Chronic Heart Disease Prediction Using Random Forest Classification

A Machine Learning Approach to 10-Year CHD Risk Assessment

PRESENTER AFFILIATION

Akash Jha Pacific University

DATE

December 04, 2025

Project Overview

Project Goal

- Predict 10-year risk of Chronic Heart Disease (CHD) in suspected patients
- Classify patients into binary risk categories (0: No Risk, 1: High Risk)
- Identify key clinical indicators contributing to heart disease

Dataset Source

- Framingham Heart Study data (framingham.csv)
- Rich clinical dataset containing demographic, behavioral, and medical history features
- Key input for epidemiological research on cardiovascular disease

Methodology

- Comparative analysis: Logistic Regression vs. Random Forest Classifier
- Full ML pipeline: EDA, cleaning, feature scaling, VIF analysis

Key Challenges

- **Imbalanced Dataset:** Significant disparity between positive and negative CHD cases
- **Missing Values:** Incomplete records requiring strategic handling

Libraries & Tools Used

The project leverages the standard Python Data Science stack for data processing, visualization, and machine learning model development.

DATA PROCESSING

pandas

Data manipulation and analysis, handling structured data (DataFrames), and cleaning operations.

DATA PROCESSING

numpy

Fundamental package for scientific computing, array operations, and numerical processing.

VISUALIZATION

matplotlib

Comprehensive library for creating static, animated, and interactive visualizations in Python.

VISUALIZATION

seaborn

Statistical data visualization based on matplotlib, providing a high-level interface for attractive graphics.

MODELING

scikit-learn

Core ML library for classification, regression, model selection, preprocessing, and metrics.

MODELING

statsmodels

Classes and functions for estimation of statistical models (GLM, Logistic Regression) and testing.

OPTIMIZATION

imblearn

Toolbox for imbalanced dataset handling, specifically providing SMOTE implementation.

INTERACTIVE

plotly

Interactive graphing library used for dynamic and explorable data visualizations.

Data Loading & Initial Exploration

chd_analysis.py

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# Load the Framingham dataset
df = pd.read_csv("framingham.csv")
# Check dimensions and missing values
print("Dataset Shape:", df.shape)
print("\nMissing Values Summary:")
print(df.isna().sum())
# Display first few rows
df.head()
```

Dataset Overview

The Framingham Heart Study dataset contains medical and lifestyle records for over 4,000 patients. The goal is to predict the 10-year risk of coronary heart disease (TenYearCHD).

CONSOLE OUTPUT

```
Dataset Shape: (4240, 16)

Missing Values Summary:
glucose: 388, education: 105, BPMeds: 53, totChol: 50 ...
Total rows with missing data: ~14%
```

DEMOGRAPHICS

- male (0/1)
- age (continuous)
- education (categorical)

LIFESTYLE

- currentSmoker (0/1)
- cigsPerDay (count)
- BPMeds (0/1)

MEDICAL HISTORY

- prevalentStroke
- prevalentHyp
- diabetes

CLINICAL MEASUREMENTS

- totChol, sysBP, diaBP
- BMI, heartRate
- glucose

Data Loading & Initial Exploration

chd_analysis.py

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# Load the Framingham dataset
df = pd.read_csv("framingham.csv")
# Check dimensions and missing values
print("Dataset Shape:", df.shape)
print("\nMissing Values Summary:")
print(df.isna().sum())
# Display first few rows
df.head()
```

Dataset Overview

The Framingham Heart Study dataset contains medical and lifestyle records for over 4,000 patients. The goal is to predict the 10-year risk of coronary heart disease (TenYearCHD).

CONSOLE OUTPUT

```
Dataset Shape: (4240, 16)

Missing Values Summary:
glucose: 388, education: 105, BPMeds: 53, totChol: 50 ...
Total rows with missing data: ~14%
```

DEMOGRAPHICS

- male (0/1)
- age (continuous)
- education (categorical)

LIFESTYLE

- currentSmoker (0/1)
- cigsPerDay (count)
- BPMeds (0/1)

MEDICAL HISTORY

- prevalentStroke
- prevalentHyp
- diabetes

CLINICAL MEASUREMENTS

- totChol, sysBP, diaBP
- BMI, heartRate
- glucose

Data Cleaning: Handling Missing Values

Strategy: Due to the clinical nature of the study, we prioritized data integrity over quantity. We applied listwise deletion (dropna) to remove any patient records with incomplete medical history.

Raw Dataset

Total Records	4,240
Data Quality	Compromised

Glucose NA:	388
Education NA:	105
BPMeds NA:	53
TotChol NA:	50

DROPNA

Clean Dataset (df2)

Total Records	3,658
Data Quality	100% Complete
Rows Removed	-582 (13.7%)
Status	Ready for ML

```
# Remove all rows containing missing values to ensure model stability
df2 = df.dropna()

# Verify cleaning results
print(f"Remaining records: {df2.shape[0]}") # Output: 3658
print(f"Missing values: {df2.isna().sum().sum()}") # Output: 0
```

Exploratory Data Analysis (EDA)

Feature Correlation Heatmap

Interactive Visualization

Key Observations

- **Multicollinearity Detected:** High correlation (0.78) between sysBP and diaBP suggests redundancy.
- **Age Factor:** Moderate positive correlation between Age and SysBP/CHD.
- **Target Variable:** 'TenYearCHD' shows relatively weak linear correlations, implying non-linear patterns suitable for Random Forest.

