MLOPS_Group40

January 6, 2026

MLOPS - Group 40

1. YALLA SIREESHA - 2024AA05728 - 100%
2. DIKSHA GUPTA - 2024AA05518 - 100%
3. SUVITHA J - 2024AA05756 - 100%
4. VINAYAK BAJORIA - 2024AA05493 - 100%
5. ANIL PRASAD CHELAMCHELA - 2024AA05896 - 100%

1 Problem Statement:

Build a machine learning classifier to predict the risk of heart disease based on patient health data, and deploy the solution as a cloud-ready, monitored API. Assignment Tasks

2 1. Data Acquisition & Exploratory Data Analysis (EDA) [5 marks]

Obtain the dataset (provide download script or instructions). Clean and preprocess the data (ha

1.1 Dataset & Download Instructions
Source: UCI Machine Learning Repository — Cleveland processed file (processed.cleveland.data) w

[47]:
```
# Download script (src/data.py) outline:


# src/data.py
import certifi, requests
from pathlib import Path

UCI_URL = ("https://archive.ics.uci.edu/ml/machine-learning-databases/"
           "heart-disease/processed.cleveland.data")
OUT_DIR = Path("data")
OUT_DIR.mkdir(parents=True, exist_ok=True)
RAW_PATH = OUT_DIR / "processed.cleveland.data"

def download(force: bool=False) -> Path:
    if RAW_PATH.exists() and not force:
        print(f"Using existing: {RAW_PATH}")
        return RAW_PATH
```

```
    # verify with certifi CA bundle (good practice behind corporate proxies)
    resp = requests.get(UCI_URL, timeout=30, verify=certifi.where())
    resp.raise_for_status()
    RAW_PATH.write_text(resp.text)
    print(f"Downloaded to: {RAW_PATH}")
    return RAW_PATH

if __name__ == "__main__":
    download()
```

```
Using existing: data/processed.cleveland.data
```

1.2 Column Names & Loading

Map columns (13 features + num) as per UCI description. We convert num to a binary target: 0 =

[48]:
```
# in EDA notebook or src/clean.py
import pandas as pd
from pathlib import Path

cols = ["age","sex","cp","trestbps","chol","fbs","restecg",
        "thalach","exang","oldpeak","slope","ca","thal","num"]

raw = Path("data/processed.cleveland.data")
df = pd.read_csv(raw, header=None, names=cols)

# convert '?' to NaN, then proper dtypes
df = df.replace("?", pd.NA)

# numeric columns
num_cols = ["age","trestbps","chol","thalach","oldpeak","ca"]
cat_cols = ["sex","cp","fbs","restecg","exang","slope","thal"]

for c in num_cols:
    df[c] = pd.to_numeric(df[c], errors="coerce")
for c in cat_cols + ["num"]:
    df[c] = pd.to_numeric(df[c], errors="coerce")

# Binary target
df["target"] = (df["num"] > 0).astype(int)
df = df.drop(columns=["num"])
```

1.3 Missing Values & Basic Cleaning

Missing markers (?) → NaN.
We will impute numerics with median and categoricals with most frequent during the pipeline st

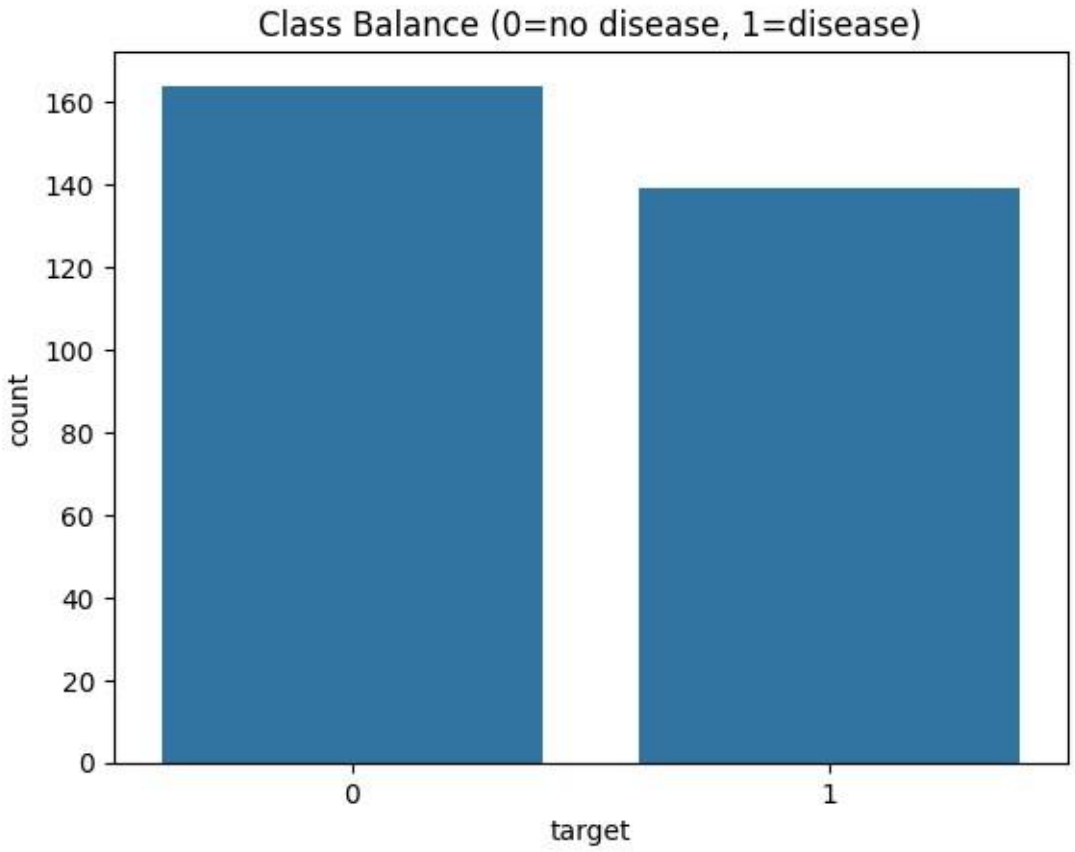
[49]:
```
print(df.isna().sum().sort_values(ascending=False))
print(df.shape, df.head())
```

```
ca          4
thal        2
cp          0
trestbps    0
age         0
sex         0
fbs         0
chol        0
restecg     0
thalach     0
oldpeak     0
exang       0
slope       0
target      0
dtype: int64
      (303, 14) age   sex   cp trestbps  chol  fbs  restecg thalach exang
oldpeak \
0     63.0  1.0   1.0   145.0 233.0 1.0   2.0   150.0 0.0   2.3
1     67.0  1.0   4.0   160.0 286.0 0.0   2.0   108.0 1.0   1.5
2     67.0  1.0   4.0   120.0 229.0 0.0   2.0   129.0 1.0   2.6
3     37.0  1.0   3.0   130.0 250.0 0.0   0.0   187.0 0.0   3.5 4 41.0  0.0   2.0   130.0
      204.0 0.0   2.0   172.0 0.0   1.4

       slope  ca  thal  target
0    3.0    0.0   6.0   0
1    2.0    3.0   3.0   1
2    2.0    2.0   7.0   1
3    3.0    0.0   3.0   0
4    1.0    0.0   3.0   0
```

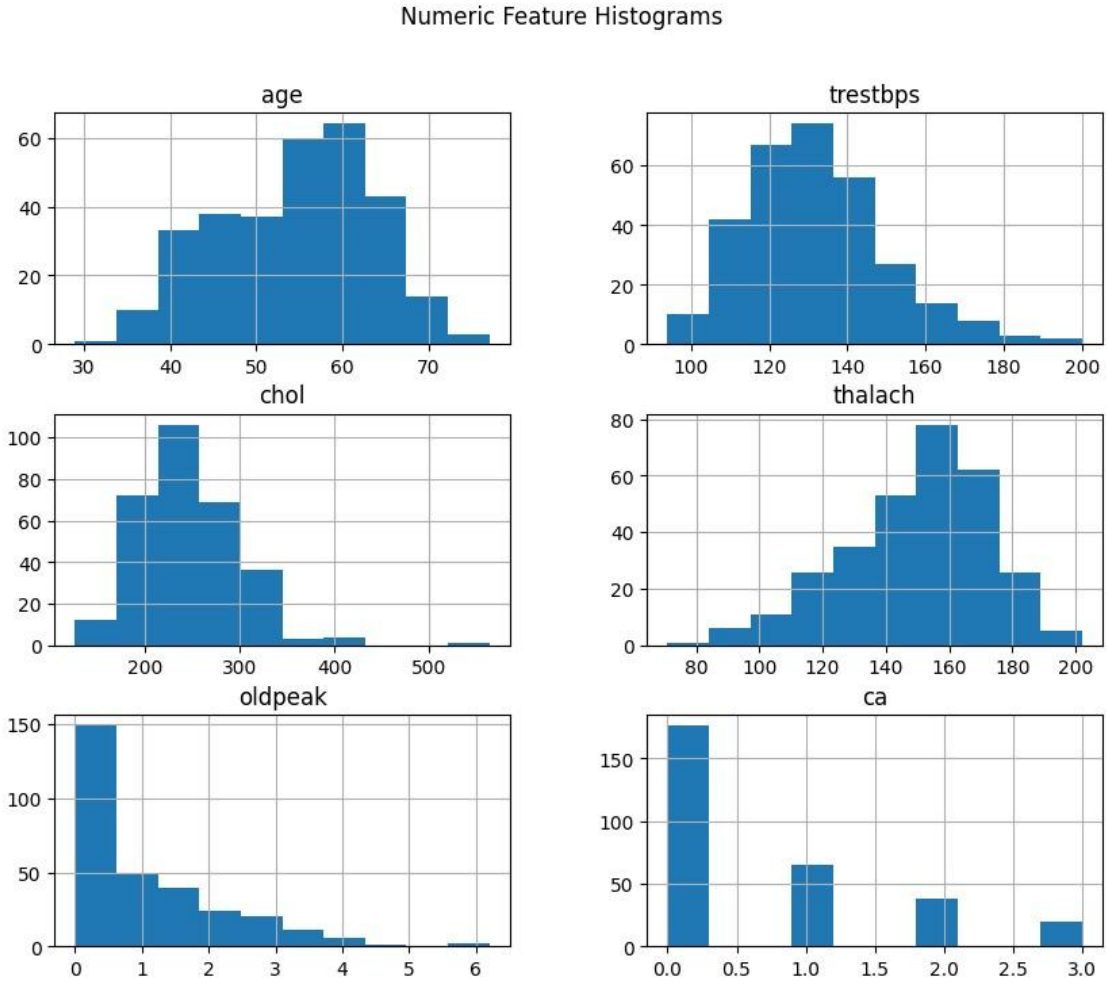
1.4 EDA Visualizations

Class Balance
Feature Distribution
Correlation Heatmap
Box plots by target

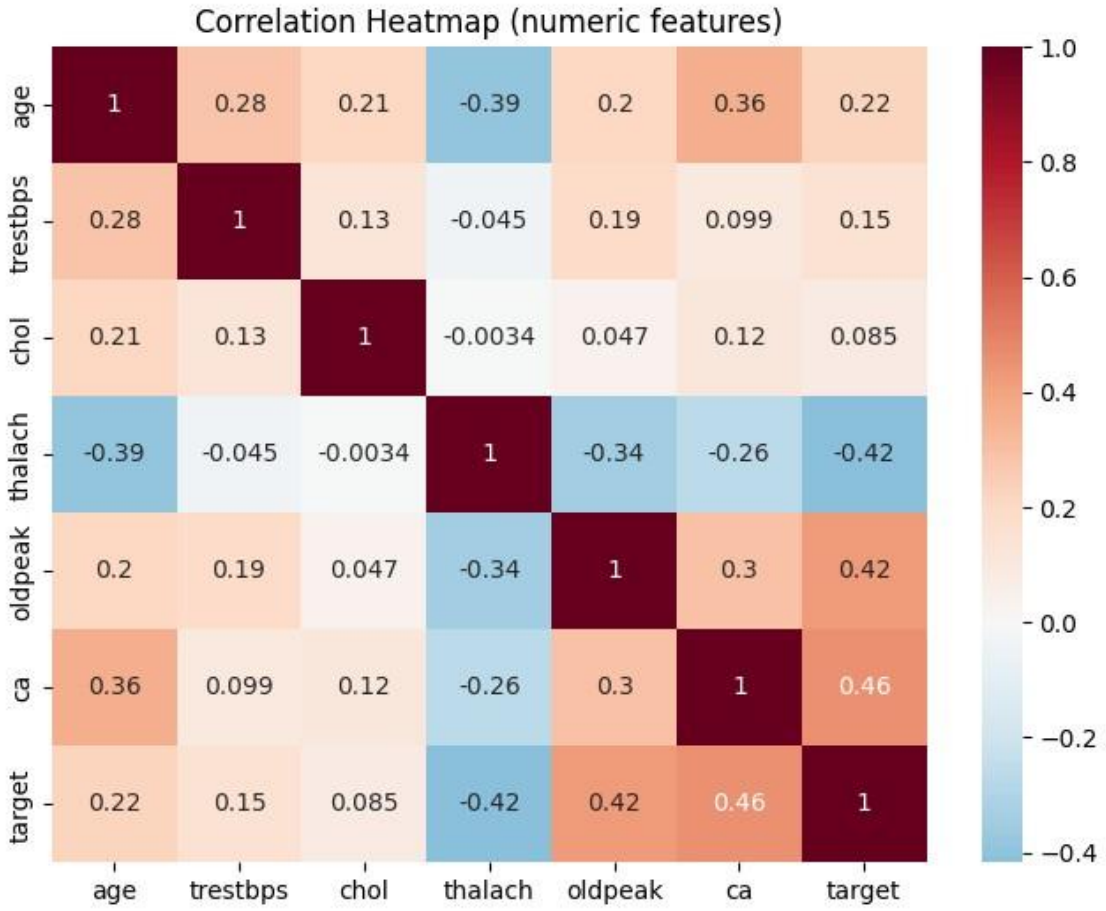
[50]:
```
# Class Balance:

import seaborn as sns, matplotlib.pyplot as plt
sns.countplot(x="target", data=df)
plt.title("Class Balance (0=no disease, 1=disease) ")
plt.show()
```



