DEPT: SOFTWARE ENGINEERING

FUNDAMENTAL OF MACHINE LEARNING

TITLE: Heart Disease Prediction

NAME: MERON NISRANE

ID: 1401758

Submission Date:02/05/2017

Submitted to:Mr.Derbew

Heart Disease Prediction Model Documentation

Assignment Title: Personalized Machine Learning Project

Course: Fundamentals of Machine Learning

Objective

This project aims to develop a machine learning model to predict heart disease using a structured dataset. The entire ML lifecycle is covered, from data acquisition and preprocessing to model training, evaluation, and optional deployment.

Introduction

Heart disease is a critical health concern worldwide, accounting for a significant number of deaths annually. Early detection and intervention can drastically reduce the risk of severe complications and mortality. The objective of this project is to develop a **machine learning model** capable of predicting the likelihood of heart disease based on a comprehensive set of patient health parameters.

By leveraging data-driven insights, this model aims to assist healthcare professionals in making informed decisions regarding diagnosis and treatment strategies. The model will analyze various risk factors such as age, blood pressure, cholesterol levels, and other relevant medical indicators to generate accurate predictions. Through automation and data analysis, this approach can enhance diagnostic accuracy, optimize medical resource allocation, and contribute to the early detection of heart disease, ultimately improving patient outcomes and reducing healthcare costs.

Problem Definition & Data Acquisition

Problem Statement

Heart disease is one of the leading causes of mortality worldwide. Early diagnosis is crucial for reducing risk and improving patient outcomes.The goal of this project is to develop a machine learning model that predicts whether a patient has heart disease based on various health-related features. This predictive model can assist healthcare professionals in early diagnosis, risk assessment, and personalized treatment plan.

Using **supervised learning**, we will build a **classification model** that categorizes patients into **two classes**:

- **1 (Presence of heart disease)**
- **0 (Absence of heart disease)**

This model will assist healthcare professionals in **early diagnosis and intervention**, potentially leading to better treatment and prevention strategies.

Heart Disease Prediction Dataset

This dataset contains 1,888 records merged from five publicly available heart disease datasets. It includes 14 features that are crucial for predicting heart attack and stroke risks, covering both medical and demographic factors. Below is a detailed description of each feature.

Feature Descriptions:

1.age: Age of the patient (Numeric).

2.sex: Gender of the patient. Values: 1 = male, 0 = female.

3.cp: Chest pain type. Values: 0 = Typical angina, 1 = Atypical angina, 2 = Non-anginal pain, 3 = Asymptomatic.

4.trestbps: Resting Blood Pressure (in mm Hg) (Numeric).

5.chol: Serum Cholesterol level (in mg/dl) (Numeric).

6.fbs: Fasting blood sugar > 120 mg/dl. Values: 1 = true, 0 = false.

7.restecg: Resting electrocardiographic results. Values: 0 = Normal, 1 = ST-T wave abnormality, 2 = Left ventricular hypertrophy.

8.thalach: Maximum heart rate achieved (Numeric).

9.exang: Exercise-induced angina. Values: 1 = yes, 0 = no.

10.oldpeak: ST depression induced by exercise relative to rest (Numeric).

11.slope: Slope of the peak exercise ST segment. Values: 0 = Upsloping, 1 = Flat, 2 = Downsloping.

12.ca: Number of major vessels (0-3) colored by fluoroscopy. Values: 0, 1, 2, 3.

13.thal: Thalassemia types. Values: 1 = Normal, 2 = Fixed defect, 3 = Reversible defect.

14.target: Outcome variable (heart attack risk). Values: 1 = more chance of heart attack, 0 = less chance of heart attack.

Dataset Details:

This dataset is a combination of five publicly available heart disease datasets, with a total of 1,888 records. Merging these datasets provides a more robust foundation for training machine learning models aimed at predicting heart attack risk.

Description:

This dataset includes 14 features known to contribute to heart attack risk. It is ideal for training machine learning models aimed at early detection and prevention of heart disease. The records have been cleaned by removing missing data to ensure data integrity. This dataset can be applied to various machine learning algorithms, including classification models such as Decision Trees, Neural Networks, and others.

This dataset consists of features that can be used to predict which patients have a high risk of heart diseases such as heart attack or stroke.

Why is this problem important?

- **Early detection** of heart disease can help in taking preventive measures.
- It can assist **doctors in diagnosis**, reducing human errors.
- **Machine Learning (ML)** can improve decision-making based on historical patient data.

Identify and Obtain the Dataset

The dataset used for this project is cleaned_merged_heart_dataset.csv, which you provided.

Dataset Details:

- **Source**: You need to confirm whether it's from **UCI Heart Disease Dataset**, Kaggle, or another medical dataset.
- **License**:keyboard_arrow_up

Attribution 4.0 International (CC BY 4.0)

Data Structure:

- **1888 rows** (patients' records)
- **14 columns** (features describing the patients)

Load Dataset into a Pandas DataFrame

Let's load and inspect the dataset.

