✅ PRACTICAL 2: Track experiments using MLflow

GOAL: Train a model and log its parameters and metrics to the MLflow UI.

1. Setup Project Folder

```
mkdir mlflow-practical
cd mlflow-practical
```

2. Create and Activate Virtual Environment

```
python -m venv venv
.\venv\Scripts\activate
```

3. Install Libraries

```
pip install mlflow scikit-learn pandas
```

4. Create Python file: `create_data.py`

```
import pandas as pd
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=100, n_features=10, n_informative=5, n_redundant=0, random_state=42)
df = pd.DataFrame(X, columns=[f'feature_{i}' for i in range(10)])
df['target'] = y

df.to_csv('dummy_data.csv', index=False)
print("Data created.")
```

5. Create Python file: `train.py`

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score
import mlflow
import mlflow.sklearn
import sys

# Set tracking URI to a local directory named 'mlruns'
mlflow.set_tracking_uri("file:./mlruns")

# Get C parameter from command line, default to 1.0
C_param = float(sys.argv[1]) if len(sys.argv) > 1 else 1.0

# Load data
df = pd.read_csv('dummy_data.csv')
X = df.drop('target', axis=1)
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Start an MLflow run
with mlflow.start_run():
    print("Starting MLflow run...")
    # Train model
    model = LogisticRegression(C=C_param, max_iter=200, random_state=42)
    model.fit(X_train, y_train)

    # Make predictions
    preds = model.predict(X_test)

    # Calculate metrics
    accuracy = accuracy_score(y_test, preds)
    precision = precision_score(y_test, preds)

    # Log parameters
    mlflow.log_param("C", C_param)
    mlflow.log_param("model_type", "LogisticRegression")

    # Log metrics
    mlflow.log_metric("accuracy", accuracy)
    mlflow.log_metric("precision", precision)

    # Log the model
    mlflow.sklearn.log_model(model, "model")

    print(f"Run complete. Accuracy: {accuracy}")
```

6. Run setup and training

```
python create_data.py
python train.py 0.5
python train.py 1.0
python train.py 2.0
```

7. Launch MLflow UI

```
mlflow ui
```

> Open your browser to `http://127.0.0.1:5000`

PRACTICAL 3: Track experiments using Weights & Biases

GOAL: Train a model and log its parameters and metrics to the W&B dashboard. **PRE-REQUISITE:** You need a free W&B account (https://wandb.ai/site).

1. Setup Project Folder

```
mkdir wandb-practical
cd wandb-practical
```

2. Create and Activate Virtual Environment

```
python -m venv venv
.\venv\Scripts\activate
```

3. Install Libraries

```
pip install wandb scikit-learn pandas
```

4. Login to W&B

```
wandb login
```

> This will ask you to paste an API key from your W&B profile settings.

5. Create Python file: `create_data.py`

```
import pandas as pd
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=100, n_features=10, n_informative=5, n_redundant=0, random_state=42)
df = pd.DataFrame(X, columns=[f'feature_{i}' for i in range(10)])
df['target'] = y

df.to_csv('dummy_data.csv', index=False)
print("Data created.")
```

6. Create Python file: `train_wandb.py`

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import wandb

# Load data
df = pd.read_csv('dummy_data.csv')
X = df.drop('target', axis=1)
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Configuration for the run
config = {
    "C": 1.0,
    "model_type": "LogisticRegression",
    "solver": "liblinear"
}

# 1. Initialize W&B run
wandb.init(project="practical-exam", config=config)

# 2. Train model
model = LogisticRegression(
    C=wandb.config.C,
    solver=wandb.config.solver,
    random_state=42
)
model.fit(X_train, y_train)

# Make predictions
preds = model.predict(X_test)
accuracy = accuracy_score(y_test, preds)

# 3. Log metrics
wandb.log({"accuracy": accuracy})

print(f"Run complete. Accuracy: {accuracy}")
wandb.finish()
```

7. Run setup and training

```
python create_data.py
python train_wandb.py
```

> Check the terminal output for the W&B link to your project dashboard.

