

Heart Disease Risk Prediction: An End-to-End MLOps Pipeline

S.No	Register Number	Name	Contribution
1	2024aa05421	Krithika Madhavan	100%
2	2024aa05435	Yarragondla Rugmangadha Reddy	100%
3	2024aa05423	Payel Karmakar	100%
4	2024aa05870	Deepak Sindhu	100%
5	2024ab05227	PARAB PRATHAMESH PRAFULLA PRADNYA	100%

1. Project Overview

- This project implements an end-to-end MLOps pipeline for predicting the presence of heart disease using patient health data. The objective is to design, build, and deploy a scalable and reproducible machine learning solution by applying modern MLOps practices across the entire model lifecycle.
- The solution uses the **Heart Disease UCI Dataset**, which contains clinical features such as age, sex, blood pressure, cholesterol levels, electrocardiographic results, and other medically relevant indicators. The target variable is binary, representing the presence or absence of heart disease.
- The project covers all major stages of an industrial machine learning workflow, including data acquisition and exploratory data analysis, feature engineering, model training and evaluation, experiment tracking, CI/CD automation, containerized model serving, deployment on Kubernetes, and runtime monitoring. Multiple classification models are trained and compared, with performance evaluated using standard metrics such as accuracy, precision, recall, and ROC-AUC.
- To ensure reproducibility and automation, **MLflow** is integrated for experiment tracking and artifact management, while **GitHub Actions** is used to implement a continuous integration pipeline that performs linting, testing, data preparation, and model training. The final trained model is exposed through a **FastAPI-based REST API**, containerized using Docker, and deployed on a Kubernetes cluster using **Minikube**. Request-level logging and latency monitoring are implemented to provide basic production observability.

- An optional **Streamlit-based user interface** was implemented as a lightweight demonstration layer to interact with the deployed FastAPI service. The Streamlit application collects user inputs and sends them to the `/predict` endpoint, displaying the predicted risk and confidence score without performing inference locally.
- Overall, this project demonstrates a complete, production-aligned MLOps workflow that mirrors real-world machine learning systems, emphasizing automation, reproducibility, deployment readiness, and monitoring.

2. Setup & Installation Instructions

This section describes the steps required to set up the project locally, execute model training, and deploy the model-serving API.

2.1 System Requirements

- Operating System: Linux / macOS / Windows
 - Python Version: Python 3.10 or higher
 - Package Manager: pip
 - Containerization: Docker
 - Orchestration: Minikube and kubectl
 - Optional UI: Streamlit
-

2.2 Clone the Repository

```
git clone https://github.com/Krithika-Madhavan-5421/MLOPS-ASSIGNMENT1-GROUP44.git
cd heart-disease-mlops
```

2.3 Create and Activate a Conda Environment

A Conda environment is used to isolate project dependencies and ensure reproducibility across different systems.

Create a new Conda environment:

```
conda create -n heart-disease-mlops python=3.10 -y
```

2.4 Install Project Dependencies

Install all required Python dependencies using the provided requirements.txt file.

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

2.5 Data Acquisition and Exploratory Data Analysis

The dataset is automatically fetched from the UCI Machine Learning Repository.

Run the EDA script:

```
python src/eda.py
```

This step:

- Downloads the Heart Disease UCI dataset
- Cleans and preprocesses the data
- Converts the target variable into a binary label
- Saves the cleaned dataset to data/heart.csv
- Generates EDA visualizations under artifacts/plots/

2.6 Model Training and Experiment Tracking

Train the machine learning models using the training script:

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

This step:

- Loads the cleaned dataset
- Applies feature preprocessing using a unified pipeline
- Trains Logistic Regression and Random Forest models
- Evaluates models using accuracy, precision, recall, and ROC-AUC
- Logs metrics and model artifacts using MLflow

To view experiments and metrics:

```
mlflow ui
```

Access the MLflow dashboard at:

```
http://localhost:5000
```

2.7 Running the FastAPI Inference Service Locally

The trained model is served through a FastAPI-based REST API.

```
uvicorn src.app:app --host 0.0.0.0 --port 8000
```

Swagger API documentation is available at:

```
http://localhost:8000/docs
```

2.8 Docker Image Build and Local Execution

Build the Docker image for the inference service:

```
docker build -t heart-disease-api:latest .
```

Run the Docker container locally:

```
docker run -p 8000:8000 heart-disease-api:latest
```

2.9 Kubernetes Deployment Using Minikube

Start the Minikube cluster:

```
minikube start
```

Deploy the application to Kubernetes:

```
kubectl apply -f manifests/deployment.yaml  
kubectl apply -f manifests/service.yaml
```

Verify deployment status:

```
kubectl get pods  
kubectl get svc
```

Access the deployed service:

```
minikube service heart-disease-service --url
```

Swagger API documentation is available at:

```
http://192.168.49.2:30533
```

2.10 Streamlit Demonstration Interface

An optional Streamlit-based user interface is provided as a demonstration layer.

```
streamlit run streamlit_app.py
```

The Streamlit UI collects user inputs, calls the FastAPI /predict endpoint, and displays the predicted risk and confidence score.

2.11 CI/CD Pipeline Execution

A CI/CD pipeline is implemented using GitHub Actions and is automatically triggered on push events and pull requests.

The pipeline performs:

- Code linting using flake8
 - Unit testing using pytest
 - Data preparation and EDA
 - Model training
 - Artifact logging and upload
-

3. Exploratory Data Analysis (EDA) and Modelling Choices

3.1 Exploratory Data Analysis

Exploratory Data Analysis was performed to understand the distribution, relationships, and quality of the dataset before model training. The following steps were carried out:

- Checked for missing and invalid values and removed incomplete records.
- Analyzed the class distribution of the target variable to understand class balance.
- Visualized feature distributions using histograms.
- Examined correlations between numerical features and the target variable using a correlation heatmap.

These analyses helped in identifying relevant features and understanding their impact on heart disease prediction.

3.2 Feature Engineering

- Numerical features were scaled to ensure consistent ranges.
- Categorical features were encoded appropriately.
- A unified preprocessing pipeline was created using a `ColumnTransformer` to ensure the same transformations are applied during both training and inference.

3.3 Model Selection and Training

Two classification models were trained and evaluated:

- **Logistic Regression:** Chosen as a baseline model due to its interpretability and efficiency for binary classification problems.
- **Random Forest Classifier:** Selected to capture non-linear relationships and feature interactions.

3.4 Evaluation Metrics

Models were evaluated using the following metrics:

- Accuracy
- Precision
- Recall
- ROC-AUC

These metrics provide a balanced assessment of classification performance, especially for medical prediction tasks.

4. Experiment Tracking Summary

MLflow was integrated into the project to track experiments, metrics, and trained model artifacts.

4.1 Tracked Components

For each training run, MLflow logs:

- Model parameters
- Evaluation metrics (accuracy, precision, recall, ROC-AUC)
- Serialized model pipeline as an artifact

4.2 Experiment Organization

- Separate experiments were maintained for CI-based training to ensure portability and avoid environment-specific conflicts.
 - **Figure 4.1:** MLflow experiment overview

The screenshot shows the MLflow UI for the 'Heart Disease Prediction' experiment. The interface includes a search bar, filters for time, state, and datasets, and tabs for Table, Chart, Evaluation, and Experimental. The experimental tab is selected, showing a table with the following data:

	Run Name	Created	Dataset	Duration	Source	Models
<input type="checkbox"/>	RandomForest	2 days ago	-	3.2s	<input type="checkbox"/> train.py	<input checked="" type="checkbox"/> sklearn
<input type="checkbox"/>	LogisticRegression	2 days ago	-	4.9s	<input type="checkbox"/> train.py	<input checked="" type="checkbox"/> sklearn

2 matching runs

- Each model run is stored as an independent MLflow run, enabling easy comparison between different models.
 - **Random Forest Model**

The screenshot shows the mlflow UI interface. At the top, there is a navigation bar with the mlflow logo (2.11.4), Experiments, Models, and a user section for Rocky Mattermost (<http://chat.rockylinux.org/>). On the right side of the top bar are settings, GitHub, and Docs links. Below the navigation bar, the page title is "Heart Disease Prediction > RandomForest". On the far right, there is a "Register model" button and a three-dot menu icon.

