THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Project Report
Heart Disease Prediction

Course Code: UML501

Course Name: Machine Learning

Department: Computer Science and Engineering Department

Submitted By:

10221[redacted]
Rachit Sinha
3CS[redacted]

CONTENT

1. **Introduction**
2. **Dataset Overview**
3. **Data Preprocessing**
    - Handling Missing Values
    - Feature Engineering
    - Data Normalisation
4. **Machine Learning Models**
    - Logistic Regression
    - Decision Tree
    - Random Forest
5. **Evaluation Metrics**
    - Accuracy
    - Precision
    - Recall
    - F1-Score
    - Confusion Matrix
6. **Python Code**
7. **Output and Analysis**
8. **Conclusion**
9. **References**

Introduction

The ability to predict heart disease is one of the most important applications of data science and machine learning in healthcare. As coronary heart disease (CHD) continues to be one of the leading causes of death worldwide, predictive models offer an opportunity for early intervention and targeted treatment. In this project, we explore the **Framingham Heart Study dataset**, a well-known dataset used to predict the likelihood of developing heart disease within the next ten years.

The goal of this project is to preprocess the data, train various machine learning models to predict coronary heart disease and compare their performances. Given the importance of heart disease prediction, this model has the potential to assist healthcare professionals in identifying high-risk individuals and providing them with preventative care.

Dataset Overview

The dataset consists of various predictors that influence heart disease, such as:

- **Age**: Represents the individual's age in years; older individuals are at higher risk of heart disease due to cumulative physiological changes.
- **Gender (male)**: Indicates whether the individual is male (1) or female (0); men have a higher early risk of heart disease.
- **Education**: Categorical variable denoting the individual's education level; removed due to minimal correlation with the target variable.
- **Current Smoker**: Binary variable (1 for smokers, 0 for non-smokers); smoking increases the risk of coronary heart disease.
- **Cigarettes per Day (cigsPerDay)**: Represents the average number of cigarettes smoked daily by smokers; higher consumption elevates CHD risk.
- **BPMeds**: Binary variable indicating whether the individual is on blood pressure medication (1 for yes, 0 for no); helps manage hypertension.
- **Prevalent Stroke**: Binary variable (1 for history of stroke, 0 for none); prior strokes indicate higher cardiovascular risk.
- **Prevalent Hypertension**: Binary variable (1 for hypertension, 0 for normal); high blood pressure is a critical heart disease risk factor.
- **Diabetes**: Binary variable (1 for diabetes, 0 for non-diabetic); diabetes significantly increases heart disease risk.

- **Total Cholesterol (totChol)**: Indicates the total cholesterol level in mg/dL; high cholesterol is a direct risk factor for CHD.
- **Systolic Blood Pressure (sysBP)**: Represents systolic blood pressure in mmHg; elevated systolic BP is a predictor of cardiovascular events.
- **Diastolic Blood Pressure (diaBP)**: Represents diastolic blood pressure in mmHg; elevated diastolic BP is linked to increased cardiovascular risk.
- **Body Mass Index (BMI)**: Body weight in kg/m^2; higher BMI values are associated with obesity and related heart disease risks.
- **Heart Rate (heartRate)**: Indicates resting heart rate in beats per minute; an elevated heart rate may signal underlying health issues.
- **Glucose**: Indicates glucose level in mg/dL; high glucose levels reflect poor metabolic health and increased CHD risk.
- **TenYearCHD**: Binary target variable (1 for CHD within ten years, 0 otherwise); used to predict heart disease outcomes.

The dataset consists of 16 columns (features), with over 4,000 entries of individuals tracked for a period of time.

Data Preprocessing

Data preprocessing is one of the most crucial steps in any machine learning project. It helps transform raw data into a format that is suitable for training machine learning models. In this project, several preprocessing steps were carried out, including handling missing data, feature engineering, and normalization.

Handling Missing Values

Real-world datasets often contain missing or incomplete data. Handling missing values correctly is essential to avoid introducing bias or inaccuracies into the model. In this project, we employed various strategies for handling missing data:

- **Dropping Irrelevant Features:** The feature education was removed because it did not exhibit a meaningful relationship with the target variable, TenYearCHD.
- **Identifying Missing Values:** To visualize the missing data distribution, a **heatmap** was created to visually confirm the extent of missingness. This helped in choosing the most appropriate imputation strategy.
- **Filling Missing Values:**

    1. **Mean Imputation:** For features like heartRate, where there was only one missing value, we used mean imputation, filling the missing value with the mean of the existing data.
    2. **Median Imputation:** Features like BMI, totChol, and BPMeds had small amounts of missing data. We opted to impute these missing values using the median, which is less sensitive to outliers compared to mean imputation.

