Objective of Week 8 Assignment

- Introduce **data poisoning attacks** into the IRIS dataset.
- Evaluate the effect of poisoning levels: **0%, 5%, 10%, 50%**
- Analyze how poisoning impacts **model performance** (Accuracy, Precision, Recall, F1).
- Log all experiments using **MLflow Tracking Server**.
- Generate and compare:
    - Confusion matrices
    - Debug JSON metadata
    - Model artifacts
- Understand how to **mitigate poisoning attacks** in real ML pipelines.

What is Data Poisoning?

- Intentional corruption of training data.
- Goal: reduce model accuracy or cause targeted misclassification.
- Introduced using:
  - Random noise
  - Label flipping
  - Outlier insertion
- Simulated here using **Gaussian noise injection** into a percentage of training samples.

Experiment Setup

Components Used

- **Dataset:** IRIS (Sklearn)
- **Model:** Random Forest Classifier
- **Poisoning Method:** Gaussian noise injection
- **MLflow:**
    - Tracks parameters
    - Logs metrics
    - Stores artifacts (confusion matrix, debug.json, models)
- **VM Setup:** MLflow server running at `http://127.0.0.1:5000`

Step-by-Step Workflow

Steps Performed

1. Setup & start MLflow server in the VM.

Export MLflow tracking URI:

```
export MLFLOW_TRACKING_URI=http://127.0.0.1:5000
```

2.

Run the poisoning experiment with multiple levels:

```
python iris_poisoning_mlflow_debug.py --poison-levels 0.0 0.05 0.10 0.50 --noise-std 1.0
```

3.
4. Log all metrics, params, and artifacts to MLflow.

5. Export run metrics to CSV (`week8_metrics.csv`).

6. Collect and package artifacts from MLflow.

7. Analyze poisoning impact and prepare observations.

Poisoning Logic in Code

Key Components of `iris_poisoning_mlflow_debug.py`

- `poison_data_debug()`
    - Adds random Gaussian noise to selected samples.
- `plot_and_save_cm()`
    - Generates confusion matrix for each poisoning level.
- `run_experiment()`
    - Runs model training + logging for each poisoning level.
- `main()`
    - Loops through poisoning fractions and logs runs to MLflow.

Metrics Observed

Sample Results (based on our run)

Poison %	# Samples Poisoned	Accuracy	Precision (Macro)	Recall (Macro)	F1 Score (Macro)	Observation
0%	0	**0.90**	0.9023	0.90	0.8997	Baseline (clean data)
5%	6	**0.9333**	0.9333	0.9333	0.9333	Slight improvement – noise acts like regularization
10%	12	**0.9667**	0.9697	0.9667	0.9665	Best performance — moderate noise increases model robustness
50%	60	**0.9333**	0.9444	0.9333	0.9326	Performance drops — heavy poisoning impacts stability

Confusion Matrices

Confusion Matrices for Each Poisoning Level

- **0% Poisoning**
  - Clean decision boundaries
  - Minimal misclassification
- **5% Poisoning**
  - Still clean — small noise has negligible impact
- **10% Poisoning**
  - Best performing model, confusion matrix almost perfectly diagonal
- **50% Poisoning**
  - More off-diagonal values
  - Model starts confusing overlapping classes due to heavy noise

Observations & Interpretation

Key Learnings from Results

- The model remains **stable** under mild poisoning (5%).
- Moderate poisoning (10%) improves accuracy — **noise acts like a regularizer**.
- Heavy poisoning (50%) begins to distort feature distributions significantly.
- MLflow artifacts confirm:
  - Mean & Std deviation shift heavily after poisoning
  - Confusion matrices become less diagonal at higher noise
  - Debug JSON files show poisoned sample indices and distribution changes
- Poisoning does not always instantly break a model; ML systems must monitor **gradual degradation**.

Mitigation Strategies

Defense Techniques Against Data Poisoning

- **Data Validation Pipelines**
  - Detect abnormal feature distributions
  - Identify outliers or sudden shifts
- **Statistical Drift Detection**
  - Monitor mean, std, KL divergence
  - Compare clean vs new data distributions
- **Robust Training Methods**
  - Noise-tolerant algorithms
  - Median loss functions
  - RANSAC-based approaches
- **Data Provenance Tools**
  - DVC remote storage
  - Audit logs
- **Human-in-the-loop Review**
  - Manually inspect flagged samples
- **Model Monitoring**
  - Track online accuracy changes using MLflow + custom metrics

Files Included

Files Submitted for Week 8

- **iris_poisoning_mlflow_debug.py**
  - Main experiment script
  - Implements poisoning, training, metric logging, and artifact generation
- **week8_metrics.csv**
  - Aggregated table of all experiment runs
  - Used to compare poisoning impact
- **myscript.txt**
  - Commands executed during the VM experimentation and MLflow setup
- **README-week8.md**
  - Documentation explaining the Week 8 workflow, results, and artifacts

The `artifacts_flat/` Folder — What It Contains

The `artifacts_flat/` folder contains **all artifacts extracted from MLflow**, flattened into one directory for easy submission.

It includes:

- **Confusion matrix PNGs**
  - Visual representation of predictions per poisoning level
  - Helps compare model performance qualitatively
- **Debug JSON files**
  - Contains:
    - poisoned sample indices
    - before/after feature stats
    - metrics per run
  - Useful for auditing poisoning impact
- **Model artifacts (`model.joblib`, MLmodel metadata, conda.yaml)**
  - Exact model used in each run
  - Enables reproducibility

🎯 Goal of the `artifacts_flat/` Folder

- To **submit MLflow artifacts** without requiring the entire mlruns directory
- To make evaluation easier for instructors
- To show:
  - you logged artifacts correctly
  - you understand debugging metadata
  - you captured model behavior under poisoning

Conclusion
Final Summary

- Successfully simulated **data poisoning attacks** at 0%, 5%, 10%, and 50%.
- Logged complete experiment pipeline using **MLflow**.
- Observed:
  - Improvement at moderate noise
  - Degradation at heavy poisoning
- Generated confusion matrices, debug metadata, and model artifacts.
- Learned the importance of:
  - Data validation
  - Continuous monitoring
  - Artifact tracking
  - Robust model training
- Delivered a reproducible and fully tracked MLOps experiment.