbalanced dataset** using sklearn's `make_classification`.

This simulates a **real-world classification problem** (e.g., fraud detection where fraud cases are rare).

- `n_samples=1000` → 1000 rows of data
- `n_features=10` → 10 features (columns)
- `n_informative=2` → only 2 features actually matter
- `n_redundant=8` → 8 noisy features
- `weights=[0.9, 0.1]` → 90% class 0, 10% class 1 (imbalanced)
- `random_state=42` → reproducibility

[2]:
```
# Step 2: Create an imbalanced dataset
X, y = make_classification(
    n_samples=1000, n_features=10, n_informative=2, n_redundant=8,
    weights=[0.9, 0.1], flip_y=0, random_state=42
)

# Check class distribution
np.unique(y, return_counts=True)
```

[2]: (array([0, 1]), array([900, 100]))

5 Train-Test Split

We split the dataset into **training and testing sets**.

- **Training set** → used to teach the model
- **Testing set** → used to evaluate performance on unseen data
- `test_size=0.3` → 30% test, 70% train
- `stratify=y` → ensures both train/test maintain same class distribution

[3]:
```
# Step 3: Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, stratify=y, random_state=42
)
```

6 Define and Train Model

We use **Logistic Regression** (a simple ML algorithm for classification).

Parameters:
- `solver="lbfgs"` → optimization method
- `max_iter=1000` → ensures convergence
- `multi_class="auto"` → handles binary/multiclass problems automatically
- `random_state=42` → reproducibility

The model is then **fitted (trained)** on training data.

[4]:
```
# Step 4: Define model parameters
params = {
    "solver": "lbfgs",
    "max_iter": 1000,
    "multi_class": "auto",
    "random_state": 42,
}

# Train the model
lr = LogisticRegression(**params)
lr.fit(X_train, y_train)
```

[4]: LogisticRegression(max_iter=1000, multi_class='auto', random_state=42)

7 Predictions and Evaluation

Once the model is trained, we test it on the **unseen test data**.

We evaluate using `classification_report`, which provides:
- **Precision** → how many predicted positives are correct
- **Recall** → how many actual positives were captured
- **F1-score** → balance between precision & recall
- **Accuracy** → overall correctness

[5]:
```python
# Step 5: Make predictions
y_pred = lr.predict(X_test)

# Generate classification report
report = classification_report(y_test, y_pred)
print(report)

# Also store as dictionary (for MLflow logging)
report_dict = classification_report(y_test, y_pred, output_dict=True)
report_dict
```

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       270
           1       0.62      0.50      0.56        30

    accuracy                           0.92       300
   macro avg       0.79      0.73      0.76       300
weighted avg       0.91      0.92      0.92       300
```

[5]:
```
{'0': {'precision': 0.9456521739130435,
  'recall': 0.9666666666666667,
  'f1-score': 0.9560439560439561,
  'support': 270.0},
 '1': {'precision': 0.625,
  'recall': 0.5,
  'f1-score': 0.5555555555555556,
  'support': 30.0},
 'accuracy': 0.92,
 'macro avg': {'precision': 0.7853260869565217,
  'recall': 0.7333333333333334,
  'f1-score': 0.7557997557997558,
  'support': 300.0},
 'weighted avg': {'precision': 0.9135869565217392,
  'recall': 0.92,
  'f1-score': 0.9159951159951161,
  'support': 300.0}}
```

8 Experiment Tracking with MLflow

Now we integrate **MLflow** to track experiments.

- `mlflow.set_experiment("1st Experiment")` → creates a new experiment
- `mlflow.set_tracking_uri("http://127.0.0.1:5000/")` → MLflow tracking server
- Inside `mlflow.start_run()` we log:
  - Parameters (`mlflow.log_params`)
  - Metrics (`mlflow.log_metrics`)
  - Model artifact (`mlflow.sklearn.log_model`)

[8]:
```python
# Step 6: Log experiment in MLflow
mlflow.set_experiment("1st Experiment")
mlflow.set_tracking_uri("http://127.0.0.1:5000/")

with mlflow.start_run():
    # Log parameters
    mlflow.log_params(params)

    # Log metrics
    mlflow.log_metrics({
        'accuracy': report_dict['accuracy'],
        'recall_class_0': report_dict['0']['recall'],
        'recall_class_1': report_dict['1']['recall'],
        'f1_score_macro': report_dict['macro avg']['f1-score']
    })