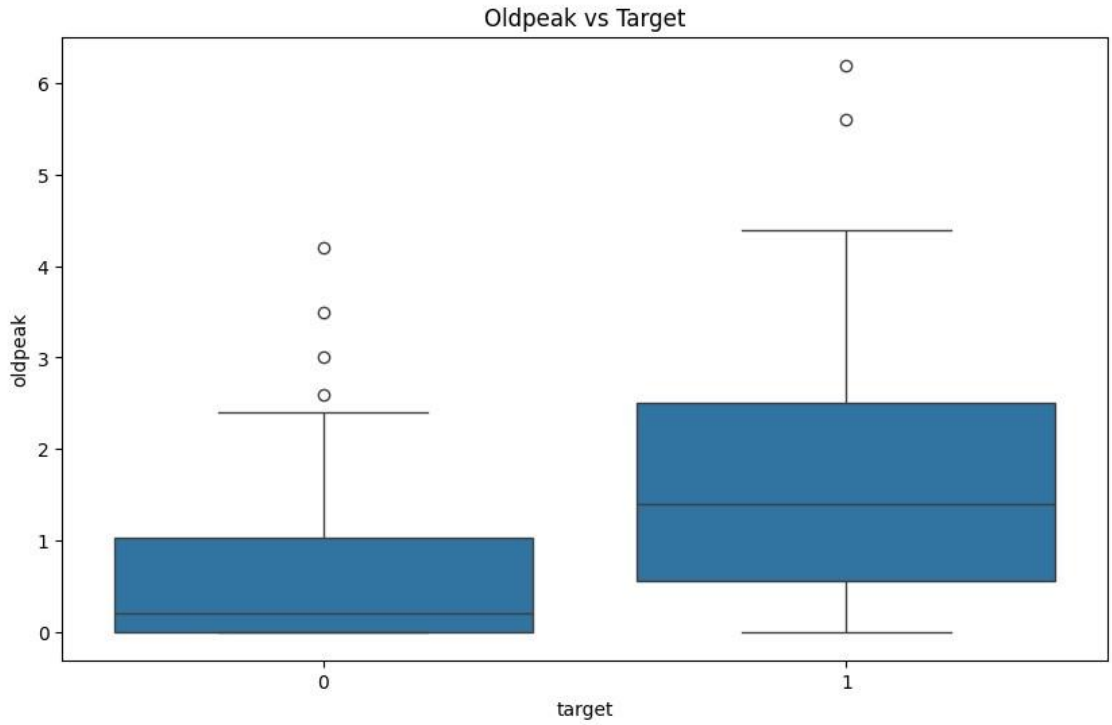
[51]:
```
# Feature Distributions (numerics):

df[num_cols].hist(figsize=(10,8))
plt.suptitle("Numeric Feature Histograms")
plt.show()
```



[52]:
```
# Correlation Heatmap (numerics):

corr = df[num_cols + ["target"]].corr()
plt.figure(figsize=(8,6))
sns.heatmap(corr, annot=True, cmap="RdBu_r", center=0)
plt.title("Correlation Heatmap (numeric features) ")
plt.show()
```



[53]:
```
# Box plots by target (optional):


plt.figure(figsize=(10,6))
sns.boxplot(x="target",  ="oldpeak", data=df)
plt.title("Oldpeak vs Target")
plt.show()
```



3 2. Feature Engineering & Model Development — [8 marks]

2.1 Final Feature Pipeline We use ColumnTransformer to:

Numeric: age, trestbps, chol, thalach, oldpeak, ca → median imputation + standard scaling

Categorical: sex, cp, fbs, restecg, exang, slope, thal → most frequent imputation + one-hot en

[54]:
```
from sklearn.compose import
ColumnTransformer from sklearn.pipeline
import Pipeline from sklearn.impute
import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler

num_features = ["age","trestbps","chol","thalach","oldpeak","ca"]
cat_features = ["sex","cp","fbs","restecg","exang","slope","thal"]

numeric_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
```

```
])
categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))
])

preprocess = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, num_features),
        ("cat", categorical_transformer, cat_features)
    ]
)
```

2.2 Train/Test Split & Cross-Validation Setup

We evaluate via Stratified K-Fold (CV) to ensure class proportions. Metrics: accuracy, precisio

[55]:
```
from sklearn.model_selection import StratifiedKFold, cross_validate
 ↪recall_score, roc_auc_score

X = df[num_features + cat_features]
y = df["target"]

scorers = {
    "accuracy": make_scorer(accuracy_score),
    "precision": make_scorer(precision_score),
    "recall": make_scorer(recall_score),
    "roc_auc": make_scorer(roc_auc_score)
}
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

```
from sklearn.metrics import make_scorer, accuracy_score, precision_score,␣
```

2.3 Models & Pipelines

We train Logistic Regression (baseline, interpretable) and Random Forest (nonlinear, robust).

[56]:
```
from sklearn.linear_model import
LogisticRegression from sklearn.ensemble import
RandomForestClassifier

# Logistic Regression (C tuned lightly)
lr_pipe = Pipeline(steps=[
    ("preprocess", preprocess),
    ("clf", LogisticRegression(max_iter=1000, C=1.0,
    class_weight="balanced"))
])
```

```
# Random Forest (shallow baseline; tune n_estimators &
max_depth) rf_pipe = Pipeline(steps=[
    ("preprocess", preprocess),
    ("clf", RandomForestClassifier( n_estimators=300,
max_depth=None, min_samples_split=2,
class_weight="balanced", random_state=42)) ])
```

2.4 Cross-Validated Results

[57]:
```
import numpy as np

def eval_model(pipe, name): scores = cross_validate(pipe, X, y,
    cv=cv, scoring=scorers, n_jobs=-1,␣
 ↪return_estimator=False) print(f"\n{name} CV
    Results (5-fold):\n") for m in
    ["accuracy","precision","recall","roc_auc"]:
        vals = scores[f"test_{m}"]
        print(f" {m:>9}: mean={vals.mean():.3f}, std={vals.std():.3f}")
    return scores

lr_scores = eval_model(lr_pipe, "LogisticRegression")
rf_scores = eval_model(rf_pipe, "RandomForest")
```

```
LogisticRegression CV Results (5-fold):

   accuracy: mean=0.845, std=0.027
  precision: mean=0.856, std=0.053
  recall: mean=0.798, std=0.030
  roc_auc: mean=0.842, std=0.026

RandomForest CV Results (5-fold):

   accuracy: mean=0.825, std=0.027
  precision: mean=0.844, std=0.047
  recall: mean=0.762, std=0.039
  roc_auc: mean=0.820, std=0.027
```

2.5 Model Selection & Tuning

We select RandomForest when ROC-AUC and recall are critical (detecting positives).
LR is still

Grid search n_estimators, max_depth, min_samples_leaf
Balance precision-recall trade-off (threshold tuning)
Calibrate probabilities if needed (CalibratedClassifierCV)

[58]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {
    "clf__n_estimators":  200, 300, 500],
    "clf__max_depth":  None, 8, 12],
    "clf__min_samples_leaf":  1, 2, 3]
}
gs = GridSearchCV(rf_pipe, param_grid=param_grid, cv=cv, scoring="roc_auc",␣
 ↪n_jobs=-1)
gs.fit(X, y)

print("Best params:", gs.best_params_)
```

```
print("Best CV ROC-AUC:", gs.best_score_)
best_rf = gs.best_estimator_
```

```
Best params: {'clf__max_depth': 8, 'clf__min_samples_leaf': 3,
'clf__n_estimators': 200}
Best CV ROC-AUC: 0.9175214446047779
```

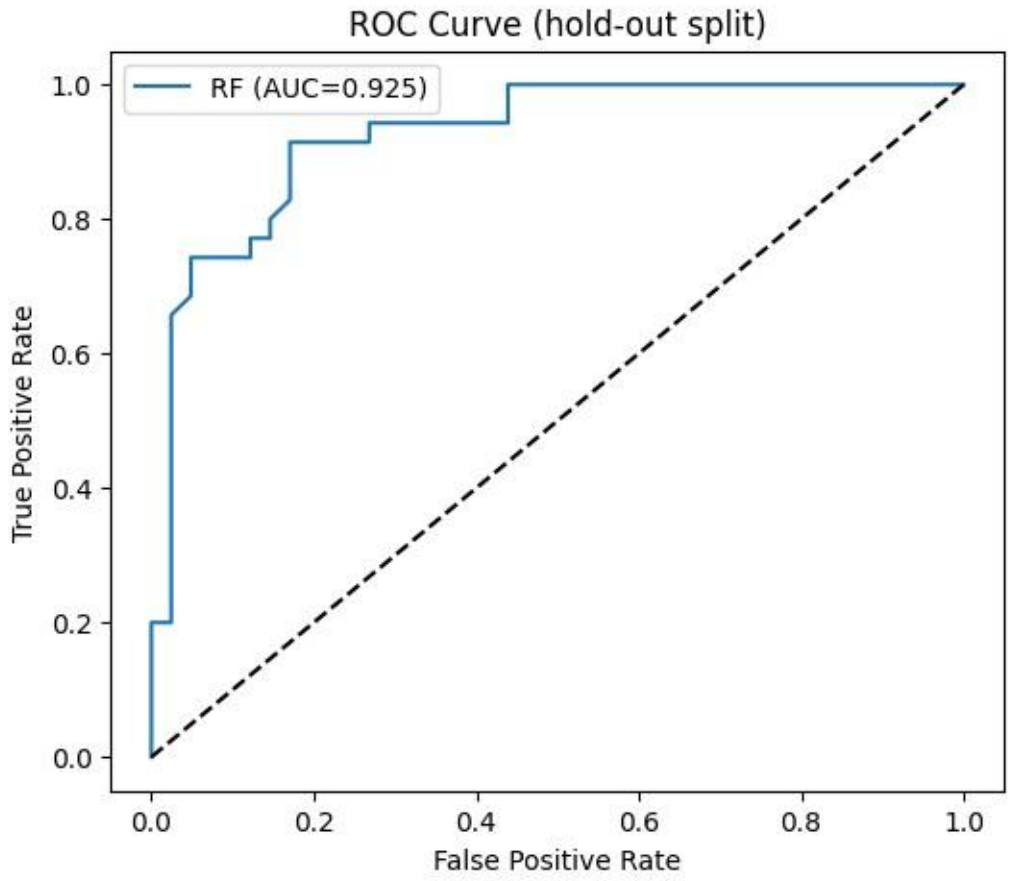
2.6 Final Training & Artifacts
Train the chosen model on full data, save artifacts for API inference.

[59]:
```
from pathlib import Path
import joblib, json

ART_DIR = Path("artifacts"); ART_DIR.mkdir(exist_ok=True)

# choose: best_rf or rf_pipe.fit(X, y)
rf_pipe.fit(X, y)

joblib.dump(rf_pipe, ART_DIR / "model.joblib")
metrics = {
    "cv_lr":  m: float(np.mean(lr_scores[f"test_{m}"])) for m in␣
 ↪["accuracy","precision","recall","roc_auc"]},
    "cv_rf":  m: float(np.mean(rf_scores[f"test_{m}"])) for m in␣
 ↪["accuracy","precision","recall","roc_auc"]}
}
json.dump(metrics, open(ART_DIR / "metrics.json","w"), indent=2)
print("Saved artifacts to:", ART_DIR)
```

```
Saved artifacts to: artifacts
```

2.7 ROC Curve & Threshold Analysis (optional visualization)

[60]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
 ↪test_size=0.25, random_state=42)
rf_pipe.fit(X_train, y_train)

# Predict probabilities (positive class)
import matplotlib.pyplot as plt
y_proba = rf_pipe.predict_proba(X_test)[:,1]
fpr, tpr, thr = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, label=f"RF (AUC={roc_auc:.3f})")
plt.plot([0,1], [0,1], "k--")
plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
plt.title("ROC Curve (hold-out split)")
plt.legend(); plt.show()
```



4 3. Experiment Tracking [5 marks]

Integrate MLflow (or a similar tool) for experiment tracking. Log parameters, metrics, artifact

A) Install & configure
MLflow 1) Requirements

```
# requirements.txt (add if missing)
mlflow==2.2.0
```

2) Local tracking URI options

File-based (simple): MLflow will default to a local ./mlruns directory if you don’t set a track

Explicit local UI:

```
mlflow ui --host 0.0.0.0 --port 5000
# open http://localhost:5000
```

B) Integrate MLflow in src/train.py

Below is a drop-in train script skeleton that logs parameters, metrics, artifacts, and plots fo

[61]:
```
# src/train.py

!pip install mlflow
import argparse, json, os, time, joblib
from pathlib import Path
import numpy as np
import pandas as pd

from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import StratifiedKFold, cross_validate,␣
 ↪train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             roc_auc_score, confusion_matrix, roc_curve, auc)

import matplotlib.pyplot as plt
import mlflow
import mlflow.sklearn

# ---------- data load ----------
def load_df():
    cols = ["age","sex","cp","trestbps","chol","fbs","restecg",
            "thalach","exang","oldpeak","slope","ca","thal","num"]
    df = pd.read_csv("data/processed.cleveland.data", header=None, names=cols)
    df = df.replace("?", pd.NA)

    num_cols = ["age","trestbps","chol","thalach","oldpeak","ca"]
    cat_cols = ["sex","cp","fbs","restecg","exang","slope","thal"]

    for c in num_cols:
        df[c] = pd.to_numeric(df[c], errors="coerce")
    for c in cat_cols + ["num"]:
        df[c] = pd.to_numeric(df[c], errors="coerce")

    df["target"] = (df["num"] > 0).astype(int)
    df = df.drop(columns=["num"])
    return df, num_cols, cat_cols

# ---------- preprocess ----------
```

```
def build_preprocess(num_cols, cat_cols):
    numeric_transformer = Pipeline(steps=[
        ("imputer", SimpleImputer(strategy="median")),
        ("scaler", StandardScaler())
    ])
    categorical_transformer = Pipeline(steps=[
        ("imputer", SimpleImputer(strategy="most_frequent")),
        ("onehot", OneHotEncoder(handle_unknown="ignore"))
    ])
    return ColumnTransformer([
        ("num", numeric_transformer, num_cols),
        ("cat", categorical_transformer, cat_cols)
    ])

def plot_and_save_roc(y_true, y_proba, out_path):
    fpr, tpr, _ = roc_curve(y_true, y_proba)
    roc_auc = auc(fpr, tpr)
    plt.figure(figsize=(6,5))
    plt.plot(fpr, tpr, label=f"AUC={roc_auc:.3f}")
    plt.plot([0,1], [0,1], "k--")
    plt.xlabel("FPR"); plt.ylabel("TPR"); plt.title("ROC Curve")
    plt.legend()
    plt.tight_layout()
    plt.show()
    plt.savefig(out_path)
    plt.close()
    return roc_auc

def plot_and_save_confusion(y_true, y_pred, out_path):
    cm = confusion_matrix(y_true, y_pred)
    fig, ax = plt.subplots(figsize=(4,4))
    im = ax.imshow(cm, cmap="Blues")
    ax.set_title("Confusion Matrix")
    ax.set_xlabel("Predicted"); ax.set_ylabel("True")
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, cm[i,j], ha="center", va="center", color="black")
    fig.colorbar(im)
    plt.tight_layout()
    plt.show()
    plt.savefig(out_path)
    plt.close()

def main(model_name):

    model = model_name
    n_estimators = 300
```

```
    max_depth = None
    min_samples_leaf = 1
    C = 1
    cv = 5
    # Prepare experiment
    experiment_name =  "HeartDisease"
    mlflow.set_experiment(experiment_name)

    df, num_cols, cat_cols = load_df()
    X = df[num_cols + cat_cols]
    y = df["target"]

    preprocess = build_preprocess(num_cols, cat_cols)

    # Choose model
    if model == "lr":
        clf = LogisticRegression(max_iter=1000, C=C, class_weight="balanced")
    elif model == "rf":
        clf = RandomForestClassifier(
            n_estimators=n_estimators, max_depth=max_depth,
            min_samples_leaf=min_samples_leaf, class_weight="balanced",
            random_state=42
        )
    else:
        raise ValueError("Unsupported model. Use 'lr' or 'rf'.")

    pipe = Pipeline(steps=[("preprocess", preprocess), ("clf", clf)])