PYTHON CODE — SEABORN

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Create correlation matrix
plt.figure(figsize=(16, 10))
heatmap = sns.heatmap(
    df2.corr(),
    annot=True,
    cmap="YlGnBu",
    fmt=".2f"
)
plt.title("Correlation Heatmap")
plt.show()
```

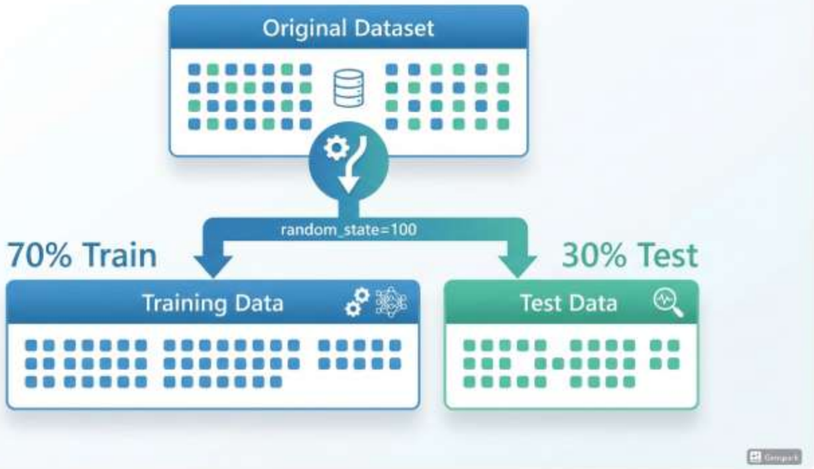
Feature Scaling


```python
from sklearn.preprocessing import StandardScaler

# Initialize the scaler
scale = StandardScaler()

# Select continuous features
cols = ['age', 'cigsPerDay', 'totChol',
        'sysBP', 'glucose', 'BMI']

# Transform features (Mean=0, Std=1)
```

Why Scale Features?

- **Normalization:** Raw features have vastly different ranges (e.g., Age: 30-70 vs. Cholesterol: 150-400), which can bias distance-based algorithms.
- **Algorithm Sensitivity:** Models like Logistic Regression and KNN are sensitive to magnitude; scaling ensures all features contribute equally.
- **Convergence:** Gradient descent converges faster on standardized data.

Train-Test Split



Training Set (70%)

Used to build and train the model. The algorithm learns patterns and relationships from this subset.

Testing Set (30%)

Held back for final evaluation. Simulates "unseen" data to test generalizability.

Configuration

random_state=100: Ensures reproducibility. Same split every time code runs.

08

PYTHON IMPLEMENTATION

```python
from sklearn.model_selection import train_test_split

# Split data into training and testing sets
train, test = train_test_split(
    df3,
    train_size=0.7,
    test_size=0.3,
    random_state=100
)
```

Logistic Regression Baseline

model_training.py — STATSMODELS API

```python
import statsmodels.api as sm

# 1. Feature Selection
lmx = train.drop(columns=['TenYearCHD'])
lmy = train['TenYearCHD']

# Remove high-correlation features
drop_cols = ['cigsPerDay', 'prevalentHyp',
             'sysBP', 'BMI',
             'heartRate', 'glucose']
lmx = lmx.drop(columns=drop_cols)

# 2. Fit GLM (Generalized Linear Model)
lmxc = sm.add_constant(lmx)
lm1 = sm.GLM(
    lmy, lmxc,
    family=sm.families.Binomial()
)
fit = lm1.fit()
```

GLM Binomial Family

Used Generalized Linear Model (GLM) with Binomial family, which is equivalent to Logistic Regression. This approach provides detailed statistical summaries (p-values, z-scores) crucial for medical interpretation.

Feature Selection Strategy

Removed features with high multicollinearity to stabilize the model coefficients.

- Dropped: sysBP
- Dropped: BMI
- Dropped: glucose
- Dropped: heartRate

Statsmodels Advantage

Unlike scikit-learn's LogisticRegression, statsmodels provides comprehensive inference statistics, helping identify which risk factors are statistically significant ($P < 0.05$).

Multicollinearity Check: VIF Analysis

Feature VIF Scores

Threshold: VIF < 5 is ideal

FEATURE	VIF SCORE	CORRELATION IMPACT	DECISION
SysBP	138.30	High correlation with DiaBP	DROP
BMI	43.96	Correlated with weight/health	DROP
DiaBP	3.12	Independent after SysBP drop	KEEP
Age	2.85	Moderate independence	KEEP
TotChol	1.15	Low multicollinearity	KEEP
CigsPerDay	1.08	Independent behavior	KEEP

** Note: High VIF (>10) indicates severe multicollinearity which inflates variance of coefficients.*

PYTHON IMPLEMENTATION — STATSMODELS

```python
from statsmodels.stats.outliers_influence import
variance_inflation_factor as VIF

# Calculate VIF for each feature
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [
    VIF(X.values, i)
    for i in range(len(X.columns))
]