    # Log model
    mlflow.sklearn.log_model(lr, "Logistic Regression")
```

```
2025/08/22 10:49:16 WARNING mlflow.models.model: `artifact_path` is deprecated.
Please use `name` instead.
2025/08/22 10:50:04 WARNING mlflow.models.model: Model logged without a
signature and input example. Please set `input_example` parameter when logging
the model to auto infer the model signature.
```

```
 View run sneaky-worm-317 at: http://127.0.0.1:5000/#/experiments/1359937480698
15002/runs/ce045d64068644bca5f83d2236cccad3
 View experiment at: http://127.0.0.1:5000/#/experiments/135993748069815002
```

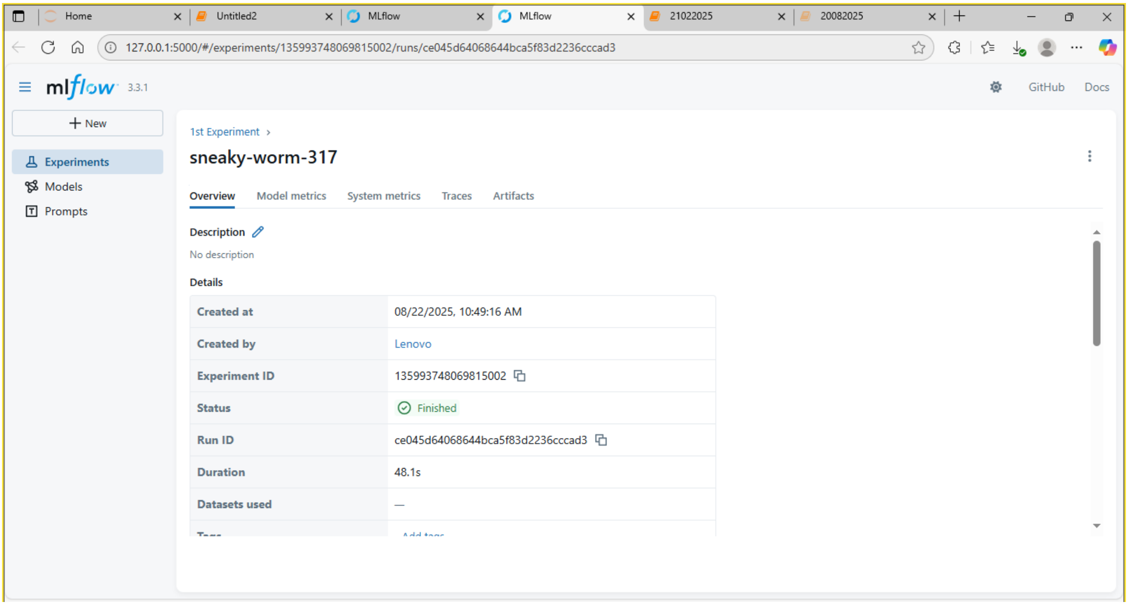
9 Opening the MLflow UI from Notebook Links

When the training cell finishes, MLflow prints **two links**:

- **Link 1 – MLflow Tracking UI (home):** Opens the main dashboard at `http://127.0.0.1:5000`. From here you can browse **Experiments**, drill into a **Run**, view **metrics**, and download **artifacts** (logged models).
- **Link 2 – Direct Run URL:** Jumps straight to the specific run you just created (the page with run details, metrics, artifacts, etc.).

Below are annotated screenshots so you know what's what.

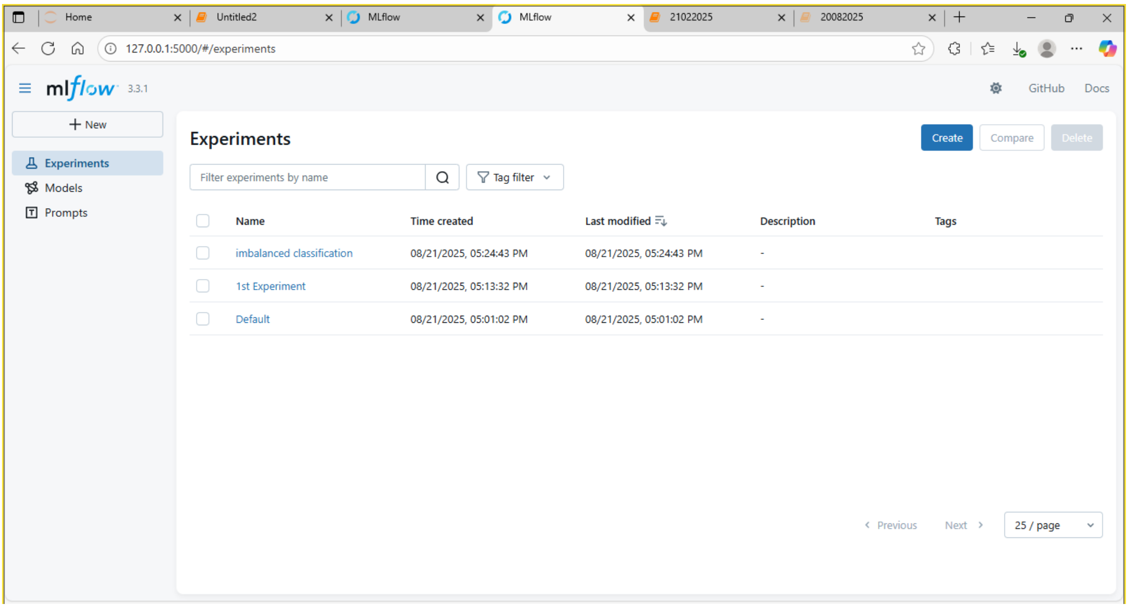
9.1 Link 1 (Main MLflow UI) — Screenshot 1: Experiments Dashboard



What you see: - **Sidebar:** *Experiments*, *Models*, *Prompts*.
- **Experiments table:** Every experiment you or your code created (e.g., **1st Experiment**, **imbalanced classification**, **Default**).
- **Columns:** *Name*, *Time created*, *Last modified*, *Description*, *Tags*.
- **Search & filters:** Use the search box / *Tag filter* to find experiments quickly.