    For filling these values, we used **fillna()**, which allows us to replace missing entries with specific values such as the mean, median, or a predefined constant.

    3. **KNN Imputation:** For the glucose feature, which had numerous missing values, we used K-Nearest Neighbours (KNN) imputation. This approach fills missing values based on the similarity of other features. The KNN method uses surrounding data points (neighbours) to estimate missing values, making it a suitable choice when missingness is not completely random.

    KNN (K-Nearest Neighbours): is a simple, non-parametric algorithm used for classification and regression tasks. It works by identifying the "k" nearest data points to a given point and making predictions based on their values.

1. **Distance Metric**: KNN calculates the distance between points, typically using Euclidean distance, but other distance metrics (like Manhattan or Minkowski) can also be used.
2. **K (Neighbours)**: The "k" in KNN refers to the number of nearest neighbours to consider when making a prediction. For classification, the most common class among the k neighbours is chosen. For regression, the average (or weighted average) of the k neighbours' values is used.
3. **Lazy Learning**: KNN is a "lazy learner," meaning it doesn't require a training phase. Instead, it stores the entire training dataset and makes predictions on the fly, which can be computationally expensive for large datasets.

Here, we used the features sysBP, diaBP, totChol to estimate the missing values in the glucose column. These features were selected because they are likely correlated with glucose. We took n_neighbors=5, meaning the imputation was based on the 5 nearest neighbours to the missing value.

Feature Engineering

Feature engineering is the process of creating new features or modifying existing ones to improve the model's performance. Several steps were taken to prepare the features for machine learning models:

- **Normalization**: Features like blood pressure (sysBP, diaBP), cholesterol levels (totChol), and glucose levels have different scales. Normalizing the data is important because machine learning algorithms, particularly those based on distance metrics, can be sensitive to the scale of the data.
  For standardizing the dataset, we used the **StandardScaler()** from the **sklearn.preprocessing library**. This scaler transforms the features to have a mean of 0 and a standard deviation of 1, making them standardized. This process is essential for ensuring that features with different scales do not dominate the model training.
- **Exploratory Data Analysis (EDA)**: Before training any models, a significant amount of EDA was performed to understand the relationships between various features and the target variable. For instance, we explored the correlation between systolic blood pressure (sysBP) and cholesterol (totChol). Although some features exhibited expected relationships (e.g., age and TenYearCHD), others had more complex or weak correlations. The insights gained during EDA helped inform decisions about data preprocessing and imputation strategies.
  For exploring relation between sysBP and totChol, we used **lmplot()** function. lmplot is a powerful visualization function in the Python library **Seaborn** that is

used to create scatter plots with an optional linear regression model fit. It combines the ability to draw scatter plots and regression lines in a single step, making it useful for exploring relationships between two variables.

Linear Regression: Linear regression is a statistical method used to model the relationship between a dependent variable (target) and one or more independent variables (predictors). It assumes that this relationship is linear, meaning the change in the dependent variable is proportional to changes in the independent variable(s).
It involves one independent variable and one dependent variable, modelled as:

$$y = mx + b$$

where:
y: Dependent variable
x: Independent variable
m: Slope (rate of change)
b: Intercept (value of y when x=0)

Machine Learning Models

In this project, we experimented with several machine learning models to predict the likelihood of coronary heart disease. The models chosen for comparison include **Logistic Regression**, **Decision Tree**, and **Random Forest**.

Logistic Regression

Logistic regression is one of the most widely used algorithms for binary classification tasks. It works by estimating the probability that a given input point belongs to a particular class (in this case, heart disease or not). The logistic regression function outputs a probability between 0 and 1, which is then thresholded (usually at 0.5) to make a classification decision.

Logistic regression uses the **logistic function** (also called the sigmoid function) to map any real-valued number to a probability. The logistic function is given by:

$$P(y = 1|X) = \frac{1}{(1 + e^{-(\beta 0 + \beta 1X1 + \beta 2X2 + \ldots + \beta nXn)})}$$

where $P(y = 1 \mid X)$ is the probability that the individual has heart disease (Class1), and $X_1, X_2, \ldots, X_n$ are the input features.