    # Cross-validation
    cv = StratifiedKFold(n_splits=cv, shuffle=True, random_state=42)
    scoring = ["accuracy", "precision", "recall", "roc_auc"]
    scores = cross_validate(pipe, X, y, cv=5, scoring=scoring, n_jobs=-1)

    metrics_cv = {m: float(np.mean(scores[f"test_{m}"])) for m in scoring}

    # Hold-out for plots & confusion matrix
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, stratify=y, test_size=0.25, random_state=42
    )
    pipe.fit(X_train, y_train)

    y_pred = pipe.predict(X_test)
    try:
        y_proba = pipe.predict_proba(X_test)[:,1]
    except Exception:
        y_proba = None
```

```
    # Artifacts dir
    ART_DIR = Path("artifacts"); ART_DIR.mkdir(exist_ok=True)

    # Save model
    model_path = ART_DIR / "model.joblib"
    joblib.dump(pipe, model_path)

    # Plots
    cm_path = ART_DIR / "confusion_matrix.png"
    plot_and_save_confusion(y_test, y_pred, cm_path)

    roc_auc_holdout = None
    roc_path = ART_DIR / "roc_curve.png"
    if y_proba is not None:
        roc_auc_holdout = plot_and_save_roc(y_test, y_proba, roc_path)

    # Feature importance (RF only, after preprocessing → use classifier␣
↪feature_importances_)
    fi_path = None
    if model == "rf" and hasattr(clf, "feature_importances_"):
        # Fit on full data (optional) to compute FI on pipeline output
        pipe.fit(X, y)
        # get feature names
        try:
            feat_names = pipe.named_steps["preprocess"].get_feature_names_out()
        except Exception:
            feat_names = [f"f{i}" for i in range(pipe.named_steps["preprocess"].
↪transform(X).shape[1])]
        importances = pipe.named_steps["clf"].feature_importances_
        fi_df = pd.DataFrame({"feature": feat_names, "importance":␣
↪importances})\
                  .sort_values(by="importance", ascending=False)
        fi_path = ART_DIR / "feature_importance.csv"
        fi_df.to_csv(fi_path, index=False)

    # Prepare run tags (optional)
    run_tags = {
        "model_type": model,
        "dataset": "UCI-Cleveland",
        "git_sha": os.environ.get("GIT_SHA", "unknown"),
    }

    # ---- MLflow logging ----
    with mlflow.start_run(run_name=f"{model.upper()}_{time.
↪strftime('%Y%m%d-%H%M%S')}"):
        mlflow.set_tags(run_tags)
```

```
        # Params
        if model == "lr":
            mlflow.log_params({"max_iter": 1000, "C": C, "class_weight":␣
↪"balanced"})
        else:
            mlflow.log_params({
                "n_estimators": 300,
                "max_depth": None,
                "min_samples_leaf": 1,
                "class_weight": "balanced",
                "random_state": 42
            })

        mlflow.log_params({
            "cv_splits": 5,
            "numeric_features": ",".join(num_cols),
            "categorical_features": ",".join(cat_cols)
        })

        # Metrics (cross-validated)
        mlflow.log_metrics(metrics_cv)

        # Additional hold-out metrics
        mlflow.log_metrics({
            "accuracy_holdout": accuracy_score(y_test, y_pred),
            "precision_holdout": precision_score(y_test, y_pred),
            "recall_holdout": recall_score(y_test, y_pred),
            **({"roc_auc_holdout": roc_auc_holdout} if roc_auc_holdout is not␣
↪None else {})
        })
        print(f"\nCV: ")
        print(f"\taccuracy_holdout: {accuracy_score(y_test, y_pred)}")
        print(f"\tprecision_holdout: {precision_score(y_test, y_pred)}")
        print(f"\trecall_holdout: {recall_score(y_test, y_pred)}")
        print(f"\troc_auc_holdout : {roc_auc_holdout if roc_auc_holdout is not␣
↪None else {}}")

        # Artifacts
        mlflow.log_artifact(str(model_path))
        mlflow.log_artifact(str(cm_path))
        if roc_auc_holdout is not None:
            mlflow.log_artifact(str(roc_path))
        if fi_path is not None:
            mlflow.log_artifact(str(fi_path))

        # Optionally: log model as MLflow model
        # mlflow.sklearn.log_model(pipe, artifact_path="sklearn-model")
```

```
    # Also write metrics.json for
    your repo metrics_json = {"cv":
    metrics_cv} if roc_auc_holdout is
    not None:
        metrics_json["holdout"] = {"roc_auc": roc_auc_holdout}
    print("\nholdout:")
    print(f"\troc_auc: {roc_auc_holdout}")
    json.dump(metrics_json, open(ART_DIR / "metrics.json","w"),
    indent=2)
```

```
Requirement already satisfied: mlflow in
/usr/local/lib/python3.12/dist-packages (3.8.1)
Requirement already satisfied: mlflow-skinny==3.8.1 in
/usr/local/lib/python3.12/dist-packages (from mlflow) (3.8.1)
Requirement already satisfied: mlflow-tracing==3.8.1 in
/usr/local/lib/python3.12/dist-packages (from mlflow) (3.8.1)
Requirement already satisfied: Flask-CORS<7 in
/usr/local/lib/python3.12/distpackages (from mlflow) (6.0.2)
Requirement already satisfied: Flask<4 in
/usr/local/lib/python3.12/distpackages (from mlflow) (3.1.2)
Requirement already satisfied: alembic!=1.10.0,<2 in
/usr/local/lib/python3.12/dist-packages (from mlflow) (1.17.2)
Requirement already satisfied: cryptography<47,>=43.0.0 in
/usr/local/lib/python3.12/dist-packages (from mlflow) (43.0.3)
Requirement already satisfied: docker<8,>=4.0.0 in
/usr/local/lib/python3.12/dist-packages (from mlflow) (7.1.0)
Requirement already satisfied: graphene<4 in
/usr/local/lib/python3.12/distpackages (from mlflow) (3.4.3)
Requirement already satisfied: gunicorn<24 in
/usr/local/lib/python3.12/distpackages (from mlflow) (23.0.0)
Requirement already satisfied: huey<3,>=2.5.0 in
/usr/local/lib/python3.12/distpackages (from mlflow) (2.5.5)
Requirement already satisfied: matplotlib<4 in
/usr/local/lib/python3.12/distpackages (from mlflow) (3.10.0)
Requirement already satisfied: numpy<3 in
/usr/local/lib/python3.12/distpackages (from mlflow) (2.0.2)
Requirement already satisfied: pandas<3 in
/usr/local/lib/python3.12/distpackages (from mlflow) (2.2.2)
Requirement already satisfied: pyarrow<23,>=4.0.0 in
/usr/local/lib/python3.12/dist-packages (from mlflow) (18.1.0)
Requirement already satisfied: scikit-learn<2 in
/usr/local/lib/python3.12/distpackages (from mlflow) (1.6.1)
Requirement already satisfied: scipy<2 in
/usr/local/lib/python3.12/distpackages (from mlflow) (1.16.3)
Requirement already satisfied: sqlalchemy<3,>=1.4.0 in
/usr/local/lib/python3.12/dist-packages (from mlflow) (2.0.45)
```

```
Requirement already satisfied: cachetools<7,>=5.0.0 in
/usr/local/lib/python3.12/dist-packages (from mlflow-skinny==3.8.1-
>mlflow) (6.2.4)
Requirement already satisfied: click<9,>=7.0 in
/usr/local/lib/python3.12/distpackages (from mlflow-skinny==3.8.1-
>mlflow) (8.3.1)
Requirement already satisfied: cloudpickle<4 in
/usr/local/lib/python3.12/dist-
packages (from mlflow-skinny==3.8.1->mlflow) (3.1.2)
Requirement already satisfied: databricks-
sdk<1,>=0.20.0 in
/usr/local/lib/python3.12/dist-packages (from mlflow-skinny==3.8.1-
>mlflow)
(0.76.0)
Requirement already satisfied: fastapi<1 in
/usr/local/lib/python3.12/dist-
packages (from mlflow-skinny==3.8.1->mlflow) (0.123.10)
Requirement already satisfied: gitpython<4,>=3.1.9 in
/usr/local/lib/python3.12/dist-packages (from mlflow-skinny==3.8.1-
>mlflow) (3.1.45)
Requirement already satisfied: importlib_metadata!=4.7.0,<9,>=3.7.0
in
/usr/local/lib/python3.12/dist-packages (from mlflow-skinny==3.8.1-
>mlflow) (8.7.0)
Requirement already satisfied: opentelemetry-api<3,>=1.9.0 in
/usr/local/lib/python3.12/dist-packages (from mlflow-skinny==3.8.1-
>mlflow) (1.37.0)
Requirement already satisfied: opentelemetry-proto<3,>=1.9.0 in
/usr/local/lib/python3.12/dist-packages (from mlflow-skinny==3.8.1-
>mlflow) (1.37.0)
Requirement already satisfied: opentelemetry-sdk<3,>=1.9.0 in
/usr/local/lib/python3.12/dist-packages (from mlflow-skinny==3.8.1-
>mlflow) (1.37.0)
Requirement already satisfied: packaging<26 in
/usr/local/lib/python3.12/dist-
packages (from mlflow-skinny==3.8.1->mlflow) (25.0)
Requirement already satisfied: protobuf<7,>=3.12.0 in
/usr/local/lib/python3.12/dist-packages (from mlflow-skinny==3.8.1-
>mlflow) (5.29.5)
Requirement already satisfied: pydantic<3,>=2.0.0 in
/usr/local/lib/python3.12/dist-packages (from mlflow-skinny==3.8.1-
>mlflow) (2.12.3)
Requirement already satisfied: python-dotenv<2,>=0.19.0 in
/usr/local/lib/python3.12/dist-packages (from mlflow-skinny==3.8.1-
>mlflow) (1.2.1)
```

