**Project Title: Heart Disease Detection AI: Machine Learning Models for Early Cardiovascular Disease Prediction**

**Project Summary:**  
This project leverages machine learning techniques to build predictive models for heart disease detection, focusing on early diagnosis and prevention. Key tools and techniques include **Logistic Regression**, **Random Forest Classifier**, and **SMOTE (Synthetic Minority Over-sampling Technique)** for data balancing. The dataset is preprocessed using **StandardScaler** for feature normalization, and performance is evaluated with accuracy metrics, confusion matrices, and classification reports. The project also demonstrates the use of **imbalanced data handling** techniques to improve model performance and ensure robust predictions.

**Data Source & Data Type Note**

The dataset used in this project was fetched directly from the **UCI Machine Learning Repository** using the Python library ucimlrepo. The specific dataset is titled **"Heart Disease" (ID: 45)**.

**Source Command Used:**

from ucimlrepo import fetch\_ucirepo

heart\_disease = fetch\_ucirepo(id=45)

The dataset contains both **numerical and categorical features**, structured as follows:

* **Total Records**: 303 patients
* **Features**: 13 attributes (e.g., age, cholesterol, chest pain type)
* **Target Variable**: Originally multiclass (0–4), later **converted to binary** for classification (0 = No Disease, 1 = Disease)

**Feature Types:**

* Numerical: age, trestbps, chol, thalach, oldpeak
* Categorical (integer-encoded): cp, restecg, slope, thal, sex, fbs, exang, ca

The data was preprocessed using **standard scaling** and balanced using **SMOTE** to ensure fair training of the model.

**Step-by-Step Breakdown**

**🔹 Step 1: Imports**

import pandas as pd, numpy as np

import seaborn as sns, matplotlib.pyplot as plt

from ucimlrepo import fetch\_ucirepo

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

from imblearn.over\_sampling import SMOTE

* **Purpose**: Load essential libraries for data analysis, model building, and class balancing.
* ucimlrepo: A wrapper to fetch UCI Machine Learning datasets easily.
* SMOTE: Synthetic Minority Over-sampling Technique to handle class imbalance.

**🔹 Step 2: Load Data**

heart\_disease = fetch\_ucirepo(id=45)

df = pd.concat([heart\_disease.data.features, heart\_disease.data.targets], axis=1)

* **What happens here**: You pull the Heart Disease dataset from UCI and combine features + target into one DataFrame.
* id=45: The dataset ID for "Heart Disease" in UCI’s repository.

**🔹 Step 3: Convert Multiclass to Binary**

df['target'] = df['target'].apply(lambda x: 0 if x == 0 else 1)

* Original target column had values from 0 to 4 (0 = no disease, 1–4 = various stages).
* **Why this matters**: We're simplifying this into a **binary classification task**:
  + 0: No disease
  + 1: Has disease (any level)

**🔹 Step 4: EDA – Visualize Target Distribution**

sns.countplot(data=df, x='target')

plt.title("Target Distribution (Binary)")

plt.show()

* **Goal**: Understand the imbalance.
* Usually, more people don't have disease (class 0), so we visualize to decide whether balancing is needed.

**🔹 Step 5: Split Features and Target**

X = df.drop('target', axis=1)

y = df['target']

* Separating:
  + X: All predictor variables (age, cholesterol, etc.)
  + y: Target variable (0 or 1)

**🔹 Step 6: Standardize Features**

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

* **Why standardize?** Logistic Regression and some other models perform better when data is scaled.
* This ensures all features have mean = 0 and std = 1, reducing bias caused by differing units.

**🔹 Step 7: Balance the Dataset (SMOTE)**

smote = SMOTE(random\_state=42)

X\_res, y\_res = smote.fit\_resample(X\_scaled, y)

* **Why SMOTE?** Your target classes were imbalanced.
* SMOTE creates **synthetic samples** of the minority class (class 1 – disease) to balance out the dataset.
* This avoids biased learning toward the majority class.

**🔹 Step 8: Train-Test Split**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X\_res, y\_res, test\_size=0.2, random\_state=42, stratify=y\_res

)

* Splits the balanced data into:
  + 80% training
  + 20% testing
* stratify=y\_res ensures the split maintains class balance.

**🔹 Step 9: Logistic Regression**

log\_model = LogisticRegression(class\_weight='balanced', max\_iter=1000)

log\_model.fit(X\_train, y\_train)

y\_pred\_log = log\_model.predict(X\_test)

* **Why class\_weight='balanced'?**
  + Even after SMOTE, we make the model give fair weight to both classes.
* max\_iter=1000: Increases iterations to ensure convergence.

**Evaluation:**

print("Accuracy:", accuracy\_score(y\_test, y\_pred\_log))

print(confusion\_matrix(y\_test, y\_pred\_log))

print(classification\_report(y\_test, y\_pred\_log))

* **Confusion Matrix**: Shows true/false positives and negatives.
* **Classification Report**: Gives precision, recall, and f1-score for each class.
  + **Precision**: Correct positive predictions out of all positive predictions.
  + **Recall**: Correct positive predictions out of all actual positives.
  + **F1-score**: Balance between precision and recall.
  + **Accuracy**: Overall correctness of the model.

**🔹 Step 10: Random Forest**

rf\_model = RandomForestClassifier(n\_estimators=200, random\_state=42, class\_weight='balanced')

rf\_model.fit(X\_train, y\_train)

y\_pred\_rf = rf\_model.predict(X\_test)

* **Random Forest**: An ensemble method of decision trees.
* n\_estimators=200: Uses 200 trees for more stable results.
* class\_weight='balanced': Prevents the model from favoring the majority class.