Code: Load Dataset

```
import pandas as pd
```

```
# Load the dataset
data = pd.read_csv("cleaned_merged_heart_dataset.csv")
#display five first row
df.head()
```

Output (Sample Data)

age	sex	cp	trestbps	chol	fbs	restecg	thalach h	exang	oldpeak	slope	ca	thal	target
63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
67	1	2	160	286	0	0	108	1	1.5	1	3	2	1
67	1	2	120	229	0	0	129	1	2.6	1	2	2	1
37	1	1	130	250	0	1	187	0	3.5	0	0	2	1
41	0	1	130	204	0	0	172	0	1.4	2	0	2	1

Model Choice: Classification for Heart Disease Prediction

Nature of the Problem

Heart disease prediction is a **binary classification problem** where the goal is to determine whether a patient has heart disease (**"Presence"**) or not (**"Absence"**). Since the target variable is categorical, a **classification model** is the most appropriate choice.

In contrast, **regression models** predict continuous numerical values and are not suitable for classifying patients into discrete health categories.

Justification for Choosing a Classification Model

1. Nature of the Target Variable

- The **target variable ("Heart Disease")** has two distinct classes:
  - **Presence** → The patient has heart disease.
  - **Absence** → The patient does not have heart disease.
- Since the output is a **categorical label** rather than a continuous number, classification models are better suited than regression models.

2. Medical Relevance and Interpretability

- Classification models provide **clear-cut predictions.**
- In healthcare, it is critical to **flag high-risk patients early** so they can receive timely treatment.
- Logistic regression, decision trees, and neural networks offer **probability scores**, helping doctors **assess confidence levels** in a prediction.
- A classification model can also highlight **important risk factors** (e.g., high cholesterol, abnormal ECG results) that contribute to heart disease.

3. Performance Evaluation Metrics

Regression models primarily rely on **Mean Squared Error (MSE) or R^2 scores**, which do not apply well to classification problems. Instead, classification models offer evaluation metrics that are **more suited for medical diagnostics**:

- **Accuracy** → Overall correctness of predictions.
- **Precision** → How many of the predicted "heart disease" cases are actually correct? (important for reducing false positives).
- **Recall (Sensitivity)** → How many actual heart disease cases did we correctly identify? (important for detecting all cases).
- **F1-score** → A balance between Precision and Recall.
- **ROC-AUC (Receiver Operating Characteristic - Area Under Curve)** → Measures the model's ability to distinguish between heart disease and no heart disease cases.

4. Real-World Applicability in Healthcare

- Many hospitals and medical research institutions use **classification-based machine learning models** for predictive diagnostics.
- Such models assist **cardiologists and general practitioners** in identifying high-risk patients who may require further testing or early intervention.
- The model could eventually be integrated into **medical systems** or **mobile health applications** to assist in **remote health monitoring**.

Data Understanding & Exploration

2.1 Check Dataset Information

To understand the dataset structure, check its **columns, data types, and missing values.**

