**Heart Disease Prediction System – End-to-End MLOps Report**

**1. Introduction**

This document describes the design, implementation, and deployment of an end-to-end MLOps pipeline for heart disease prediction. The project demonstrates the full machine learning lifecycle, including data ingestion, preprocessing, exploratory analysis, model training, experiment tracking, CI/CD automation, containerization, Kubernetes deployment, and monitoring.

The objective is not only to build an accurate model, but also to ensure **reproducibility, automation, scalability, and observability**, which are core principles of modern machine learning engineering.

**2. System Overview**

The system is composed of the following high-level components:

* Data ingestion and preprocessing pipeline
* Machine learning model training and evaluation
* Experiment tracking with MLflow
* CI/CD automation using GitHub Actions
* REST API for inference using FastAPI
* Containerization using Docker
* Deployment on local Kubernetes
* Monitoring and logging of API requests

Each component is designed to be modular and reproducible.

**3. Setup & Installation Instructions**

**3.1 Prerequisites**

* Python 3.11+
* Git
* Docker
* Kubernetes (Docker Desktop Kubernetes or Minikube)
* pip / virtual environment

**3.2 Repository Setup**

**git clone https://github.com/2024aa05018/heart-disease-mlops.git**

**cd heart-disease-mlops**

Create and activate virtual environment:

python -m venv .venv

source .venv/bin/activate

Install dependencies:

pip install -r requirements.txt

**3.3 Data Preparation**

Raw data is **not committed** to version control. Instead, the pipeline uses reproducible scripts.

python -m src.download\_data

python -m src.preprocess

This produces a cleaned dataset under data/processed/.

**4. Dataset Description**

The project uses the **UCI Heart Disease dataset**, containing clinical features such as:

* Age
* Sex
* Chest pain type
* Resting blood pressure
* Cholesterol
* ECG results
* Maximum heart rate
* Exercise-induced angina

The target variable is binary:

* 0: No heart disease
* 1: Presence of heart disease

**5. Exploratory Data Analysis (EDA)**

EDA was conducted to understand the data distribution, feature relationships, and class imbalance.

**5.1 Class Distribution**

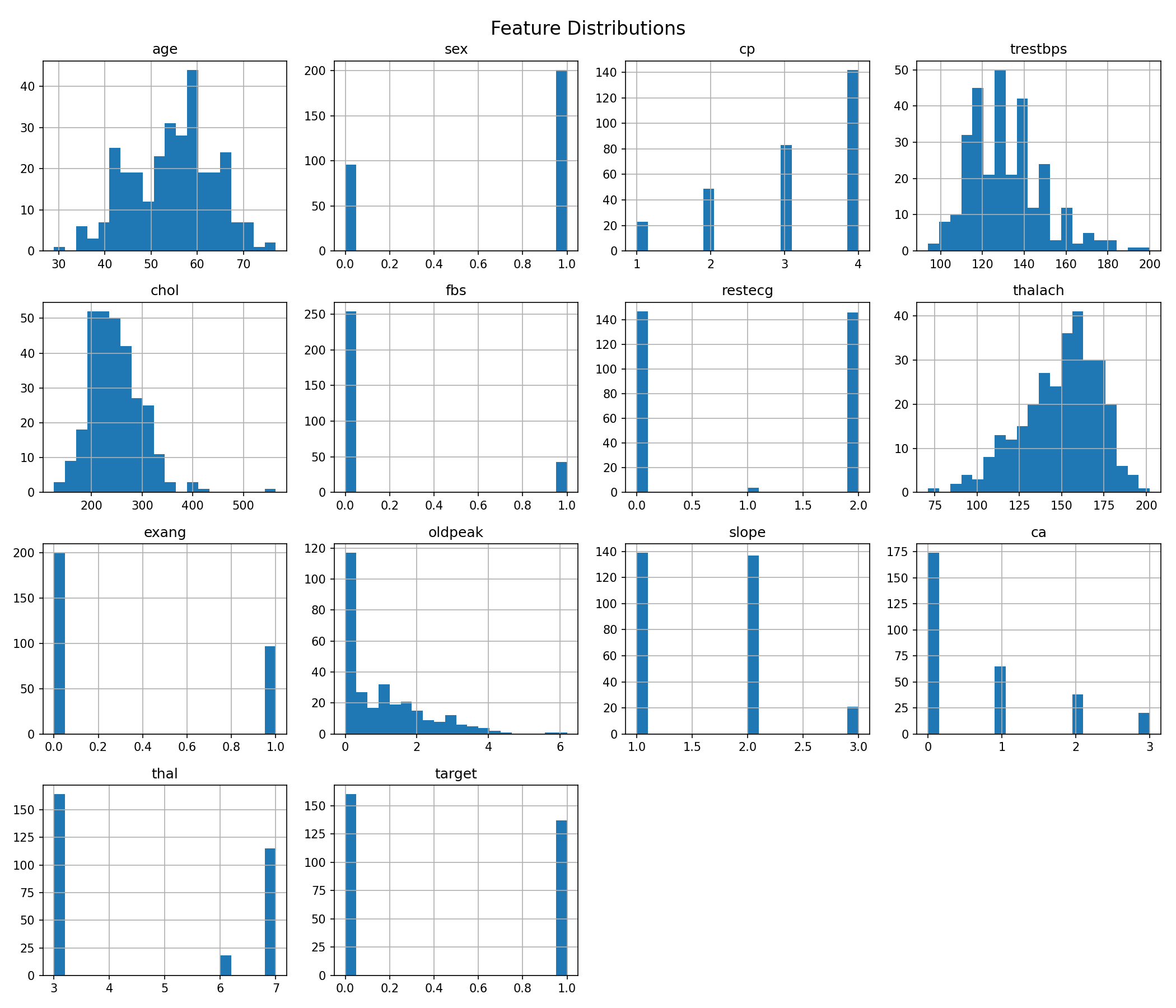
A class distribution plot shows the balance between positive and negative cases.