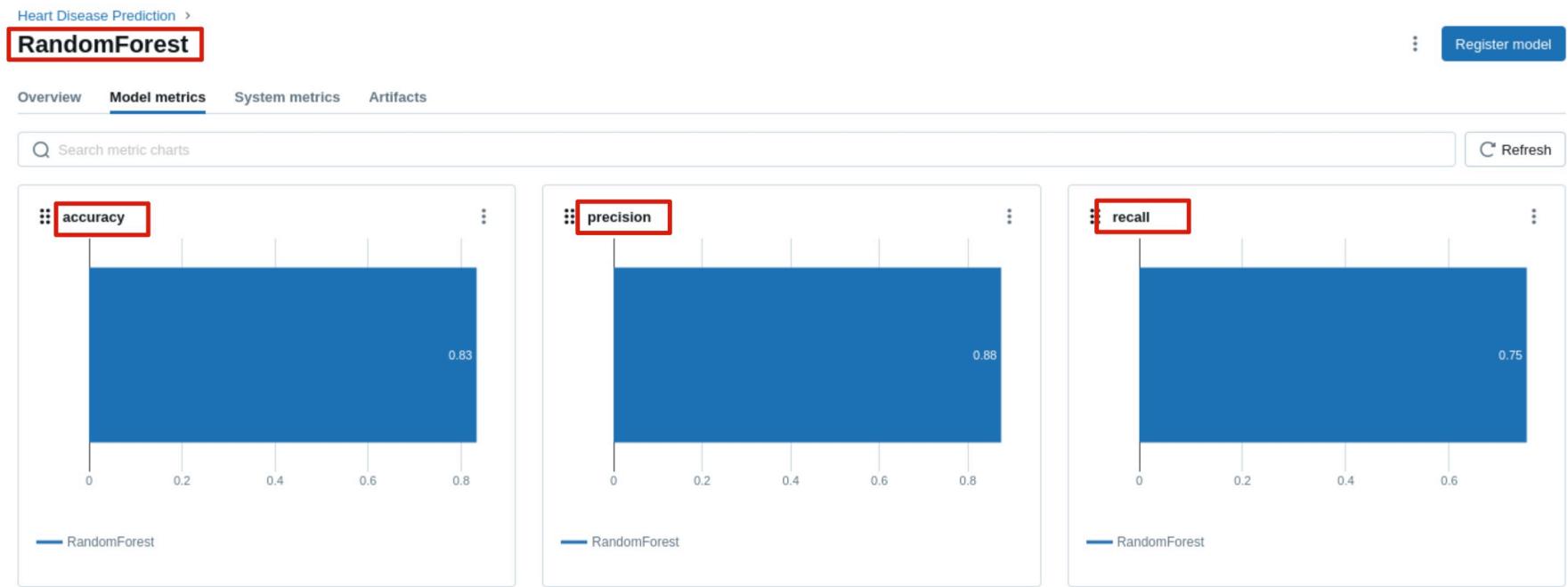
The main content area has tabs for Overview, Model metrics, System metrics, and Artifacts. The Overview tab is selected. Under the Description section, it says "No description". The Details section contains the following information:

Created at	2026-01-02 11:53:01
Created by	cloud
Status	Finished
Run ID	13763c2f91164f85a927b6194afb8b3b
Duration	3.2s
Datasets used	—
Tags	Add

The "Source" row is highlighted with a red box and contains the following details:

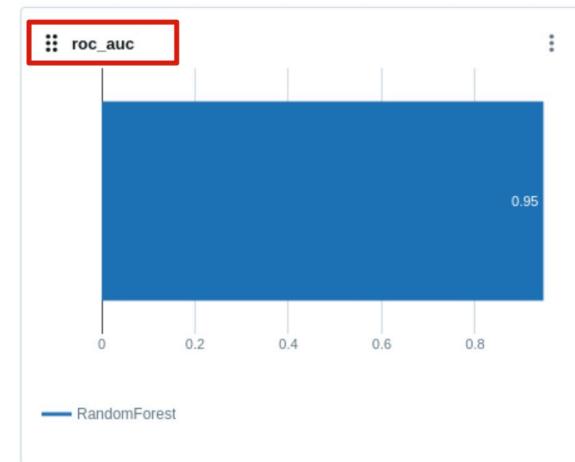
- train.py
- c76b3df600cac4ffa60e38997c2c8865f0d4e2c3

At the bottom left, there is a link to <http://chat.rockylinux.org> and a sklearn icon.

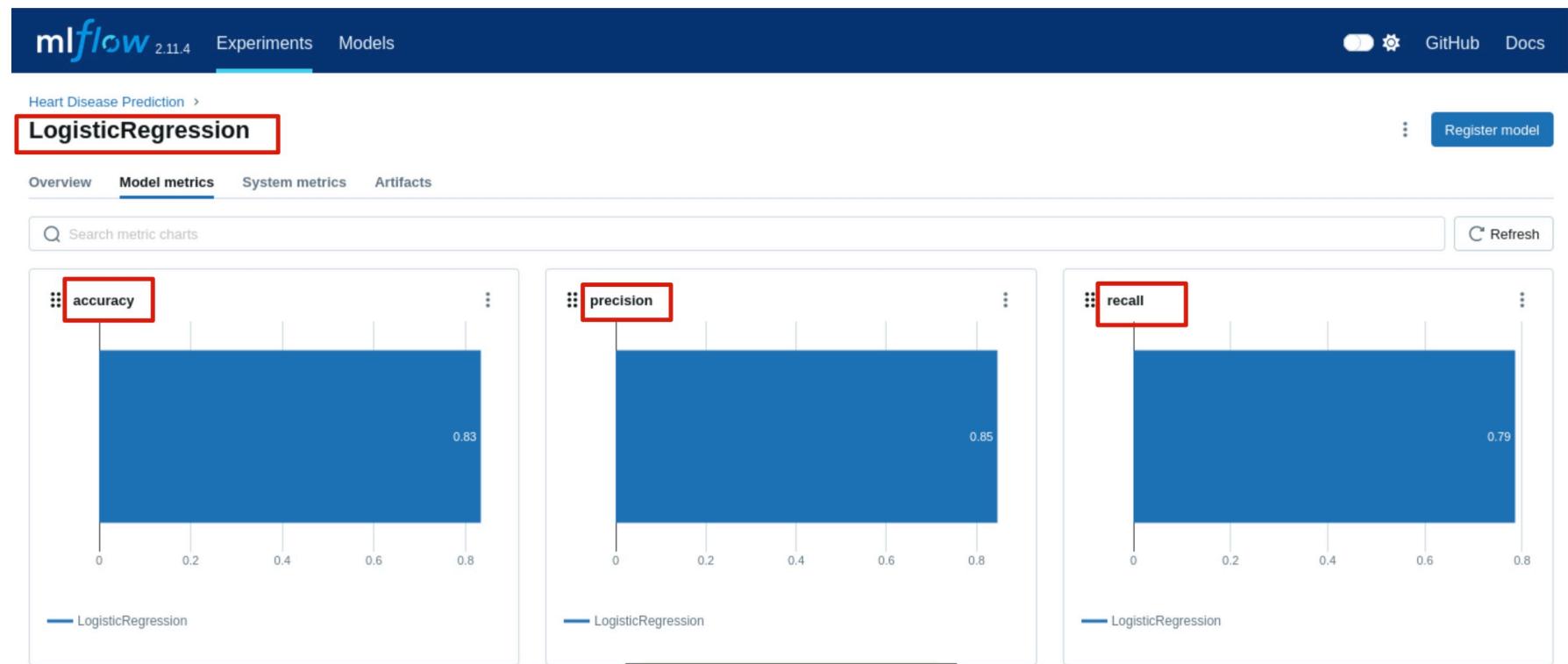


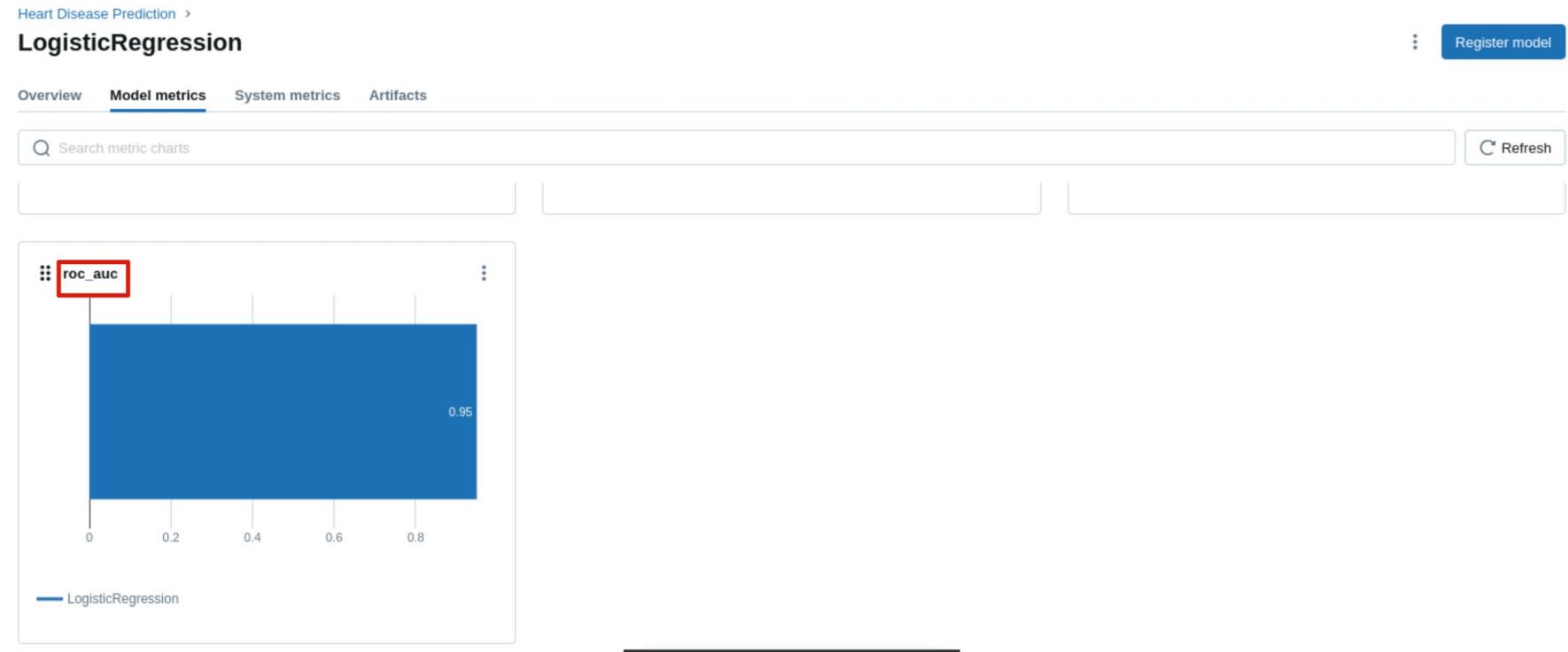
Heart Disease Prediction >

RandomForest

[Register model](#)[Overview](#) [Model metrics](#) [System metrics](#) [Artifacts](#) Search metric charts[Refresh](#)