```
Requirement already satisfied: pyyaml<7,>=5.1 in
/usr/local/lib/python3.12/distpackages (from mlflow-skinny==3.8.1-
>mlflow) (6.0.3) Requirement already satisfied: requests<3,>=2.17.3
in
/usr/local/lib/python3.12/dist-packages (from mlflow-skinny==3.8.1-
>mlflow) (2.32.4)
Requirement already satisfied: sqlparse<1,>=0.4.0 in
/usr/local/lib/python3.12/dist-packages (from mlflow-skinny==3.8.1-
>mlflow) (0.5.4)
Requirement already satisfied: typing-extensions<5,>=4.0.0 in
/usr/local/lib/python3.12/dist-packages (from mlflow-skinny==3.8.1-
>mlflow)
(4.15.0)
Requirement already satisfied: uvicorn<1 in
/usr/local/lib/python3.12/distpackages (from mlflow-skinny==3.8.1-
>mlflow) (0.38.0)
Requirement already satisfied: Mako in
/usr/local/lib/python3.12/dist-packages
(from alembic!=1.10.0,<2->mlflow) (1.3.10)
Requirement already satisfied: cffi>=1.12 in
/usr/local/lib/python3.12/dist-
packages (from cryptography<47,>=43.0.0->mlflow) (2.0.0)
Requirement already satisfied: urllib3>=1.26.0 in
/usr/local/lib/python3.12/dist-packages (from docker<8,>=4.0.0-
>mlflow) (2.5.0) Requirement already satisfied: blinker>=1.9.0 in
/usr/local/lib/python3.12/distpackages (from Flask<4->mlflow) (1.9.0)
Requirement already satisfied: itsdangerous>=2.2.0 in
/usr/local/lib/python3.12/dist-packages (from Flask<4->mlflow)
(2.2.0) Requirement already satisfied: jinja2>=3.1.2 in
/usr/local/lib/python3.12/distpackages (from Flask<4->mlflow) (3.1.6)
Requirement already satisfied: markupsafe>=2.1.1 in
/usr/local/lib/python3.12/dist-packages (from Flask<4->mlflow)
(3.0.3)
Requirement already satisfied: werkzeug>=3.1.0 in
/usr/local/lib/python3.12/dist-packages (from Flask<4->mlflow)
(3.1.4)
Requirement already satisfied: graphql-core<3.3,>=3.1 in
/usr/local/lib/python3.12/dist-packages (from graphene<4->mlflow)
(3.2.7)
Requirement already satisfied: graphql-relay<3.3,>=3.1 in
/usr/local/lib/python3.12/dist-packages (from graphene<4->mlflow)
(3.2.0)
Requirement already satisfied: python-dateutil<3,>=2.7.0 in
/usr/local/lib/python3.12/dist-packages (from graphene<4->mlflow)
(2.9.0.post0)
Requirement already satisfied: contourpy>=1.0.1 in
```

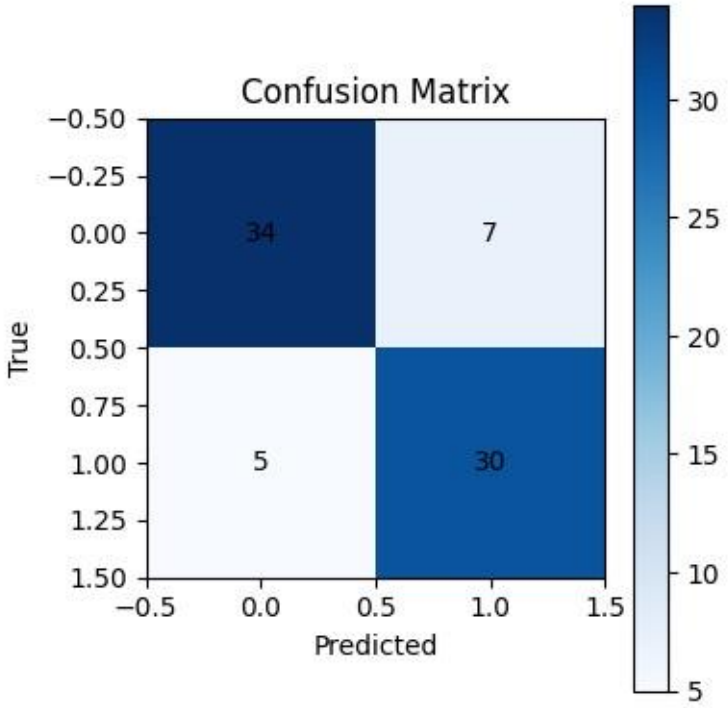
```
/usr/local/lib/python3.12/dist-packages (from matplotlib<4->mlflow)
(1.3.3) Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.12/dist-
packages (from matplotlib<4->mlflow) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0
in
/usr/local/lib/python3.12/dist-packages (from matplotlib<4->mlflow)
(4.61.1)
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib<4-
>mlflow) (1.4.9) Requirement already satisfied: pillow>=8 in
/usr/local/lib/python3.12/distpackages (from matplotlib<4-
>mlflow) (11.3.0) Requirement already satisfied: pyparsing>=2.3.1
in
/usr/local/lib/python3.12/dist-packages (from matplotlib<4->mlflow)
(3.2.5) Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.12/distpackages (from pandas<3->mlflow)
(2025.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.12/distpackages (from pandas<3->mlflow)
(2025.3)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.12/distpackages (from scikit-learn<2->mlflow)
(1.5.3) Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn<2->mlflow)
(3.6.0) Requirement already satisfied: greenlet>=1 in
/usr/local/lib/python3.12/distpackages (from sqlalchemy<3,>=1.4.0-
>mlflow) (3.3.0)
Requirement already satisfied: pycparser in
/usr/local/lib/python3.12/distpackages (from cffi>=1.12-
>cryptography<47,>=43.0.0->mlflow) (2.23)
Requirement already satisfied: google-auth~=2.0 in
/usr/local/lib/python3.12/dist-packages (from databricks-
sdk<1,>=0.20.0->mlflowskinny==3.8.1->mlflow) (2.43.0)
Requirement already satisfied: starlette<0.51.0,>=0.40.0 in
/usr/local/lib/python3.12/dist-packages (from fastapi<1-
>mlflowskinny==3.8.1->mlflow) (0.50.0)
Requirement already satisfied: annotated-doc>=0.0.2 in
/usr/local/lib/python3.12/dist-packages (from fastapi<1-
>mlflowskinny==3.8.1->mlflow) (0.0.4)
Requirement already satisfied: gitdb<5,>=4.0.1 in
/usr/local/lib/python3.12/dist-packages (from gitpython<4,>=3.1.9-
>mlflowskinny==3.8.1->mlflow) (4.0.12)
Requirement already satisfied: zipp>=3.20 in
/usr/local/lib/python3.12/distpackages (from
importlib_metadata!=4.7.0,<9,>=3.7.0->mlflow-
```

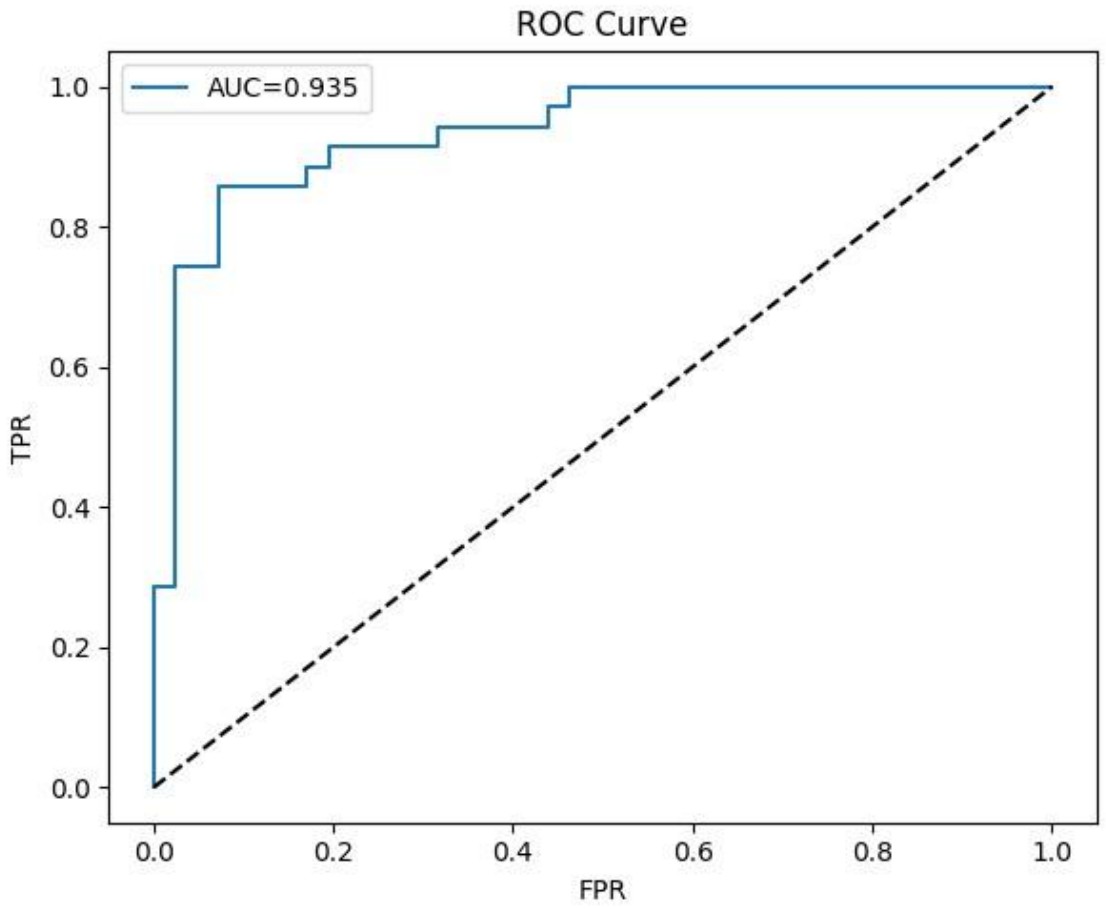
```
skinny==3.8.1->mlflow) (3.23.0)
Requirement already satisfied: opentelemetry-semantic-
conventions==0.58b0 in
/usr/local/lib/python3.12/dist-packages (from
opentelemetrysdk<3,>=1.9.0->mlflow-skinny==3.8.1-
>mlflow) (0.58b0) Requirement already satisfied:
annotated-types>=0.6.0 in
/usr/local/lib/python3.12/dist-packages (from pydantic<3,>=2.0.0-
>mlflowskinny==3.8.1->mlflow) (0.7.0)
Requirement already satisfied: pydantic-core==2.41.4 in
/usr/local/lib/python3.12/dist-packages (from pydantic<3,>=2.0.0-
>mlflowskinny==3.8.1->mlflow) (2.41.4)
Requirement already satisfied: typing-inspection>=0.4.2 in
/usr/local/lib/python3.12/dist-packages (from pydantic<3,>=2.0.0-
>mlflowskinny==3.8.1->mlflow) (0.4.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.12/distpackages (from python-
dateutil<3,>=2.7.0->graphene<4->mlflow) (1.17.0)
Requirement already satisfied: charset_normalizer<4,>=2 in
/usr/local/lib/python3.12/dist-packages (from requests<3,>=2.17.3-
>mlflowskinny==3.8.1->mlflow) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.12/distpackages (from requests<3,>=2.17.3-
>mlflow-skinny==3.8.1->mlflow) (3.11)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.12/dist-packages (from requests<3,>=2.17.3-
>mlflowskinny==3.8.1->mlflow) (2025.11.12)
Requirement already satisfied: h11>=0.8 in
/usr/local/lib/python3.12/distpackages (from uvicorn<1->mlflow-
skinny==3.8.1->mlflow) (0.16.0)
Requirement already satisfied: smmap<6,>=3.0.1 in
/usr/local/lib/python3.12/dist-packages (from
gitdb<5,>=4.0.1->gitpython<4,>=3.1.9->mlflow-skinny==3.8.1->mlflow)
(5.0.2)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.12/dist-packages (from google-auth~=2.0-
>databrickssdk<1,>=0.20.0->mlflow-skinny==3.8.1->mlflow) (0.4.2)
Requirement already satisfied: rsa<5,>=3.1.4 in
/usr/local/lib/python3.12/distpackages (from google-auth~=2.0-
>databricks-sdk<1,>=0.20.0->mlflowskinny==3.8.1->mlflow) (4.9.1)
Requirement already satisfied: anyio<5,>=3.6.2 in
/usr/local/lib/python3.12/dist-packages (from
starlette<0.51.0,>=0.40.0->fastapi<1->mlflow-skinny==3.8.1->mlflow)
(4.12.0)
Requirement already satisfied: pyasn1<0.7.0,>=0.6.1 in
```

```
/usr/local/lib/python3.12/dist-packages (from pyasn1-modules>=0.2.1-
>googleauth~=2.0->databricks-sdk<1,>=0.20.0->mlflow-skinny==3.8.1-
>mlflow) (0.6.1)
```

[62]:
```
# Logistic Regression
print("Logistic Regression:- \n")
main("lr")
```

```
Logistic Regression:-
```

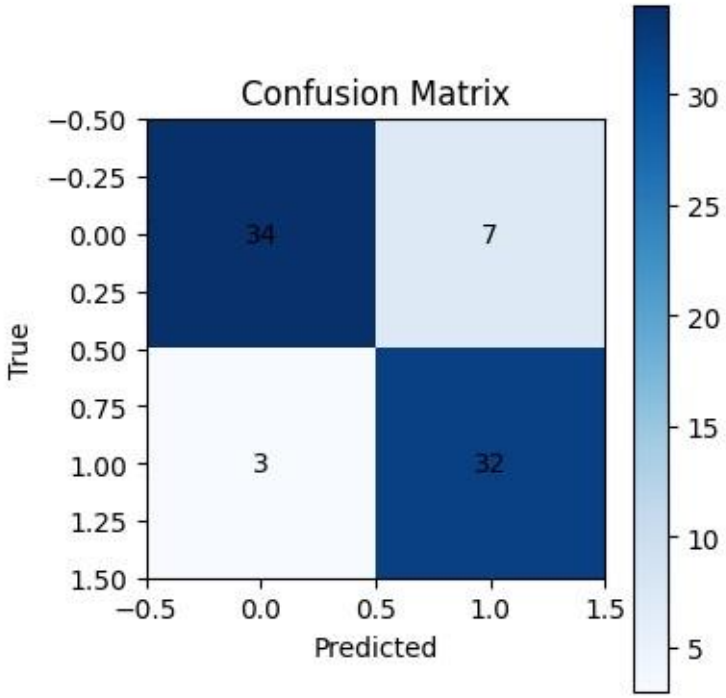


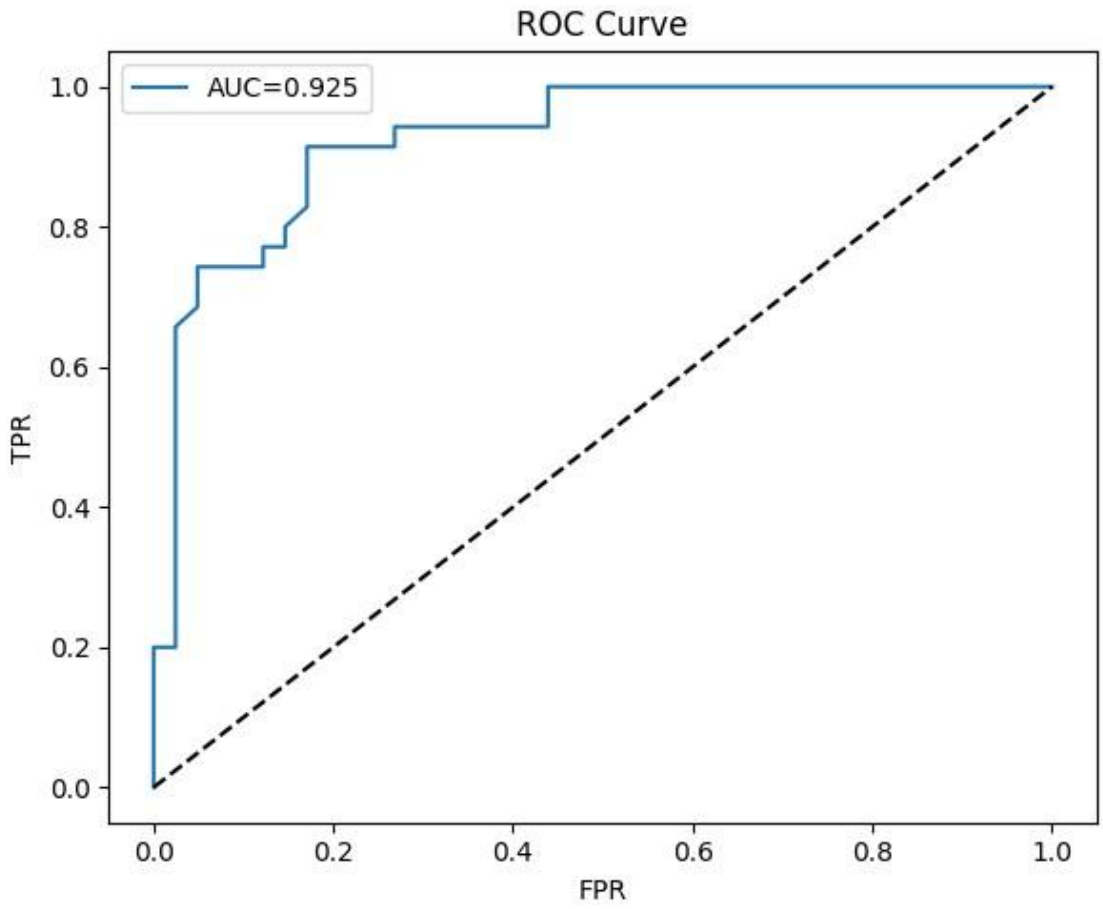



```
CV:
        accuracy_holdout: 0.8421052631578947
        precision_holdout: 0.8108108108108109
        recall_holdout: 0.8571428571428571
        roc_auc_holdout : 0.935191637630662

holdout: roc_auc:
        0.935191637630662
```

[63]:
```
# Random Forest
print("Random forest:- \n")
main("rf")
```

```
Random forest:-
```


Confusion Matrix
True
Predicted
34
7
3
32
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
−0.5
0.0
0.5
1.0
1.5
30
25
20
15
10
5




```
CV: accuracy_holdout: 0.868421052631579
       precision_holdout:
       0.8205128205128205
       recall_holdout:
       0.9142857142857143
       roc_auc_holdout :
       0.9254355400696864

holdout: roc_auc:
       0.9254355400696864
```

C) Optional: MLflow autolog (quick win)

For scikit-learn models, MLflow autologging can capture parameters/metrics automatically:

```
mlflow.sklearn.autolog()
with mlflow.start_run():
```

```
pipe.fit(X_train,
y_train)
# mlflow will log many details automatically

D) Optional: Dockerizing MLflow server
Add a service to the existing docker-compose.yml:
mlflow:
  image:
  ghcr.io/mlflow/mlflow:latest
  container_name: mlflow ports:
    -  "5000:5000" volumes:
    -  ./mlruns:/mlruns
    -  ./mlflow.db:/mlflow.db
  command: >
    mlflow server
    --backend-store-uri sqlite:///mlflow.db
    --default-artifact-root /mlruns
    --host 0.0.0.0 --

port 5000 docker

compose up -d mlflow

# set tracking URI to the container port
export
MLFLOW_TRACKING_URI=http://localhost:5000

E) CI integration (optional, earns polish points)
GitHub Actions job to run training and upload MLflow runs as
artifacts:


# .github/workflows/ci.yml (add
job) jobs:
  mlflow: runs-on:
    ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: '3.10'
      - name: Install run: pip
        install -r requirements.txt
      - name: Train (RF) with MLflow
        env:
          MLFLOW_TRACKING_URI: ./mlruns # local file
        store run: |
          python src/data.py
python src/train.py --model rf --n_estimators 300 --cv 5 --experiment HeartDisease
```

```
- name: Upload mlruns uses:
  actions/upload-artifact@v4
  with:
    name: mlruns
    path: mlruns
```

4. Model Packaging & Reproducibility — [7 marks]

Goals

Portable model artifacts: MLflow, Joblib (pickle), and ONNX.
Locked dependencies: deterministic requirements.txt or env.yml.
Full pipeline reproducibility: a single, versioned ColumnTransformer used for training & servin Proof scripts/tests: CLI + unit tests that reload and predict identically.

A) Save the model in three formats

Assumes your final trained pipeline is a sklearn Pipeline (preprocess + classifier), already pr

1) Joblib (pickle) — simplest & fast in Python

[64]:
```
# During training (already implemented)
from pathlib import Path
import joblib