**Evaluation:**

print("Accuracy:", accuracy\_score(y\_test, y\_pred\_rf))

print(confusion\_matrix(y\_test, y\_pred\_rf))

print(classification\_report(y\_test, y\_pred\_rf))

**✅ Outcome Summary**

**🔹 Logistic Regression**

* Accuracy: ~83%
* Balanced results
* Lightweight, interpretable

**🔹 Random Forest**

* Accuracy: May vary (usually 80–90%)
* Robust, less interpretable
* Can be improved with tuning

**Statistics Formulas Used in the Project**

1. **Accuracy**:  
   Accuracy is the proportion of the total number of correct predictions to the total predictions made.

Accuracy=TP + TNTP + TN + FP + FN\text{Accuracy} = \frac{\text{TP + TN}}{\text{TP + TN + FP + FN}}Accuracy=TP + TN + FP + FNTP + TN​

Where:

* + **TP** = True Positives (Correctly predicted positives)
  + **TN** = True Negatives (Correctly predicted negatives)
  + **FP** = False Positives (Incorrectly predicted as positives)
  + **FN** = False Negatives (Incorrectly predicted as negatives)

1. **Precision**:  
   Precision is the proportion of positive predictions that were correct.

Precision=TPTP + FP\text{Precision} = \frac{\text{TP}}{\text{TP + FP}}Precision=TP + FPTP​

1. **Recall** (or Sensitivity):  
   Recall is the proportion of actual positives that were correctly identified.

Recall=TPTP + FN\text{Recall} = \frac{\text{TP}}{\text{TP + FN}}Recall=TP + FNTP​

1. **F1-Score**:  
   F1-Score is the harmonic mean of Precision and Recall, balancing both metrics.

F1-Score=2⋅Precision⋅RecallPrecision+Recall\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}F1-Score=2⋅Precision+RecallPrecision⋅Recall​

1. **Confusion Matrix**:  
   A confusion matrix is a table used to describe the performance of a classification algorithm. It compares the predicted values with actual values.

Predicted PositivePredicted NegativeActual PositiveTPFNActual NegativeFPTN\begin{array}{|c|c|c|} \hline & \text{Predicted Positive} & \text{Predicted Negative} \\ \hline \text{Actual Positive} & \text{TP} & \text{FN} \\ \hline \text{Actual Negative} & \text{FP} & \text{TN} \\ \hline \end{array}Actual PositiveActual Negative​Predicted PositiveTPFP​Predicted NegativeFNTN​​

**Project Story: Why This Project?**

This project tackles **heart disease classification**, a critical issue for public health. Here's the *story*:

**1. The Problem:**

Heart disease remains one of the leading causes of death worldwide. Early detection can significantly improve treatment outcomes, saving lives and reducing healthcare costs. However, detecting heart disease involves complex and expensive tests, which may not always be available or timely.

**2. The Importance of Predictive Models:**

With **predictive modeling**, machine learning (ML) models like Logistic Regression and Random Forest can help automate the detection process. Instead of relying solely on expensive tests, these models can predict the likelihood of heart disease based on readily available health data.

By building a machine learning model, we aim to **predict the presence or absence of heart disease** from common features like:

* Age
* Blood Pressure
* Cholesterol levels
* Resting ECG
* Maximum Heart Rate
* etc.

These models can be implemented as an easy-to-use tool in **healthcare settings**, empowering doctors to make more informed decisions quickly.

**3. Challenges in the Data:**

Heart disease data often suffers from **imbalanced classes**, where one class (e.g., no disease) outweighs the other (e.g., has disease). This imbalance can lead to biased models that predict the majority class more often.  
**Solution**: By using **SMOTE (Synthetic Minority Over-sampling Technique)**, we balanced the dataset, creating synthetic samples for the underrepresented class to prevent the model from favoring the majority class.

**4. Why Logistic Regression and Random Forest?**

* **Logistic Regression** is a simple, interpretable model often used for binary classification tasks (i.e., has disease or not). It provides insights into the relationships between each feature and the target variable.
* **Random Forest** is an ensemble method that combines multiple decision trees to improve accuracy and stability. It's less interpretable but more robust to overfitting and can handle non-linear relationships in the data.

**5. Impact of the Project:**

* **Automated Diagnosis**: The model can be deployed in low-resource environments or as part of a **decision support system** for healthcare providers. It can also assist patients who may not have immediate access to a specialist, helping them make timely decisions about their health.
* **Saving Lives**: Faster, more accurate diagnosis can lead to earlier interventions, improving treatment outcomes.
* **Scalable Model**: With minor modifications, this model can be expanded to other health-related datasets, helping to predict various diseases (diabetes, cancers, etc.).

**Real-Time Use of This Project: Heart Disease Detection AI**

This project can be applied in **clinical decision support systems** where hospitals or healthcare platforms use predictive algorithms to assist doctors in early diagnosis. By inputting patient health data (like age, cholesterol, blood pressure, etc.), the trained model can flag high-risk individuals **before critical symptoms appear**, helping:

* **Doctors** prioritize cases needing urgent attention
* **Telemedicine platforms** auto-screen patients remotely
* **Insurance companies** assess health risk more accurately
* **Health apps** give early lifestyle alerts to users