- **Create button:** Make a new experiment manually if needed.

What to do next: Click **1st Experiment** to view all its **Runs**.

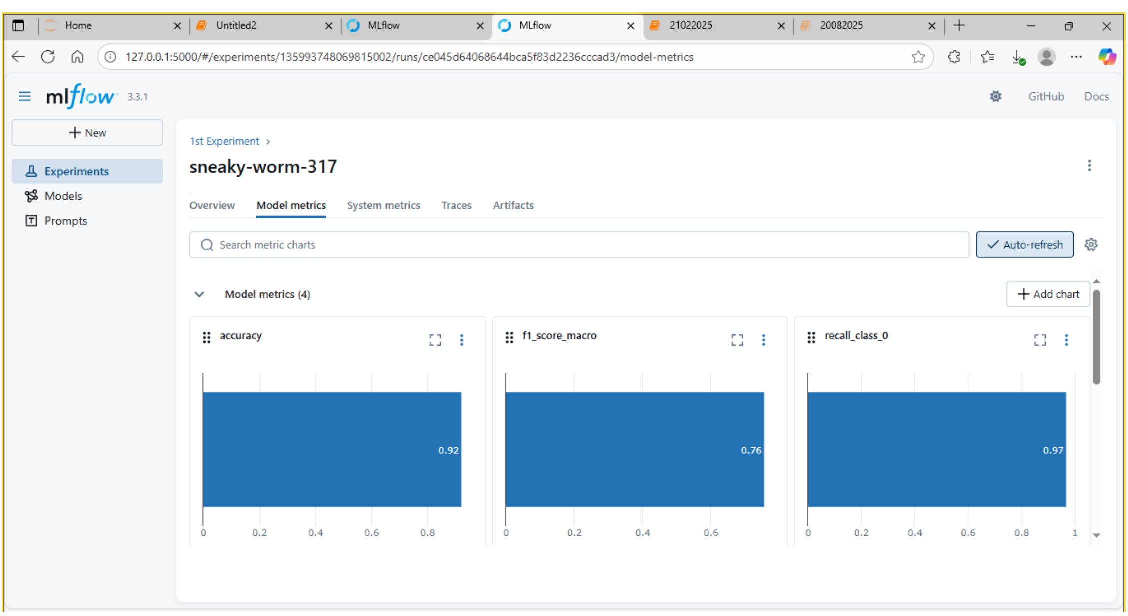
9.2 Link 1 — Screenshot 2: Run Overview (Details tab)



Key parts: - **Breadcrumbs:** *Experiments → 1st Experiment → {run_name}*.
- **Tabs:** *Overview*, *Model metrics*, *System metrics*, *Traces*, *Artifacts*.
- **Details card:**
- **Experiment ID** and **Run ID** (unique identifiers)
- **Status** (e.g., *Finished*) and **Duration** (how long training took)
- **Created at / Created by** (audit info)

Why it matters: This is the canonical page for a single run—use it to navigate to metrics and artifacts.

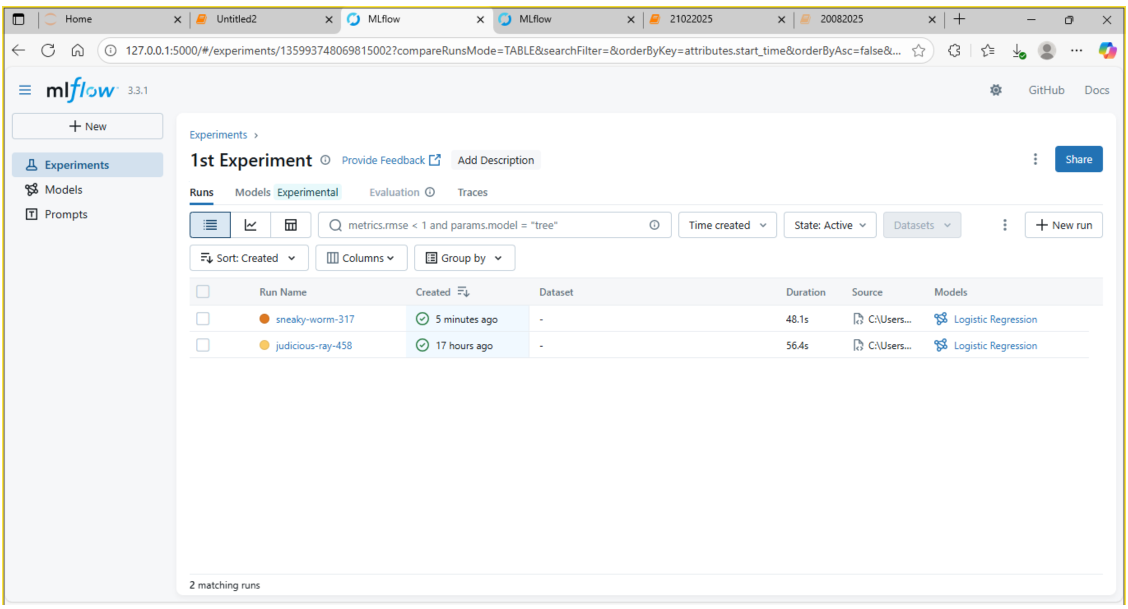
9.3 Link 1 — Screenshot 3: Model Metrics tab



What you see: - Metric charts logged from code: **accuracy**, **f1_score_macro**, **recall_class_0**, **recall_class_1**.
- Each tile shows the **latest value**; click the menu to customize, or **Add chart** for more.
- Use the search bar to quickly locate a metric by name.

Tip: If you log metrics during training across steps/epochs, charts render as **time series**.

9.4 Link 2 (Direct Run / Runs Table) — Screenshot 1: Runs list & Compare



What you see: - A table of all **runs** inside *1st Experiment* (e.g., *sneaky-worm-317*, *judicious-ray-458*).
- **Columns:** *Created*, *Duration*, *Source* (which notebook/script), **Models** (logged artifacts like *Logistic Regression*).
- **Filter bar:** Write queries (e.g., `metrics.accuracy > 0.9`) to find best runs.
