

Technical Report: Heart Disease Prediction Pipeline – Group 32

This document details the end-to-end MLOps process implemented for the Heart Disease (Cleveland) dataset. All visualizations and performance metrics shown below were generated automatically during the execution of the project's source code.

Setup/install instructions

Prerequisites and Local Environment Setup

- **Python Environment:**
 - Create a virtual environment: `python -m venv venv`
 - Activate it:
 - Linux/Mac: `source venv/bin/activate`
 - Windows: `venv\Scripts\activate`
 - Install dependencies: `pip install -r requirements.txt`
- **IDE:** Visual Studio Code
- **Dataset:** Heart disease dataset (likely heart.csv)
- **Docker:** Required for containerization
- **Kubernetes:** Minikube for local cluster setup
- **Git:** Client for version control and pushing to GitHub

Running the Application Locally

1. Activate the virtual environment (as above).
2. Train the model and make predictions: `python train_automl.py && python src/predict.py`
3. Start MLflow UI: `mlflow ui --port 5000` (**access at <http://127.0.0.1:5000>**)
4. Run the main app: `python heart_disease.py`
5. Alternative MLflow command: `python -m mlflow ui`

Docker Containerization

- Build the image: `docker build -t heart-disease-api:2.0`
- Tag the image: `docker tag heart-disease-api:2.0 heart-disease-api:latest`
- Run the container: `docker run -p 8000:8000 heart-disease-api`

Kubernetes Deployment with Minikube

1. Start Minikube: minikube start
2. Load the Docker image into Minikube: minikube image load heart-disease-api:latest
 - o Alternative: docker save my-app:latest | minikube image load -
3. Apply Kubernetes manifests:
 - o kubectl apply -f k8s/deployment.yaml
 - o kubectl apply -f k8s/service.yaml
4. Check status:
 - o kubectl get deployments
 - o kubectl get pods
 - o kubectl get services
5. Access the service: minikube service heart-disease-service
6. For external IP (load balancer): minikube tunnel

Monitoring Setup (Prometheus and Grafana)

- **Standalone Docker:**
 - o Prometheus: docker run --name prometheus -p 9090:9090 -v C:/ddrive/AIML-MS/courses/sem3/MLOPs/Assignment/prometheus.yml:/etc/prometheus/prometheus.yml prom/prometheus
 - o Grafana: docker run -d --name grafana -p 3000:3000 grafana/grafana

- **Kubernetes with Helm:**
 - Add repo: helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
 - Update: helm repo update
 - Install: helm install monitoring prometheus-community/kube-prometheus-stack --namespace monitoring --create-namespace
 - Port forward:
 - Prometheus: kubectl port-forward svc/monitoring-kube-prometheus-prometheus 9090 -n monitoring
 - Grafana: kubectl port-forward svc/monitoring-grafana 3000:80 -n monitoring
 - Grafana admin
password: TS6bN7QZp1IVaB6KyUG8M9ng6HTGAN4SoljdBZHM (decoded from secret)
- Check endpoints: kubectl get endpoints heart-disease-service

Additional Notes

- MLflow backend store: mlflow ui --backend-store-uri file:C:\ddrive\AIML-MS\courses\sem3\MLOPs\Assignment\mlflow-runs
- Docker registry secret for GitHub Container Registry (ghcr.io) is pre-configured with specific credentials.
- The project includes automated EDA, model training, prediction, and testing scripts in the src and tests directories.

Exploratory Data Analysis (EDA) & Modelling Choices

Data Acquisition Strategy

The dataset used in this project is the Cleveland Heart Disease Dataset obtained from the UCI Machine Learning Repository. We implemented an automated download script in `src/data_loader.py` that fetches the raw data directly from the source. If the URL is unavailable, it falls back to a local copy.

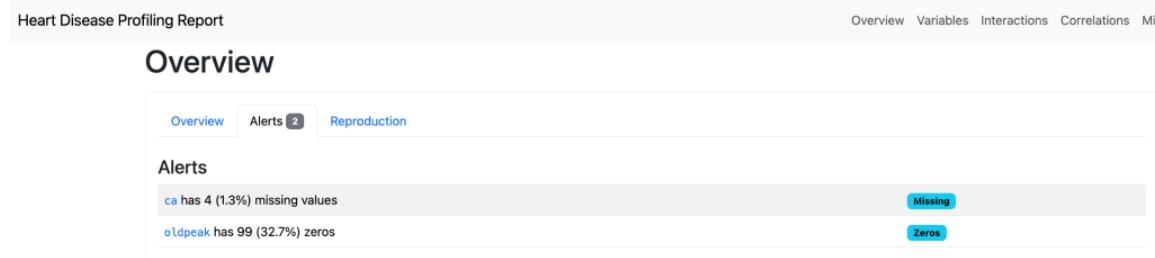
Professional Exploratory Data Analysis

Before any cleaning, we performed comprehensive EDA using Pandas Profiling and Sweetviz. These reports provide a baseline diagnostic of the raw dataset.