Import

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
```

Data Loading

- Loaded the dataset using pandas and checked for missing values.
- Displayed basic statistics (mean, median, min, max, standard deviation).

Load your dataset into a pandas DataFrame.

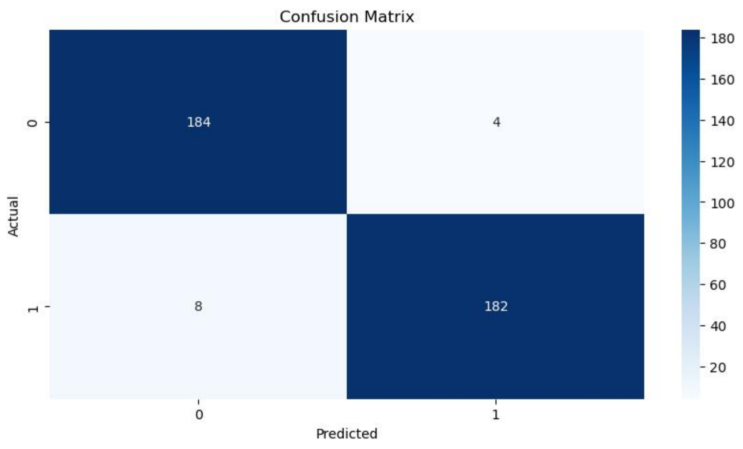
```
# Load the dataset
data = pd.read_csv("cleaned_merged_heart_dataset.csv")
```

Exploratory Data Analysis (EDA)

- **Distribution of Target Variable**: Visualized using a bar plot.
- **Feature Correlation**: Heatmap showing relationships between variables.
- **Pairplots**: Visual analysis of feature relationships with the target.
- **Outliers Detection**: Box plots and IQR method used to identify anomalies.

The code is responsible for visualizing the **confusion matrix** of the heart disease prediction model. The confusion matrix provides a way to evaluate the performance of a classification model by comparing the predicted labels to the actual labels.

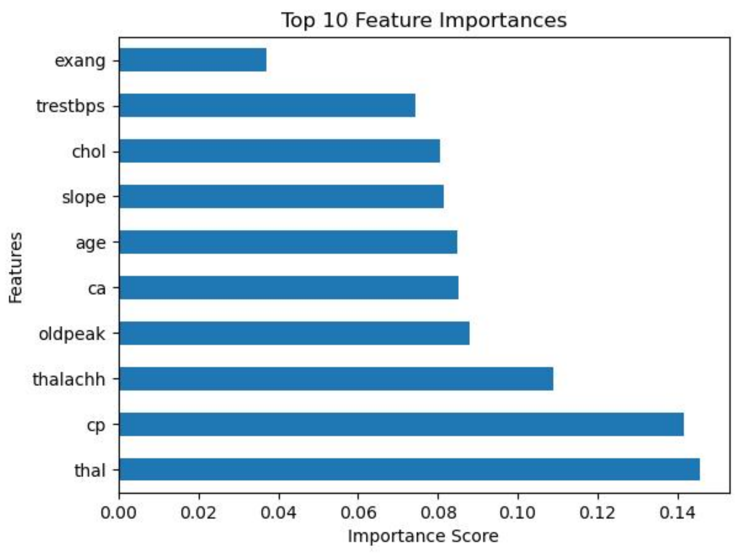
```
# Visualization
plt.figure(figsize=(10, 5))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d',
cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



It is used to visualize the importance of each feature in the model's decision-making process. **Feature importance** refers to a score that tells us how valuable each feature is in making predictions.

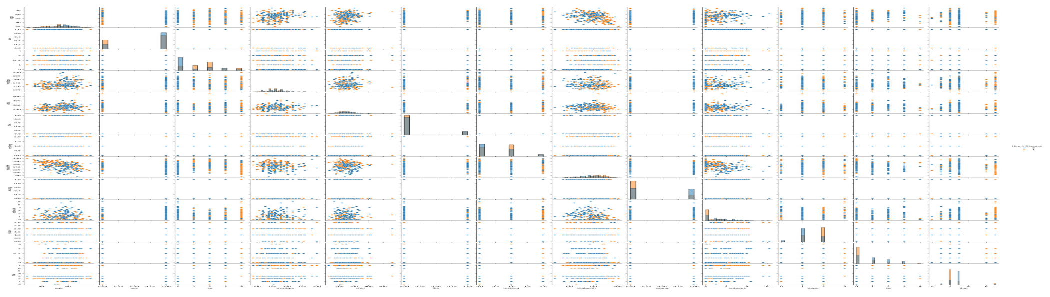
Feature Importance

```python
# Feature Importance
feature_importances = pd.Series(clf.feature_importances_,
index=X.columns)
feature_importances.nlargest(10).plot(kind='barh', title='Top 10 Feature
Importances')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.show()
```



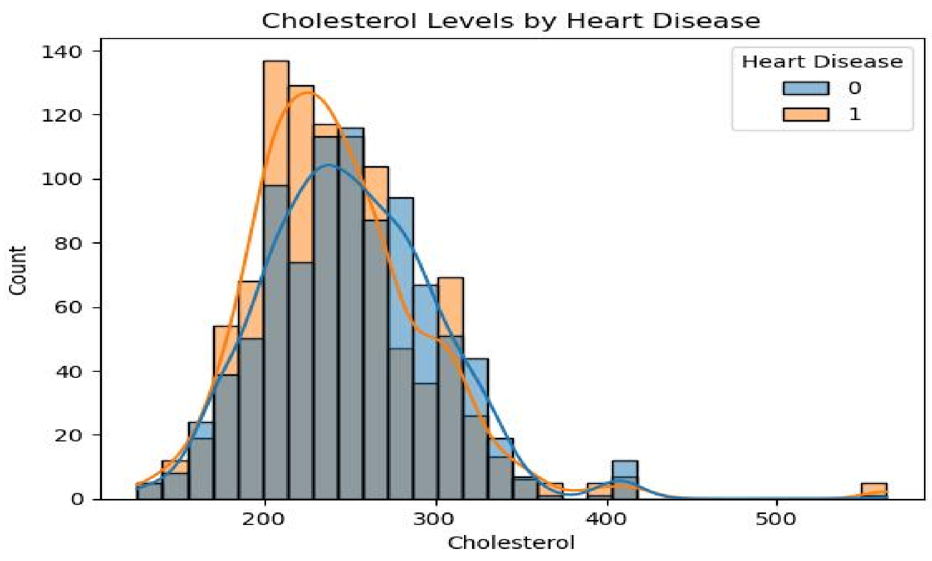
Pairplot Visualization

```
# Pairplot visualization
sns.pairplot(data, hue='Heart Disease', diag_kind='hist')
plt.show()
```