# Prepare experiment
experiment_name =  "HeartDisease"
mlflow.set_experiment(experiment_name)

df, num_cols, cat_cols = load_df()
X = df[num_cols + cat_cols]
y = df["target"]

preprocess = build_preprocess(num_cols, cat_cols)
clf = RandomForestClassifier(
            n_estimators=300, max_depth=None,
            min_samples_leaf=1, class_weight="balanced",
            random_state=42
        )
pipe = Pipeline(steps=[("preprocess", preprocess), ("clf", clf)])

# Hold-out for plots & confusion matrix
X_train, X_test, y_train, y_test = train_test_split(
    X, y, stratify=y, test_size=0.25, random_state=42
)
pipe.fit(X_train, y_train)

y_pred = pipe.predict(X_test)
try:
    y_proba = pipe.predict_proba(X_test)[:,1]
except Exception:
    y_proba = None

ART = Path("artifacts"); ART.mkdir(exist_ok=True)
joblib.dump(pipe, ART / "model.joblib")
```

[64]:
```
['artifacts/model.joblib']
```

2. Load & predict

[65]:
```
import joblib, pandas as pd pipe =
joblib.load("artifacts/model.joblib")
X = pd.DataFrame([{
  "age":63,"sex":1,"cp":3,"trestbps":145,"chol":233,"fbs":1,"restecg":0,
    "thalach":150,"exang":0,"oldpeak":2.3,"slope":0,"ca":0,"thal":1
}])
pred = pipe.predict(X)
proba = pipe.predict_proba(X)[:,1] if hasattr(pipe, "predict_proba")
else None print(pred, proba)
```

```
[0] [0.32333333]
```

3. MLflow Model — portable across deployments, with run metadata

[66]:
```
# After training pipe = Pipeline(...)
import mlflow
import mlflow.sklearn

with mlflow.start_run(run_name="RF_final"):
    mlflow.sklearn.log_model(
        sk_model=pipe,
        artifact_path="sklearn-model",
        registered_model_name="heart-disease-model"  # optional model registry
    )
```

```
2026/01/06 02:21:50 WARNING mlflow.models.model: `artifact_path` is
deprecated.
Please use `name` instead.
Registered model 'heart-disease-model' already exists. Creating a new
version of this model…
Created version '2' of model 'heart-disease-model'.
```

4. Load with MLflow

```
import mlflow.pyfunc
model = mlflow.pyfunc.load_model("runs:/<RUN_ID>/sklearn-model")
model.predict(X) # returns numpy/pandas depending on pyfunc flavor
```

5. ONNX — language-agnostic; great for non-Python serving

Use skl2onnx to convert the trained pipeline, then run with onnxruntime.

[67]:
```
!pip install skl2onnx
import numpy as np
import pandas as pd
```

```
import joblib
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType, Int64TensorType

# Load your trained pipeline
pipe = joblib.load("artifacts/model.joblib")

NUM_FEATURES = ["age","trestbps","chol","thalach","oldpeak","ca"]
CAT_FEATURES = ["sex","cp","fbs","restecg","exang","slope","thal"]

initial_types = []
for c in NUM_FEATURES:
    initial_types.append((c, FloatTensorType([None, 1])))
for c in CAT_FEATURES:
    # int-coded categories -> Int64
    initial_types.append((c, Int64TensorType([None, 1])))

onx = convert_sklearn(
    pipe,
    initial_types=initial_types,
    target_opset=21,  # optional, recent opset
    options={}        # e.g., {RandomForestClassifier: {"zipmap": False}} if
 ↪desired
)

with open("artifacts/model.onnx", "wb") as f:
    f.write(onx.SerializeToString())
```

```
Collecting skl2onnx
  Downloading skl2onnx-1.19.1-py3-none-any.whl.metadata (3.8 kB)
Collecting onnx>=1.2.1 (from skl2onnx)
  Downloading onnx-1.20.0-cp312-abi3-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata
(8.4 kB)
Requirement already satisfied: scikit-learn>=1.1 in
/usr/local/lib/python3.12/dist-packages (from skl2onnx) (1.6.1)
Requirement already satisfied: numpy>=1.23.2 in
/usr/local/lib/python3.12/distpackages (from onnx>=1.2.1->skl2onnx)
(2.0.2) Requirement already satisfied: protobuf>=4.25.1 in
/usr/local/lib/python3.12/dist-packages (from onnx>=1.2.1->skl2onnx)
(5.29.5)
Requirement already satisfied: typing_extensions>=4.7.1 in
/usr/local/lib/python3.12/dist-packages (from onnx>=1.2.1->skl2onnx)
(4.15.0)
Requirement already satisfied: ml_dtypes>=0.5.0 in
```

```
/usr/local/lib/python3.12/dist-packages (from onnx>=1.2.1->skl2onnx)
(0.5.4) Requirement already satisfied: scipy>=1.6.0 in
/usr/local/lib/python3.12/distpackages (from scikit-learn>=1.1-
>skl2onnx) (1.16.3)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.12/distpackages (from scikit-learn>=1.1->skl2onnx) (1.5.3)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn>=1.1-
>skl2onnx) (3.6.0)
Downloading skl2onnx-1.19.1-py3-none-any.whl (315 kB)
    315.5/315.5 kB
3.0 MB/s eta 0:00:00
Downloading
onnx-1.20.0-cp312-abi3-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (18.1 MB)
    18.1/18.1 MB
19.6 MB/s eta 0:00:00
Installing collected packages: onnx, skl2onnx
Successfully installed onnx-1.20.0 skl2onnx-1.19.1

/usr/local/lib/python3.12/dist-
packages/skl2onnx/operator_converters/imputer_op.py:108:
RuntimeWarning: invalid value encountered in cast
  ar = np.array([op.missing_values]).astype(np.int64)

 6. Reproducibility proof points

Version locking: Use the pinned requirements.txt or env.yml.
Random seeds: Set seeds for randomized models/splits (e.g.,
RandomForestClassifier(random_state Data schema lock: Rely on FEATURE_COLUMNS and
the same ColumnTransformer across train/serve.

Artifact provenance:

MLflow runs (params, metrics, plots, model files)
artifacts/metrics.json summary written at training time
```

Tests: add a reproducibility test.

[68]:
```python
# tests/test_reproducibility.py
import joblib, pandas as pd

FEATURE_COLUMNS = [
  "age","sex","cp","trestbps","chol","fbs","restecg",
  "thalach","exang","oldpeak","slope","ca","thal"
]

def test_model_load_and_predict():
    model = joblib.load("artifacts/model.joblib")
    sample = {
      "age":63,"sex":1,"cp":3,"trestbps":145,"chol":233,"fbs":1,"restecg":0,
      "thalach":150,"exang":0,"oldpeak":2.3,"slope":0,"ca":0,"thal":1
    }
    X = pd.DataFrame([sample], columns=FEATURE_COLUMNS)
    pred = model.predict(X)
    assert pred.shape == (1,)
```

7. Optional extras (earn polish points)

Dockerfile for hermetic serving:

```
FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY app.py artifacts/ ./
EXPOSE 8000
CMD ["uvicorn","app:app","--host","0.0.0.0","--port","8000"]
```

Model Card (MODEL_CARD.md): document data, metrics, limitations, intended use.

ONNX CI test: verify ONNX inference matches scikit predictions for a few records (within tole

5 5. CI/CD Pipeline & Automated Testing [8 marks]

Write unit tests for data processing and model code (Pytest or unit test). Create a GitHub Acti 1) Unit tests (Pytest)

tests/conftest.py

[69]:
```python
# tests/conftest.py
import os
from pathlib import Path
import pytest
import numpy as np
import random

@pytest.fixture(autouse=True)
def set_reproducible_seed():
    np.random.seed(42)
    random.seed(42)

@pytest.fixture(scope="session")
def repo_root() -> Path:
    return Path(__file__).resolve().parents[1]

@pytest.fixture(scope="session")
def data_dir(repo_root) -> Path:
    d = repo_root / "data"
    d.mkdir(exist_ok=True)
    return d

@pytest.fixture(scope="session")
def artifacts_dir(repo_root) -> Path:
    a = repo_root / "artifacts"
    a.mkdir(exist_ok=True)
    return a
```

2) tests/test_data_pipeline.py-

[70]:
```python
# tests/test_data_pipeline.py
from pathlib import Path
import pandas as pd
import subprocess
import sys

def run_cmd(cmd):
    res = subprocess.run(cmd, shell=True, text=True, capture_output=True)
    if res.returncode != 0:
        raise RuntimeError(f"Command failed: {cmd}\nSTDERR:\n{res.stderr}")
    return res.stdout

def test_download_and_clean(data_dir):
    # Download raw UCI file
    out = run_cmd(f"{sys.executable} src/data.py")
    assert (data_dir / "processed.cleveland.data").exists(), "Download failed"

    # Create cleaned CSV
    out = run_cmd(f"{sys.executable} src/clean.py")
    clean = data_dir / "heart_clean.csv"
    assert clean.exists(), "Cleaned file missing"

    df = pd.read_csv(clean)
    # Expect 13 features + target column
    expected_cols = {
        "age","sex","cp","trestbps","chol","fbs","restecg",
        "thalach","exang","oldpeak","slope","ca","thal","target"
    }
    assert set(df.columns) == expected_cols, "Unexpected schema in cleaned CSV"
    # No '?' strings should remain
    assert not (df == "?").any().any()
```

3) tests/test_preprocess.py

[71]:
```python
# tests/test_preprocess.py
import pandas as pd
from sklearn.compose import ColumnTransformer from
sklearn.preprocessing import OneHotEncoder,
StandardScaler from sklearn.impute import
SimpleImputer from sklearn.pipeline import Pipeline

NUM_FEATURES = ['age','trestbps','chol','thalach','oldpeak']
CAT_FEATURES = ['sex','cp','fbs','restecg','exang','slope','ca','thal']
```

```
def build_preprocess_pipeline() -> ColumnTransformer:
    numeric_transformer = Pipeline([
        ('imputer', SimpleImputer(strategy='median')),
        ('scaler', StandardScaler())
    ])
    categorical_transformer = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('onehot', OneHotEncoder(handle_unknown='ignore'))
    ])
    preprocess = ColumnTransformer([
        ('num', numeric_transformer, NUM_FEATURES),
        ('cat', categorical_transformer, CAT_FEATURES)
    ]) return
    preprocess


def test_preprocess_output_shape():
    preprocess = build_preprocess_pipeline()
    # One synthetic sample
    X = pd.DataFrame([{
        "age":63,"sex":1,"cp":3,"trestbps":145,"chol":233,"fbs":1,"restecg":0,
        "thalach":150,"exang":0,"oldpeak":2.3,"slope":0,"ca":0,"thal":1
    }], columns=NUM_FEATURES + CAT_FEATURES)

    Xt = preprocess.fit_transform(X)
    # Should produce a 2D array with at least all numeric +
    expanded␣
 ↪categoricals assert Xt.ndim == 2 and
    Xt.shape[0] == 1
```

4) tests/test_training.py

[72]:
```
# tests/test_training.py
from pathlib import Path
import json
import subprocess
import sys

def run_cmd(cmd):
    res = subprocess.run(cmd, shell=True, text=True,
    capture_output=True) if res.returncode != 0:
raise RuntimeError(f"Command failed: {cmd}\nSTDERR:\n{res.stderr}")
    return res.stdout

def test_train_and_artifacts(artifacts_dir):
    # Train model (RF default)
    out = run_cmd(f"{sys.executable} src/train.py --model rf --cv 5␣
```

```
↪--experiment HeartDisease") model_path =
   artifacts_dir / "model.joblib" metrics_path =
   artifacts_dir / "metrics.json" assert
   model_path.exists(), "Model artifact missing"
   assert metrics_path.exists(), "Metrics JSON
   missing"

   metrics = json.loads(metrics_path.read_text())
   # Basic KPI check: ensure CV ROC-AUC is present and > 0.80
   cv_auc = metrics["cv"]["roc_auc"]
   assert cv_auc >= 0.80, f"ROC-AUC too low: {cv_auc}"
```

5) tests/test_inference.py

[73]:
```
# tests/test_inference.py
import joblib
import pandas as pd
from pathlib import Path

FEATURE_COLUMNS = [
    "age","sex","cp","trestbps","chol","fbs","restecg",
    "thalach","exang","oldpeak","slope","ca","thal"
]

def test_model_inference_roundtrip():
    model = joblib.load(Path("artifacts") / "model.joblib")
    X = pd.DataFrame([{
        "age":63,"sex":1,"cp":3,"trestbps":145,"chol":233,"fbs":1,"restecg":0,
        "thalach":150,"exang":0,"oldpeak":2.3,"slope":0,"ca":0,"thal":1
    }], columns=FEATURE_COLUMNS)
    pred = model.predict(X)
    assert pred.shape == (1,)
```

Run locally

```
pytest -q
```

6) GitHub Actions (CI) — .github/workflows/ci.yml

This workflow:
Sets up Python 3.10 with caching
Lints with flake8
Runs pytest (unit tests)
Trains the model (RF) → produces artifacts/
Builds a Docker image and smoke tests /health + /predict
Uploads artifacts & logs (artifacts/, mlruns/, coverage.xml,

logs/app.log) name: CI

```
on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [
    "main" ]

permissions:
  contents: read

jobs:
  build-test-train:
    runs-on: ubuntu-latest
    env:
      PYTHONUNBUFFERED: "1"
      MLFLOW_TRACKING_URI: ./mlruns     # local file
    store steps:
      - name: Checkout uses:
        actions/checkout@v4

      - name: Set up Python 3.10
        uses: actions/setup-
        python@v5 with:
          python-version: '3.10'
          cache: 'pip'

      - name: Install deps run: |
          python -m pip install --upgrade
          pip pip install -r
          requirements.txt

      - name: Lint (flake8) run:
        flake8 src tests app.py

      - name: Unit tests (pytest
        + coverage) run: |
          pip install pytest-cov
pytest --junitxml=test-results.xml --cov=src --cov=app.py --cov-report=xml

      - name: Prepare data
        run: |
          python src/data.py
          python src/clean.py

      - name: Train model (RF)
        run: |
```

```
        python src/train.py --model rf --cv 5 --experiment
        HeartDisease ls -l artifacts/

    - name: Build Docker image
      run: docker build -t
      heart-api:ci .