- o **Logistic Regression Model**





4.3 Benefits

Using MLflow enables:

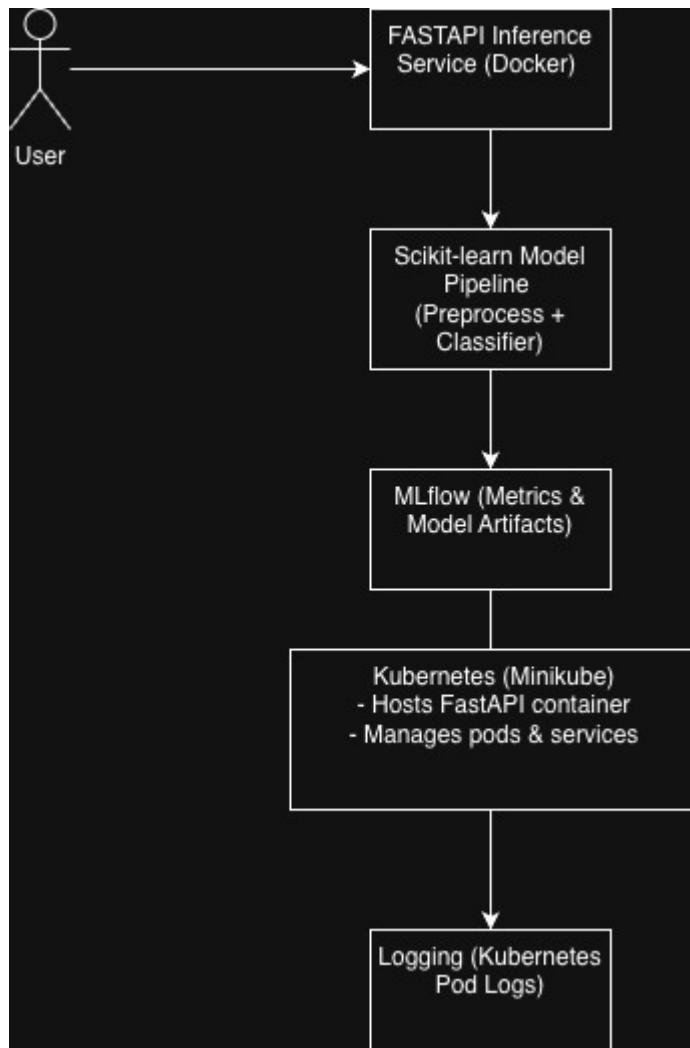
- Reproducibility of experiments
- Traceability of model versions
- Easy comparison of multiple model runs

Screenshots of the MLflow UI showing logged runs and metrics are included in the report.

5. Architecture Diagram

The overall system architecture of the project is shown below:

Figure 5.1: End-to-end system architecture illustrating data flow from ingestion to deployment and inference.



6. CI/CD and Deployment Workflow

6.1 CI/CD Pipeline

A CI/CD pipeline was implemented using GitHub Actions. The pipeline is triggered automatically on pushes and pull requests and performs the following steps:

1. Code checkout

← MLOps CI Pipeline

Ignore mlruns and stabilize MLflow CI tracking #11

Annotations
2 warnings

lint-test-train
succeeded 2 days ago in 56s

Search logs

Checkout code

```
1 ► Run actions/checkout@v4
16 Syncing repository: Krithika-Madhavan-5421/MLOPS-ASSIGNMENT1-GROUP44
17 ► Getting Git version info
21 Temporarily overriding HOME='/home/runner/work/_temp/600cacfc-65d4-4cba-b9de-c2c5e6db3244' before making global git config changes
22 Adding repository directory to the temporary git global config as a safe directory
23 /usr/bin/git config --global --add safe.directory /home/runner/work/MLOPS-ASSIGNMENT1-GROUP44/MLOPS-ASSIGNMENT1-GROUP44
24 Deleting the contents of '/home/runner/work/MLOPS-ASSIGNMENT1-GROUP44/MLOPS-ASSIGNMENT1-GROUP44'
25 ► Initializing the repository
42 ► Disabling automatic garbage collection
44 ► Setting up auth
52 ► Fetching the repository
56 ► Determining the checkout info
57 /usr/bin/git sparse-checkout disable
58 /usr/bin/git config --local --unset-all extensions.worktreeConfig
59 ► Checking out the ref
63 /usr/bin/git log -1 --format=%H
64 f00c016159ca640972fdf3f9d7ddf54c6312b95c
```

2. Dependency installation

The screenshot shows the GitHub Actions interface for a workflow named 'lint-test-train'. The 'lint-test-train' job has succeeded 2 days ago in 56s. The workflow consists of two main steps: 'Set up Python' and 'Install dependencies'. The 'Install dependencies' step is highlighted with a red box. The log output for this step shows the execution of pip commands to upgrade pip and install dependencies from requirements.txt, including packages like numpy, scipy, pandas, scikit-learn, matplotlib, and seaborn.

```
1 Run actions/setup-python@v5
9 Installed versions
1 ▶ Run pip install --upgrade pip
2 pip install --upgrade pip
3 pip install -r requirements.txt
4 pip install -e .
5 shell: /usr/bin/bash -e {0}
6 env:
7   pythonLocation: /opt/hostedtoolcache/Python/3.10.19/x64
8   PKG_CONFIG_PATH: /opt/hostedtoolcache/Python/3.10.19/x64/lib/pkgconfig
9   Python_ROOT_DIR: /opt/hostedtoolcache/Python/3.10.19/x64
10  Python2_ROOT_DIR: /opt/hostedtoolcache/Python/3.10.19/x64
11  Python3_ROOT_DIR: /opt/hostedtoolcache/Python/3.10.19/x64
12  LD_LIBRARY_PATH: /opt/hostedtoolcache/Python/3.10.19/x64/lib
13 Requirement already satisfied: pip in /opt/hostedtoolcache/Python/3.10.19/x64/lib/python3.10/site-packages (25.3)
14 Collecting numpy<1.29.0,>=1.22.4 (from -r requirements.txt (line 1))
15   Downloading numpy-1.26.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (61 kB)
16 Collecting scipy<1.12,>=1.10 (from -r requirements.txt (line 2))
17   Downloading scipy-1.11.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (60 kB)
18 Collecting pandas<2.3,>=1.5 (from -r requirements.txt (line 3))
19   Downloading pandas-2.2.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (89 kB)
20 Collecting scikit-learn<1.5,>=1.2 (from -r requirements.txt (line 4))
21   Downloading scikit_learn-1.4.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (11 kB)
22 Collecting matplotlib<3.9,>=3.7 (from -r requirements.txt (line 5))
23   Downloading matplotlib-3.8.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (5.8 kB)
24 Collecting seaborn<0.14,>=0.12 (from -r requirements.txt (line 6))
25   Downloading seaborn-0.13.2-py3-none-any.whl.metadata (5.4 kB)
26   Collecting mlflow>2.12 -->2.2 (from -r requirements.txt (line 7))
```

3. Code linting using flake8



The screenshot shows a CI pipeline interface with a sidebar on the left and a main log viewer on the right.