Key Concepts of Logistic Regression

1. **Logistic Function**:
    - The logistic function maps the linear combination of input features to a probability value between 0 and 1. In multiple logistic regression, this function is used with multiple predictors $(X^1, X^2, \ldots, X_k)$.
2. **Multiple Independent Variables**:
    - Multiple logistic regression allows for several independent variables $(X^1, X^2, \ldots, X_k)$, which can be either continuous or categorical. The model estimates the contribution of each variable to the target variable.
3. **Log-Odds**:
    - MLR works by modeling the log-odds of the target variable, where log-odds is the logarithmic transformation of the odds ratio of the event occurring.

4. **Maximum Likelihood Estimation (MLE)**:
    - The parameters $(\beta^0, \beta^1, \dots, \beta_k)$ of the logistic regression model are estimated using **maximum likelihood estimation (MLE)**, which maximizes the likelihood of observing the given data based on the model parameters.
5. **Multicollinearity**:
    - In multiple logistic regression, multicollinearity refers to the correlation between independent variables. High multicollinearity can lead to unreliable coefficient estimates and is often addressed using techniques like **variance inflation factor (VIF)** to detect and address correlation among predictors.

Implementation: The model was trained using the **LogisticRegression**() from the sklearn.linear_model module. It was evaluated using accuracy, precision, recall, and F1-score.

Decision Tree

A **Decision Tree** is a non-linear model that recursively splits the dataset based on feature values, creating a tree-like structure where each node represents a decision rule based on one of the features.

The structure of a Decision Tree resembles a tree, where:

- **Root Node**: This is the starting point of the tree where the first split occurs based on a feature. It contains the entire dataset before any decisions are made.
- **Decision Nodes**: These are the intermediate nodes where the data is further split based on the value of another feature. Each node represents a decision based on a feature's value.
- **Leaf Nodes**: These are the final nodes that contain the predicted output. For classification tasks, these represent class labels, while for regression, they represent continuous values.

Key Concepts of Decision Trees

1. **Splitting**:
    - At each node, the algorithm selects the feature that best splits the data into distinct groups based on an impurity measure. Common impurity measures are:

- **Gini Impurity**
  The formula for **Gini Impurity** is:
  $$G = 1 - \Sigma(p_i^2)$$
  Where p_i is the proportion of instances belonging to class i. A lower Gini value indicates a better split.

- **Entropy**
  The formula for **Entropy** is:
  $$H = -\Sigma(p_i \times log^2(p_i))$$
  Where p_i is the probability of class i at a node. Lower entropy indicates less uncertainty, leading to better splits.

2. **Stopping Criteria**:
   - A **Decision Tree** stops growing when it meets a predefined stopping condition, such as:
     - The tree reaches a maximum depth.
     - The number of samples in a node falls below a minimum threshold.
     - All data points in a node belong to the same class (pure node).

Implementation: The model was trained using the **DecisionTreeClassifier()** class from the sklearn.tree module. It was evaluated using accuracy, precision, recall, and F1-score.

Random Forest

A **Random Forest (RF)** is an ensemble learning technique that combines multiple **Decision Trees** to produce a more robust and accurate model. It operates by constructing a collection (or "forest") of decision trees, typically trained using random subsets of the training data and random selections of features. Each tree in the forest is built independently, and their predictions are combined to make the final decision.

Key Concepts of Random Forest

1. **Ensemble Learning**:
   - **Random Forest** is based on the concept of ensemble learning, where multiple models (decision trees, in this case) are trained independently and their predictions are aggregated to improve accuracy and reduce variance. This is in contrast to a single decision tree, which may overfit to the training data.

2. **Bagging (Bootstrap Aggregating)**:
    - **Bagging** is a technique used in **Random Forest** where multiple subsets of the training data are randomly sampled with replacement. This means that each tree is trained on a different subset of the data, which introduces diversity in the trees and helps to reduce overfitting.
3. **Random Feature Selection**:
    - In **Random Forest**, at each split in a decision tree, only a random subset of features is considered. This randomness prevents individual trees from becoming too similar and overfitting the data. It improves the model's performance by increasing diversity among the trees.
4. **Majority Voting for Classification**:
    - For classification tasks, **Random Forest** uses a majority voting scheme, where each individual tree in the forest provides a class prediction, and the class with the most votes across all trees is selected as the final prediction.

      Majority Voting Formula:

      $$\hat{Y} = majority\ vote\big(T_1(X), T_2(X), \dots, T_k(X)\big)$$

      Where:

        - $T_i(X)$ is the prediction of the i_{th} tree.
        - k is the total number of trees in the forest.
5. **Averaging for Regression**:
    - For regression tasks, **Random Forest** takes the average of the predictions from all individual trees to make the final prediction.

