# ML OPS: **Build, Track, Package, Deploy and Monitor an ML Model**

|  |  |  |
| --- | --- | --- |
| **Name** | **BITS ID** | **Contribution** |
| SUBHRANSU MISHRA | 2023AC05489 | 100% |
| DULAL DAS | 2023AC05041 | 100% |
| LAKSHMISRINIVAS PERAKAM | 2023AC05540 | 100% |
| SHAILESH KUMAR SINGH | 2023AC05475 | 100% |

## Selected Application Area: **California Housing dataset**

## For this project, I have selected the **California Housing dataset** from the **StatLib repository** (originally from the 1990 U.S. Census data) as the foundation for building and deploying a machine learning regression model. The dataset contains demographic and geographic information for various districts in California, with the **median house value** as the target variable.

## Deliverables / Links to Work:

|  |  |  |
| --- | --- | --- |
| 01 | ***GitHub: For Code and Pipeline*** | Access Link |
| 02 | ***Docker-Hub: For Image of the running application*** | Access Link |
| 03 | ***Walkthrough Video: Explaining the entire work*** | Access Link |
| 04 | ***Architecture Diagram of Lifecycle Phases*** | Access Link |
| 05 | ***Link to Dataset:* California Housing** | Access Link |

## Technologies Used:

* **Git + GitHub** – Managed version control for code, configuration, and pipeline scripts, enabling collaborative development and history tracking.
* **DVC** – Versioned and tracked the California Housing dataset, intermediate artifacts, and pipeline stages for reproducibility and efficient storage.
* **MLflow** – Tracked model training experiments, logged parameters/metrics, and registered the best-performing regression model in the model registry.
* **Docker** – Containerized the FastAPI prediction service for consistent deployment across environments with a lightweight, reproducible runtime.
* **FastAPI** – Built the REST API to serve real-time house price predictions with Pydantic-based input validation and auto-generated API documentation.
* **GitHub Actions** – Automated CI/CD pipeline for linting, testing, pulling data from DVC remote, reproducing the pipeline, building, and pushing Docker images to Docker Hub.
* **Logging Module** – Implemented structured logging to capture incoming prediction requests, model outputs, and errors, with logs persisted to files and SQLite.
* **Prometheus + Grafana** – Collected and visualized application performance metrics, request latency, error rates, and model version changes via custom metrics and dashboards.
* **Pydantic** – Enforced schema validation for API inputs to ensure data integrity before prediction.
* **SQLite** – Stored API request/response logs in a lightweight database for auditing and analysis.

## Project Architecture Overview

For Visual representation, please access this link : <https://github.com/DulalDas59/mlops-california-housing-pipeline/blob/main/docs/Architecture.md>

This project follows all main steps of the machine learning lifecycle, and each step is implemented with proper tools so that the system is easy to use, maintain, and monitor.

1. **Data & Pipeline Versioning**
   * The California Housing dataset is stored and versioned using **DVC**.
   * Both raw and processed data are saved, and a copy is kept in remote storage so the work can be reproduced anytime.
2. **Model Training & Experiment Tracking**
   * Two models (Linear Regression, Decision Tree Regressor) are trained from scripts in the src/ folder.
   * **MLflow** keeps a record of parameters, accuracy, and model files. The best model is stored in the MLflow Model Registry.
3. **Continuous Integration & Deployment (CI/CD)**
   * **GitHub Actions** automatically runs code checks, pulls data from DVC, runs the pipeline, builds a Docker image, pushes it to Docker Hub, and deploys the container.
4. **Model Serving**
   * A **FastAPI** service inside Docker serves the model with these endpoints:
     + /predict – for predictions
     + /healthz – for service health check
     + /retrain – to retrain when new data is available (API key required)
     + /metrics – to share performance data with Prometheus
5. **Monitoring & Logging**
   * **Prometheus** collects metrics like number of requests, speed of predictions, errors, retraining count, and model version changes.
   * **Grafana** shows these metrics on easy-to-read dashboards.
   * All requests and responses are logged into files and an SQLite database for later review.
6. **Model Retraining**
   * If new data is added in DVC or the /retrain API is called, the system trains a new model, logs it in MLflow, updates the registry, and starts using the new model without stopping the service.