Left Sidebar:

- Summary
- All jobs
- lint-test-train** (selected, indicated by a blue bar)
- Run details
- Usage
- Workflow file

Main Log View:

Job Name: lint-test-train
Status: succeeded 2 days ago in 56s

Log Sections:

- Lint** (highlighted with a red box)
- Run flake8 src/
flake8 src/
shell: /usr/bin/bash -e {0}
env:
pythonLocation: /opt/hostedtoolcache/Python/3.10.19/x64
PKG_CONFIG_PATH: /opt/hostedtoolcache/Python/3.10.19/x64/lib/pkgconfig
Python_ROOT_DIR: /opt/hostedtoolcache/Python/3.10.19/x64
Python2_ROOT_DIR: /opt/hostedtoolcache/Python/3.10.19/x64
Python3_ROOT_DIR: /opt/hostedtoolcache/Python/3.10.19/x64
LD_LIBRARY_PATH: /opt/hostedtoolcache/Python/3.10.19/x64/lib

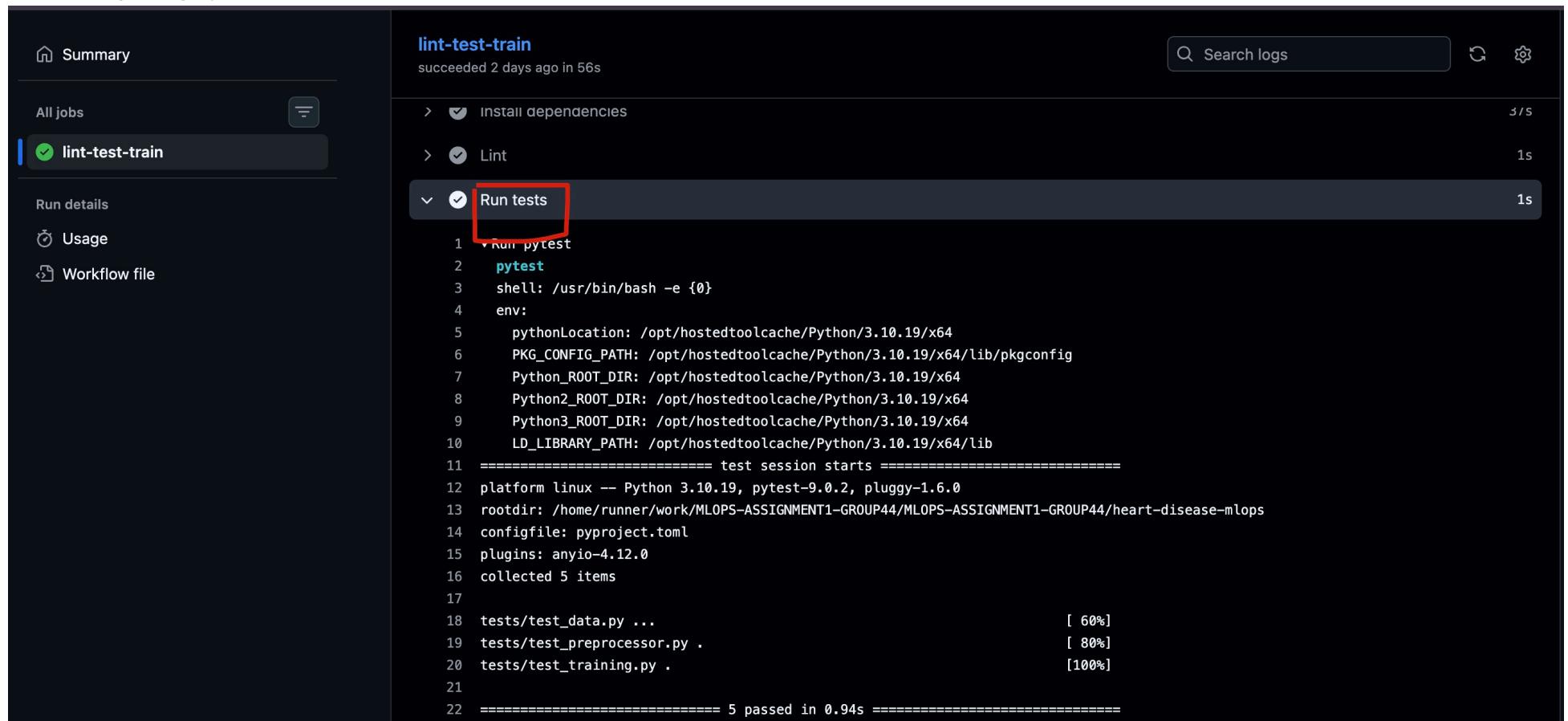
Top Right:

- Search logs
- Refresh
- Settings

Bottom Right:

1s

4. Unit testing using pytest



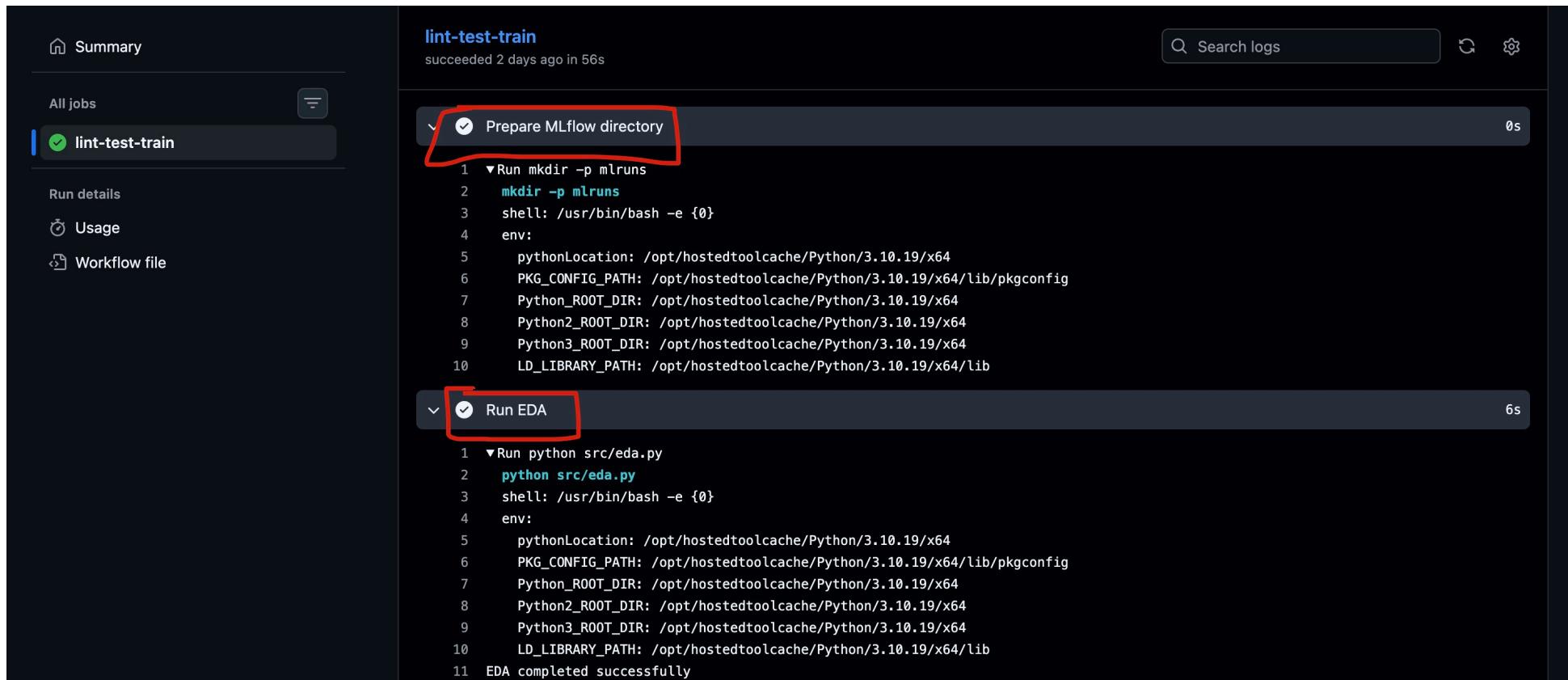
The screenshot shows a CI pipeline interface with a summary page on the left and a detailed log view on the right. The log view for the 'lint-test-train' job shows the following steps:

- > 🛠 Install dependencies
- > ✅ Lint
- ✓ Run tests (highlighted with a red box)

The 'Run tests' step details the command execution:

```
1 1 run pytest
2 2 pytest
3 shell: /usr/bin/bash -e {0}
4 env:
5   pythonLocation: /opt/hostedtoolcache/Python/3.10.19/x64
6   PKG_CONFIG_PATH: /opt/hostedtoolcache/Python/3.10.19/x64/lib/pkgconfig
7   Python_ROOT_DIR: /opt/hostedtoolcache/Python/3.10.19/x64
8   Python2_ROOT_DIR: /opt/hostedtoolcache/Python/3.10.19/x64
9   Python3_ROOT_DIR: /opt/hostedtoolcache/Python/3.10.19/x64
10  LD_LIBRARY_PATH: /opt/hostedtoolcache/Python/3.10.19/x64/lib
11 ===== test session starts =====
12 platform linux -- Python 3.10.19, pytest-9.0.2, pluggy-1.6.0
13 rootdir: /home/runner/work/MLOPS-ASSIGNMENT1-GROUP44/MLOPS-ASSIGNMENT1-GROUP44/heart-disease-mlops
14 configfile: pyproject.toml
15 plugins: anyio-4.12.0
16 collected 5 items
17
18 tests/test_data.py ... [ 60%]
19 tests/test_preprocessor.py . [ 80%]
20 tests/test_training.py . [100%]
21
22 ===== 5 passed in 0.94s =====
```