Implementation: The model was trained using the **RandomForestClassifier()** class from the sklearn.ensemble module. It was evaluated using accuracy, precision, recall, and F1-score.

Evaluation Parameters

Evaluation is critical in understanding how well a machine learning model performs, especially when predicting outcomes like heart disease. In this project, several key evaluation metrics were used to assess the performance of the classification models. These metrics provide insights into the model's ability to make accurate predictions, particularly with imbalanced datasets like this one. Below is an in-depth discussion of the evaluation parameters used:

Accuracy

Accuracy is one of the most commonly used evaluation metrics in classification tasks. It is the ratio of correctly predicted instances to the total instances in the dataset. The formula for accuracy is:

Accuracy = (True Positives + True Negatives) / Total Instances

Where:

- **True Positives (TP)**: Instances where the model correctly predicted the positive class (e.g., predicting "heart disease" when it was actually present).
- **True Negatives (TN)**: Instances where the model correctly predicted the negative class (e.g., predicting "no heart disease" when it was actually absent).
- **Total Instances**: The total number of samples in the dataset.

While accuracy is a useful metric, it may be misleading when the dataset is imbalanced (for example, when one class is much larger than the other). In these cases, a model could achieve high accuracy by always predicting the majority class, even if it performs poorly for the minority class. Thus, accuracy alone is not always a sufficient measure of performance.

Precision

Precision is the proportion of positive predictions that are actually correct. It measures how many of the instances predicted as positive are truly positive. The formula for precision is:

Precision = True Positives / (True Positives + False Positives)

Where:

- **False Positives (FP)**: Instances where the model incorrectly predicted the positive class (e.g., predicting "heart disease" when it was actually absent).

Precision is important when the cost of false positives is high, such as in medical diagnoses where unnecessary treatments could result from false positive predictions.

Recall

Recall, also known as sensitivity or true positive rate, is the proportion of actual positive instances correctly identified by the model. The formula for recall is:

Recall = True Positives / (True Positives + False Negatives)

Where:

- **False Negatives (FN)**: Instances where the model incorrectly predicted the negative class (e.g., predicting "no heart disease" when it was actually present).

Recall is crucial when missing positive instances has severe consequences, such as failing to detect heart disease, which could be life-threatening.

F1-Score

The **F1-Score** is the harmonic mean of precision and recall, providing a balanced measure that accounts for both false positives and false negatives. The formula for F1-Score is:

F1-Score = 2 * (Precision * Recall) / (Precision + Recall)

The F1-Score ranges from 0 to 1, with 1 indicating perfect precision and recall. A high F1-Score indicates that the model is both precise and capable of correctly identifying most positive instances. The F1-Score is particularly useful in cases of class imbalance, where it balances the trade-off between precision and recall.

Confusion Matrix

The **confusion matrix** is a tool used to evaluate the performance of a classification model by displaying the actual versus predicted classifications. It shows four key components:

1. **True Positives (TP)**: Instances where the model correctly predicted the positive class.
2. **True Negatives (TN)**: Instances where the model correctly predicted the negative class.
3. **False Positives (FP)**: Instances where the model incorrectly predicted the positive class.

4. **False Negatives (FN)**: Instances where the model incorrectly predicted the negative class.

The confusion matrix can be visualized as follows:

	Predicted: 0	**Predicted: 1**
Actual: 0	TN	FP
Actual: 1	FN	TP

From this matrix, we can derive various metrics:

- **Accuracy** = (TP + TN) / (TP + TN + FP + FN)
- **Precision** = TP / (TP + FP)
- **Recall** = TP / (TP + FN)
- **Specificity (True Negative Rate)** = TN / (TN + FP)

The confusion matrix helps identify the types of errors the model is making, which can guide further improvements.