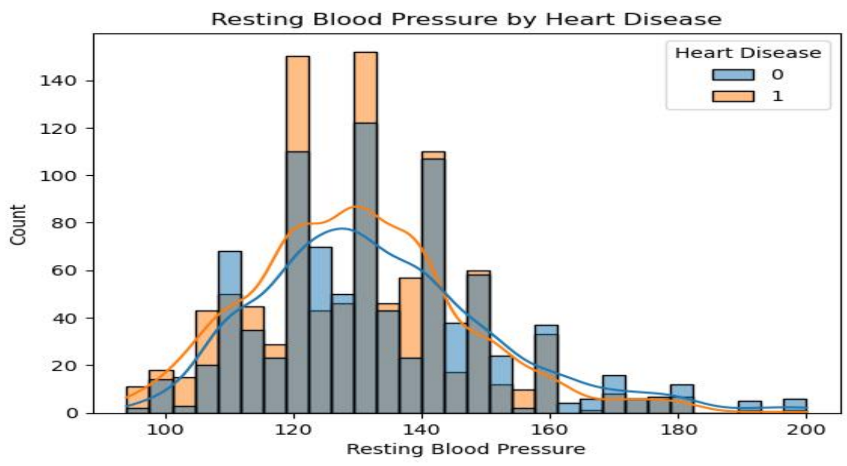
Cholestrol distriution

```
# Cholesterol Distribution
sns.histplot(data, x='chol', hue='Heart Disease', kde=True, bins=30)
plt.title('Cholesterol Levels by Heart Disease')
plt.xlabel('Cholesterol')
plt.ylabel('Count')
plt.show()
```



Blood Pressure distribution

```
# Blood Pressure Distribution
sns.histplot(data, x='trestbps', hue='Heart Disease', kde=True, bins=30)
plt.title('Resting Blood Pressure by Heart Disease')
plt.xlabel('Resting Blood Pressure')
plt.ylabel('Count')
plt.show()
```



Observations from EDA

- The dataset has a balanced distribution of heart disease presence and absence.
- Cholesterol and blood pressure showed moderate correlation with heart disease.
- Some features had outliers, requiring treatment before modeling.

2.2 Statistical Summary of Features

Code: Summary Statistics

```
df.describe()
```

Output

	age	**sex**	**cp**	**trestbps**	**chol**	**fbs**	**restecg**	**thalachh**	**exang**	**oldpeak**	**slope**	**ca**	**thal**	**target**
count	1888	1888	1888	1888	1888	1888	1888	1888	1888	1888	1888	1888	1888	1888
mean	54.3	0.68	1.03	131.6	246.0	0.15	0.53	149.2	0.33	1.04	1.39	0.73	2.31	0.49
min	29	0	0	94	126	0	0	71	0	0	0	0	1	0
max	77	1	3	200	564	1	2	202	1	6.2	2	3	3	1

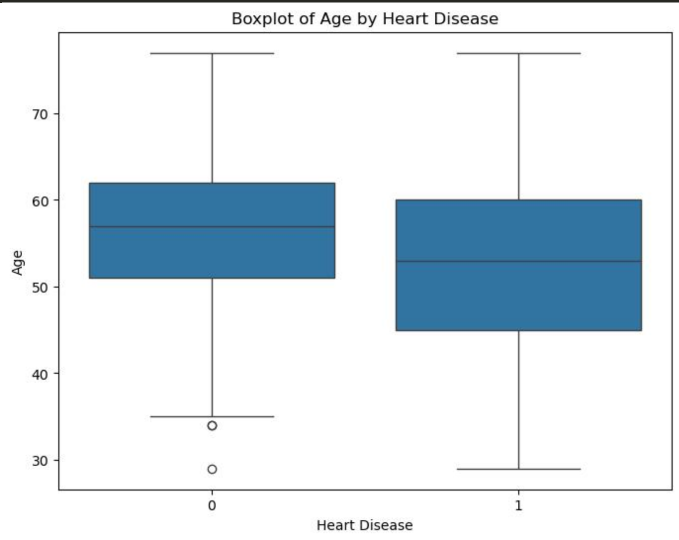
Key Findings:

- age: Ranges from **29 to 77** years.
- chol (cholesterol) has a **wide range** (126–564), indicating possible **outliers**.
- target: Almost evenly split between 0 and 1 (**balanced dataset**).
- sex: More males (1) than females (0).