5. Data preparation and EDA execution



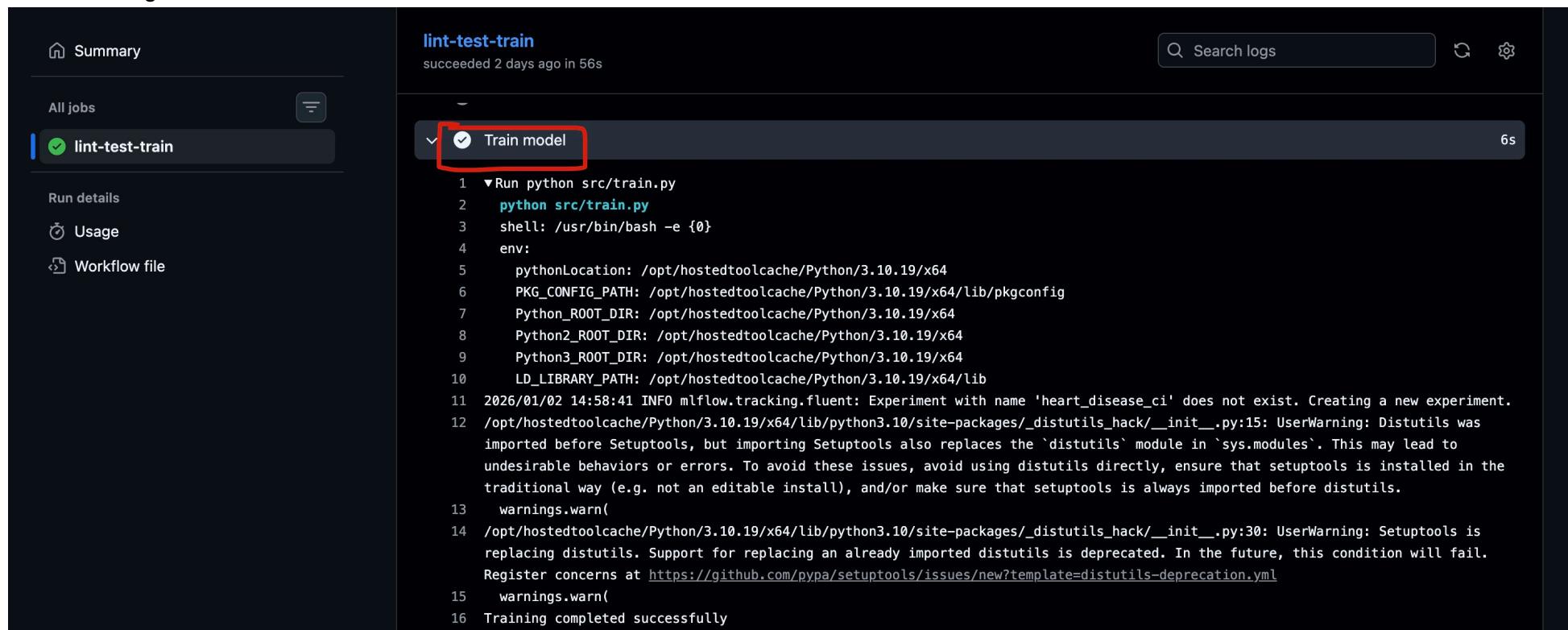
The screenshot shows a pipeline named "lint-test-train" that has completed successfully 2 days ago in 56s. The pipeline consists of two main steps:

- Prepare MLflow directory**: This step includes the command `mkdir -p mlruns`. It was run via a shell command with environment variables like `pythonLocation`, `PKG_CONFIG_PATH`, and `LD_LIBRARY_PATH`.
- Run EDA**: This step includes the command `python src/eda.py`. It was also run via a shell command with similar environment variables.

Both steps are marked as successful with green checkmarks and are highlighted with red boxes.

```
1 1 ▶Run mkdir -p mlruns
2 2  mkdir -p mlruns
3 3  shell: /usr/bin/bash -e {0}
4 4  env:
5 5  pythonLocation: /opt/hostedtoolcache/Python/3.10.19/x64
6 6  PKG_CONFIG_PATH: /opt/hostedtoolcache/Python/3.10.19/x64/lib/pkgconfig
7 7  Python_ROOT_DIR: /opt/hostedtoolcache/Python/3.10.19/x64
8 8  Python2_ROOT_DIR: /opt/hostedtoolcache/Python/3.10.19/x64
9 9  Python3_ROOT_DIR: /opt/hostedtoolcache/Python/3.10.19/x64
10 10 LD_LIBRARY_PATH: /opt/hostedtoolcache/Python/3.10.19/x64/lib
11 11 EDA completed successfully
```

6. Model training



The screenshot shows a pipeline interface with a sidebar on the left containing 'Summary', 'All jobs', 'lint-test-train' (selected), 'Run details', 'Usage', and 'Workflow file'. The main area displays a log for a job named 'lint-test-train' that succeeded 2 days ago in 56s. A red box highlights the 'Train model' step in the log list. The log content is as follows:

```
1  ▼Run python src/train.py
2  python src/train.py
3  shell: /usr/bin/bash -e {0}
4  env:
5    pythonLocation: /opt/hostedtoolcache/Python/3.10.19/x64
6    PKG_CONFIG_PATH: /opt/hostedtoolcache/Python/3.10.19/x64/lib/pkgconfig
7    Python_ROOT_DIR: /opt/hostedtoolcache/Python/3.10.19/x64
8    Python2_ROOT_DIR: /opt/hostedtoolcache/Python/3.10.19/x64
9    Python3_ROOT_DIR: /opt/hostedtoolcache/Python/3.10.19/x64
10   LD_LIBRARY_PATH: /opt/hostedtoolcache/Python/3.10.19/x64/lib
11  2026/01/02 14:58:41 INFO mlflow.tracking.fluent: Experiment with name 'heart_disease_ci' does not exist. Creating a new experiment.
12  /opt/hostedtoolcache/Python/3.10.19/x64/lib/python3.10/site-packages/_distutils_hack/_init__.py:15: UserWarning: Distutils was
     imported before Setuptools, but importing Setuptools also replaces the `distutils` module in `sys.modules`. This may lead to
     undesirable behaviors or errors. To avoid these issues, avoid using distutils directly, ensure that setuptools is installed in the
     traditional way (e.g. not an editable install), and/or make sure that setuptools is always imported before distutils.
13  warnings.warn(
14  /opt/hostedtoolcache/Python/3.10.19/x64/lib/python3.10/site-packages/_distutils_hack/_init__.py:30: UserWarning: Setuptools is
     replacing distutils. Support for replacing an already imported distutils is deprecated. In the future, this condition will fail.
     Register concerns at https://github.com/pypa/setuptools/issues/new?template=distutils-deprecation.yml
15  warnings.warn(
16  Training completed successfully
```

7. Artifact upload (MLflow runs and plots)

The screenshot shows a GitHub Actions pipeline named 'lint-test-train' that has just succeeded. The pipeline consists of two main steps:

- Upload MLflow artifacts**: This step uploaded 83 files. The log output for this step is highlighted with a red box and includes the following text:


```

1 ► Run actions/upload-artifact@v4
16
16 With the provided path, there will be 83 files uploaded
17 Artifact name is valid!
18 Root directory input is valid!
19 Beginning upload of artifact content to blob storage
20 Uploaded bytes 309092
21 Finished uploading artifact content to blob storage!
22 SHA256 digest of uploaded artifact zip is 4b7bd5a0652ded027e9542ea420ea701846551e7ebdc81c3e7d481533d24cfe3
23 Finalizing artifact upload
24 Artifact mlflow-artifacts.zip successfully finalized. Artifact ID 5016646950
25 Artifact mlflow-artifacts has been successfully uploaded! Final size is 309092 bytes. Artifact ID is 5016646959
26 Artifact download URL: https://github.com/Krithika-Madhavan-5421/MLOPS-ASSIGNMENT1-GROUP44/actions/runs/20689445220/artifacts/5016646959
      
```
- Upload EDA plots**: This step uploaded 3 files. The log output for this step is also highlighted with a red box and includes the following text:


```

1 ► Run actions/upload-artifact@v4
16
16 With the provided path, there will be 3 files uploaded
17 Artifact name is valid!
18 Root directory input is valid!
19 Beginning upload of artifact content to blob storage
20 Uploaded bytes 100669
21 Finished uploading artifact content to blob storage!
22 SHA256 digest of uploaded artifact zip is 6f03733d31f12f7097674cfb6d0eb06050dd6ec0f01afb71f5e72df7be6056ed
23 Finalizing artifact upload
24 Artifact eda-plots.zip successfully finalized. Artifact ID 5016646996
25 Artifact eda-plots has been successfully uploaded! Final size is 100669 bytes. Artifact ID is 5016646996
26 Artifact download URL: https://github.com/Krithika-Madhavan-5421/MLOPS-ASSIGNMENT1-GROUP44/actions/runs/20689445220/artifacts/5016646996
      
```

Screenshots of successful pipeline executions and uploaded artifacts are included.