Pandas Profiling: Feature Correlations Heatmap



Figure 1: Heatmap showing the correlation between medical features (from Pandas Profiling)



Sweetviz: Missing Values Detection

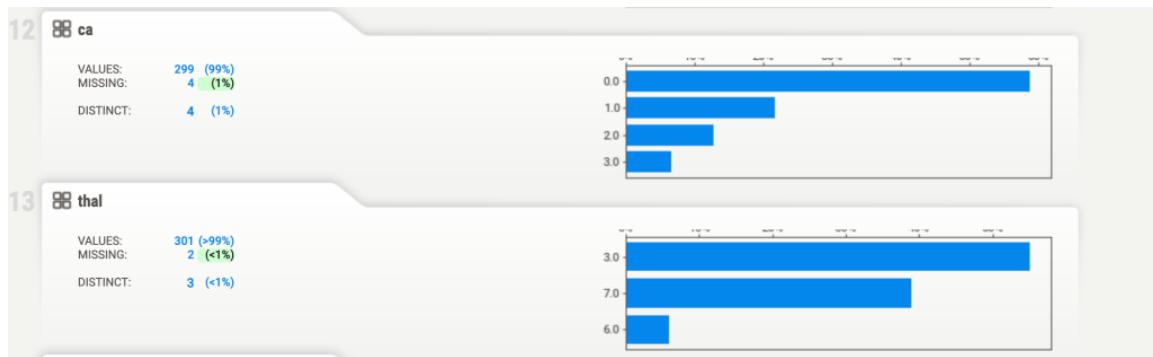


Figure 2: Identification of missing fields in clinical data (from Sweetviz)

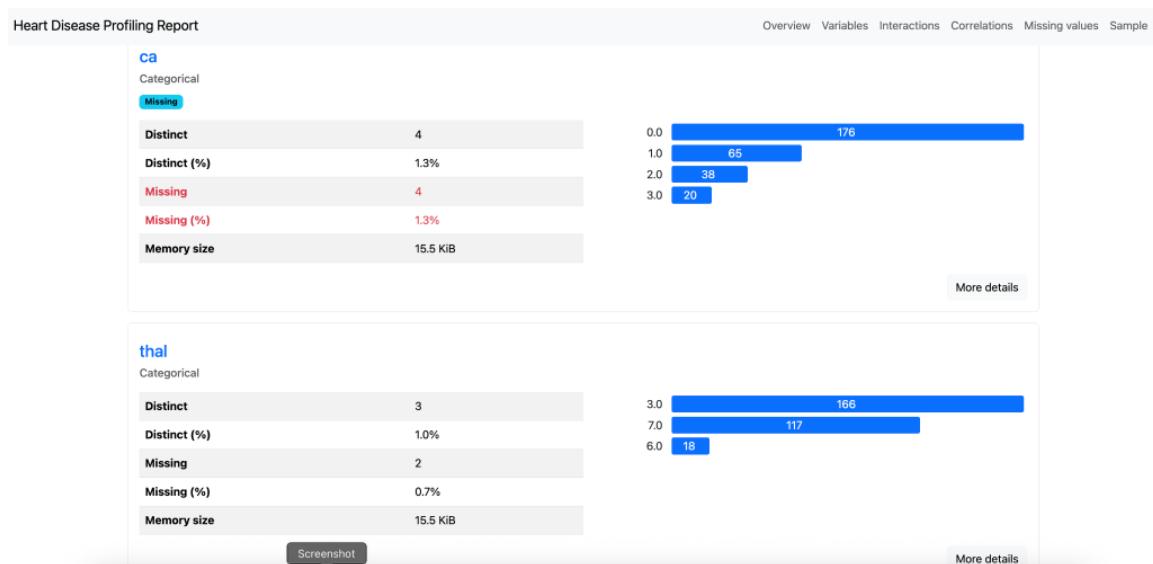


Figure 3: Identification of missing fields in clinical data (from Pandas-Profilng)

Data Cleaning & Label Engineering

Based on the EDA findings, several cleaning steps were integrated into the modelling pipeline:

1. Missing value identification (ca and thal columns).
2. Multi-class target binarization (stages 1-4 mapped to 1).

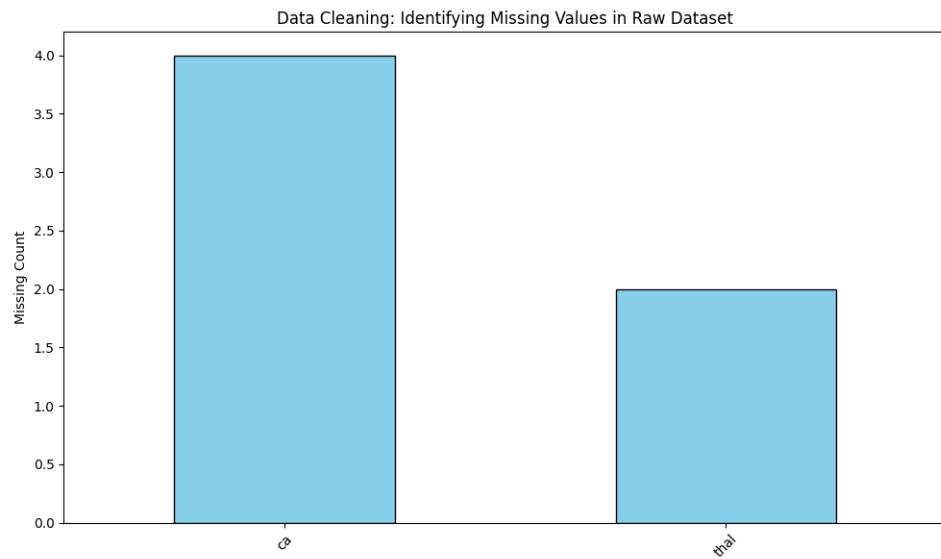


Figure 3: Automated Detection of Missing Values

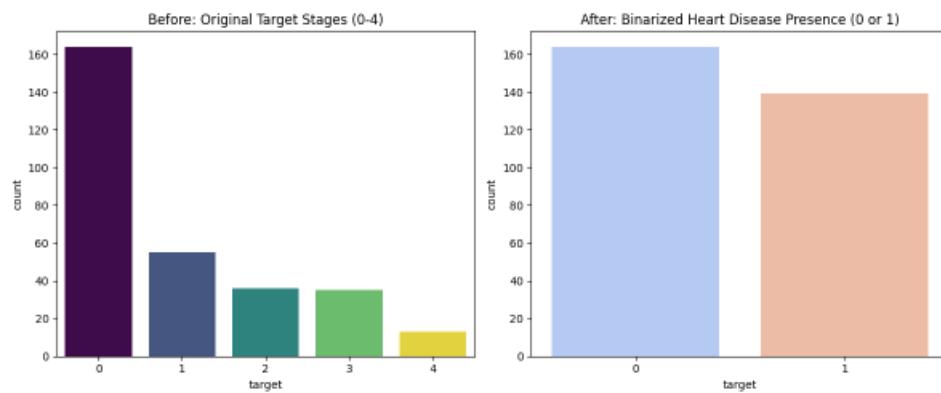


Figure 4: Transformation of Diagnostic stages into Binary target

Feature Engineering & Model Development

Feature Engineering

Advanced feature engineering was performed using Sklearn ColumnTransformers:

- One-Hot Encoding for categorical data (e.g., Sex, CP).
- Polynomial Interaction Features (degree=2) to capture synergy between variables.
- Standardization (Z-score scaling) for numerical and engineered inputs.