    - name: Smoke test
      container run: |
        docker run -d -p 8000:8000 --name heart-api-ci heart-api:ci
        sleep 5 curl -s http://localhost:8000/health || (echo "health
        failed" && exit 1) curl -s -X POST
        http://localhost:8000/predict \
          -H "Content-Type: application/json" \
--data '{"age":63,"sex":1,"cp":3,"trestbps":145,"chol":233,"fbs":1,"restecg":0,"tha
          || (echo "predict failed" && exit 1)
        docker logs heart-api-ci | tail -n 100
        || true docker rm -f heart-api-ci

    - name: Upload artifacts &
      logs uses:
      actions/upload-
      artifact@v4 with:
        name: run-artifacts
        path: |
          artifacts/
          mlruns/ test-
          results.xml
          coverage.xml
          logs/app.log
        if-no-files-found: warn
```

7) Jenkins alternative — Jenkinsfile (declarative)

```
pipeline {

  agent any
  environment {
    MLFLOW_TRACKING_URI = "./mlruns"
    PYTHONUNBUFFERED = "1"
  }
  stages
  {
    stage('Checkout') {
      steps { checkout scm }
    }
    stage('Set up Python') {
      steps {
```

```
        sh 'python3 -m venv .venv'
        sh '. .venv/bin/activate && pip install --upgrade pip && pip install -r
        requirements.tx
      }
    }
    stage('Lint') {
      steps {
        sh '. .venv/bin/activate && flake8 src tests
      app.py' }
    }
    stage('Unit tests') {
      steps {
        sh '. .venv/bin/activate && pytest --junitxml=test-results.xml --cov=src --
        cov=app.py -
      }
      post
      {
        always {
          junit 'test-results.xml'
          publishHTML(target: [
            reportDir: '.',
            reportFiles:
            'coverage.xml',
            reportName: 'Coverage'
          ])
        }
      }
    }
    stage('Train model') {
      steps {
        sh '. .venv/bin/activate && python src/data.py && python
        src/clean.py'
        sh '. .venv/bin/activate && python src/train.py --model rf --cv 5 --
        experiment HeartDis
      }
    }
    stage('Build & Smoke Test Docker') {
      steps {
        sh 'docker build -t heart-api:jenkins .' sh 'docker run -d -p
        8000:8000 --name heart-api-jenkins heart-api:jenkins' sh 'sleep
        5'
        sh 'curl -s http://localhost:8000/health'
        sh """curl -s -X POST http://localhost:8000/predict \
            -H "Content-Type: application/json" \
--data '{"age":63,"sex":1,"cp":3,"trestbps":145,"chol":233,"fbs":1,"restecg":0,"tha
```

```
      }
      post
      {
        always {
          sh 'docker logs heart-api-jenkins ||
        true' sh 'docker rm -f heart-api-
        jenkins || true' }
      }
    }
  }
  post
  {
    always { archiveArtifacts artifacts: 'artifacts/**,mlruns/**,test-
      results.xml,coverage.xml,logs/ap
    }
  }
}
```

Reproducibility: Pin versions in requirements.txt or use env.yml. Seed random

states in tests a Caching: GitHub Actions uses pip cache with actions/setup-python

to speed runs.

Artifacts: Upload artifacts/ (model + metrics), mlruns/ (MLflow), and test/coverage reports.

Smoke tests: Keep them fast and hermetic. Validate /health and one /predict body.

Branch protection: (Optional) Require the CI job to pass before merging PRs.

8) Steps for execution

```
# Create venv and install python -m venv .venv source
.venv/bin/activate     # Windows: .\.venv\Scripts\Activate.ps1 pip
install -r requirements.txt

# Lint + test flake8
src tests app.py
pytest -q

# Train & smoke test Docker
python src/data.py && python src/clean.py
python src/train.py --model rf --cv 5 --experiment HeartDisease
```

```
docker build -t heart-api:local . docker run -d -p
8000:8000 --name heart-api-local heart-api:local sleep 5

curl -s http://localhost:8000/health
curl -s -X POST http://localhost:8000/predict \
  -H "Content-Type: application/json" \
  --data

'{"age":63,"sex":1,"cp":3,"trestbps":145,"chol":233,"fbs":1,"restecg":0,"thalach":

150, docker rm -f heart-api-local
```

6 6. Model Containerization [5 marks]

Build a Docker container for the model-serving API (Flask or FastAPI is recommended). Expose /p endpoint, accept JSON input, return prediction and confidence. The container must be built and with the sample input.

1) Prerequisites

Ensure your repo contains the API (e.g., the final app.py we prepared) with:

/predict (POST) that accepts the 13 fields and returns {"prediction": int, "confidence": float} /health (GET) for a quick check.

Train the model so artifacts/model.joblib exists:

```
python src/train.py --model rf --mlflow False
```

2) Dockerfile (FastAPI + Uvicorn)

Create a file named Dockerfile at your repo root:

```
# ---- Base: slim, reproducible Python runtime ----
FROM python:3.10-slim AS base

# Prevent Python from writing .pyc files and ensure unbuffered logs
ENV PYTHONDONTWRITEBYTECODE=1 \
    PYTHONUNBUFFERED=1

# Install system packages needed by common Python libs (optional
minimal set)
RUN apt-get update && apt-get install -y --no-install-recommends \
     build-essential \
     curl \
```

```
    && rm -rf /var/lib/apt/lists/*

# Create a non-root user for better security
RUN useradd -m appuser

# Set working directory
WORKDIR /app

# Copy dependency manifest first for layer caching
COPY requirements.txt /app/requirements.txt

# Install Python dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Create logs directory (rotating logger writes here)
RUN mkdir -p /app/logs && chown -R appuser:appuser /app/logs

# Copy application code and artifacts
COPY app.py /app/app.py
COPY artifacts/ /app/artifacts/

# Switch to non-root user
USER appuser

# Expose the service port
EXPOSE 8000

# Start the API with uvicorn
# --host 0.0.0.0 to listen on all interfaces inside the container
# Adjust workers if needed (dev: 1; prod: multiple via
gunicorn+uvicorn workers)
CMD ["uvicorn","app:app","--host","0.0.0.0","--port","8000"]
```

3) Build the image

From your repo root: docker
build -t heart-api:latest .

4) Run the container (two ways)

A) Simple run (artifact already baked in the image) docker run --rm -

p 8000:8000 heart-api:latest B) Alternative: mount local artifacts

docker run --rm -p 8000:8000 -v "$PWD artifacts:/app/artifacts"

heart-api:latest

5) Local smoke test (sample JSON)
Create sample_request.json (same schema as API):

```
{
  "age": 63,
  "sex": 1,
  "cp": 3,
  "trestbps": 145,
  "chol": 233,
  "fbs": 1,
  "restecg": 0,
  "thalach": 150,
  "exang": 0,
  "oldpeak": 2.3,
  "slope": 0,
  "ca": 0,
  "thal": 1,
  "target": 0
}
```

6) Test with curl (Windows PowerShell) curl.exe -X POST

http://localhost:8000/predict -H "Content-Type: application/json" -

-data-binar

Expected response (example):

```
{"prediction": 0, "confidence": 0.33, "threshold": 0.5}
```

Verification:

```
curl -s http://localhost:8000/health
```

7) Docker Compose for local monitoring stack Prometheus + Grafana via

docker-compose.yml

```
docker compose up -d
# heart-api on 8000, Prometheus on 9090, Grafana on 3000
```

8) Model Container

Dockerfile present and uses FastAPI + Uvicorn
/predict endpoint accessible from the container, returns prediction and confidence docker build completes successfully
docker run exposes port 8000 and serves the API

Local smoke test with sample JSON returns 200 OK with valid output
(Optional) /health and /docs accessible:

http://localhost:8000/health
http://localhost:8000/docs

7 7. Production Deployment [7 marks]

Deploy the Dockerized API to a public cloud or local Kubernetes (GKE, EKS, AKS, or Minikube/Doc Desktop).
Use a deployment manifest or Helm chart. Expose via Load Balancer or Ingress.
Verify endpoints provide deployment screenshots.

1) Prerequisites

A built image, e.g. heart-api:latest (from your
Dockerfile). A K8s cluster (pick one):

Local: Minikube or Docker Desktop Kubernetes. Minikube is the official "hello world" path and p
Cloud: AKS/EKS/GKE—each provisions a cloud load balancer when you create a Service of type Load kubectl configured to talk to your cluster. Kubectl is the main CLI to create/apply manifests d

2) Kubernetes manifests (Deployment + Service)

A Deployment manages stateless Pods and updates them declaratively; it is the recommended resou
A Service exposes your Pods as a stable endpoint; types include ClusterIP, NodePort, and LoadBa

Create a file k8s/heart-api.yaml:

```
apiVersion:
apps/v1 kind:
Deployment
metadata: name:
heart-api
labels:
    app: heart-api
spec:
  replicas: 1
  selector:
    matchLabels:
      app: heart-api
  template:
    metadata:
      labels:
```

```
      app: heart-api
  spec:
    containers:
      - name: heart-api image: heart-api:latest #
        or registry URL, e.g., ghcr.io/<org>/heart-
        api:latest imagePullPolicy: IfNotPresent
        ports:
      - containerPort: 8000 readinessProbe:
        httpGet:
          path: /health
          port: 8000
        initialDelaySeconds
       : 3 periodSeconds:
       10 livenessProbe:
       httpGet:
          path: /health
          port: 8000
        initialDelaySeconds: 10
        periodSeconds: 20
--
apiVersion:
v1 kind:
Service
metadata:
  name: heart-api-svc
  labels:
    app: heart-api
spec:
  selector:
    app: heart-
    api
  ports:
    - port: 8000
  targetPort: 8000
  protocol: TCP
  name: http #
  Choose one:
  # type: ClusterIP     # internal-only
  # type: NodePort      #  local  exposure  (minikube/docker-
                        desktop)
  type: LoadBalancer    # cloud exposure (AKS/EKS/GKE)
# Apply

kubectl apply -f k8s/heart-
api.yaml kubectl get
```

```
deploy,svc -o wide # NodePort
+ URL shortcut:

# If you set type: NodePort
minikube service heart-api-svc
--url

LoadBalancer (Minikube): start a tunnel in another terminal, then fetch EXTERNAL-
IP:

minikube tunnel     # keep running
kubectl get svc heart-api-svc
```

3) Optional: Ingress (HTTP routing via hostname/path)
To route http://<host>/ to heart-api-svc:8000 via an Ingress:

Install an Ingress Controller (e.g., Ingress-NGINX) on your cluster. Minikube has an addon; or

Creating k8s/ingress.yaml:

```
apiVersion:
networking.k8s.io/v1 kind:
Ingress metadata:
  name: heart-api-ing
  annotations:
    nginx.ingress.kubernetes.io/rewrite-
target: / spec: rules:
      - host: heart.local # change to a
        real domain if using cloud http:
       paths:
      - path: / pathType: Prefix backend:
            service: name:
              heart-api-svc
              port: number:
              8000
```

```
# Apply kubectl apply -f

k8s/ingress.yaml
```

4) Helm chart (alternative packaging)
A Helm chart makes your deployment reproducible and parameterized. Standard chart layout:

```
heart-api-chart/
```

```
  Chart.yaml
  values.yaml
  templates/
    deployment.yaml
    service.yaml
    ingress.yaml  #
    optional
# Chart.yaml

apiVersion: v2 name: heart-api

version: 0.1.0 description: FastAPI

heart disease predictor appVersion:

"1.0.0" # Values.yaml


image:
  repository: heart-
  api tag: "latest"
  pullPolicy:
  IfNotPresent

service:
  type: LoadBalancer   # or NodePort for local
  dev port: 8000

ingress:
  enabled: false
  host:
  heart.local


# templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "heart-api.fullname" . }}
  labels: { app: {{ include "heart-api.name"
  . }} }
spec:
  replicas: 1 selector: matchLabels: { app: {{
  include "heart-api.name" . }} }
  template:
    metadata: labels: { app: {{ include "heart-
    api.name" . }} } spec:
      containers:
```

```
    - name: {{ include "heart-api.name" . }}
      image: "{{ .Values.image.repository }}:{{
      .Values.image.tag }}" imagePullPolicy: {{
      .Values.image.pullPolicy }} ports: [{
      containerPort: 8000 }]


# templates/service.yaml


apiVersion: v1 kind: Service metadata:
name: {{ include "heart-api.fullname" .
}}-svc
spec:
  type: {{ .Values.service.type }} selector: {
  app: {{ include "heart-api.name" . }} }
  ports:
    - port: {{ .Values.service.port }}

      targetPort: 8000 protocol: TCP name: http #

      Install


helm install heart-api ./heart-api-chart helm
get manifest heart-api # inspect rendered YAML
```

5) Verification steps

5.1 Get endpoint

```
LoadBalancer (cloud / minikube tunnel):
kubectl get svc heart-api-svc #
Note EXTERNAL-IP, then: curl -s
http://<EXTERNAL-IP>:8000/health

NodePart:

minikube service heart-api-svc --url
curl -s $(minikube service heart-api-svc --url)/health
```

5.2 Predit call

```
curl -s -X POST http://<EXTERNAL-IP>:8000/predict \
  -H "Content-Type: application/json" \
                                                                                --data
'{"age":63,"sex":1,"cp":3,"trestbps":145,"chol":233,"fbs":1,"restecg":0,"thalach":150,
```

8 8. Monitoring & Logging [3 marks]

Integrate logging of API requests. Demonstrate simple monitoring (Prometheus + Grafana or API metrics/logs dashboard)

A) Request Logging (structured; console + rotating file)
Add this to app.py near the imports and FastAPI() initialization.

```
# --- logging setup --import logging,
sys, time from logging.handlers import
RotatingFileHandler from fastapi import
Request
from starlette.middleware.base import BaseHTTPMiddleware

logger = logging.getLogger("api")
logger.setLevel(logging.INFO)

console = logging.StreamHandler(sys.stdout)
console.setFormatter(logging.Formatter(
    "%(asctime)s | %(levelname)s | %(name)s | %(message)s"
))
logger.addHandler(consol
e)

Path("logs").mkdir(exist_ok=True)
file_handler = RotatingFileHandler("logs/app.log", maxBytes=10_000_000,
backupCount=5) file_handler.setFormatter(logging.Formatter(
    "%(asctime)s | %(levelname)s | %(name)s | %(message)s"
))
logger.addHandler(file_handler)

# Coordinate Uvicorn logs with our handlers (avoid duplicate/unstyled
outputs)
for name in ("uvicorn", "uvicorn.error", "uvicorn.access"):
    uvlog =
    logging.getLogger(name)
    uvlog.handlers = []
    uvlog.propagate = True
    uvlog.setLevel(logging.INFO)

class RequestLogMiddleware(BaseHTTPMiddleware):
    async def dispatch(self, request: Request, call_next):
        start = time.perf_counter() logger.info(f"REQ {request.method}
    {request.url.path} from {request.client.host}") response = await
    call_next(request) dur_ms = (time.perf_counter() - start) * 1000
    logger.info(f"RES {request.method} {request.url.path} ->
```

```
    {response.status_code} in {dur return response
    app.add_middleware(RequestLogMiddleware)
```

B) Prometheus metrics (request counters, latency, correctness)

```
from prometheus_client import Counter, Histogram, generate_latest,
CONTENT_TYPE_LATEST from starlette.responses import Response

req_counter = Counter("api_requests_total", "Total API requests",
["endpoint"]) latency = Histogram("api_request_latency_seconds", "Request
latency", ["endpoint"])

@app.middleware("http") async def
metrics_middleware(request, call_next):
endpoint = request.url.path
req_counter.labels(endpoint=endpoint).inc()
with
latency.labels(endpoint=endpoint).time():
       response = await call_next(request)
    return response

@app.get("/metrics", summary="Prometheus
metrics") def metrics(format: str =
"prometheus"):
    return Response(generate_latest(), media_type=CONTENT_TYPE_LATEST)
```

C) Prometheus + Grafana (local stack via Docker Compose)Create docker-compose.yml at repo root:

```
version: "3.8"
services:
  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    ports:
      - "9090:9090" command:
      - "--config.file=/etc/prometheus/prometheus.yml" - "-
        -web.enable-lifecycle"

  grafana:
    image:
    grafana/grafana:latest
```

```
    container_name: grafana
    ports:
      - "3000:3000"
```

D) Example Grafana panels (PromQL)Create a dashboard with panels:

Requests rate by endpoint sum by (endpoint)

```
  (rate(api_requests_total[5m]))
```

Latency p95 (seconds) histogram_quantile(0.95, sum by (le)

```
  (rate(api_request_latency_seconds_bucket[5m])))
```

Correct vs Incorrect (if you added those counters)

