Day72_CICD_Pipeline_for_ML_Project

August 25, 2025

CI/CD Pipeline for ML Projects

In this notebook, we will build a **CI/CD pipeline for ML projects**.
The goal is to automate the process from **data preparation → model training → evaluation → deployment**.

1. Why CI/CD for ML?

- **CI (Continuous Integration):** Every time code changes, it is automatically tested, validated, and merged.
- **CD (Continuous Delivery/Deployment):** New versions of the model/code are automatically delivered or deployed to production.

Tools we use: - **Git + GitHub** → Version control & code hosting.
- **GitHub Actions (CI/CD)** → Automates testing, training, and deployment.
- **Docker / Cloud (AWS, Azure, GCP)** → For real deployments (optional in this demo).

2. Workflow Steps

1. Data extraction & cleaning.
2. Train/Test split (`X_train`, `y_train`, `X_test`, `y_test`).
3. Build models (Logistic Regression, Random Forest).
4. Evaluate (confusion matrix, f1, precision, recall, accuracy).
5. Save outputs (plots + metrics + pickle file).
6. Push code & files to GitHub.
7. Create a GitHub Actions Workflow (`.github/workflows/run.yml`).
8. CI/CD auto-runs pipeline on every push.

1 Model Training

[1]:
```
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, precision_score, f1_score,␣
 ↪recall_score
import pickle
```

[3]:
```
sns.set(style='white')

# Load Data
dataset = pd.read_csv(r"C:\Users\Lenovo\OneDrive\Desktop\Python Everyday␣
 ↪work\Class work\AI\MlOps\CICD\code-2\code-2\iris.csv")

# Feature names (Ensure no extra spaces or parentheses)
dataset.columns = [colname.strip(' (cm)').replace(" ", "_") for colname in␣
 ↪dataset.columns.tolist()]
features_names = dataset.columns.tolist()[:4]
```

[4]:
```
# Feature Engineering
dataset['sepal_length_width_ratio'] = dataset['sepal_length'] /␣
 ↪dataset['sepal_width']
dataset['petal_length_width_ratio'] = dataset['petal_length'] /␣
 ↪dataset['petal_width']

# Select Features
dataset = dataset[['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
                   'sepal_length_width_ratio', 'petal_length_width_ratio',␣
 ↪'target']]
```

[5]:
```
# Split Data
train_data, test_data = train_test_split(dataset, test_size=0.2,␣
 ↪random_state=44)
X_train = train_data.drop('target', axis=1).values.astype('float32')
y_train = train_data.loc[:, 'target'].values.astype('int32')
X_test = test_data.drop('target', axis=1).values.astype('float32')
y_test = test_data.loc[:, 'target'].values.astype('int32')
```

[6]:
```
# Logistic Regression
```

```python
logreg = LogisticRegression(C=0.0001, solver='lbfgs', max_iter=100,␣
↪multi_class='multinomial')
logreg.fit(X_train, y_train)
predictions_lr = logreg.predict(X_test)
```

```
C:\Users\Lenovo\anaconda3\Lib\site-
packages\sklearn\linear_model\_logistic.py:1264: FutureWarning: 'multi_class'
was deprecated in version 1.5 and will be removed in 1.7. From then on, it will
always use 'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
```

[7]:
```python
cm_lr = confusion_matrix(y_test, predictions_lr)
f1_lr = f1_score(y_test, predictions_lr, average='micro')
prec_lr = precision_score(y_test, predictions_lr, average='micro')
recall_lr = recall_score(y_test, predictions_lr, average='micro')
```

[8]:
```python
train_acc_lr = logreg.score(X_train, y_train) * 100
test_acc_lr = logreg.score(X_test, y_test) * 100
```

[9]:
```python
# Random Forest
rf_reg = RandomForestRegressor()
rf_reg.fit(X_train, y_train)
predictions_rf = rf_reg.predict(X_test)
predictions_rf_class = np.round(predictions_rf).astype(int)
```

[10]:
```python
f1_rf = f1_score(y_test, predictions_rf_class, average='micro')
prec_rf = precision_score(y_test, predictions_rf_class, average='micro')
recall_rf = recall_score(y_test, predictions_rf_class, average='micro')
```

[11]:
```python
train_acc_rf = rf_reg.score(X_train, y_train) * 100
test_acc_rf = rf_reg.score(X_test, y_test) * 100
```