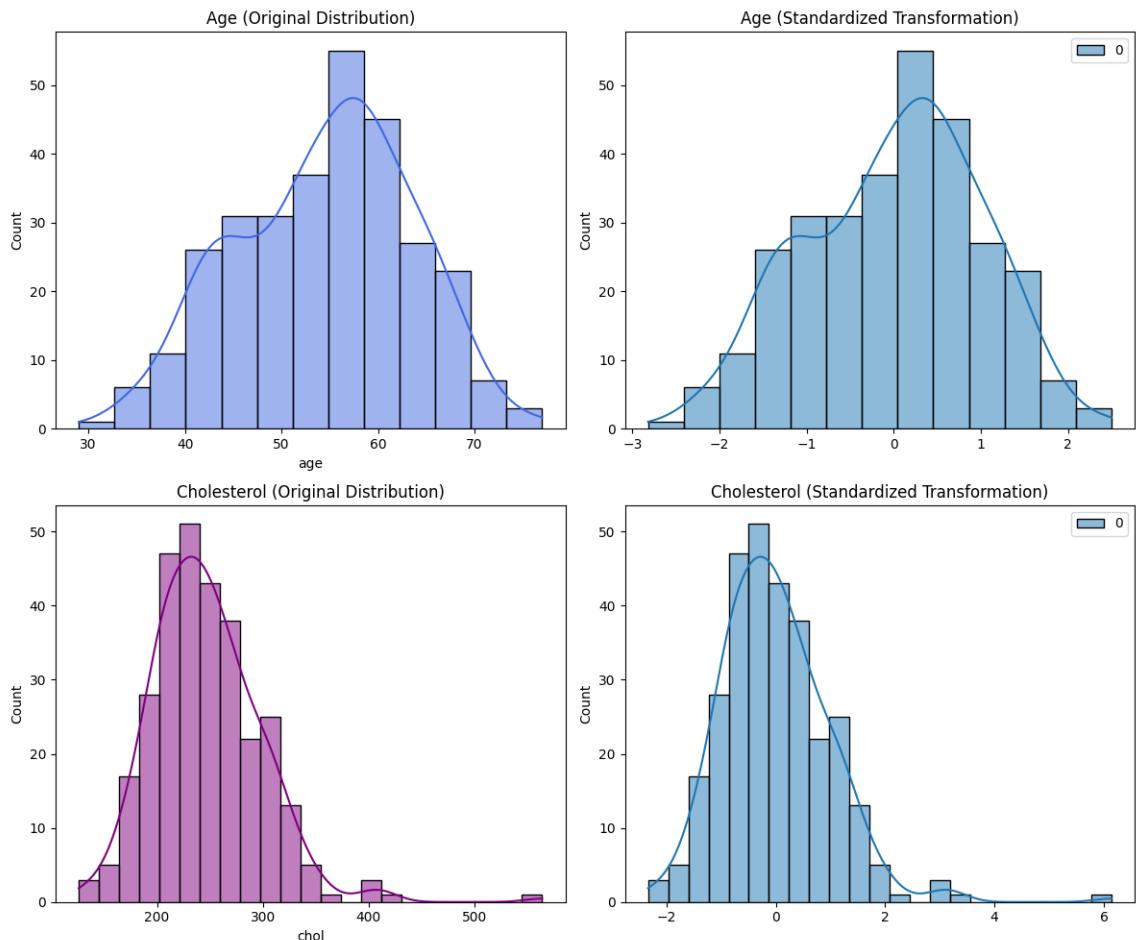


Figure 5: Impact of Standardization on Feature Distributions

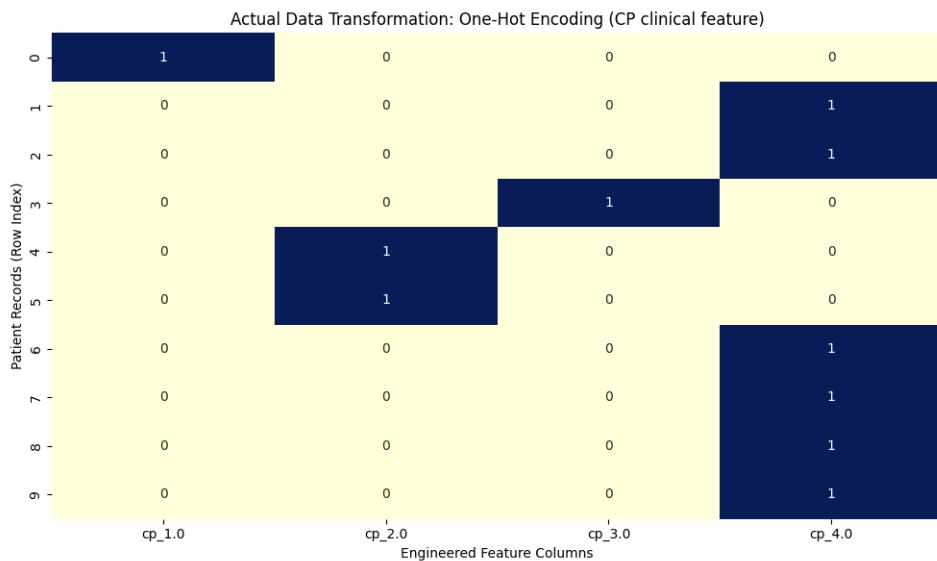


Figure 6: Actual One-Hot Encoding transformation on patient records

Model Selection and Metrics

Using FLAML AutoML, we tuned various classifiers.

The winning model was XGBoost, optimized for ROC-AUC.

Metric	Final Test Performance
ROC-AUC	0.9496
Accuracy	90.16%
Precision	0.9038
Recall	0.9016

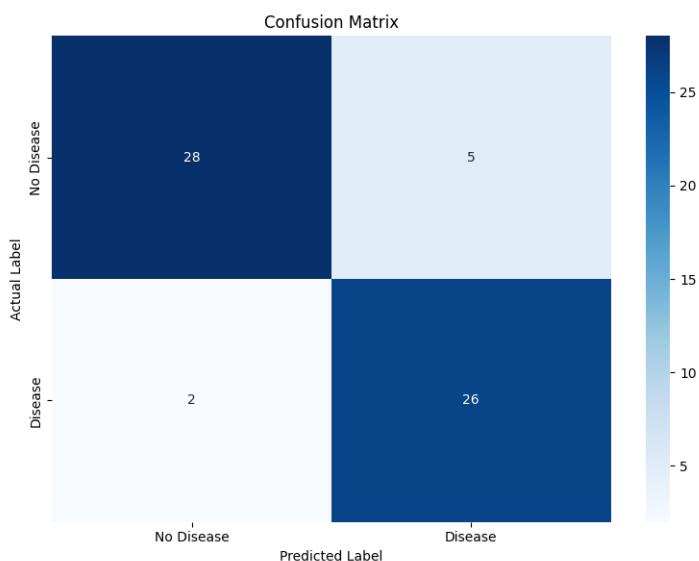


Figure 7: Confusion Matrix of Final Model

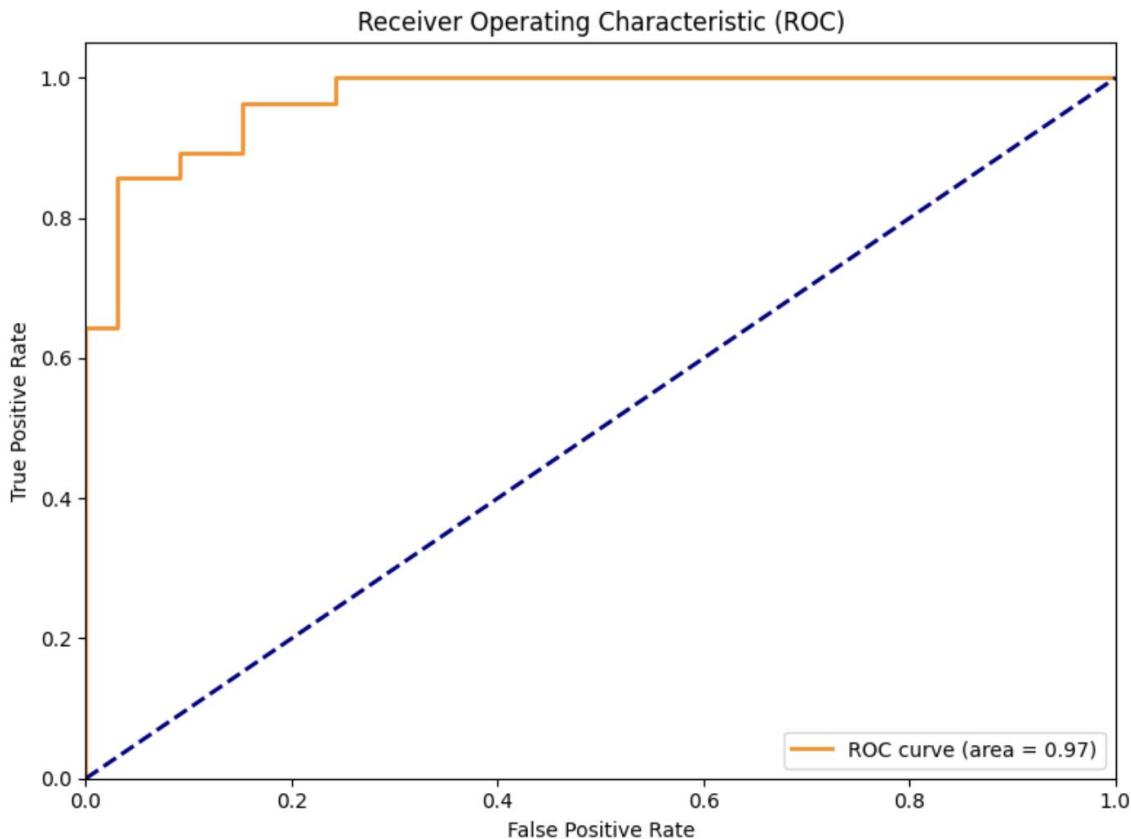


Figure 8: ROC Curve on Held-out Test Set

Experiment Tracking Summary

The project uses a **hierarchical experiment tracking design**:

- **Experiment Name:**
Heart_Disease_Prediction_AutoML
- **Parent Run:**
Represents a single AutoML execution, including:
 - Dataset split
 - Preprocessing strategy
 - AutoML configuration
 - Final model evaluation
- **Nested Child Runs:**
Each FLAML trial (model + hyperparameters) is logged as a nested MLflow run using a custom callback.

mlflow 3.8.0

Heart_Disease_Prediction_AutoML Machine learning

Runs

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

Time created

State: Active

Datasets

Sort: Created

Columns

New run

Show more columns (62 total)

Run Name	Created	Duration	Source	Models	learner
AutoML_Run_20260105_190505	48 minutes ago	1 min	train_automl.py	Heart_Disease_Pre...	xgboost
AutoML_Run_20260105_190505_child_529	47 minutes ago	14ms	train_automl.py	-	xgboost
AutoML_Run_20260105_190505_child_528	47 minutes ago	15ms	train_automl.py	-	lgbm
AutoML_Run_20260105_190505_child_527	47 minutes ago	14ms	train_automl.py	-	lgbm
AutoML_Run_20260105_190505_child_526	47 minutes ago	15ms	train_automl.py	-	xgboost
AutoML_Run_20260105_190505_child_525	47 minutes ago	14ms	train_automl.py	-	xgboost
AutoML_Run_20260105_190505_child_524	47 minutes ago	14ms	train_automl.py	-	lgbm
AutoML_Run_20260105_190505_child_523	47 minutes ago	14ms	train_automl.py	-	xgboost
AutoML_Run_20260105_190505_child_522	47 minutes ago	15ms	train_automl.py	-	rf
AutoML_Run_20260105_190505_child_521	47 minutes ago	14ms	train_automl.py	-	xgboost
AutoML_Run_20260105_190505_child_520	47 minutes ago	16ms	train_automl.py	-	xgboost
AutoML_Run_20260105_190505_child_519	47 minutes ago	14ms	train_automl.py	-	xgboost
AutoML_Run_20260105_190505_child_518	47 minutes ago	14ms	train_automl.py	-	rf
AutoML_Run_20260105_190505_child_517	47 minutes ago	21ms	train_automl.py	-	rf2
AutoML_Run_20260105_190505_child_516	47 minutes ago	12ms	train_automl.py	-	rf

601 matching runs

Logged Parameters:

Parent Run Parameters

The following parameters are logged once per AutoML run:

mlflow 3.8.0

Heart_Disease_Prediction_AutoML > Runs

AutoML_Run_20260105_190505

Overview Model metrics System metrics Traces Artifacts

Description

No description

Metrics (12)

Search metrics

Metric	Value	Models
best_validation_loss	0.15341204689030777	inference_pipeline +2
validation_loss	0.15341204689030777	inference_pipeline +2
wall_clock_time	15.1437349319458	inference_pipeline +2
accuracy	0.9016393442622951	inference_pipeline +1
iter_counter	132	inference_pipeline +2
customized metric	0.15341204689030777	inference_pipeline +2
recall	0.9016393442622951	inference_pipeline +1
roc_auc	0.949675324753247	inference_pipeline +1
trial_time	0.11898946762084961	inference_pipeline +2
precision	0.9038956460426582	inference_pipeline +1
best_loss	0.15341204689030777	inference_pipeline +1
f1_score	0.9017983323911815	inference_pipeline +1

Screenshot

About this run

Created at: 01/06/2026, 12:35:05 AM

Created by: runner

Experiment ID: 418609388340197443

Status: Finished

Run ID: 4d61d3cbd35645d4b154cb3d4475c5df

Duration: 1.1min

Child runs:

- AutoML_Run_20260105_190505_child_529
- AutoML_Run_20260105_190505_child_528
- AutoML_Run_20260105_190505_child_527
- AutoML_Run_20260105_190505_child_526
- AutoML_Run_20260105_190505_child_525
- AutoML_Run_20260105_190505_child_524
- AutoML_Run_20260105_190505_child_523
- AutoML_Run_20260105_190505_child_522
- AutoML_Run_20260105_190505_child_521
- AutoML_Run_20260105_190505_child_520
- AutoML_Run_20260105_190505_child_519
- AutoML_Run_20260105_190505_child_518
- AutoML_Run_20260105_190505_child_517
- AutoML_Run_20260105_190505_child_516

Source: train_automl.py -> 12a4235

Registered prompts: -

Final Test Metrics

After model selection, the best model is evaluated on the held-out test set. The following metrics are logged:

- Accuracy

- Precision (weighted)
- Recall (weighted)
- F1-score (weighted)
- ROC AUC

Parameters (28)

Parameter	Value
re:lambda	0.476223299246811
best_config_colsample_bytree	0.6993108042372532
metric_strategy	Minimize 1 - (0.5 * Recall + 0.5 * ROC_AUC)
colsample_bylevel	0.898448035850516
best_config_subsample	0.7164280511605149
time_budget	60
best_config_rg_alpha	0.006896809387654332
mr_child_weight	7.70275277431409
best_config_max_leaves	44
selection_reason	Score: 0.8466 (Rec:0.9016, AUC:0.9497)
learning_rate	0.7135003922079828
best_config_learning_rate	0.7135003922079828
model_type	XGBoost
best_estimator	xgboost