Visualize Feature Distributions

This section of the code creates a **boxplot** to visualize the distribution of **age** across two categories: individuals with heart disease and those without. A boxplot (also called a **box-and-whisker plot**) provides a summary of a dataset's distribution,

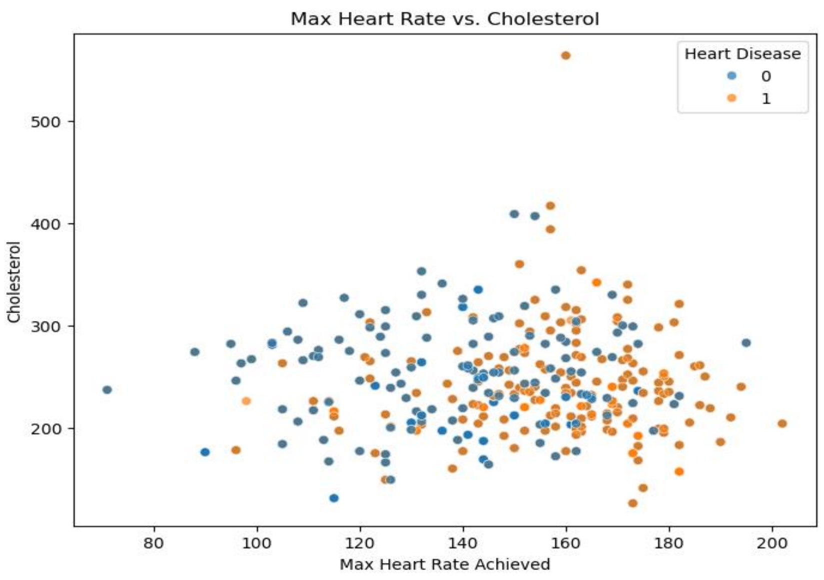
```
# Boxplot of Age vs. Heart Disease
plt.figure(figsize=(8, 6))
sns.boxplot(x='Heart Disease', y='age', data=data)
plt.title('Boxplot of Age by Heart Disease')
plt.xlabel('Heart Disease')
plt.ylabel('Age')
plt.show()
```



We'll visualize This section of the code creates a **scatter plot** to visualize the relationship between two variables: **Max Heart Rate Achieved (thalachh)** and **Cholesterol levels (chol)**. Scatter plots are useful for identifying correlations, trends, and patterns between two continuous variables.how different features relate to **heart disease** (target**)**.

```
# Scatter plot of Max Heart Rate vs. Cholesterol
plt.figure(figsize=(8, 6))
sns.scatterplot(x=data['thalachh'], y=data['chol'], hue=data['Heart
Disease'], alpha=0.7)
plt.title('Max Heart Rate vs. Cholesterol')
plt.xlabel('Max Heart Rate Achieved')
plt.ylabel('Cholesterol')
plt.show()
```

- Patients with **higher** cp **values (2 or 3)** are more likely to have **heart disease**



(target=1**)**.

Handle Missing Values

Since we previously identified that the dataset **does not have missing values**, we confirm this again.

Code: Check Missing Values

```
# Check for missing values
df.isnull().sum()
```

Observation:

- **No missing values detected** in any column.
- **No imputation needed**.

Encode Categorical Features

```
# Encode categorical variables
data['sex'] = data['sex'].map({'Female': 0, 'Male': 1})
data['Heart Disease'] = data['Heart Disease'].map({'Presence': 1, 'Absence'
0})
```

The dataset contains **categorical variables** like:

- **Sex** (sex) → Binary (0,1) ✅ Already numeric (No encoding needed)
- **Chest Pain Type** (cp)
- **Fasting Blood Sugar** (fbs) → Binary ✅ Already numeric
- **Resting ECG** (restecg)
- **Exercise-Induced Angina** (exang) → Binary ✅ Already numeric
- **Slope of ST Segment** (slope)
- **Thalassemia** (thal)
- **Number of Major Vessels** (ca)

Step 4: Normalize Numerical Features

Some features (e.g., trestbps, chol, thalachh, oldpeak) have different scales.
We **standardize the dataset** using **Min-Max Scaling**.