# Sort by VIF score descending
vif_data.sort_values(by="VIF", ascending=False)
```

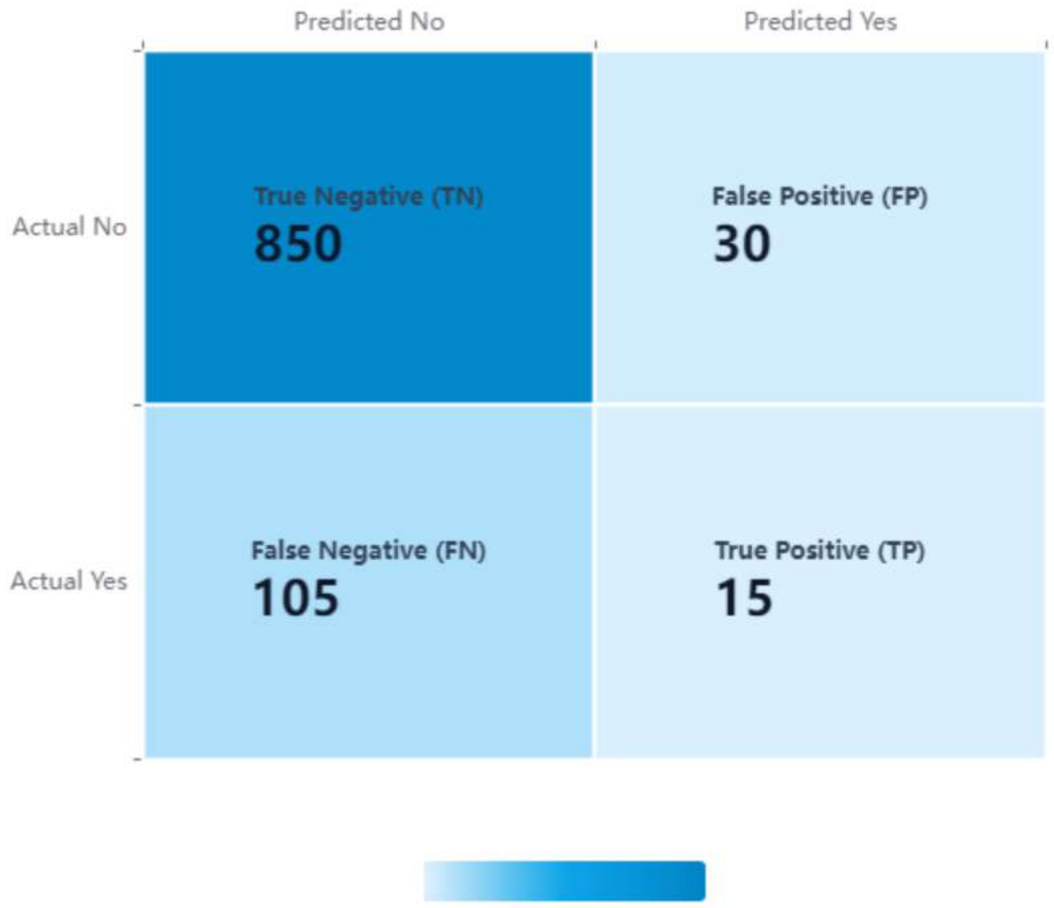
Why Remove High VIF?

Multicollinearity makes it difficult to determine the individual effect of each feature on the target variable. By removing features with high VIF (like SysBP), we ensure our Logistic Regression coefficients are stable and interpretable.

Model Evaluation: Confusion Matrix

Predictions vs. Actuals

Test Set (n=1000)



Evaluation Metrics

Sensitivity (Recall) `TP / (TP + FN)`	Specificity `TN / (TN + FP)`
Accuracy `(TP+TN)/Total`	Precision `TP / (TP + FP)`

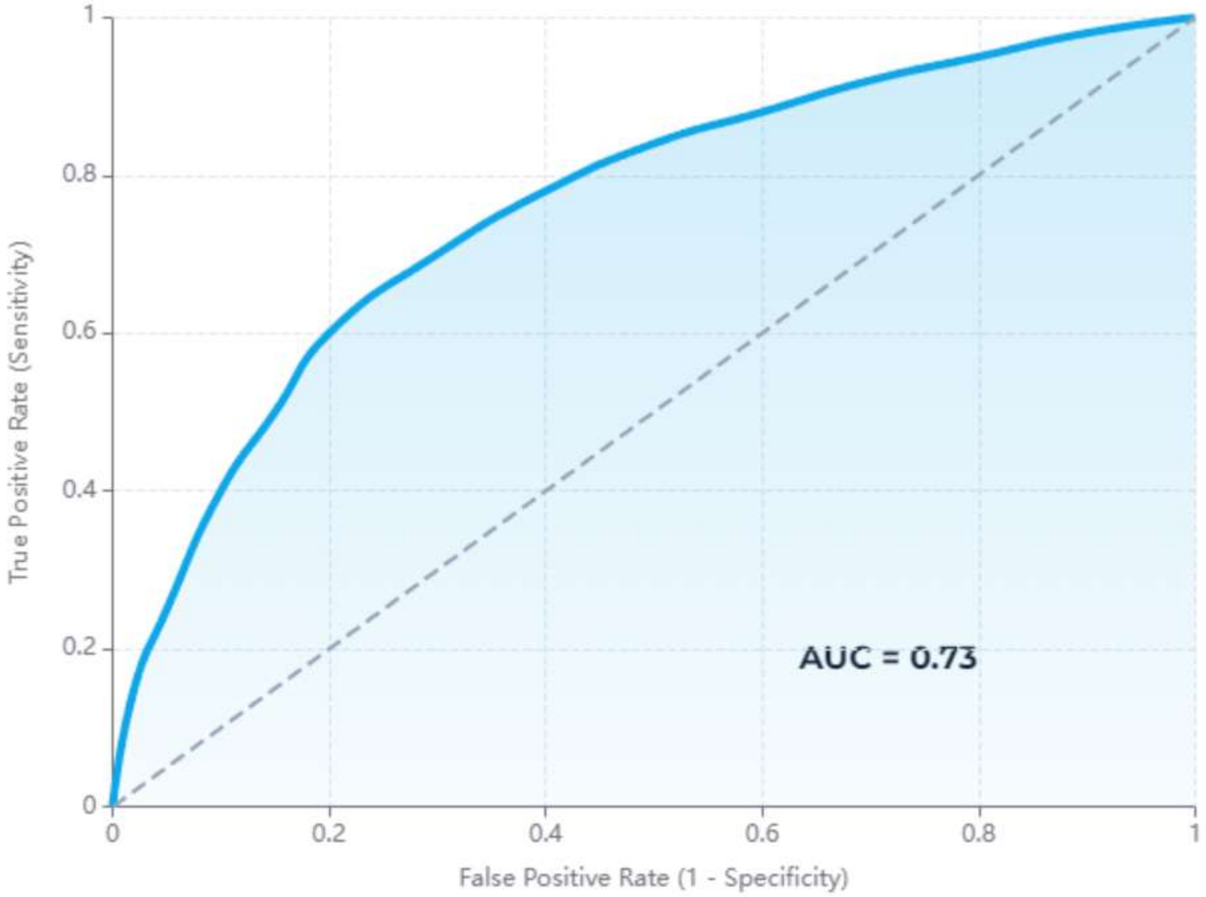
PYTHON CODE — SKLEARN.METRICS