Datasets
None

Tags

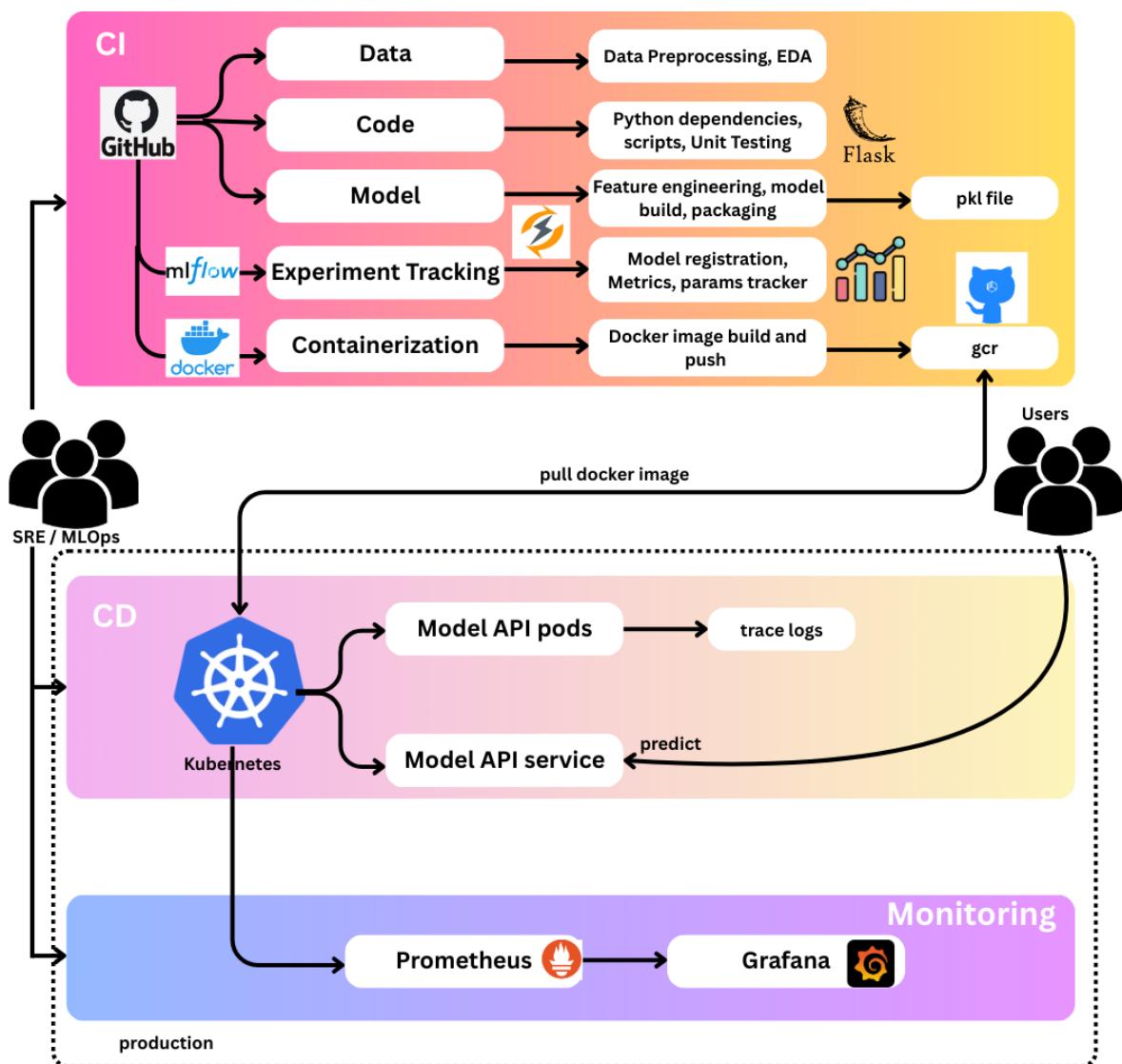
```
flaml.estimator_name: xgboost flaml.version: 2.9.6
flaml.learner: xgboost flaml.best_run: True
selection_reason: Best Val Loss: 0.1534 flaml.run_source: flaml-automl
flaml.log_type: manual run_type: parent flaml.sample_size: 242
flaml.automl_user_configurations: {}
flaml.estimator_class: XGBBoosterEstimator
flaml.metric: customized metric flaml.iteration_number: 132
```

Registered models
Heart_Disease_Pipeline v1

Logged models (3)

Model attributes			No dataset					Parameters	
Type	Model name	Status	t	best_validation_loss	accuracy	iter_counter	recall	f1_score	learning_rate
Output	model	Ready	54	0.15341204689030777	0.90163934	132	0.9016393	0.9017983323	0.713500392207
Output	best_model_skies	Ready	54	0.15341204689030777	0.90163934	132	0.9016393	0.9017983323	0.713500392207
Output	inference_pipeline	Ready	54	0.15341204689030777	0.90163934	132	0.9016393	0.9017983323	0.713500392207

Architecture diagram:



CI/CD and deployment workflow screenshots:

CI Screenshots:

The screenshot shows a GitHub Actions pipeline summary for a job named "build-test-train". The job succeeded 10 hours ago in 6m 43s. It consists of 17 steps, all of which have been completed successfully. The steps are listed below:

- > Set up Python
- > Install Dependencies
- > Lint Code (flake8)
- > Run Unit Tests
- > Run EDA
- > Upload EDA Reports
- > Train AutoML Model
- > Upload MLflow Artifacts
- > Prepare Model for Docker
- > Log in to GitHub Container Registry
- > Build and Push Docker Image
- > Post Log in to GitHub Container Registry
- > Post Set up Python
- > Post Checkout Repository
- > Complete job