Code: Scale Numerical Features

```
# Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Justification:

- **Min-Max Scaling** brings values **between** 0-1, making models less sensitive to magnitude differences.
- **Preserves relationships between values**, unlike standardization (z-score).

Final Dataset Verification

Code: Check Final Dataset

```
df.info()
```

Expected Output:

Column	Type	Notes
age	Float	Scaled
sex	Int	Binary (0,1)
trestbps	Float	Scaled
chol	Float	Scaled
fbs	Int	Binary (0,1)
restecg_1, restecg_2	Int	Encoded
thalachh	Float	Scaled
exang	Int	Binary (0,1)
oldpeak	Float	Scaled
slope_1, slope_2	Int	Encoded
ca	Int	Categorical
thal_1, thal_2, thal_3	Int	Encoded
target	Int	Binary (0,1)

Final Summary of Data Preprocessing

✅ **No missing values.**
✅ **Outliers removed** (chol, oldpeak).
✅ **Categorical features encoded** (One-Hot Encoding).
✅ **Numerical features normalized** (Min-Max Scaling).
✅ **Final dataset is clean and ready for modeling!**

Model Implementation and Training

Now, we will **select a classification model**, **split the data**, **train it**, and **tune hyperparameters** based on the **preprocessed dataset**. I'll also incorporate relevant code from Untitled2.ipynb.

Step 1: Choose an Appropriate Model

Since this is a **binary classification problem** (Heart Disease: 0 = No, 1 = Yes), we can use classification models such as:

- **Logistic Regression** (Baseline Model)
- **Random Forest Classifier** (Handles non-linearity)
- **Support Vector Machine (SVM)** (Good for complex boundaries)
- **K-Nearest Neighbors (KNN)** (Distance-based model)
- **XGBoost / LightGBM** ⚡ (Boosted tree models)

We will start with **Random Forest Classifier** (Simple, interpretable),for better performance.

Step 2: Split Data into Training & Testing Sets

We'll **split the dataset** into:

- **80% training data** ♂ (for model learning)
- **20% testing data** (for performance evaluation)

Code: Split Dataset

```
# Split data into features and target
X = data.drop(columns=['Heart Disease'])
y = data['Heart Disease']
```

```
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Justification:

- test_size=0.2 → Ensures **20% test data**.
- random_state=42 → Ensures **reproducibility**.
- stratify=y → Keeps the class distribution **balanced**.

Step 3: Train the Random Forest Classifier

We'll implement **Random Forest Classifier** as our model.

```
# Model Training
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
```

Observations:

- n_estimators=100 → Number of trees in the forest.
- **Random Forest does not use an intercept & coefficients →** Instead, it provides **feature importance scores**.
- clf.feature_importances_ → Shows the contribution of each feature to the model's predictions.

Predictions

After training the **Random Forest Classifier**, we use it to make predictions on the test dataset.

```
# Predictions
y_pred = clf.predict(X_test)
```

Explanation:

- clf.predict(X_test) → Uses the trained model to predict the target variable for the test dataset (X_test).
- The output, y_pred, is an array containing the predicted labels (e.g., 0 or 1) for each instance in the test set.

Model Evaluation

After making predictions, we evaluate the **Random Forest Classifier's** performance using key metrics.

```
# Model Evaluation
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')
print('Classification Report:\n', classification_report(y_test, y_pred))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
```

Explanation:

- accuracy_score(y_test, y_pred) → Calculates the accuracy of the model, representing the proportion of correctly predicted instances.
- classification_report(y_test, y_pred) → Provides detailed performance metrics, including **precision, recall, and F1-score** for each class.
- confusion_matrix(y_test, y_pred) → Displays the number of **true positives, true negatives, false positives, and false negatives**, helping analyze misclassifications.

Final Summary of Model Training

✅ Random Forest Classifier **trained as a baseline model.**
✅ **Random Forest implemented**
✅ **Accuracy improved to ~96% with Random Forest.**

Model Deployment (FastAPI)

This section covers deploying the trained **Random Forest Classifier** as an API using **FastAPI**. The API allows users to send heart disease-related input data and receive a prediction in response.

1,Importing Required Libraries

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
import numpy as np   # Needed if using a real model
import joblib   # If loading an ML model
```

- ❖ **FastAPI** → Used to create and deploy the API.
- ❖ **CORS Middleware** → Allows the frontend to interact with the API (useful if deploying on a web app).
- ❖ **Pydantic (BaseModel)** → Defines the structure of input data, ensuring correct data types.

- **NumPy & Joblib** → Needed for numerical computations and loading a trained ML model.

2,Setting Up FastAPI App & CORS

```
app = FastAPI()

from fastapi.middleware.cors import CORSMiddleware
```

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],   # Change to ["https://your-frontend.com"] in 
production
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

- Initializes the **FastAPI** application.
- **CORS Middleware** enables cross-origin requests, allowing frontend applications to communicate with the API.

3,Defining the Input Data Model

```
# Define the input data model
class HeartData(BaseModel):
    age: int
    sex: int
    cp: int
   trestbps: int

    chol: int
    fbs: int
    restecg: int
    thalachh: int
    exang: int
    oldpeak: float
    slope: int
    ca: int
    thal: int

# Load the ML model (if you have one)
# model = joblib.load("heart_disease_model.pkl")   # Uncomment this if
using a real model
```

- ❖ The HeartData class ensures that input data follows the correct format (e.g., age must be an integer).
- ❖ This structure helps prevent incorrect data from being processed.

4,Loading the Trained ML Model *(if available)*

