Pipeline Documentation

End-to-End MLOps Pipeline for California Housing Prediction

This repository contains a complete, end-to-end MLOps pipeline for training, versioning, containerizing, deploying, and monitoring a regression model for the California Housing dataset.

Architectural Summary

This project demonstrates a modern MLOps workflow, taking a model from experimentation to an automated, observable production service. The architecture is designed for reproducibility, scalability, and automation.

Architectural Flow Diagram

```
Developer         GitHub & DVC         GitHub Actions (CI/CD)      Docker
Hub           Production Server
+---------+       +-------------+       +----------------------+  +-----
-------+       +------------------+
|         |       |             |       |                      |  |
|       |                   |
| git push|------>| Code & Data |------>| 1. Test & Lint (CI)    |  |
|       | ./deploy.sh pulls |
|         |       | Versioning  |       | 2. Build Docker Image  |->| Imag
e      |<------| & runs container  |
+---------+       +-------------+       | 3. Push to Registry(CD)|  | Regi
stry   |       |                   |
                                        +----------------------+  +-----
-------+       +------------------+

|     ^
     ^
|     |
     |
|     |
     |    MLflow (Tracks Experiments & Registers Models)
|  Prometheus
     +----------------------------------------------------------------------
-------------------+  (Scrapes

/metrics)
```

Stages of the Pipeline

Foundation & Experimentation (Reproducibility)

Code Versioning (Git): All source code is tracked using Git for collaboration and history tracking.

Data Versioning (DVC): The raw dataset (`housing.csv`) is versioned with DVC. This keeps the Git repository lightweight by tracking only a small pointer file, while the actual data is stored in a separate cache, ensuring full data reproducibility.

Experiment Tracking (MLflow): Every model training run logs its parameters (e.g., `max_depth`), metrics (e.g., R^2, RMSE), and the resulting model file as an artifact. This creates a detailed, auditable history of all experiments.

Model Registry (MLflow): The best-performing model is programmatically identified and registered in the MLflow Model Registry. This assigns it an official name and version (e.g., `california-housing-regressor:1`), turning it into a governed asset ready for production.

Application Packaging & Observability (Production Readiness)

API Service (FastAPI): A robust, high-performance API is built to serve predictions from the registered model. It exposes a `/predict` endpoint that accepts JSON input and returns model predictions.

Containerization (Docker): The API, model artifacts, and all dependencies are packaged into a self-contained Docker image. This guarantees that the service runs identically in any environment, from a local machine to a cloud server.

Logging & Monitoring: The API is instrumented for observability. It generates structured JSON logs for every request and exposes a `/metrics` endpoint with real-time performance data (request counts, latency) in a Prometheus-compatible format.

CI/CD & Deployment (Automation)

Automation (GitHub Actions): A CI/CD pipeline automatically tests, lints, builds, and pushes the Docker image to a registry (Docker Hub) on every `git push` to the `main` branch.

Deployment: A simple deployment script (`deploy.sh`) pulls the latest container image from the registry and runs it on a target server, completing the automated lifecycle from code commit to a running production service.

Step-by-Step Instructions to Build the Pipeline

Follow these commands to replicate the entire pipeline.

Part 0: Prerequisites

Clone the Repository:

```
git clone <your-repository-url>
```

```
cd <repository-name>
```

Install Tools: Ensure you have Python 3.9+, Git, and Docker Desktop installed and running.

Create and Activate a Virtual Environment:

```
python3 -m venv venv
source venv/bin/activate  # On macOS/Linux
# venv\Scripts\activate   # On Windows
```

Install Dependencies:

```
pip install --upgrade pip
pip install -r requirements.txt
```

Part 1: Data Versioning

Fetch and Preprocess Data:

```
python src/preprocess.py
```

Initialize DVC and Track Data:

```
dvc init
dvc add data/raw/housing.csv
git add data/raw/housing.csv.dvc .gitignore
git commit -m "feat: track raw data with DVC"
```

Part 2: Model Training & Registration

Run the Training Script: This trains multiple models and registers the best one.

```
python src/train.py
```

View Experiments (Optional):

```
mlflow ui
```

Open `http://127.0.0.1:5000` in your browser. Go to the **Models** tab to see your registered model.

Part 3 & 5: API, Docker, and Monitoring

Build the Docker Image:

```
docker build -t housing-api .
```

Run the Docker Container:

```
docker run -d -p 8001:8000 --name housing-predictor housing-api
```

Test the Running Service:

API Prediction:

```
curl -X 'POST' \
  'http://localhost:8001/predict/' \
  -H 'Content-Type: application/json' \
  -d '{"MedInc": 8.3, "HouseAge": 41, "AveRooms": 7, "AveBedrms": 1,
"Population": 322, "AveOccup": 2.5, "Latitude": 37.88, "Longitude":
-122.23}'
```

Check Logs:

```
docker exec housing-predictor cat api_log.log
```

View Metrics: Open `http://localhost:8001/metrics` in your browser.

Part 4: CI/CD Setup and Deployment

Set Up GitHub & Docker Hub:

Create a public repository on [Docker Hub](https://hub.docker.com).

In your GitHub repository, go to `Settings > Secrets and variables > Actions`.

Create two repository secrets: `DOCKER_USERNAME` (your Docker Hub username) and `DOCKER_PASSWORD` (your Docker Hub password or access token).

Trigger the CI/CD Pipeline: Commit all your code and push it to GitHub. This will automatically trigger the workflow.

```
git add .
git commit -m "feat: complete initial pipeline setup"
git push origin main
```

Go to the **Actions** tab on your GitHub repository to watch the pipeline run.

Deploy the Latest Version: Once the pipeline succeeds, run the deployment script.

Important: Edit `deploy.sh` and replace `"your-dockerhub-username"` with your actual username.

Make the script executable:

```
chmod +x deploy.sh
```

Run the deployment:

```
./deploy.sh
```

Your updated service is now running and available at `http://localhost:8001`.