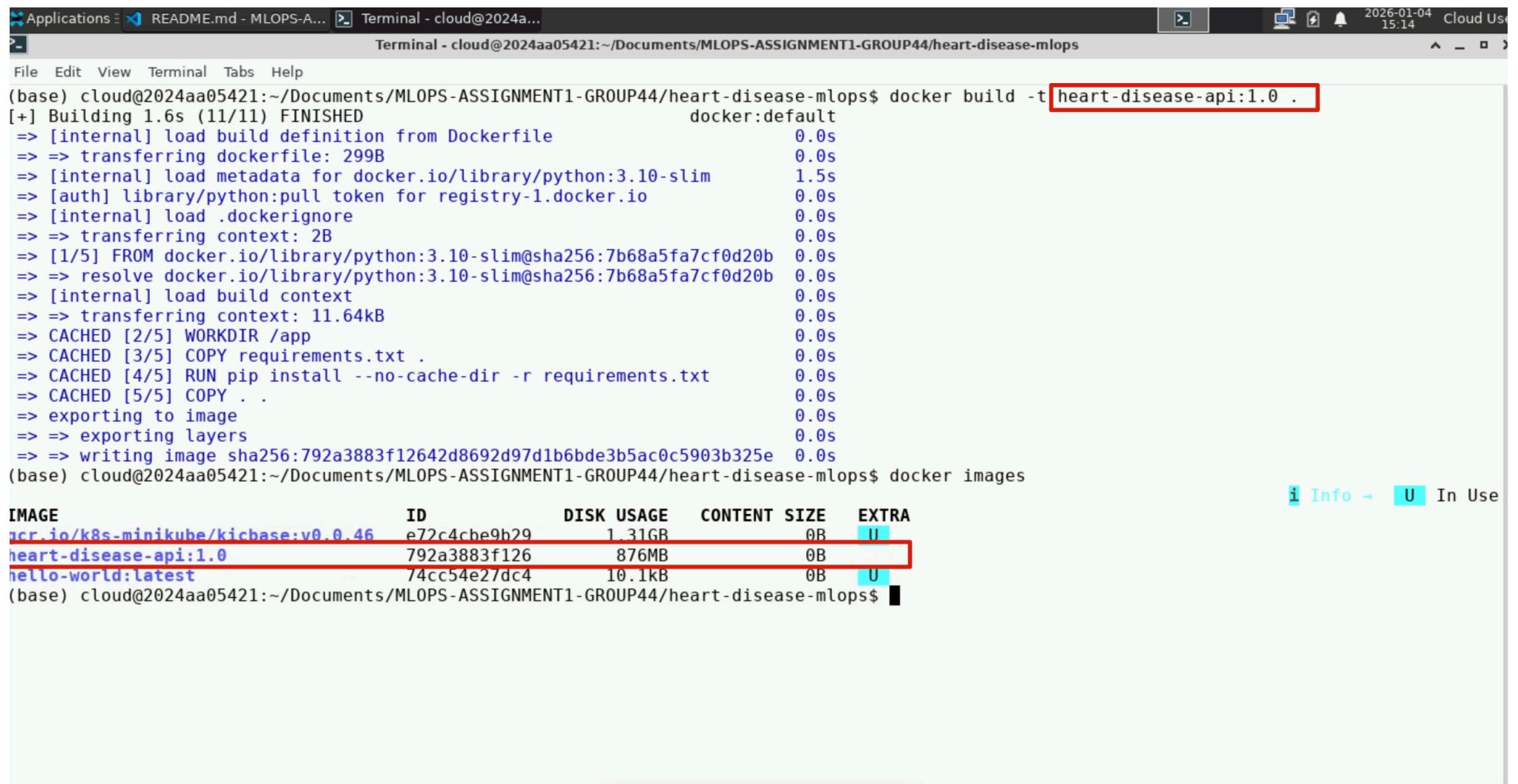
6.2 Deployment Workflow

The trained model is containerized using Docker and deployed on a Kubernetes cluster using Minikube.

Deployment steps include:

- Building the Docker image

```
docker build -t heart-disease-api:latest .
```



The screenshot shows a terminal window titled "Terminal - cloud@2024aa05421:~/Documents/MLOPS-ASSIGNMENT1-GROUP44/heart-disease-mlops". The window displays the command "docker build -t heart-disease-api:1.0 ." followed by the build logs. The logs show the process of building the image, including loading the Dockerfile, transferring metadata, pulling Python, and installing requirements. The final output shows the image was successfully built with ID 792a3883f12642d8692d97d1b6bde3b5ac0c5903b325e. Below this, the command "docker images" is run, listing three images: "k8s-minikube/kicbase:v0.0.46", "heart-disease-api:1.0", and "hello-world:latest". The "heart-disease-api:1.0" image is highlighted with a red box.

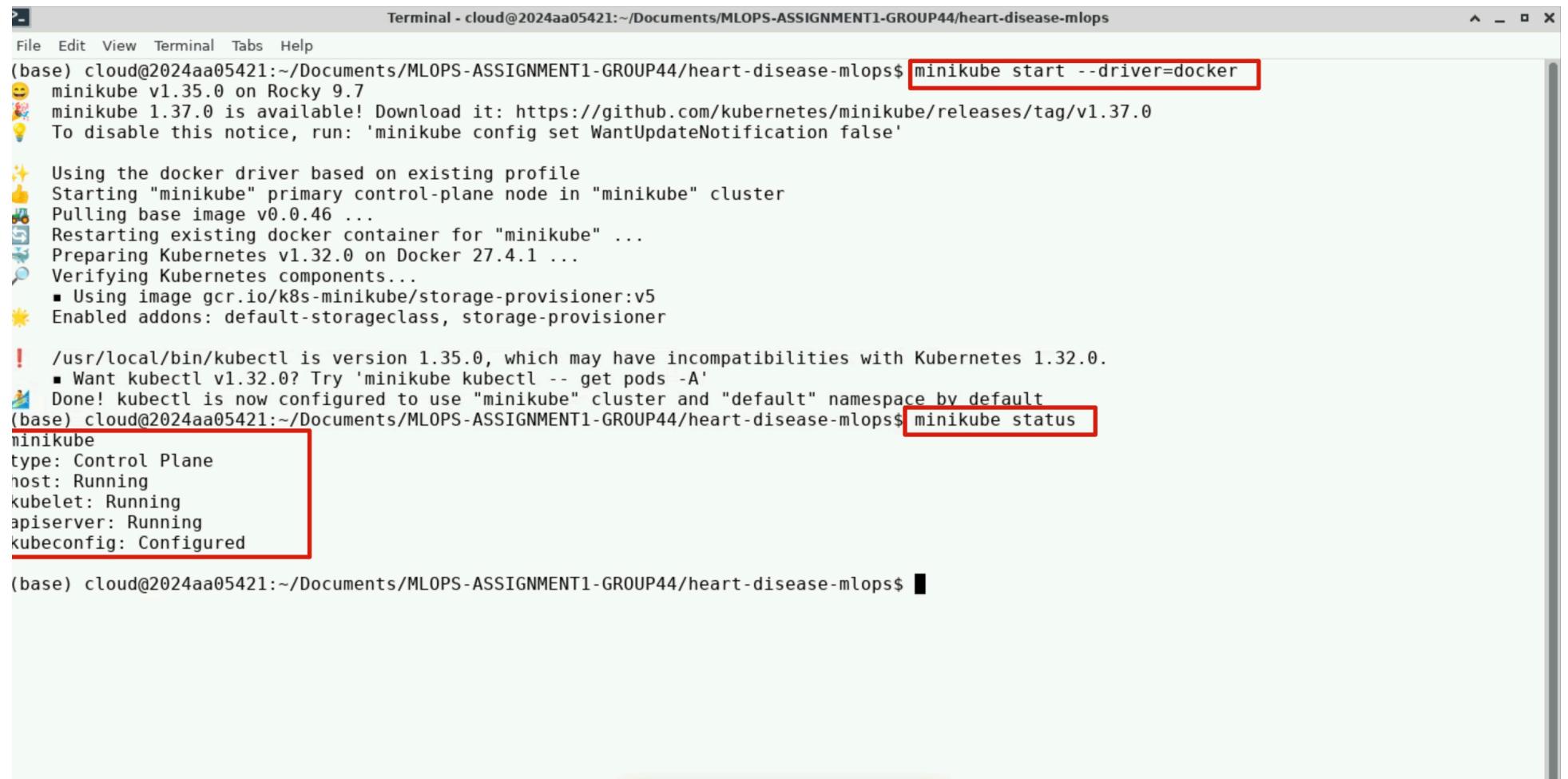
```
(base) cloud@2024aa05421:~/Documents/MLOPS-ASSIGNMENT1-GROUP44/heart-disease-mlops$ docker build -t heart-disease-api:1.0 .
[+] Building 1.6s (11/11) FINISHED
          docker:default
=> [internal] load build definition from Dockerfile                               0.0s
=> => transferring dockerfile: 299B                                            0.0s
=> [internal] load metadata for docker.io/library/python:3.10-slim             1.5s
=> [auth] library/python:pull token for registry-1.docker.io                   0.0s
=> [internal] load .dockerrcignore                                              0.0s
=> => transferring context: 2B                                                 0.0s
=> [1/5] FROM docker.io/library/python:3.10-slim@sha256:7b68a5fa7cf0d20b      0.0s
=> => resolve docker.io/library/python:3.10-slim@sha256:7b68a5fa7cf0d20b      0.0s
=> [internal] load build context                                               0.0s
=> => transferring context: 11.64kB                                           0.0s
=> CACHED [2/5] WORKDIR /app                                                 0.0s
=> CACHED [3/5] COPY requirements.txt .                                       0.0s
=> CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt           0.0s
=> CACHED [5/5] COPY . . .                                                 0.0s
=> exporting to image                                                       0.0s
=> => exporting layers                                                       0.0s
=> => writing image sha256:792a3883f12642d8692d97d1b6bde3b5ac0c5903b325e 0.0s
(base) cloud@2024aa05421:~/Documents/MLOPS-ASSIGNMENT1-GROUP44/heart-disease-mlops$ docker images


| IMAGE                        | ID           | DISK USAGE | CONTENT SIZE | EXTRA |
|------------------------------|--------------|------------|--------------|-------|
| k8s-minikube/kicbase:v0.0.46 | e72c4che9b29 | 1.31GB     | 0B           | U     |
| heart-disease-api:1.0        | 792a3883f126 | 876MB      | 0B           |       |
| hello-world:latest           | 74cc54e27dc4 | 10.1KB     | 0B           | U     |


(base) cloud@2024aa05421:~/Documents/MLOPS-ASSIGNMENT1-GROUP44/heart-disease-mlops$
```