[12]:
```python
# Save Model
with open("model.pkl", "wb") as f:
    pickle.dump(rf_reg, f)

# Save Scores
with open("scores.txt", "w") as score:
    score.write(f"Random Forest Train Accuracy: {train_acc_rf:.2f}%\n")
    score.write(f"Random Forest Test Accuracy: {test_acc_rf:.2f}%\n")
    score.write(f"F1 Score: {f1_rf:.2f}\n")
    score.write(f"Recall Score: {recall_rf:.2f}\n")
    score.write(f"Precision Score: {prec_rf:.2f}\n")
```

2 GitHub Setup for CI/CD

1. Create new repo → `cicdpipeline`

2. Upload files:
    - `iris.csv`
    - `train.py` (the model training code above)
    - `requirements.txt`

`requirements.txt` example:

```
pandas
numpy
scikit-learn
matplotlib
seaborn
```

3. Inside repo, create folder: `.github/workflows/`

4. Add file `run.yml`:

```yaml
- name: Set up Python
  uses: actions/setup-python@v4
  with:
    python-version: '3.10'

- name: Install dependencies
  run: |
    pip install -r requirements.txt

- name: Run training script
  run: |
    python train.py
```

5. Commit & Push → Go to **Actions** tab → See pipeline running.

3 Conclusion

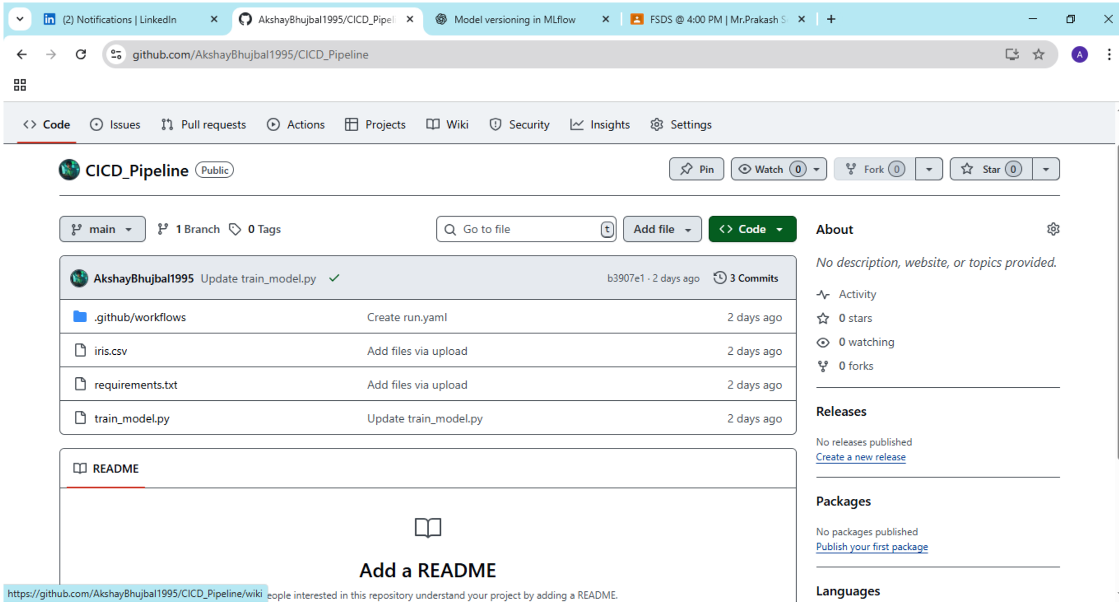
We successfully: - Trained ML models (Logistic Regression & Random Forest).
- Saved metrics and model artifacts.
- Set up GitHub Actions CI/CD workflow.
- Automated model training on each push.

This demonstrates a **basic MLOps CI/CD pipeline** which can be extended with:
- Docker + Kubernetes for deployment.
- Monitoring with Prometheus / Grafana.
- Full ML lifecycle automation.

4 Repository Setup

Step 1: Repository Setup

We created a new GitHub repository **CICD_Pipeline** and uploaded the required files.