The screenshot shows the logs for the "Build and Push Docker Image" step of the "build-test-train" job. The step succeeded 10 hours ago in 6m 43s. The logs output the following command and its execution details:

```
#11 exporting to image
#11 exporting layers
#11 exporting layers 6.8s done
#11 writing image sha256:7a9cc28c49070e8e38f88e7975ba812831731132f87914aa5cabe524443962c1 done
#11 naming to ghr.io/sriramgunda/heart-disease-api:latest done
#11 DONE 6.8s
The push refers to repository [ghr.io/sriramgunda/heart-disease-api]
#43 c93c6fa5369: Preparing
#43 5c4116736a7: Preparing
#44 9f481054af9: Preparing
#45 46c055d853a: Preparing
#46 b4dd5dc2e02: Preparing
#47 75b5fb5f56130: Preparing
#48 b12fd0d942e9: Preparing
#49 6a7f953aa30c: Preparing
#50 75b5fb5f56130: Waiting
#51 b12fd0d942e9: Waiting
#52 6a7f953aa30c: Waiting
#53 b4dd5dc2e02: Layer already exists
#54 75b5fb5f56130: Layer already exists
#55 b12fd0d942e9: Layer already exists
#56 9f481054af9: Pushed
#57 46c055d853a: Pushed
#58 c93c6fa5369: Pushed
#59 6a7f953aa30c: Pushed
#60 5c4116736a7: Pushed
#61 latest: digest: sha256:85acb178b650c31bf79333815544890507379048128ee8211b524ef02fe2e780 size: 1993
```

Deployment Screenshots:

The screenshot shows the Kubernetes Dashboard interface. On the left, a sidebar lists various workload types: Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers, Stateful Sets, Services, Ingresses, and Services. The main area is titled "Workloads" and contains three tabs: "Deployments", "Pods", and "Replica Sets".

- Deployments:** Shows one deployment named "heart-disease-deployment" with two pods. The first pod is labeled "app: heart-disease" and "pod-template-hash: b4694f77c-46lnb". The second pod is labeled "app: heart-disease" and "pod-template-hash: b4694f77c-4zcdg". Both pods are running.
- Pods:** Shows the same two pods listed above.
- Replica Sets:** Shows one replica set named "heart-disease-deployment-b4694f77c" with two pods. The first pod is labeled "app: heart-disease" and "pod-template-hash: b4694f77c-46lnb". The second pod is labeled "app: heart-disease" and "pod-template-hash: b4694f77c-4zcdg". Both pods are running.

```
C:\Users\srira>kubectl get pods,deployments,services
NAME                                     READY   STATUS    RESTARTS   AGE
pod/heart-disease-deployment-b4694f77c-46lnb   1/1     Running   1 (6m40s ago)   12h
pod/heart-disease-deployment-b4694f77c-4zcdg   1/1     Running   1 (6m40s ago)   12h

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/heart-disease-deployment   2/2      2           2           12h

NAME                  TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/heart-disease-service   LoadBalancer   10.103.64.6  127.0.0.1   80:32301/TCP   17h
service/kubernetes          ClusterIP   10.96.0.1    <none>       443/TCP    13d
```

The screenshot shows the API documentation for the "Heart Disease Prediction API". The URL is `127.0.0.1/docs#default/predict_heart_disease_predict_post`. The page title is "Heart Disease Prediction API" with version "0.1.0" and "OAS 3.1".

The "default" section contains two endpoints:

- GET /Health Check**: A simple health check endpoint.
- POST /predict Predict Heart Disease**: An endpoint for predicting heart disease. It has a "Parameters" section (No parameters) and a "Request body" section (required). The request body schema is defined as follows:

```
{
  "age": 58.0,
  "sex": 1.0,
  "cp": 3.0,
  "trtbps": 130.0,
  "chol": 256.0,
  "fbs": 1.0,
  "restecg": 2.0,
  "thalach": 142.0,
  "exang": 1.0,
  "oldpeak": 2.3,
  "slope": 2.0,
  "ca": 1.0,
  "thal": 6.0
}
```

```

Logs from heart-disease-... in heart-disease-...
INFO: 10.244.0.143:51244 - "GET /metrics HTTP/1.1" 200 OK
2026-01-06 07:58:12,480 - INFO - GET /metrics status=200 duration=0.002s
INFO: 10.244.0.143:47586 - "GET /metrics HTTP/1.1" 200 OK
2026-01-06 07:58:12,480 - INFO - GET /metrics status=200 duration=0.002s
INFO: 10.244.0.143:47586 - "GET /metrics HTTP/1.1" 200 OK
INFO: 10.244.0.143:47586 - "GET /metrics HTTP/1.1" 200 OK
2026-01-06 07:58:42,479 - INFO - GET /metrics status=200 duration=0.003s
INFO: 10.244.0.143:56468 - "GET /metrics HTTP/1.1" 200 OK
2026-01-06 07:58:57,709 - INFO - GET /metrics status=200 duration=0.099s
INFO: 10.244.0.143:37744 - "GET /metrics HTTP/1.1" 200 OK
2026-01-06 07:59:05,450 - INFO - GET /status=200 duration=0.052s
INFO: 10.244.0.142:2849 - "GET / HTTP/1.1" 200 OK
2026-01-06 07:59:05,566 - INFO - GET /favicon.ico status=404 duration=0.007s
INFO: 10.244.0.142:2849 - "GET /favicon.ico HTTP/1.1" 404 Not Found
2026-01-06 07:59:05,566 - INFO - GET /metrics status=200 duration=0.01s
INFO: 10.244.0.143:42488 - "GET /metrics HTTP/1.1" 200 OK
2026-01-06 07:59:23,179 - INFO - GET /docs status=200 duration=0.016s
INFO: 10.244.0.143:51244 - "GET /docs HTTP/1.1" 200 OK
2026-01-06 07:59:23,716 - INFO - GET /openapi.json status=200 duration=0.449s
INFO: 10.244.0.143:51244 - "GET /openapi.json HTTP/1.1" 200 OK
2026-01-06 07:59:27,486 - INFO - GET /metrics status=200 duration=0.004s
INFO: 10.244.0.143:52566 - "GET /metrics HTTP/1.1" 200 OK
2026-01-06 07:59:42,485 - INFO - GET /metrics status=200 duration=0.004s
INFO: 10.244.0.143:52566 - "GET /metrics HTTP/1.1" 200 OK
2026-01-06 07:59:47,342 - INFO - POST /predict status=200 duration=0.159s
2026-01-06 07:59:47,344 - INFO - POST /predict status=200 duration=0.815s
INFO: 10.244.0.1:5761 - "POST /predict HTTP/1.1" 200 OK
Logs from Jan 6, 2026 to Jan 6, 2026 UTC