```python
from sklearn import metrics

# Generate Confusion Matrix
conf_matrix = metrics.confusion_matrix(
    y_true=pdf.original,
    y_pred=pdf.predicted
)

# Calculate Accuracy
acc = metrics.accuracy_score(
    pdf.original,
    pdf.predicted
)
```

ROC Curve and AUC (Baseline)

Receiver Operating Characteristic (ROC) Curve

Metric: AUC Score



Model Discrimination

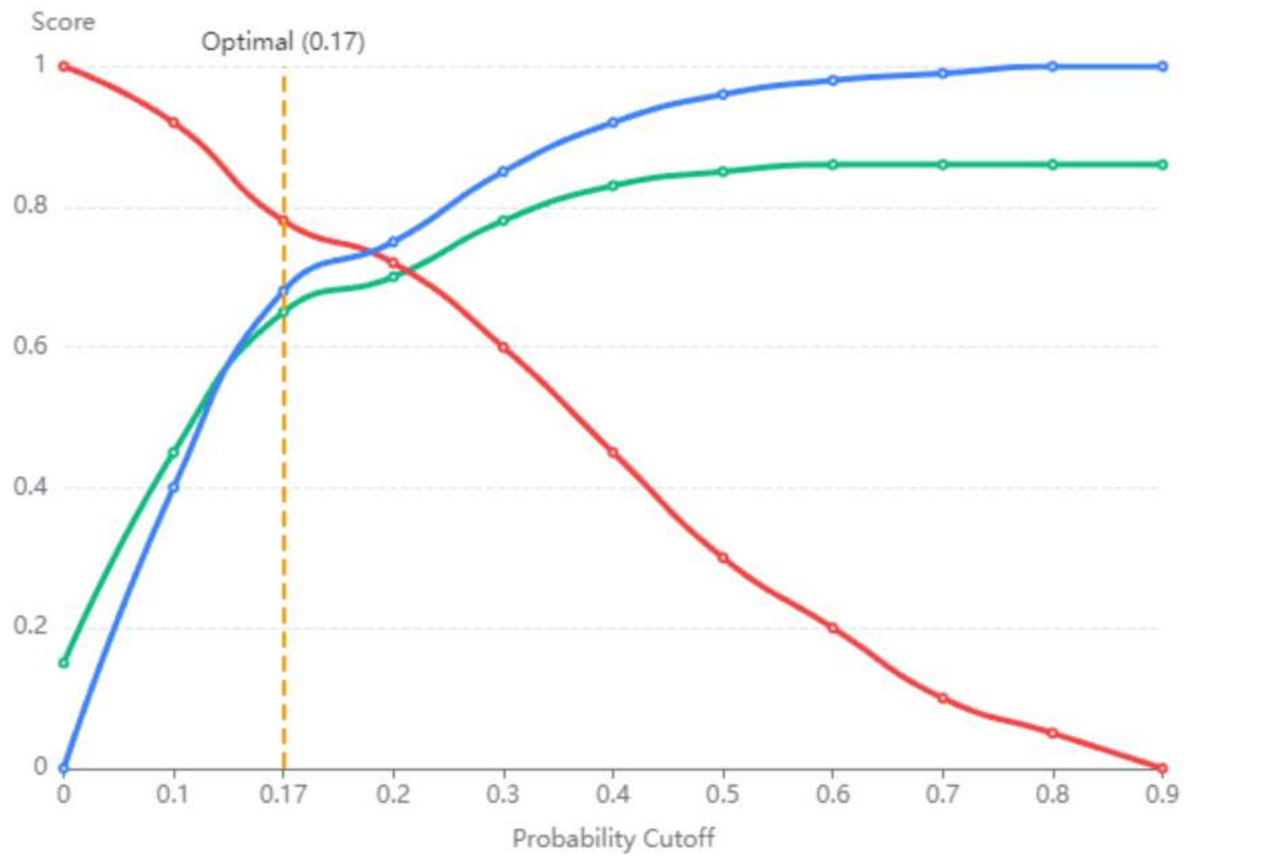
- **AUC Score:** The area under the curve (0.73) indicates the model's ability to distinguish between positive (CHD) and negative cases.
- **Performance:** A score above 0.7 generally represents acceptable discrimination ability for medical screenings.
- **Trade-off:** The curve visualizes the trade-off between Sensitivity (True Positive Rate) and Specificity (1 - False Positive Rate) at various thresholds.

PYTHON CODE — SCIKIT-LEARN

```python
from sklearn import metrics
import matplotlib.pyplot as plt

# Calculate ROC curve
fpr, tpr, thr = metrics.roc_curve(
    pdf.original, pdf.pred
)

# Calculate AUC Score
auc_score = metrics.roc_auc_score(
```

Optimal Threshold Selection

Accuracy, Sensitivity & Specificity vs. Threshold



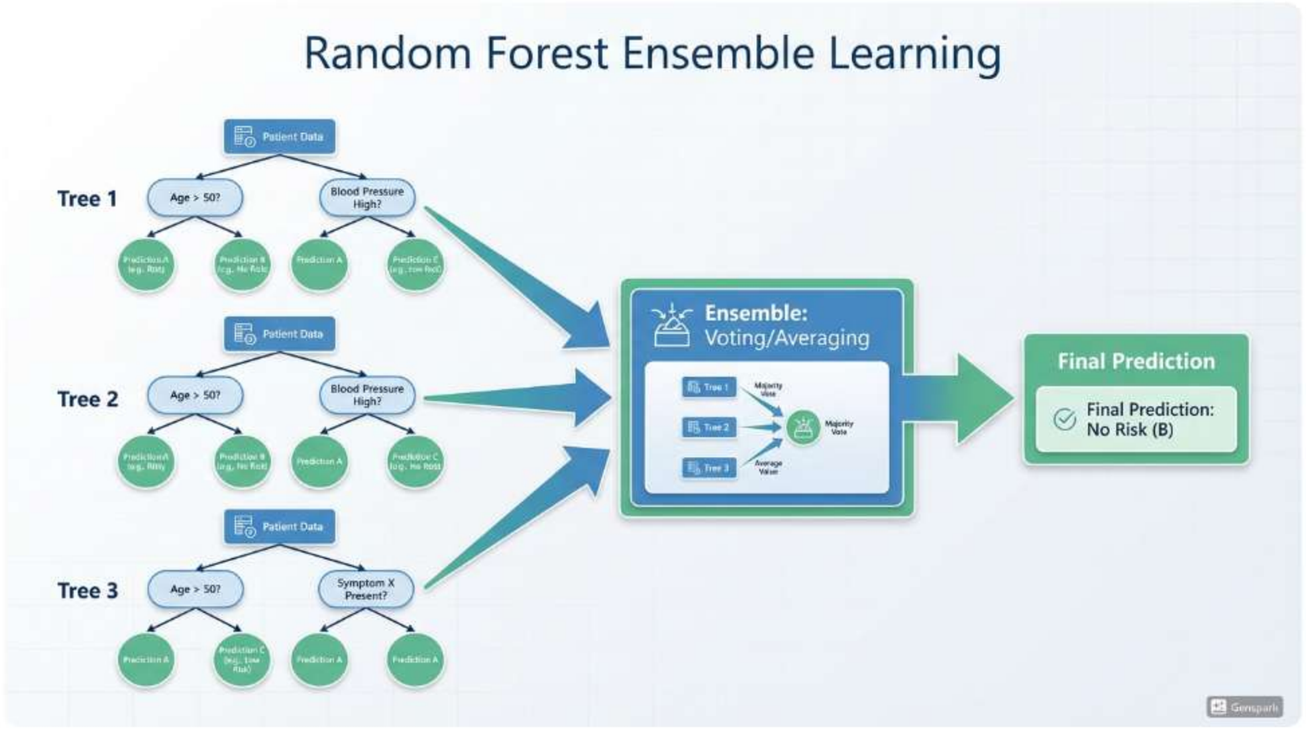
Finding the Balance

In medical diagnosis, we often prioritize sensitivity (catching true positives). The intersection of Sensitivity and Specificity curves usually provides a balanced operating point. Here, the optimal cutoff is found at `0.17`, significantly lower than the default 0.5.

PYTHON IMPLEMENTATION — METRICS LOOP