- Deploying the application using Kubernetes manifests

```
# Start Minikube Server  
minikube start --driver=docker  
  
# Check the Status of minikube  
minikube status
```

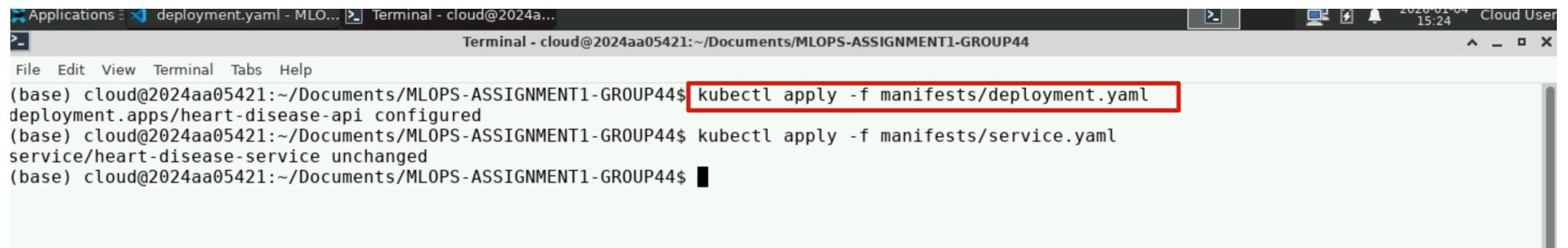


The screenshot shows a terminal window titled "Terminal - cloud@2024aa05421:~/Documents/MLOPS-ASSIGNMENT1-GROUP44/heart-disease-mlops". The window contains the following text:

```
File Edit View Terminal Tabs Help  
(base) cloud@2024aa05421:~/Documents/MLOPS-ASSIGNMENT1-GROUP44/heart-disease-mlops$ minikube start --driver=docker  
😄 minikube v1.35.0 on Rocky 9.7  
💡 minikube 1.37.0 is available! Download it: https://github.com/kubernetes/minikube/releases/tag/v1.37.0  
💡 To disable this notice, run: 'minikube config set WantUpdateNotification false'  
  
💡 Using the docker driver based on existing profile  
👉 Starting "minikube" primary control-plane node in "minikube" cluster  
📦 Pulling base image v0.0.46 ...  
🔄 Restarting existing docker container for "minikube" ...  
🛠 Preparing Kubernetes v1.32.0 on Docker 27.4.1 ...  
🔍 Verifying Kubernetes components...  
▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5  
☀️ Enabled addons: default-storageclass, storage-provisioner  
  
❗ /usr/local/bin/kubectl is version 1.35.0, which may have incompatibilities with Kubernetes 1.32.0.  
▪ Want kubectl v1.32.0? Try 'minikube kubectl -- get pods -A'  
💡 Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default  
(base) cloud@2024aa05421:~/Documents/MLOPS-ASSIGNMENT1-GROUP44/heart-disease-mlops$ minikube status  
minikube  
type: Control Plane  
host: Running  
kubelet: Running  
apiserver: Running  
kubecfg: Configured  
  
(base) cloud@2024aa05421:~/Documents/MLOPS-ASSIGNMENT1-GROUP44/heart-disease-mlops$ █
```

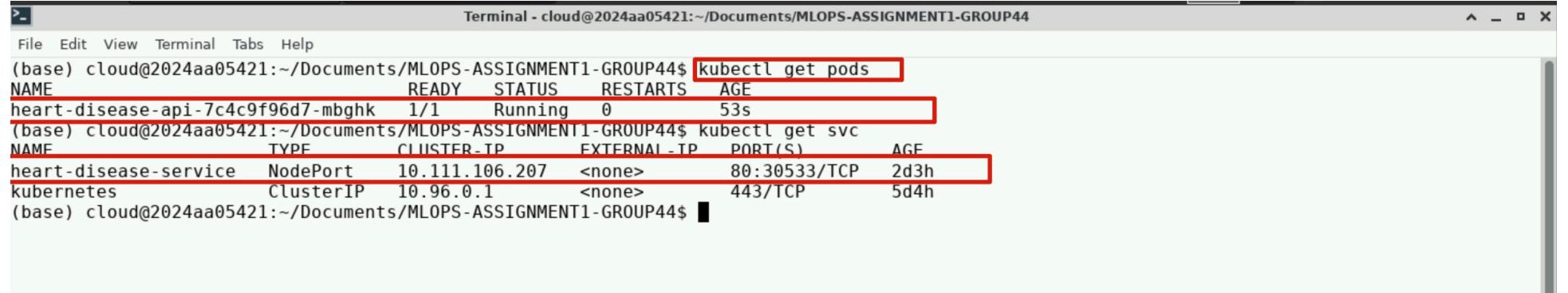
The command "minikube start --driver=docker" and the command "minikube status" are highlighted with red boxes.

```
# Apply the manifests
# Deployment
kubectl apply -f manifests/deployment.yaml
# Service
kubectl apply -f manifests/service.yaml
```



```
Applications > deployment.yaml - MLOPS-ASSIGNMENT1-GROUP44 Terminal - cloud@2024aa05421:~/Documents/MLOPS-ASSIGNMENT1-GROUP44
File Edit View Terminal Tabs Help
(base) cloud@2024aa05421:~/Documents/MLOPS-ASSIGNMENT1-GROUP44$ kubectl apply -f manifests/deployment.yaml
deployment.apps/heart-disease-api configured
(base) cloud@2024aa05421:~/Documents/MLOPS-ASSIGNMENT1-GROUP44$ kubectl apply -f manifests/service.yaml
service/heart-disease-service unchanged
(base) cloud@2024aa05421:~/Documents/MLOPS-ASSIGNMENT1-GROUP44$
```

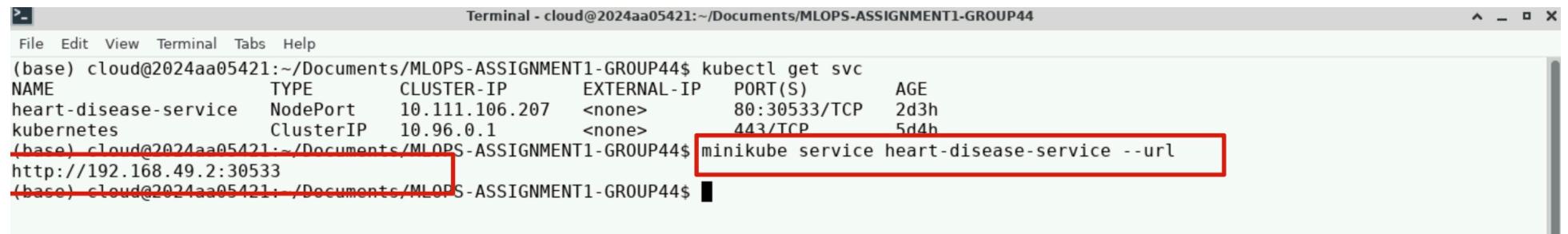
```
# Check the pods
kubectl get pods
```



```
Terminal - cloud@2024aa05421:~/Documents/MLOPS-ASSIGNMENT1-GROUP44
File Edit View Terminal Tabs Help
(base) cloud@2024aa05421:~/Documents/MLOPS-ASSIGNMENT1-GROUP44$ kubectl get pods
NAME READY STATUS RESTARTS AGE
heart-disease-api-7c4c9f96d7-mbghk 1/1 Running 0 53s
(base) cloud@2024aa05421:~/Documents/MLOPS-ASSIGNMENT1-GROUP44$ kubectl get svc
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
heart-disease-service NodePort 10.111.106.207 <none> 80:30533/TCP 2d3h
kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 5d4h
(base) cloud@2024aa05421:~/Documents/MLOPS-ASSIGNMENT1-GROUP44$
```

- Exposing the service using a NodePort

```
# Check the svc  
kubectl get svc  
# Get Endpoint  
minikube service heart-disease-service --url
```



A screenshot of a terminal window titled "Terminal - cloud@2024aa05421:~/Documents/MLOPS-ASSIGNMENT1-GROUP44". The window shows the following command history:

```
File Edit View Terminal Tabs Help  
(base) cloud@2024aa05421:~/Documents/MLOPS-ASSIGNMENT1-GROUP44$ kubectl get svc  
NAME           TYPE      CLUSTER-IP    EXTERNAL-IP   PORT(S)        AGE  
heart-disease-service  NodePort  10.111.106.207 <none>       80:30533/TCP  2d3h  
kubernetes     ClusterIP  10.96.0.1    <none>       443/TCP       5d4h  
(base) cloud@2024aa05421:~/Documents/MLOPS-ASSIGNMENT1-GROUP44$ minikube service heart-disease-service --url  
http://192.168.49.2:30533  
(base) cloud@2024aa05421:~/Documents/MLOPS-ASSIGNMENT1-GROUP44$
```

The command `minikube service heart-disease-service --url` and its output `http://192.168.49.2:30533` are highlighted with a red rectangular box.