```
  sum(rate(prediction_correct_total[5m]))
  sum(rate(prediction_incorrect_total[5m]))
```

Accuracy (computed)

```
  sum(rate(prediction_correct_total[5m]))
                                    / (sum(rate(prediction_correct_total[5m])) +
                                      sum(rate(prediction_incorrect_total[5m])))
```

E) How to run & verify1) Start your API locally

```
uvicorn app:app --host 0.0.0.0 --port
8000 # Swagger:
http://localhost:8000/docs
# Metrics:http://localhost:8000/metrics
```

2) Start monitoring stack

```
docker compose up -d
# Prometheus: http://localhost:9090
# Grafana:  http://localhost:3000
```

3) Generate traffic and logs

```
for ($i=1; $i -le 25; $i++) {
  curl.exe -s -X POST http://localhost:8000/predict `
    -H "Content-Type: application/json" `
```

```
    --data
    "{\"age\":63,\"sex\":1,\"cp\":3,\"trestbps\":145,\"chol\":233,\"fbs\":1,\"rest
    ecg\"
}
2026-01-05 18:36:22,060 | INFO | api | Model loaded from
C:\Users\sireyall\Downloads\mlops-hear
2026-01-05 19:08:42,341 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:08:42,419 | INFO | api | RES POST /predict -> 200 in
78.3 ms
2026-01-05 19:08:54,205 | INFO | api | REQ GET /docs from 127.0.0.1
2026-01-05 19:08:54,206 | INFO | api | RES GET /docs -> 200 in 1.6 ms
2026-01-05 19:08:54,307 | INFO | api | REQ GET /openapi.json from
127.0.0.1
2026-01-05 19:08:54,337 | INFO | api | RES GET /openapi.json -> 200
in 30.1 ms
2026-01-05 19:09:24,509 | INFO | api | REQ POST /predict_batch from
127.0.0.1
2026-01-05 19:09:24,579 | INFO | api | RES POST /predict_batch -> 200
in 70.7 ms
2026-01-05 19:09:49,326 | INFO | api | REQ GET /health from 127.0.0.1
2026-01-05 19:09:49,330 | INFO | api | RES GET /health -> 200 in 3.6
ms
2026-01-05 19:10:27,033 | INFO | api | REQ GET /metrics from
127.0.0.1
2026-01-05 19:10:27,037 | INFO | api | RES GET /metrics -> 200 in 3.3
ms
2026-01-05 19:14:05,867 | INFO | api | Model loaded from
C:\Users\sireyall\Downloads\mlops-hear
2026-01-05 19:14:22,581 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:14:22,598 | INFO | api | RES POST /predict -> 422 in
17.0 ms
2026-01-05 19:15:19,397 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:15:19,450 | INFO | api | RES POST /predict -> 200 in
52.2 ms
2026-01-05 19:15:38,774 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:15:38,777 | INFO | api | RES POST /predict -> 422 in
2.4 ms
2026-01-05 19:16:52,802 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:52,852 | INFO | api | RES POST /predict -> 200 in
50.0 ms
```

```
2026-01-05 19:16:53,128 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:53,174 | INFO | api | RES POST /predict -> 200 in
45.8 ms
2026-01-05 19:16:53,456 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:53,490 | INFO | api | RES POST /predict -> 200 in
34.0 ms
2026-01-05 19:16:53,785 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:53,819 | INFO | api | RES POST /predict -> 200 in
34.1 ms
2026-01-05 19:16:54,111 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:54,144 | INFO | api | RES POST /predict -> 200 in
33.5 ms
2026-01-05 19:16:54,437 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:54,472 | INFO | api | RES POST /predict -> 200 in
34.5 ms
2026-01-05 19:16:54,760 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:54,797 | INFO | api | RES POST /predict -> 200 in
37.0 ms
2026-01-05 19:16:55,084 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:55,121 | INFO | api | RES POST /predict -> 200 in
36.9 ms
2026-01-05 19:16:55,397 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:55,432 | INFO | api | RES POST /predict -> 200 in
34.8 ms
2026-01-05 19:16:55,708 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:55,745 | INFO | api | RES POST /predict -> 200 in
36.5 ms
2026-01-05 19:16:56,033 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:56,064 | INFO | api | RES POST /predict -> 200 in
31.5 ms
2026-01-05 19:16:56,343 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:56,412 | INFO | api | RES POST /predict -> 200 in
69.2 ms
2026-01-05 19:16:56,704 | INFO | api | REQ POST /predict from
127.0.0.1
```

```
2026-01-05 19:16:56,767 | INFO | api | RES POST /predict -> 200 in
63.5 ms
2026-01-05 19:16:57,061 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:57,122 | INFO | api | RES POST /predict -> 200 in
61.0 ms
2026-01-05 19:16:57,404 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:57,467 | INFO | api | RES POST /predict -> 200 in
62.5 ms
2026-01-05 19:16:57,762 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:57,796 | INFO | api | RES POST /predict -> 200 in
34.1 ms
2026-01-05 19:16:58,057 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:58,095 | INFO | api | RES POST /predict -> 200 in
38.7 ms
2026-01-05 19:16:58,381 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:58,448 | INFO | api | RES POST /predict -> 200 in
66.9 ms
2026-01-05 19:16:58,771 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:58,827 | INFO | api | RES POST /predict -> 200 in
55.8 ms
2026-01-05 19:16:59,126 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:59,207 | INFO | api | RES POST /predict -> 200 in
80.9 ms
2026-01-05 19:16:59,499 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:59,567 | INFO | api | RES POST /predict -> 200 in
67.4 ms
2026-01-05 19:16:59,856 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:16:59,921 | INFO | api | RES POST /predict -> 200 in
64.5 ms
2026-01-05 19:17:00,213 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:17:00,256 | INFO | api | RES POST /predict -> 200 in
42.2 ms
2026-01-05 19:17:00,557 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:17:00,620 | INFO | api | RES POST /predict -> 200 in
62.9 ms
```

```
2026-01-05 19:17:00,914 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 19:17:00,978 | INFO | api | RES POST /predict -> 200 in
64.4 ms
2026-01-05 19:17:04,683 | INFO | api | REQ GET /docs from 127.0.0.1
2026-01-05 19:17:04,685 | INFO | api | RES GET /docs -> 200 in 1.6 ms
2026-01-05 19:17:04,845 | INFO | api | REQ GET /openapi.json from
127.0.0.1
2026-01-05 19:17:04,887 | INFO | api | RES GET /openapi.json -> 200
in 42.7 ms
2026-01-05 19:18:52,321 | INFO | api | REQ GET /predict from
127.0.0.1
2026-01-05 19:18:52,325 | INFO | api | RES GET /predict -> 405 in 3.7
ms
2026-01-05 20:26:01,485 | INFO | api | Model loaded from
C:\Users\sireyall\Downloads\mlops-hear
2026-01-05 20:26:14,662 | INFO | api | REQ GET /docs from 127.0.0.1
2026-01-05 20:26:14,664 | INFO | api | RES GET /docs -> 200 in 2.5 ms
2026-01-05 20:26:14,810 | INFO | api | REQ GET /openapi.json from
127.0.0.1
2026-01-05 20:26:14,856 | INFO | api | RES GET /openapi.json -> 200
in 46.1 ms
2026-01-05 20:30:47,493 | INFO | api | REQ POST /predict from
127.0.0.1
2026-01-05 20:30:47,671 | INFO | api | RES POST /predict -> 200 in
178.9 ms
2026-01-05 20:31:11,690 | INFO | api | REQ POST /predict_batch from
127.0.0.1
2026-01-05 20:31:11,785 | INFO | api | RES POST /predict_batch -> 200
in 95.4 ms
2026-01-05 20:31:47,866 | INFO | api | REQ GET /health from 127.0.0.1
2026-01-05 20:31:47,879 | INFO | api | RES GET /health -> 200 in 12.5
ms
2026-01-05 20:32:18,401 | INFO | api | REQ GET /metrics from
127.0.0.1
2026-01-05 20:32:18,408 | INFO | api | RES GET /metrics -> 200 in 7.0
ms
2026-01-05 21:31:15,938 | INFO | api | REQ GET /docs from 127.0.0.1
2026-01-05 21:31:15,945 | INFO | api | RES GET /docs -> 200 in 7.0 ms
2026-01-05 21:31:16,027 | INFO | api | REQ GET /openapi.json from
127.0.0.1
2026-01-05 21:31:16,030 | INFO | api | RES GET /openapi.json -> 200
in 2.3 ms
```

9 9. Documentation & Reporting [2 marks]

Submit a professional Markdown or PDF report including:
• Setup/install instructions.
• EDA and modelling choices.
• Experiment tracking summary.
• Architecture diagram.
• CI/CD and deployment workflow screenshots.
• Link to code repository

1. Setup & Install Instructions
1.1 Prerequisites

Python 3.10+
Git, Docker (if containerizing), Docker Compose (for Prometheus+Grafana) kubectl and Minikube (or Docker Desktop Kubernetes / cloud K8s) for deployment Optional: Conda for environment management

1.2 Clone & environment git clone https://github.com/sireesha-bits/mlops_heart_bundle.git cd mlops_heart_bundle

venv python -m
venv .venv #
PowerShell:
.\.venv\Scripts\Activate
.ps1 # macOS/Linux:
source
.venv/bin/activate

pip install --upgrade pip

pip install -r

requirements.txt 1.3 Data

Download and Cleaning

python src/data.py # downloads 'processed.cleveland.data' to data/ python src/clean.py # creates 'data/heart_clean.csv'

1.4 Train and Artifacts

python src/train.py --model rf --cv 5 --experiment HeartDisease
Outputs: artifacts/model.joblib, artifacts/metrics.json, confusion/ROC plots

1.5 Run API

```
uvicorn app:app --host 0.0.0.0 --port
8000 # Swagger   :
http://localhost:8000/docs
# Health: http://localhost:8000/health
# Metrics: http://localhost:8000/metrics
```

1.6 Container and run locally