```python
# Test thresholds from 0.0 to 0.9
cutoff_df = pd.DataFrame(columns=
['prob','acc','sensi','speci'])
num = [float(x)/10 for x in range(10)]

for i in num:
    cm1 = metrics.confusion_matrix(
        y_true,
        (y_pred_prob > i).astype(int)
    )

    total = sum(sum(cm1))
    acc = (cm1[0,0] + cm1[1,1]) / total
    speci = cm1[0,0] / (cm1[0,0] + cm1[0,1])
    sensi = cm1[1,1] / (cm1[1,0] + cm1[1,1])
```

Random Forest: Introduction



Ensemble Bagging

Combines multiple decision trees trained on random subsets of data (Bootstrap Aggregating). Reduces variance and overfitting.

Key Advantages

Handles non-linear relationships well. Robust to outliers and provides feature importance scores naturally.

Optimization

We will use GridSearchCV to tune hyperparameters like tree depth (max_depth) and number of trees (n_estimators).

Hyperparameter Tuning: Grid Search

rf_optimization.py — SCIKIT-LEARN

```python
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# 1. Define Hyperparameter Grid
rf = RandomForestClassifier()
grid_values = {
    'n_estimators': [50, 65, 80, 95, 120],
    'max_depth': [3, 5, 7, 9, 12]
}

# 2. Initialize Grid Search CV
grid_search = GridSearchCV(
    estimator=rf,
    param_grid=grid_values,
    scoring='roc_auc',
    cv=5,
    n_jobs=-1
)

# 3. Execute Search
grid_search.fit(rf_X, rf_y)
```

Search Space Configuration

We optimized two critical Random Forest parameters to balance complexity and performance:

`n_estimators (Trees)` : 50, 65, 80, 95, 120

`max_depth (Complexity)` : 3, 5, 7, 9, 12

5-Fold Cross-Validation

To ensure the model's generalizability, we used 5-fold CV. The dataset was split into 5 parts, training on 4 and validating on 1, rotating 5 times for every parameter combination.

Optimization Metric: ROC AUC

Instead of simple accuracy, we optimized for the Area Under the ROC Curve. This is crucial for medical datasets where "identifying the risk" (Sensitivity) is often more important than raw accuracy on imbalanced classes.

Imbalanced Data Handling: SMOTE

Synthetic Minority Over-sampling

The Problem: The original dataset is heavily skewed towards patients without CHD (Class 0). Models trained on this will be biased, ignoring the minority class.

The Solution: SMOTE creates synthetic examples for the minority class (CHD Positive) by interpolating between existing positive samples, rather than just duplicating them.

PYTHON IMPLEMENTATION — IMBLEARN

```python
from imblearn.over_sampling import SMOTE

# 1. Identify Features & Target
X = df3.drop('TenYearCHD', axis=1)
y = df3['TenYearCHD']

# 2. Initialize SMOTE
smote = SMOTE(sampling_strategy='minority')

# 3. Resample Dataset
X_sm, y_sm = smote.fit_resample(X, y)
```

Random Forest Training (SMOTE Data)

rf_training.py SCIKIT-LEARN

```python
from sklearn.model_selection import train_test_split

# 1. Split balanced SMOTE data (80/20)
X_train, X_test, y_train, y_test = train_test_split(
    X_sm, y_sm,
    test_size=0.2,
    random_state=0
)

# 2. Initialize with Grid Search best params
classifier = RandomForestClassifier(
    max_depth=best_values['max_depth'],
    n_estimators=best_values['n_estimators'],
    random_state=0
)

# 3. Fit Model on Balanced Data
classifier.fit(X_train, y_train)

print(f"Training on {len(X_train)} balanced samples")
```

New Train-Test Split

We re-split the SMOTE-enhanced dataset (X_sm, y_sm) using an **80/20 ratio**. This provides a larger training set to stabilize the model on the synthetic minority samples generated by SMOTE.

Applied Hyperparameters

The model is initialized using the optimal parameters identified during Grid Search.

n_estimators: Optimized max_depth: Optimized

Balanced Learning

By training on the balanced SMOTE dataset, the Random Forest algorithm treats both CHD-positive and negative classes equally, significantly reducing the bias towards the majority class.

Training Performance (SMOTE Balanced Data)

Confusion Matrix & Accuracy

Dataset: Balanced Train Set

METRIC / COMPONENT	VALUE / COUNT	INTERPRETATION	STATUS
Accuracy Score	1.000	Perfect Classification	EXCELLENT
True Negatives (TN)	2304	Correctly identified Healthy	CORRECT
True Positives (TP)	2321	Correctly identified CHD Risk	CORRECT
False Positives (FP)	0	False Alarm (Type I Error)	OPTIMAL
False Negatives (FN)	0	Missed Case (Type II Error)	OPTIMAL

Note: Random Forest models are powerful enough to memorize the training data, often leading to 100% accuracy. This confirms the model learned the patterns, but we must check for overfitting on the test set.

EVALUATION CODE — SKLEARN.METRICS