## Project Workflow & Phases

### ****Phase 1 – Repository and Data Versioning****

* Created GitHub repository with clean directory structure (src/, data/, models/, api/, .github/, grafana/).
* Loaded California Housing dataset from sklearn.datasets.
* Used **DVC** to track raw dataset, processed files, and model artifacts.
* Configured DVC remote for storage and ensured .dvc files are committed while large files are excluded from Git.

### ****Phase 2 – Model Development & Experiment Tracking****

* Trained two regression models: **Linear Regression** and **Decision Tree Regressor**.
* Logged experiments in **MLflow** (parameters, metrics, artifacts, models). -> Screen Shots
* Selected the best model based on RMSE and R² scores. -> Screen Shots
* Registered the best model in the MLflow Model Registry for deployment. -> Screen Shots

**Phase 3 – API & Docker Packaging**

* Built a **FastAPI** service exposing /predict, /healthz, /retrain, and /metrics endpoints.
* Added **Pydantic** input validation to ensure correct data format before predictions. -> Screen Shots
* Containerized the API using **Docker** with a lightweight Python base image.

**Phase 4 – CI/CD with GitHub Actions**

* Configured GitHub Actions workflow to:
  + Pull dataset from DVC remote and reproduce the pipeline (dvc repro).
  + Run linting and automated tests.
  + Build Docker image and push to Docker Hub.
* Ran container image locally for inferencing, same can be deployed in any cloud service like AWS, Azure etc. -> Screen Shots

**Phase 5 – Logging and Monitoring**

* Implemented logging using Python’s logging module, with logs stored both in files and **SQLite**. -> Check If used or only files
* Added **Prometheus** metrics for request count, prediction latency, error rates, and retraining events. -> Screen Shots
* Created a **Grafana** dashboard to visualize metrics over time. -> Screen Shots

**Bonus Implementation**

In this project, we have completed all three bonus tasks. These features make the system more reliable, easier to monitor, and ready for future updates:

* **Input Validation with Pydantic** –  
  We used **Pydantic** in the /predict API to check that the input data is in the correct format before making a prediction.  
  It checks for correct data types, value ranges, and that all required fields are present.  
  This helps in avoiding wrong or incomplete data from going into the model. -> Screen Shots
* **Integration with Prometheus & Grafana** –  
  We connected the API to **Prometheus** using prometheus-fastapi-instrumentator.  
  This collects important metrics like:
  + Total number of prediction requests
  + Time taken for each prediction
  + Number of retraining jobs done
  + Number of failed predictions  
    These metrics are shown in a **Grafana dashboard**, so we can see in real time how the system is performing, whether there are errors, and how fast predictions are happening. -> Screen Shots
* **Model Re-Training on New Data** –  
  We added a secure /retrain API (with API key protection) which runs the full training process again whenever new data is available in the DVC storage.  
  After retraining:
  + New performance metrics are saved in MLflow.
  + The new model is stored with a version number in the model registry.
  + The API starts using the latest model without stopping the service. -> This Section needs change

## Conclusion:

This project demonstrates a complete and practical MLOps workflow for the California Housing regression problem, using proven tools and best practices.  
From **data versioning with DVC** to **experiment tracking in MLflow**, and from **automated CI/CD via GitHub Actions** to **real-time monitoring with Prometheus and Grafana**, every phase of the machine learning lifecycle has been implemented in a reproducible and maintainable way.

The use of **FastAPI** for model serving ensures fast, reliable predictions with proper input validation, while **Docker** guarantees consistent deployment across environments.  
Logging into both files and SQLite provides transparency and easy debugging, and the retraining mechanism ensures the model can adapt quickly to new data without downtime.

Overall, this pipeline is not just a one-time solution but a **scalable, production-ready framework** that can be extended to other datasets and ML problems. It highlights the importance of automation, reproducibility, and monitoring in delivering reliable machine learning services — key elements for any modern data-driven application.