⚙ PRACTICAL 5: Automate using Prefect

GOAL: Create a simple ETL and training pipeline using Prefect. **NOTE:** Prefect is simpler for local setup than Airflow.

1. Setup Project Folder

```
mkdir prefect-practical
cd prefect-practical
```

2. Create and Activate Virtual Environment

```
python -m venv venv
.\venv\Scripts\activate
```

3. Install Libraries

```
pip install prefect scikit-learn pandas
```

4. Create Python file: `pipeline.py`

```
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from prefect import task, flow

@task
def extract_data():
    """Generates dummy data and saves it."""
    X, y = make_classification(n_samples=100, n_features=10, random_state=42)
    df = pd.DataFrame(X, columns=[f'feature_{i}' for i in range(10)])
    df['target'] = y
    df.to_csv('dummy_data.csv', index=False)
    print("Task: Data extracted and saved.")
    return 'dummy_data.csv'

@task
def transform_data(data_path: str):
    """Loads and splits the data."""
    df = pd.read_csv(data_path)
    X = df.drop('target', axis=1)
    y = df['target']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    print("Task: Data transformed.")
    return X_train, X_test, y_train, y_test

@task
def load_model(X_train, y_train, X_test, y_test):
    """Trains a model and prints accuracy."""
    model = LogisticRegression()
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    acc = accuracy_score(y_test, preds)
    print(f"Task: Model trained. Accuracy: {acc}")
    return acc

@flow(name="Practical Exam Flow")
def run_pipeline():
    """Main flow to run the ETL and training pipeline."""
    data_path = extract_data()
    X_train, X_test, y_train, y_test = transform_data(data_path)
    accuracy = load_model(X_train, y_train, X_test, y_test)
    print(f"Flow complete. Final accuracy: {accuracy}")

if __name__ == "__main__":
    run_pipeline()
```

5. Run the pipeline

```
python pipeline.py
```

6. (Optional) Start Prefect UI

```
# In a new terminal
prefect server start
```

> Go to `http://127.0.0.1:4200` to see the dashboard. Run `python pipeline.py` again, and your flow run will appear in the UI.

✅ PRACTICAL 9: Apply explainability using SHAP

GOAL: Train a model and use SHAP to explain its predictions.

1. Setup Project Folder

```
mkdir shap-practical
cd shap-practical
```

2. Create and Activate Virtual Environment

```
python -m venv venv
.\venv\Scripts\activate
```

3. Install Libraries

```
pip install shap scikit-learn pandas matplotlib
```

4. Create Python file: `create_data.py`

```
import pandas as pd
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=100, n_features=10, n_informative=5, n_redundant=0, random_state=42)
df = pd.DataFrame(X, columns=[f'feature_{i}' for i in range(10)])
df['target'] = y

df.to_csv('dummy_data.csv', index=False)
print("Data created.")
```

5. Create Python file: `explain.py`

```
 import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import shap
import matplotlib.pyplot as plt
df = pd.read_csv('dummy_data.csv')


X = df.drop('target', axis=1)
y = df['target']


X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
print(f"Model trained. Accuracy: {model.score(X_test, y_test):.4f}")
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test)

if isinstance(shap_values, list):
    shap_to_plot = shap_values[1]
    expected_value = explainer.expected_value[1]
else:
    shap_to_plot = shap_values
    expected_value = explainer.expected_value

shap.summary_plot(shap_to_plot, X_test, show=False)
plt.savefig('shap_summary.png', bbox_inches='tight')
plt.close()
print("SHAP summary plot saved to 'shap_summary.png'")

shap.force_plot(
    expected_value,
    shap_to_plot[0, :],
    X_test.iloc[0, :],
    matplotlib=True,
    show=False
)
plt.savefig('shap_force_plot.png', bbox_inches='tight')
plt.close()
print("SHAP force plot saved to 'shap_force_plot.png'")
```

6. Run setup and explanation

```
 python create_data.py
python explain.py
```

> Check the folder for `shap_summary.png` and `shap_force_plot.png`