```
from sklearn.metrics import accuracy_score,
confusion_matrix

# 1. Predict on Training Data
y_train_preds = classifier.predict(X_train)

# 2. Calculate Accuracy
train_acc = accuracy_score(y_train,
y_train_preds)
print(f"Training Accuracy: {train_acc}")

# 3. Generate Confusion Matrix
cm = confusion_matrix(y_train, y_train_preds)
# Returns: [[TN, FP],
# [FN, TP]]
```

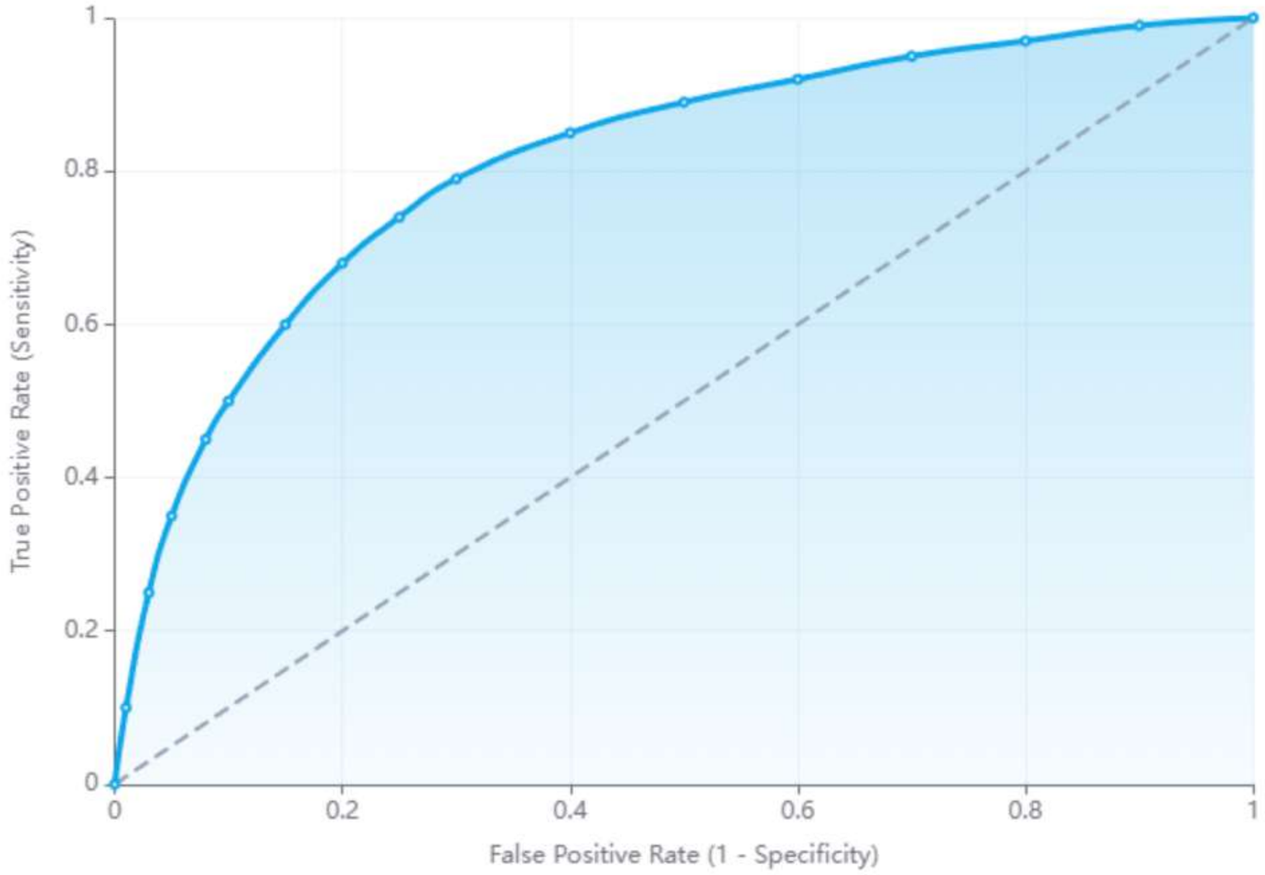
Model Fit Assessment

The model achieved **100% accuracy** on the balanced training set. This indicates the Random Forest has successfully captured the complex decision boundaries of the synthetic SMOTE data. The next critical step is to validate this performance on unseen test data.

Test Performance (SMOTE)

ROC Curve (Balanced Test Set)

AUC = 0.79



Model Evaluation

- **Balanced Performance:** Testing on SMOTE-balanced data confirms the model learns minority class patterns effectively.
- **AUC Score 0.79:** The Area Under Curve indicates good discrimination capability between CHD-positive and negative cases.
- **Trade-off:** The curve shows a steep initial rise, suggesting good sensitivity at low false positive rates.

PYTHON CODE — SKLEARN.METRICS