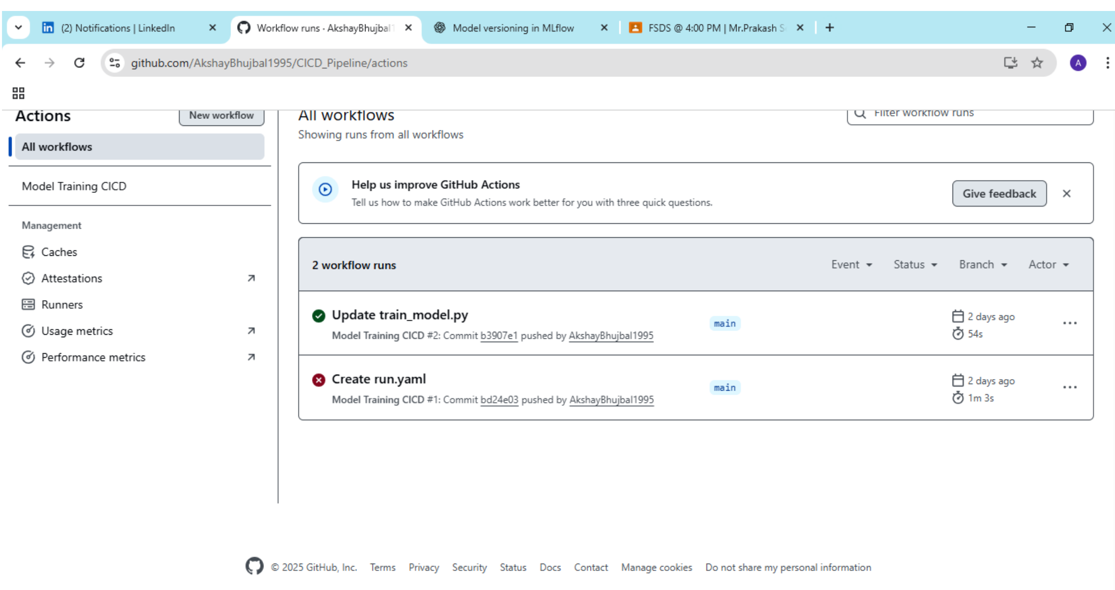


Here we can see:
- `iris.csv` → Dataset
- `train_model.py` → Model training script
- `requirements.txt` → Dependencies
- `.github/workflows/run.yaml` → Workflow definition

5 Repository Setup

Step 2: GitHub Actions Workflow

GitHub Actions automatically triggered the workflow whenever changes were pushed.

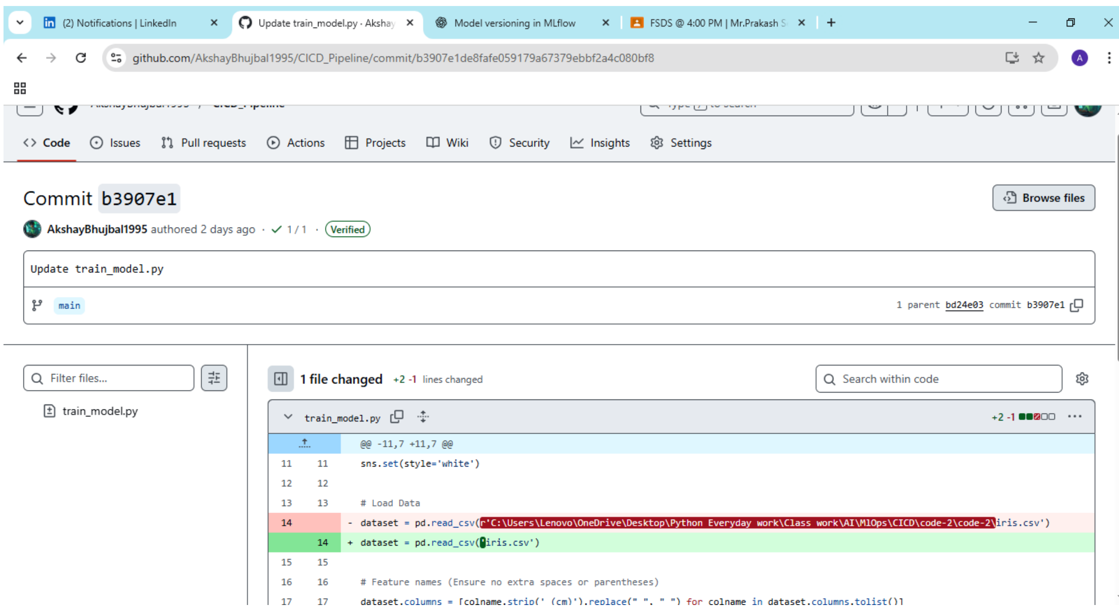


- The first workflow (`Create run.yaml`) failed because of a setup issue.
- After fixing, the second workflow (`Update train_model.py`) passed successfully.

6 Pipeline Capturing Code Changes

Step 3: Pipeline Capturing Code Changes

The pipeline also captures even **small code changes**.



Here we see that:
- **1 file was changed**
- **+2 lines added, -1 line removed**
- GitHub Actions re-ran the pipeline automatically.

This shows the **Continuous Integration (CI)** in action.