A graph of heart disease class distribution

AI-generated content may be incorrect.

**5.2 Feature Distributions**

Histograms were generated for all numerical features to analyze spread, skewness, and outliers.



**5.3 Correlation Analysis**

A correlation heatmap was used to identify relationships between features and the target.

A graph of a heatmap

AI-generated content may be incorrect.

**6. Data Preprocessing**

Preprocessing steps include:

* Handling missing values
* Converting categorical fields to numeric form
* Normalizing the target label
* Feature scaling using StandardScaler

All preprocessing logic is encapsulated in reusable Python scripts and pipelines.

**7. Model Training & Evaluation**

Two models were trained:

* Logistic Regression
* Random Forest Classifier

**7.1 Evaluation Metrics**

* Accuracy
* Precision
* Recall
* ROC-AUC

Cross-validation was used to ensure robust evaluation.

**7.2 Model Selection**

Logistic Regression was selected as the final model due to:

* Strong ROC-AUC score
* Simplicity and interpretability
* Lower risk of overfitting

**8. Experiment Tracking (MLflow)**

MLflow was used to track:

* Hyperparameters
* Metrics
* Model artifacts

Each training run is logged under a dedicated experiment.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

MLflow tracking was configured to use a project-local directory for compatibility with CI environments.

**9. Model Packaging**

The final model was packaged as a single pipeline combining preprocessing and classification.  
The pipeline was serialized using joblib, ensuring consistent inference across environments.

The packaged model is stored under:

models/final\_model.joblib

**10. CI/CD Pipeline**

A CI/CD pipeline was implemented using **GitHub Actions**.

**10.1 Pipeline Stages**

* Code checkout
* Dependency installation
* Linting using flake8
* Unit testing using pytest
* Data download and preprocessing
* Model training

**10.2 CI/CD Design Decisions**

* Tests use **synthetic data** to avoid dependency on external datasets
* Data is generated dynamically to ensure clean CI environments
* Pipeline fails on test or lint errors

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**11. API Development**

A REST API was built using **FastAPI**.

**11.1 Endpoints**

* /predict: Performs inference on input features
* /metrics: Returns basic monitoring metrics
* /docs: Swagger UI

**11.2 Inference Workflow**

1. Input JSON is validated
2. Preprocessing pipeline is applied
3. Model predicts probability
4. Prediction and confidence are returned

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**12. Containerization**

The application was containerized using Docker.

**12.1 Docker Components**

* Python base image
* Installed dependencies
* API code
* Trained model

The container exposes port 8000.

A screenshot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**13. Kubernetes Deployment**

The Dockerized API was deployed on **local Kubernetes**.

**13.1 Kubernetes Resources**

* Deployment
* ClusterIP Service

Manifests are stored under k8s/.

**13.2 Access Instructions**

kubectl apply -f k8s/

kubectl port-forward service/heart-disease-service 8000:80

A close up of a white background

AI-generated content may be incorrect.