- Verifying the deployment via Swagger UI and API responses

The screenshot shows a web browser window with the title "Heart Disease Prediction" and the URL "http://192.168.49.2:30533/docs". The page displays the OpenAPI specification for the "Heart Disease Prediction API".

The main header includes the API title, version "0.1.0", and OAS 3.1 compliance. Below the header, there is a link to "/openapi.json".

The "default" section is highlighted with a red box, containing a "POST /predict Predict" operation.

The "Schemas" section lists three schema definitions:

- HTTPValidationError** > Expand all object
- PatientData** > Expand all object
- ValidationError** > Expand all object



The screenshot shows a terminal window titled "Terminal - cloud@2024aa05421:~". The command entered is:

```
(base) cloud@2024aa05421:~$ curl -X POST "http://192.168.49.2:30533/predict" -H "Content-Type: application/json" -d '{  
    "age": 55,  
    "sex": 1,  
    "cp": 2,  
    "trestbps": 140,  
    "chol": 250,  
    "fbs": 0,  
    "restecg": 1,  
    "thalach": 150,  
    "exang": 0,  
    "oldpeak": 1.2,  
    "slope": 1,  
    "ca": 0,  
    "thal": 2  
}'  
{"prediction":0,"confidence":0.0845}(base) cloud@2024aa05421:~$ █
```

A red box highlights the JSON response from the prediction service, which includes the prediction value and confidence level.

Screenshots of running pods, services, and API responses are included.

- Monitoring and Logging:
 - Inference requests were monitored using Kubernetes pod logs, capturing request paths, response status, and latency.
 - MLflow was primarily used for training-time experiment tracking, while inference observability was handled via application logs.
 - MLFlow Metrics

mlflow 2.11.4 Experiments Models

Default Provide Feedback Add Description

Search Experiments Default Heart Disease Prediction

metrics.rmse < 1 and params.model = "tree"

Time created State: Active Datasets

Sort: Created Columns Group by

+ New run

Share

Table Chart Evaluation Experimental

Run Name	Created	Dataset	Duration	Source	Models
inference	9 minutes ago	-	16ms	unicorn	-
inference	28 minutes ago	-	18ms	unicorn	-

2 matching runs

Default > inference

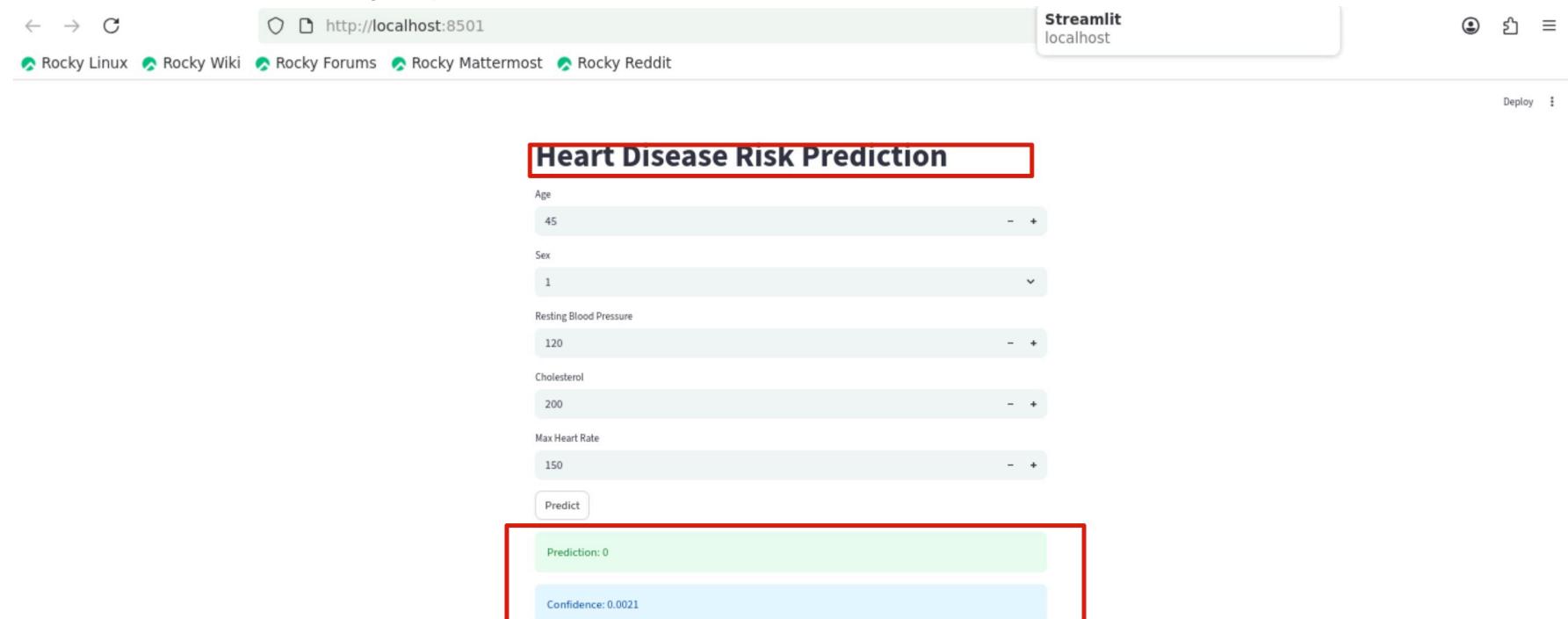


- Kubectl Logs

```
[MONITORING] method=POST path=/predict status=200 latency=0.3555s
INFO: 10.244.0.1:25938 - "POST /predict HTTP/1.1" 200 OK
[MONITORING] method=POST path=/predict status=200 latency=0.0291s
INFO: 10.244.0.1:47455 - "POST /predict HTTP/1.1" 200 OK
```

- Streamlit Inferencing

- Streamlit is an UI interface for invoking the prediction service.



- ### Access Instructions (for Local Testing)

The deployed Heart Disease Prediction API is hosted on a local Kubernetes cluster using Minikube and is not publicly exposed. The service is made accessible for local testing via a NodePort service.

Steps to Access the Deployed API

- Steps to Access the Deployed API

```
# Start minikube  
minikube start --driver=docker
```

- Verify that the application pods and service are running:

```
# Check the pods  
kubectl get pods  
kubectl get svc
```

- Retrieve the service endpoint URL:

```
# Get the endpoint  
minikube service heart-disease-service --url
```

- API Endpoints:

- Swagger UI (API documentation):

```
# Get the endpoint  
http://192.168.49.2:30533/docs
```

- Sample Curl Request:

```
curl -X POST http://192.168.49.2:30533/predict \
-H "Content-Type: application/json" \
-d '{
    "age": 55,
    "sex": 1,
    "cp": 2,
    "trestbps": 140,
    "chol": 250,
    "fbs": 0,
    "restecg": 1,
    "thalach": 150,
    "exang": 0,
    "oldpeak": 1.2,
    "slope": 1,
    "ca": 0,
    "thal": 2
}'
```

7. Code Repository

The complete source code, configuration files, CI/CD pipeline, and documentation are available at the following GitHub repository:

 <https://github.com/Krithika-Madhavan-5421/MLOPS-ASSIGNMENT1-GROUP44>

8. Conclusion

This project demonstrates a complete end-to-end MLOps workflow, covering data preparation, model development, experiment tracking, CI/CD automation, containerized deployment, and production monitoring. By integrating industry-standard tools such as MLflow, Docker, Kubernetes, and GitHub Actions, the solution ensures reproducibility, scalability, and deployment readiness.

The inclusion of an optional Streamlit interface further enhances usability, while the Kubernetes-based deployment simulates a real-world production environment. Overall, this project reflects best practices in modern MLOps and provides a strong foundation for deploying machine learning systems in production settings.

Video Demonstration Link:

https://drive.google.com/file/d/1krQUMoG8Ufc1O5A3TJk-2_r4FcP1A-Kd/view?usp=sharing