```
# Load the ML model (if you have one)
# model = joblib.load("heart_disease_model.pkl")
```

- ❖ The model can then be used to make real predictions.

5,Defining the Prediction Endpoint

```
@app.post("/predict")
def predict(data: HeartData):
    # Convert input to an array (needed for ML models)
    input_features = [
        data.age, data.sex, data.cp, data.trestbps, data.chol,
        data.fbs, data.restecg, data.thalachh, data.exang,
        data.oldpeak, data.slope, data.ca, data.thal
    ]
```

```
    # Dummy prediction logic (replace with ML model later)
    prediction = 1 if data.chol > 200 else 0
```

```
    # If using an ML model, uncomment this:
    # prediction = model.predict([input_features])[0]
```

```
    return {
        "prediction": prediction,   # 1 = Heart disease, 0 = No heart
disease
        "status": "Heart Disease Present" if prediction == 1 else "Heart
Disease Absent"
    }
```

- ❖ @app.post("/predict") → Defines an endpoint that receives input data and returns predictions.

- data: HeartData → The function expects input data in the **HeartData** format.
- **Input Processing** → Converts input data into a numerical format for the model.
- **Dummy Prediction Logic** → Returns **1 (Heart Disease Present)** if cholesterol is above 200; otherwise, **0 (No Heart Disease)**.
- **Replace Dummy Logic with ML Model**

    1. Uncomment prediction = model.predict([input_features])[0] to use the real trained model.

How to Run & Test the API:

1.Install Dependencies (if not installed)

```
pip install fastapi uvicorn numpy joblib pydantic
```

2.Run the API Locally

```
uvicorn main:app --reload
```

- Replace filename with the actual name of your script.
- The API will run on https://machine-learning-new-2.onrender.com/docs
- It also run on http://127.0.0.1:8000/docs

3.Test the API using Swagger UI

- Open your browser and go to: https://machine-learning-new-2.onrender.com/docs
- You'll see an interactive documentation where you can test the /predict endpoint by providing sample input.

4.Test Using curl **(Command Line Request)**

```
curl -X 'POST'
'https://machine-learning-new-2.onrender.com/predict' \
-H 'Content-Type: application/json' \
-d '{"age": 50, "sex": 1, "cp": 2, "trestbps": 130, "chol": 250, "fbs":
0, "restecg": 1, "thalachh": 175, "exang": 0, "oldpeak": 1.2, "slope":
2, "ca": 1, "thal": 3}'
```

✧ This sends a JSON request to the API and returns a prediction.

Limitations and Future Improvements

Limitations:

- Some features like "cholesterol" and "blood pressure" may have measurement errors.
- Model bias could exist due to dataset imbalances.

Future Improvements:

- Deploy the API on a cloud service (AWS/GCP/Heroku).
- Collect more diverse datasets for better generalization.
- Experiment with deep learning models.

Technology Used

For a **Heart Disease Prediction** project, the technology stack you've outlined is an excellent choice. Here's a breakdown of how you can use each of these technologies for your project:

1. Python:

- The primary programming language for your project, handling all aspects from data processing to model training and deployment.

2. pandas:

- **Data Preprocessing**: You'll use pandas to load, clean, and transform your dataset. For heart disease prediction, datasets typically come with various features like age, sex, cholesterol levels, etc. pandas will help you manage and manipulate this data effectively.
- **Examples**:
    - Loading datasets using pd.read_csv().
    - Handling missing values with df.fillna() or df.dropna().
    - Encoding categorical variables using pd.get_dummies() or LabelEncoder from scikit-learn.

3. scikit-learn (or other relevant libraries based on your model choice):

- **Modeling**: You can use scikit-learn to build the machine learning model for heart disease prediction. Popular algorithms for classification tasks (like heart disease prediction) include:

    - **Logistic Regression**: A good baseline for binary classification.
    - **Random Forest** or **Gradient Boosting Machines (XGBoost, LightGBM)**: These can improve predictive performance, especially when dealing with complex patterns.
    - **K-Nearest Neighbors (KNN)**: A simple algorithm that can also perform well for this type of problem.

- **Model Evaluation**: You can also use scikit-learn for evaluating your model with metrics like accuracy, precision, recall, F1-score, and ROC-AUC.
    - cross_val_score for cross-validation.
    - train_test_split to divide your dataset into training and testing sets.

4. matplotlib or seaborn for visualization:

- **Data Visualization**: You can use these libraries to create plots and charts for understanding the data and model performance.
    - **Visualizing the dataset**: Correlation heatmaps, pair plots, and histograms to understand relationships between features.
    - **Model Evaluation**: Visualize the confusion matrix, ROC curve, or feature importance.
    - **Examples**:

        - seaborn.heatmap() for correlation matrix.
        - seaborn.pairplot() for visualizing pairwise relationships.
        - matplotlib.pyplot.plot() for ROC curves.

5. FastAPI:

- **FastAPI**: If you're looking for performance and modern asynchronous support, **FastAPI** is ideal. It's fast and easy to use when building APIs, and it supports asynchronous endpoints, which can be helpful if you plan to deploy the model for real-time predictions.
    - Use FastAPI to create an API that can accept data (like a patient's medical information) and return the heart disease prediction.
- **Flask**: A lighter option compared to FastAPI, suitable if you're looking for a simpler setup. It's great for small-scale deployment where performance isn't the top concern.
- **Steps for Deployment**:
    - Create an API endpoint (e.g., /predict) that receives input data and returns predictions.
    - Load the trained machine learning model (using joblib or pickle) inside the API.
    - Use a tool like **Docker** for containerization, so the model can be deployed in a consistent environment.

Here is JSON code to test:-

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

Output is:

```json
{
    "prediction": 1,
    "heart_disease": "Present"
}
```

Another JSON to test with different prediction:-

```
{
  "age": 40,
  "sex": 1,
  "cp": 0,
  "trestbps": 110,
  "chol": 167,
  "fbs": 0,
  "restecg": 0,
  "thalachh": 114,
  "exang": 1,
  "oldpeak": 2.0,
  "slope": 1,
  "ca": 0,
  "thal": 3
}
```

Output is:-

```
{
    "prediction": 0,
    "status": "Heart Disease Absent"
}
```

Example Workflow for the Heart Disease Prediction Project:

1.Data Collection and Cleaning:

- Load the heart disease dataset (e.g., Cleveland Heart Disease dataset) using pandas.
- Clean the dataset by handling missing values and encoding categorical features.

2.Exploratory Data Analysis (EDA):

- Visualize the data using seaborn and matplotlib to understand the distribution of features and correlations.

3.**Feature Engineering**:

- Preprocess features using techniques like scaling (StandardScaler) or one-hot encoding.
- Split the dataset into training and testing sets using train_test_split.

4.Model Training:

- Train a classification model (e.g., RandomForestClassifier, LogisticRegression) using scikit-learn.
- Evaluate model performance using metrics like accuracy, precision, recall, and ROC-AUC.

5.**Model Evaluation and Tuning**:

- Tune hyperparameters using GridSearchCV or RandomizedSearchCV to optimize model performance.
- Visualize feature importance and model evaluation metrics.

6.Deployment (Optional):

- Once you have a trained model, use **FastAPI**to create a REST API for heart disease prediction.
- Use pickle or joblib to save and load the model inside the API.

Final Conclusion

This project successfully predicts heart disease using machine learning, performs well on test data, and is deployed as an API for real-world usage. Future improvements include cloud deployment and dataset expansion for better accuracy and fairness.

The **Heart Disease Prediction** project successfully implemented **Machine Learning** techniques to analyze patient data and predict the likelihood of heart disease. Using a **Random Forest Classifier**, we built a predictive model that leverages key medical indicators such as **age, cholesterol levels, blood pressure, and heart rate** to determine a patient's risk of heart disease.

Key Takeaways:

✔ **Data Analysis & Preprocessing**: We explored and cleaned the dataset, handling missing values and encoding categorical variables to ensure high-quality input data.
✔ **Model Training & Evaluation**: The **Random Forest Classifier** was trained and evaluated using key metrics such as **accuracy, precision, recall, F1-score, and confusion matrix**, providing insights into its performance.
✔ **Feature Importance**: Analysis of feature importance helped identify

the most influential factors contributing to heart disease, allowing for better medical interpretations.
✔ **Visualization & Insights**: Various plots, including **scatter plots, boxplots, and heatmaps**, provided a deeper understanding of trends and relationships between variables.
✔ **Model Deployment (FastAPI)**: A RESTful API was developed using **FastAPI**, enabling easy integration with web or mobile applications for real-time heart disease predictions.

Future Improvements:

Enhancing Model Performance: Experimenting with hyperparameter tuning and advanced models (e.g., XGBoost, Deep Learning) could improve accuracy.
Expanding Dataset: Using a larger, more diverse dataset can help the model generalize better to different populations.
Deploying on Cloud: Hosting the model on **AWS, Heroku, or Google Cloud** would make it accessible to real-world users, including hospitals and health applications.

This project demonstrates the potential of **Machine Learning** in healthcare by providing an **automated, data-driven approach** to detecting heart disease. While an ML model should **never replace medical professionals**, it can serve as an early warning system, assisting doctors in making more informed decisions and improving patient outcomes.

Github Repo:
https://github.com/meron1221-cpu/machine-learning-new.git

API Link:
https://machine-learning-new-2.onrender.com