```

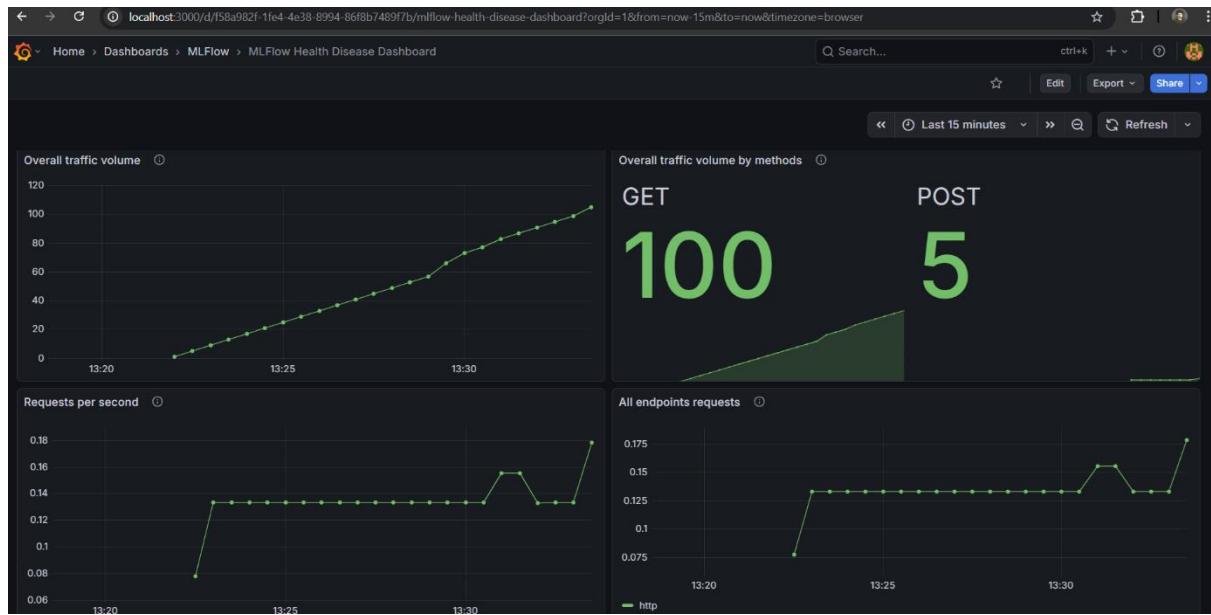
Prometheus metrics:

```

process_resident_memory_bytes 2.08457728e+08
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.76768590473e+09
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 8.129999999999999
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 9.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 1.048576e+06
# HELP api_requests_total Total API requests
# TYPE api_requests_total counter
api_requests_total{endpoint="/metrics",method="GET"} 36.0
api_requests_total{endpoint="/predict",method="GET"} 1.0
api_requests_total{endpoint="/predict",method="POST"} 2.0
api_requests_total{endpoint="/",method="GET"} 1.0
# HELP api_requests_created Total API requests
# TYPE api_requests_created gauge
api_requests_created{endpoint="/metrics",method="GET"} 1.767685927748295e+09
api_requests_created{endpoint="/predict",method="GET"} 1.7676863576104758e+09
api_requests_created{endpoint="/predict",method="POST"} 1.7676863720557714e+09
api_requests_created{endpoint="/",method="GET"} 1.7676864481392646e+09
# HELP api_request_latency_seconds Request latency
# TYPE api_request_latency_seconds histogram
api_request_latency_seconds_bucket{le="0.005"} 17.0
api_request_latency_seconds_bucket{le="0.01"} 29.0
api_request_latency_seconds_bucket{le="0.025"} 32.0
api_request_latency_seconds_bucket{le="0.05"} 35.0
api_request_latency_seconds_bucket{le="0.075"} 35.0
api_request_latency_seconds_bucket{le="0.1"} 36.0
api_request_latency_seconds_bucket{le="0.25"} 37.0
api_request_latency_seconds_bucket{le="0.5"} 38.0
api_request_latency_seconds_bucket{le="0.75"} 38.0
api_request_latency_seconds_bucket{le="1.0"} 38.0
api_request_latency_seconds_bucket{le="2.5"} 39.0
api_request_latency_seconds_bucket{le="5.0"} 39.0
api_request_latency_seconds_bucket{le="7.5"} 39.0
api_request_latency_seconds_bucket{le="10.0"} 39.0
api_request_latency_seconds_bucket{le="+Inf"} 39.0
api_request_latency_seconds_count 39.0
api_request_latency_seconds_sum 2.03809096900045
# HELP api_request_latency_seconds_created Request latency
# TYPE api_request_latency_seconds_created gauge
api_request_latency_seconds_created 1.7676859109234667e+09

```

Grafana Dashboard:



Link to code repository:

GitHub Link: <https://github.com/sriramgunda/MLOPS>

Additional Screenshots:

Please refer documentation folder for additional screenshots.
<https://github.com/sriramgunda/MLOPS/tree/main/documentation>

Demonstration Link:

Please refer the video uploaded to YouTube: <https://www.youtube.com/watch?v=SqjqfYa2Kyc>