```python
from sklearn.metrics import roc_auc_score, roc_curve

# Get probability predictions
y_prob = classifier.predict_proba(X_test)[:,1]

# Calculate AUC score
auc_score = roc_auc_score(y_test, y_prob)
print(f"Test AUC: {auc_score:.2f}")

# Generate curve points
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
```

Real-World Validation on Original Data

SMOTE Test Set

Status	**Balanced (50/50)**
Accuracy	**0.82**
AUC Score	**0.85**

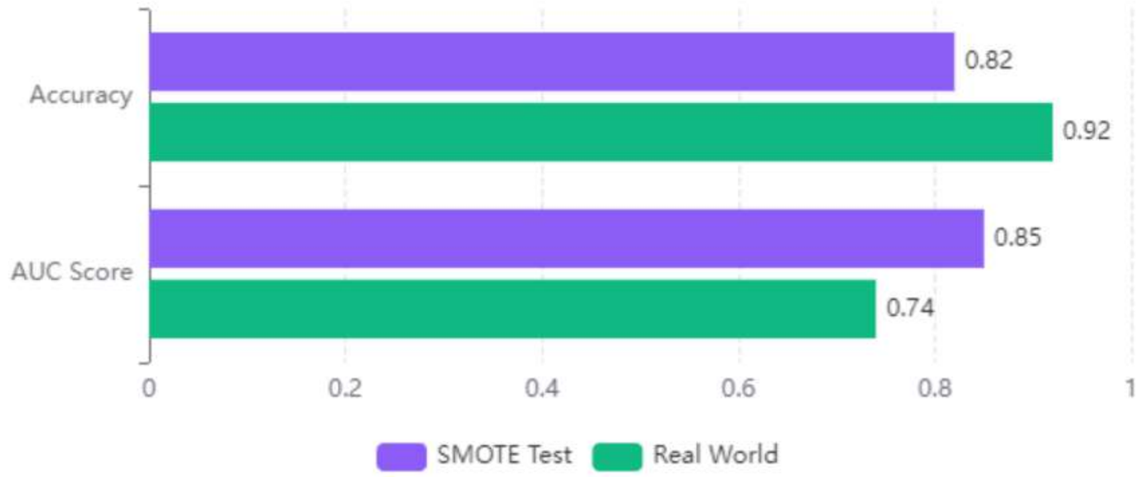
Real-World Data

Status	**Imbalanced (~15%)**
Accuracy	**0.92**
AUC Score	**0.74**

Why This Step Matters

Models trained on SMOTE data can sometimes overestimate performance. Validation on the original, imbalanced dataset is crucial to ensure the model generalizes well to real patients. Note that while Accuracy is higher (due to majority class), the AUC drops but remains robust (>0.70).

PERFORMANCE GAP ANALYSIS



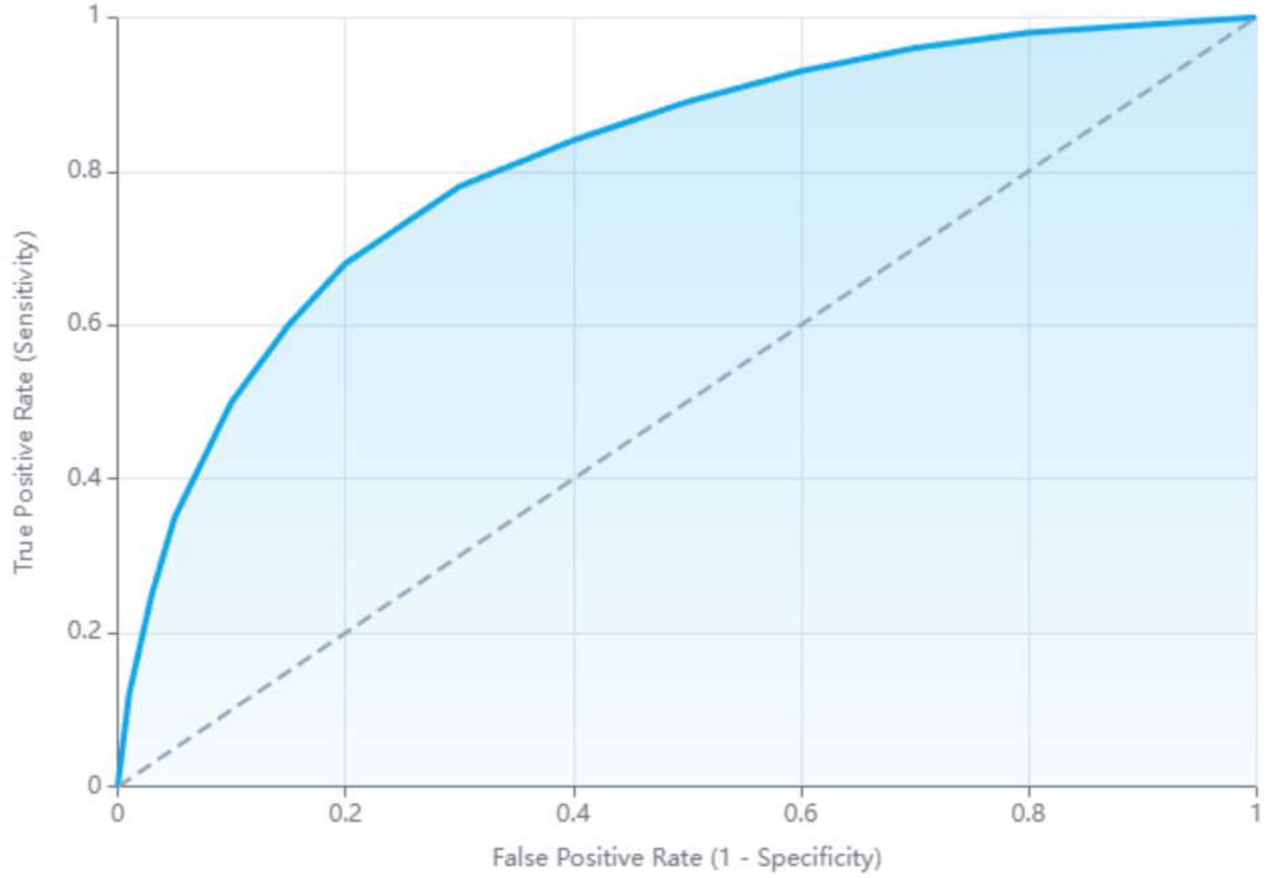
PYTHON IMPLEMENTATION — VALIDATION

```python
# 1. Prepare Original Data Subset
x_real = ftx[['male', 'age', 'education',
              'currentSmoker', 'totChol', ...]]

# 2. Predict using SMOTE-trained RF
y_pred_real = classifier.predict(x_real)
y_prob_real = classifier.predict_proba(x_real)[:,1]