- **Sort / Group by:** Organize runs by time, metric, or parameter.
- **Compare:** Select two+ runs → click **Compare** to open side-by-side charts and parameters.

Why it matters: This page is perfect for **experiment selection**—you can pick the best model based on metrics and then download or register it.

9.4.1 What to click (quick guide)

1. **From Link 1 (Home):**
   - Click your experiment (e.g., **1st Experiment**) → see **runs table**.
   - Click a **Run name** → opens the **Run Overview** page.
   - Use **Artifacts** tab to download the model (`MLmodel`, `conda.yaml`, `requirements.txt`, serialized model).
2. **From Link 2 (Direct Run):**

- You land directly on that run's detail page; open **Model metrics** or **Artifacts** immediately.

10 Summary & Next Steps

In this notebook, we explored the **foundations of MLOps**:

- Created and trained a Logistic Regression model.
- Logged parameters, metrics, and models using **MLflow**.
- Learned how to run **mlflow ui** and explored the experiment dashboard.
- Visualized performance metrics like accuracy, recall, and F1-score.
- Understood how experiments and runs are tracked for reproducibility.

10.1 Next Steps

- Try running multiple models (e.g., Random Forest, XGBoost) and compare results in MLflow.
- Explore **model registry** in MLflow for production-ready workflows.
- Learn how to deploy the trained model to a web app / API.
- Move from Notebook → VS Code → CI/CD pipeline for automation.

With MLOps, our ML journey becomes **scalable, reliable, and production-ready**.
This is just the **first step** toward mastering **end-to-end ML pipelines** .