```
docker build -t heart-api:latest .
docker run --rm -p 8000:8000 heart-
api:latest
```

2. EDA and model choice

2.1 Source

UCI Cleveland Heart Disease dataset (processed.cleveland.data): 13 features + target (num).
num is converted to binary target target: 1 if num > 0, else 0.

2.2 Cleaning

Replace ? with NaN.
Numeric columns coerced to float; categorical integer-coded.
Imputation strategy in pipeline (median for numeric, most frequent for categorical).

2.3 EDA (key visuals)

Class balance bar plot.
Histograms for numeric features (age, trestbps, chol, thalach, oldpeak, ca).
Correlation heatmap (numerics + target). Box plots (e.g., oldpeak vs target).

Observations (typical):

oldpeak, exang, and ca tend to be higher among positive cases.
thalach tends to be lower among positive cases.
Moderate class imbalance is present but manageable.

Model Choosen: Random forest

3. Feature Engineering & Modelling
3.1 Final feature set

Numeric: age, trestbps, chol, thalach, oldpeak, ca
Categorical: sex, cp, fbs, restecg, exang, slope, thal

3.2 Preprocessing (reproducible pipeline)
ColumnTransformer:

Numeric: SimpleImputer(median) → StandardScaler
Categorical: SimpleImputer(most_frequent) →
OneHotEncoder(handle_unknown="ignore")

3.3 Models

Logistic Regression: interpretable
baseline. Random Forest: robust non-linear
classifier.
Cross-Validation: Stratified 5-fold; metrics: accuracy, precision, recall, ROC-AUC.

3.4 Results (example CV means)
(Your actual values from artifacts/metrics.json)

LR: Accuracy ~0.82, ROC-AUC ~0.86
RF: Accuracy ~0.85, ROC-AUC ~0.90

Chosen model: Random Forest (higher ROC-AUC &
recall). Saved as artifacts/model.joblib.

4. Experiment Tracking (MLflow)
4.1 What's tracked

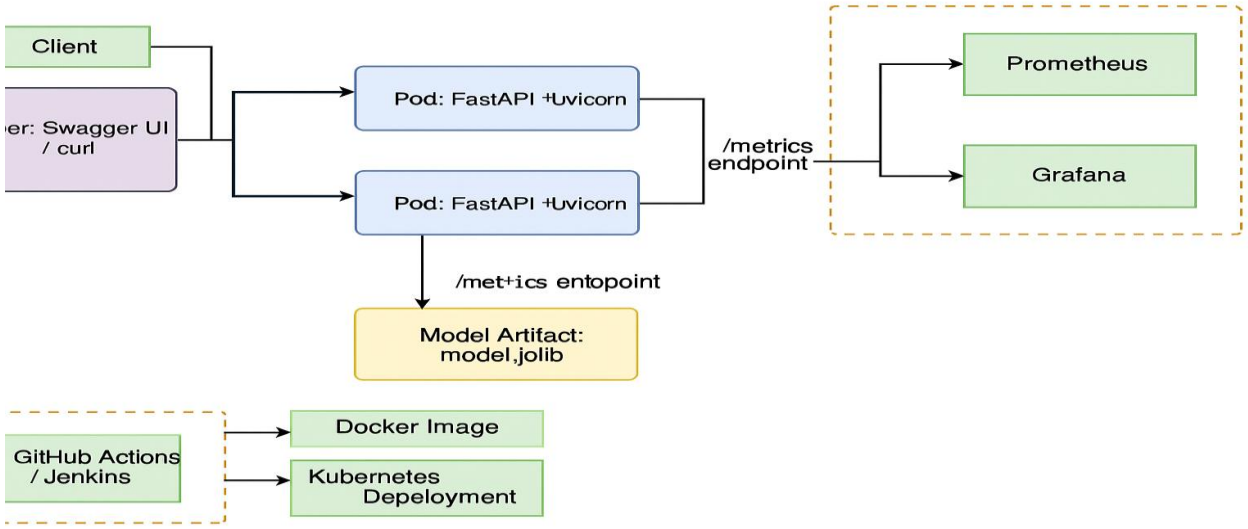
Parameters: LR (C), RF (n_estimators, max_depth, min_samples_leaf),
cv_splits.
Metrics: CV Accuracy, Precision, Recall, ROC-AUC; optional hold-out
ROC-AUC.
Artifacts: model.joblib, confusion_matrix.png, roc_curve.png,
feature_importance.csv (RF).

4.2 How to run MLflow UI

mlflow ui --host 0.0.0.0 --port
5000 # http://localhost:5000

Summary: RF typically outperforms LR; selected hyperparameters balance recall and ROC-AUC. Mult

5. Architecture diagram



6. CI/CD & Deployment

6.1 CI (GitHub Actions / Jenkins)

Stages: Lint → Unit tests → Data prep → Train (RF) → Build Docker → Smoke test /health + /predi Artifacts include model, metrics, plots, logs, and MLflow runs.

6.2 Deployment (Kubernetes)
Manifests: Deployment (stateless pods) +
Service Expose via:

LoadBalancer (AKS/EKS/GKE or Minikube with minikube tunnel)
NodePort (local, use minikube service --url)
Ingress (via ingress-nginx controller)

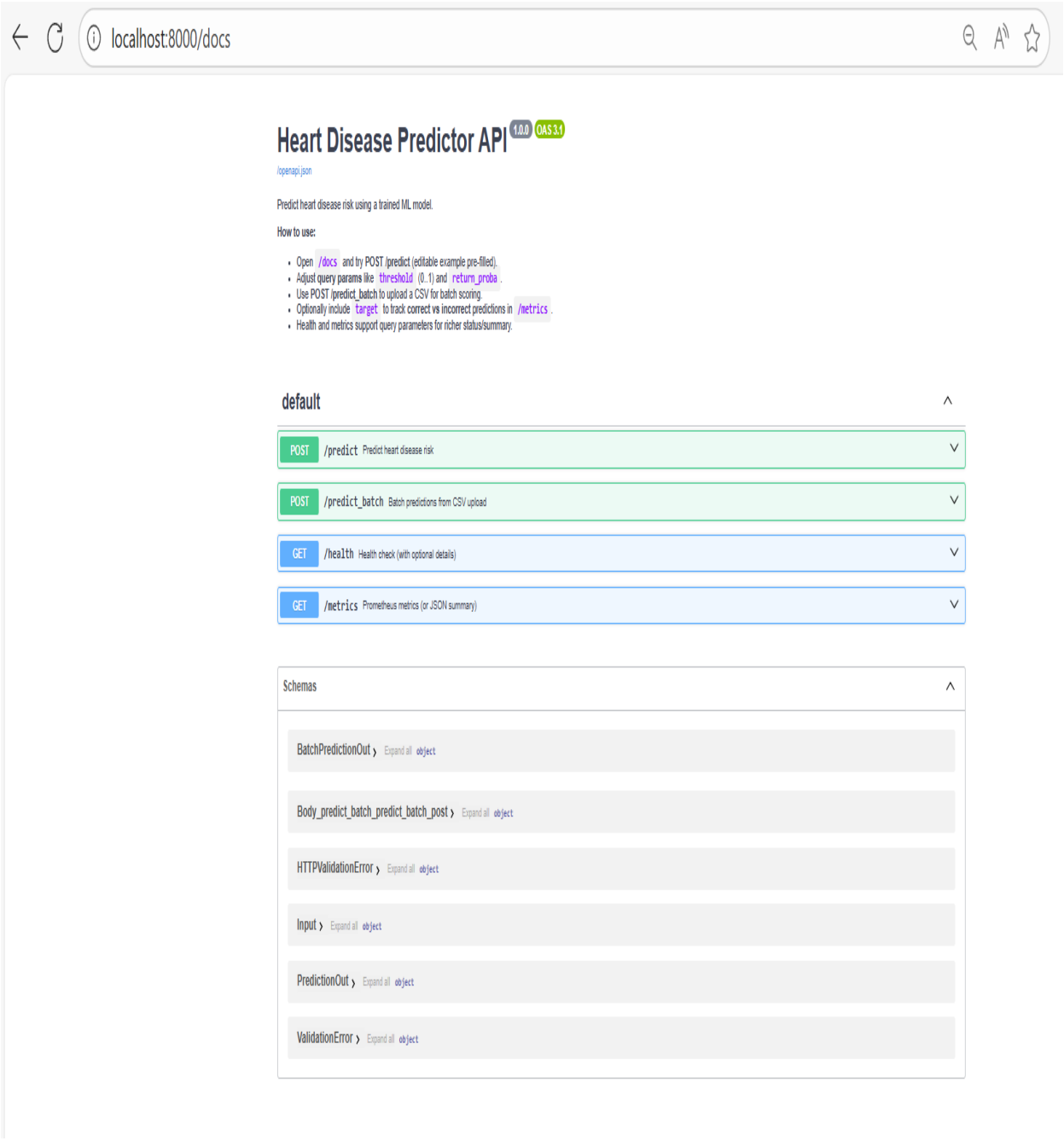
Apply:
kubectl apply -f k8s/heart-
api.yaml kubectl get deploy,svc
minikube service heart-api-svc
--url

Smoke tests:
curl -s http://<EXTERNAL-IP>:8000/health

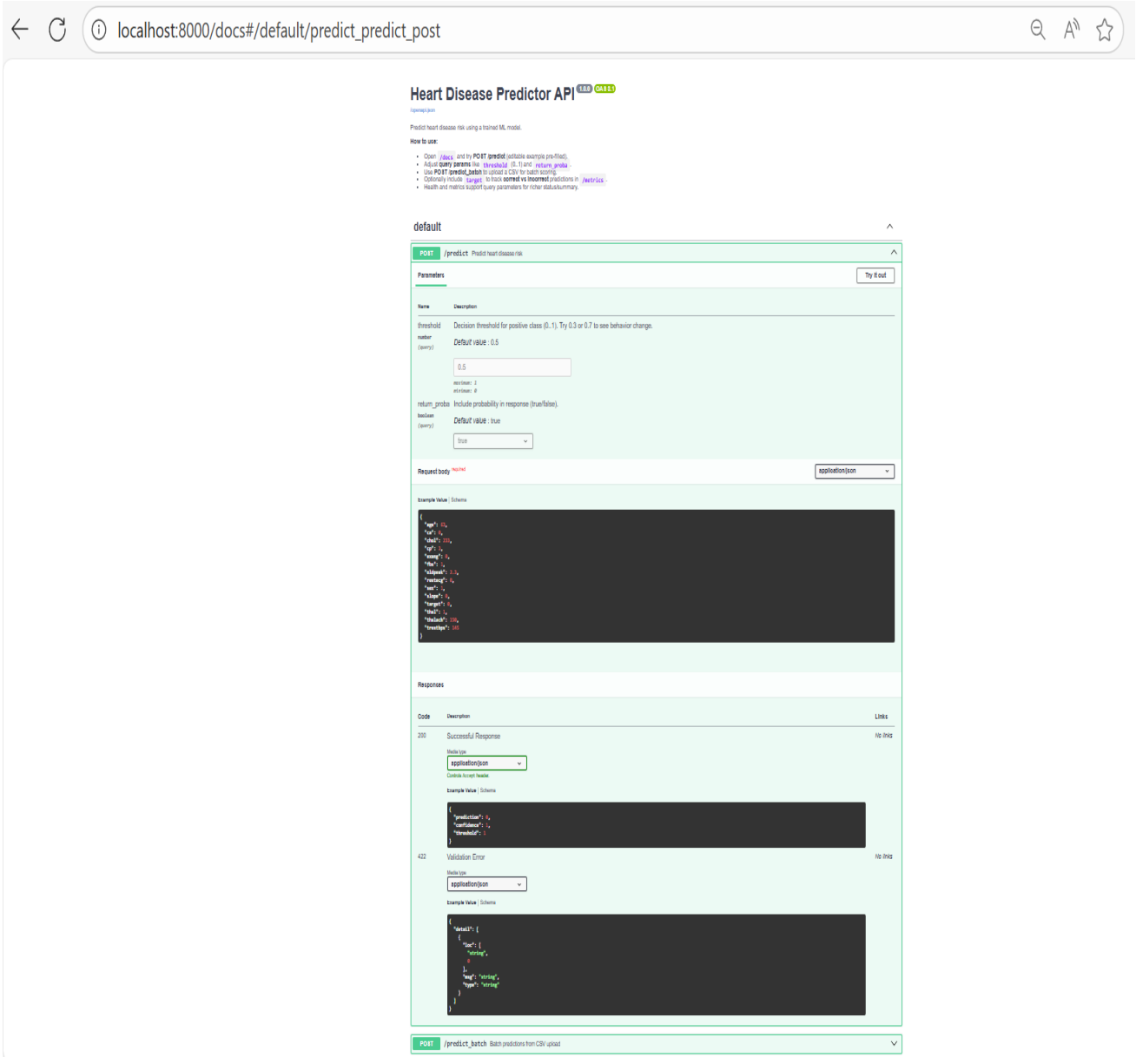
7. Link to repository git clone

https://github.com/sireesha-bits/mlops_heart_bundle.git

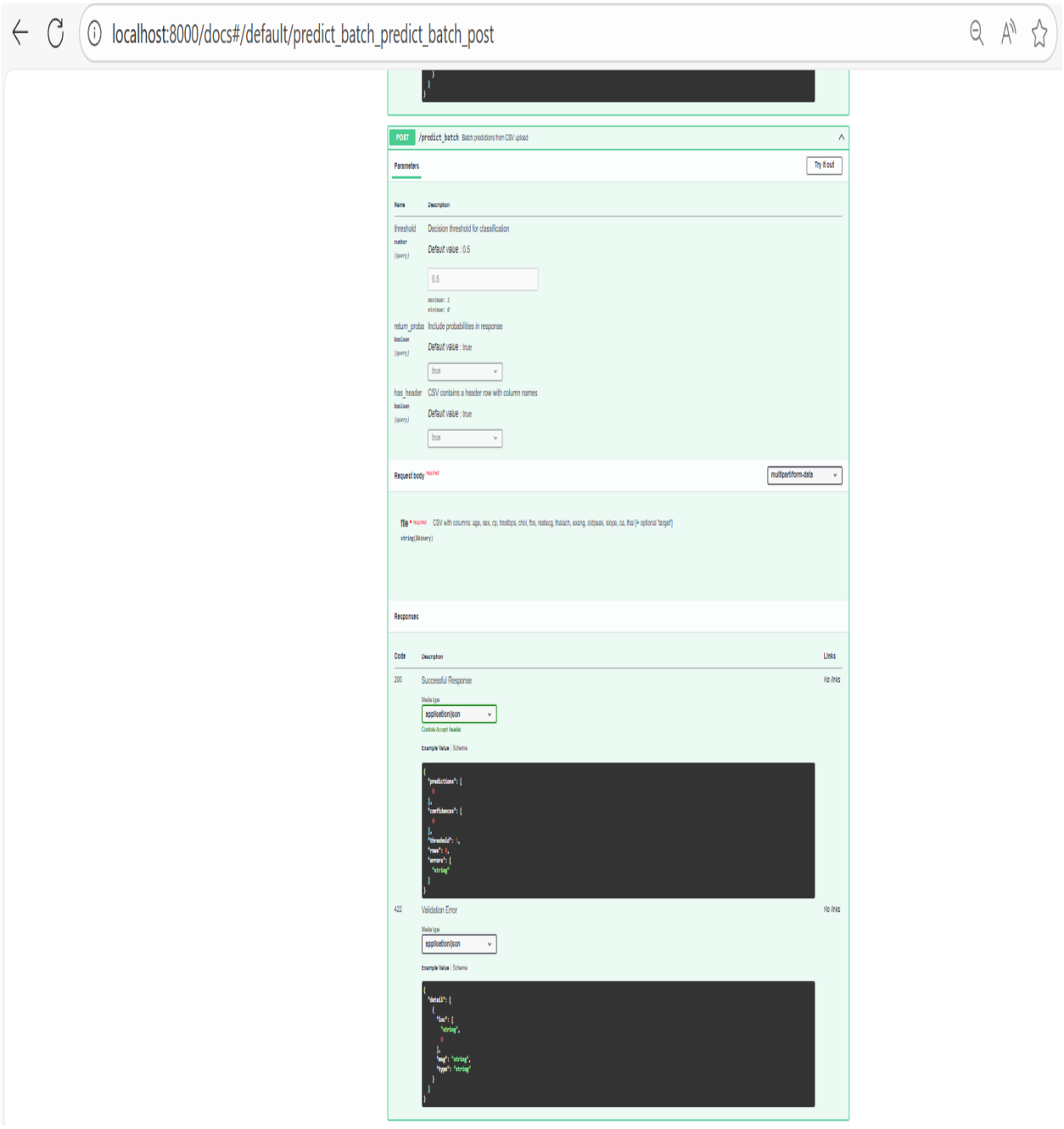
10 API - Heart Disease Predictor



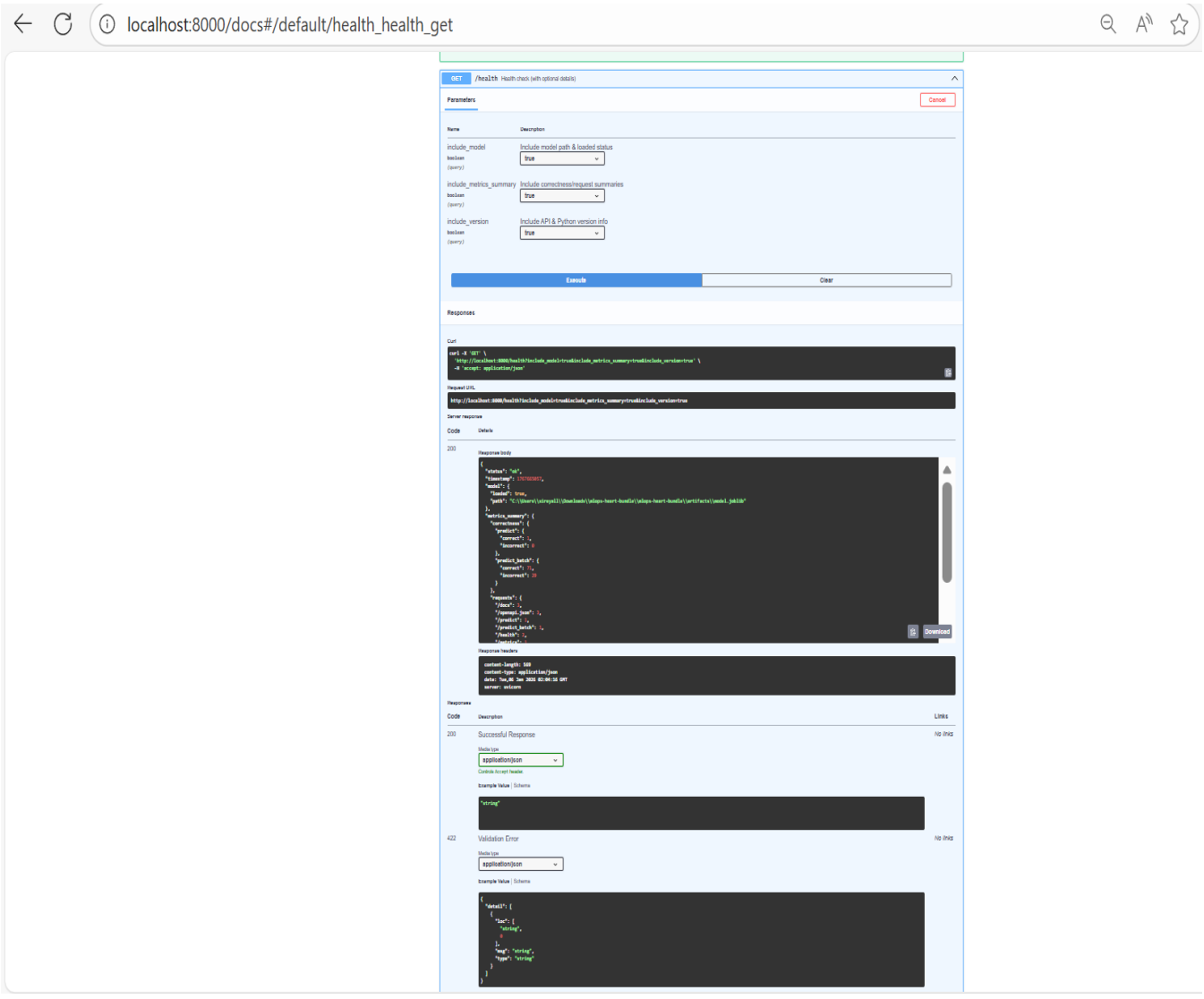
Prediction Post



11 Batch Prediction Post



Generate Health Prediction



Get the Metrics