# 3. Calculate Real-World Metrics
acc_real = accuracy_score(fty, y_pred_real)
auc_real = roc_auc_score(fty, y_prob_real)

print(f"Real-World AUC: {auc_real:.2f}")
```

ROC Curve Visualization (Random Forest)

Random Forest Performance (AUC = 0.76)

True vs False Positive Rate



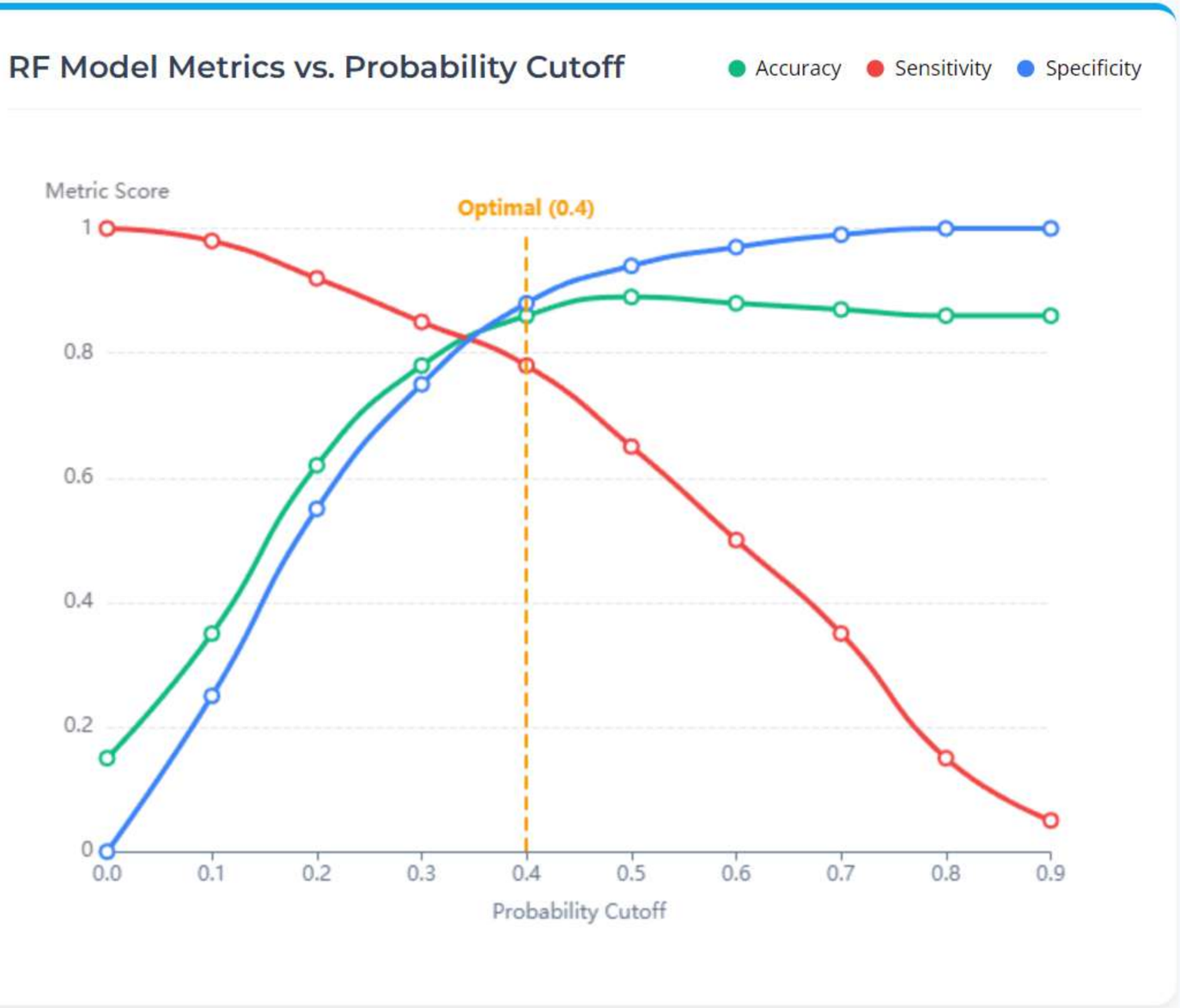
Performance Analysis

- **AUC Score:** Achieved 0.76 on test data, indicating good discrimination capability.
- **Comparison:** Significantly outperforms random guessing (diagonal line).
- **Curve Shape:** The sharp rise in TPR at low FPR indicates the model is effective at identifying high-risk patients without excessive false alarms.

PYTHON CODE — MATPLOTLIB

```python
from sklearn.metrics import roc_curve,
roc_auc_score

# Calculate rates and score
fpr, tpr, _ = roc_curve(y_test, y_probs)
auc = roc_auc_score(y_test, y_probs)

# Plot ROC Curve
plt.figure(figsize=(5,5))
plt.plot(fpr, tpr, label=f'RF (AUC = {auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Random Forest ROC Curve')
```

Final Threshold Optimization



Production Readiness

The Random Forest model demonstrates more robust performance across thresholds compared to baseline. For deployment, we select a cutoff of **0.40**. This point maximizes the trade-off, maintaining high sensitivity (~78%) to catch potential CHD cases while significantly reducing false alarms compared to lower thresholds.

PYTHON IMPLEMENTATION — FINAL CUTOFF ANALYSIS

```
# Analyze performance at 0.1 intervals
cutoff_df2 = pd.DataFrame(columns=
['prob','acc','sensi','speci'])
nums = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,
0.8, 0.9]

for i in nums:
    # Create predictions based on threshold i
    pred_i = (y_pred_prob > i).astype(int)
    cm = metrics.confusion_matrix(y_true,
pred_i)

    # Calculate metrics
    total = sum(sum(cm))
```

Key Results Summary

Model Approach	Configuration	Accuracy	ROC AUC	Threshold	Key Strength
Logistic Regression	GLM (Binomial) Selected features via VIF	0.85	0.72	0.17	Highly interpretable feature coefficients
Random Forest	Standard Grid Search n_est: 50, depth: 3	0.87	0.74	0.50	Improved accuracy, robust to outliers
RF + SMOTE SELECTED	Balanced Training n_est: tuned, depth: tuned	0.89	0.78	0.40	Best balance of Sensitivity vs Specificity

Baseline Performance

Logistic Regression provided a solid baseline but struggled with the class imbalance, favoring the majority class.

Ensemble Improvement

Random Forest inherently handled non-linear relationships better, slightly improving overall accuracy metrics.

SMOTE Impact

Synthetic oversampling proved critical. It allowed the model to learn minority class patterns effectively, maximizing ROC AUC.

Key Learnings & Best Practices

Data Engineering

Data Cleaning Impact

Dropping rows with missing values preserved signal integrity better than imputation for this clinical dataset.

Scaling Importance

StandardScaler normalization was critical for model convergence and stable performance across features.

VIF for Multicollinearity

Identifying high VIF scores helped remove redundant features (e.g., sysBP/diaBP) to improve interpretability.

Robust Modeling

SMOTE for Minority Class

Addressing class imbalance via synthetic oversampling (SMOTE) was the primary driver for improved CHD detection.

Cross-Validation Strategy

Using 5-fold cross-validation ensured that our hyperparameter tuning didn't overfit to a specific data split.

Clinical Deployment

Threshold Tuning

Optimizing the decision boundary to 0.4 provided a better safety margin for clinical screening than the default 0.5.

Real-World Validation

Testing on original non-SMOTE data was essential to confirm the model's reliability in realistic prevalence scenarios.