A white background with black text

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**14. Monitoring & Logging**

**14.1 Logging**

FastAPI middleware logs:

* HTTP method
* Endpoint
* Status code
* Request latency

Logs are observable using:

kubectl logs <pod-name>

**14.2 Metrics**

A /metrics endpoint exposes:

* Total request count
* Average response latency

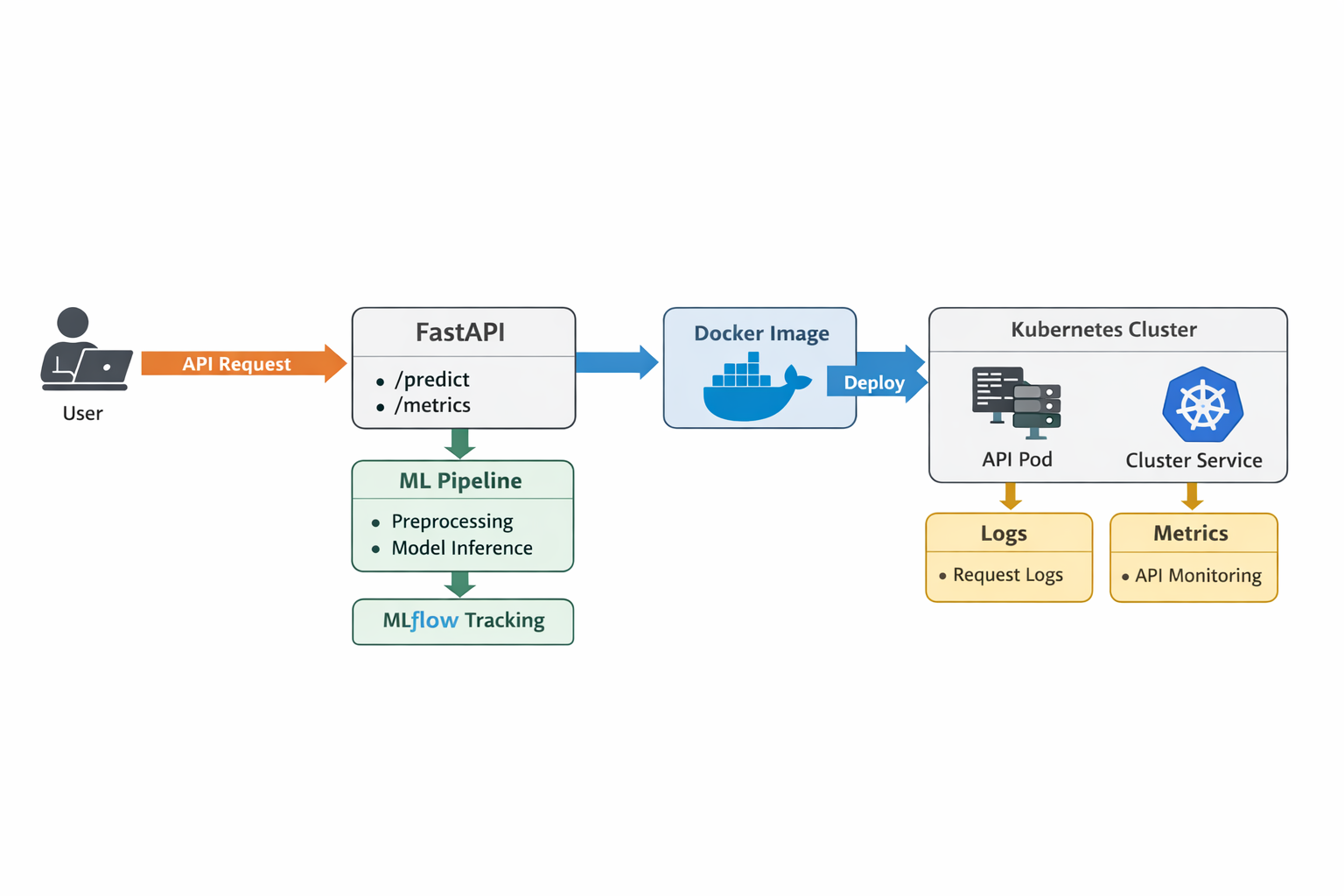
A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**15. Architecture Diagram**

**Description:**  
User → FastAPI → ML Pipeline → MLflow  
FastAPI → Docker → Kubernetes  
Kubernetes → Logs & Metrics

**16. Project Structure**

heart-disease-mlops/

├── api/

├── src/

├── tests/

├── k8s/

├── models/

├── reports/

├── screenshots/

├── .github/workflows/

├── Dockerfile

├── requirements.txt

├── README.md

**17. Repository Link**

[https://github.com/2024aa05018/heart-disease-mlops](https://github.com/2024aa05018/heart-disease-mlops?utm_source=chatgpt.com)

**18. Conclusion**

This project demonstrates a production-grade MLOps pipeline implementing best practices for automation, reproducibility, deployment, and monitoring. The system can be extended to cloud deployment, advanced monitoring stacks, and continuous retraining workflows.