Python Code:

```
import pandas as pd
from sklearn import preprocessing
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np


#dataset
disease_df = pd.read_csv("framingham.csv")


#preprocessing
#dropping column with no relation to TenYearCHD
disease_df.drop(['education'], inplace = True, axis = 1)


#analyzing missing values
print(disease_df.isnull().sum())


plt.figure(figsize=(10, 6))
sns.heatmap(disease_df.isnull(), cbar=False, cmap='viridis',
yticklabels=False)
plt.title("Missing Data Heatmap")
plt.show()


#filling missing values
#heartrate has just 1 missing value so it can be filled with simple
mean
disease_df['heartRate'].fillna(disease_df['heartRate'].mean(),
inplace=True)


#BMI has very few missing values and can be easily filled using
median of all BMI
disease_df['BMI'].fillna(disease_df['BMI'].median(), inplace=True)
```

```
#smokers also have a low amount of missing values and thus can be filled by median easily

#but we need to ignore non-smokers here to accurately identify smoking patterns of actual smokers (assuming all missing values belong to smokers)

smokers_median = disease_df[disease_df['cigsPerDay'] > 0]['cigsPerDay'].median()

disease_df['cigsPerDay'].fillna(smokers_median, inplace=True)


#there could be a close relation b/w sysBP and totChol as both are related to each other and it may help us in filling the values


sns.lmplot(x='sysBP', y='totChol', data=disease_df, ci=None, aspect=1.5)

plt.title('Linear Trend Between sysBP and totChol')

plt.xlabel('Systolic Blood Pressure (sysBP)')

plt.ylabel('Total Cholesterol (totChol)')

plt.show()


#the relation is not very close so we will not use it


#filling missing totchol and BPMeds values by simple median values


disease_df['totChol'].fillna(disease_df['totChol'].median(), inplace=True)

disease_df['BPMeds'].fillna(disease_df['BPMeds'].median(), inplace=True)


#glucose has a lot of missing values so simple mean or median will give inaccurate results, KNN is a good alternative as it uses relations with sBP, dBP, Chol, diab to fill glucose values which is much more accurate

from sklearn.preprocessing import MinMaxScaler


#create a new dataframe for imputation
```

```
features = ['sysBP', 'diaBP', 'totChol']
knn_df = disease_df[features + ['glucose']].copy()


#initialize a scaler
scaler = MinMaxScaler()


#normalize the selected features
knn_scaled = scaler.fit_transform(knn_df)


from sklearn.impute import KNNImputer


#initialize the KNN imputer with 5 neighbors
imputer = KNNImputer(n_neighbors=5)


#perform the imputation
knn_imputed = imputer.fit_transform(knn_scaled)


#transform back to the original scale
knn_df_imputed = pd.DataFrame(scaler.inverse_transform(knn_imputed),
columns=features + ['glucose'])


#replace the 'glucose' column in the original dataframe with the
imputed values
disease_df['glucose'] = knn_df_imputed['glucose']


#verifying all missing values are filled
print(disease_df.isnull().sum())


#seperating Features and Target
X = np.asarray(disease_df[['age',
'male','currentSmoker','BPMeds','prevalentStroke','prevalentHyp','di
abetes','diaBP','BMI','heartRate', 'cigsPerDay', 'totChol', 'sysBP',
'glucose']])
```

```
y = np.asarray(disease_df['TenYearCHD'])

#normalization of the dataset
X = preprocessing.StandardScaler().fit(X).transform(X)

#train and Test Split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size = 0.3)

#counting no. of patients affected with CHD
plt.figure(figsize=(7, 5))
sns.countplot(x='TenYearCHD', data=disease_df,
              palette="BuGn_r")
plt.show()

#training a Logistic Regression Model
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred_lr = logreg.predict(X_test)

#evaluation and accuracy
from sklearn.metrics import accuracy_score
print('Accuracy of Logistic Regression model is =',
      accuracy_score(y_test, y_pred_lr))

#confusion matrix
from sklearn.metrics import confusion_matrix, classification_report

cm_lr = confusion_matrix(y_test, y_pred_lr)
conf_matrix_lr = pd.DataFrame(data = cm_lr,
```

```
                            columns = ['Predicted:0', 'Predicted:1'],
                            index =['Actual:0', 'Actual:1'])

plt.figure(figsize = (8, 5))
sns.heatmap(conf_matrix_lr, annot = True, fmt = 'd', cmap = "Reds")
plt.title("Confusion Matrix - Logistic Regression")
plt.show()
print('The details for confusion matrix is =')
report_lr=classification_report(y_test, y_pred_lr, output_dict=True)
print(classification_report(y_test, y_pred_lr))

#training a Decision Tree Model
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)

#evaluation and accuracy
print('Accuracy of Decision Tree model is =',
      accuracy_score(y_test, y_pred_dt))

#confusion Matrix for Decision Tree

cm_dt = confusion_matrix(y_test, y_pred_dt)
conf_matrix_dt = pd.DataFrame(data=cm_dt,
                              columns=['Predicted:0',
'Predicted:1'],
                              index=['Actual:0', 'Actual:1'])

plt.figure(figsize=(8, 5))
sns.heatmap(conf_matrix_dt, annot=True, fmt='d', cmap="Blues")
plt.title("Confusion Matrix - Decision Tree")
```

```
plt.show()

print('The details for confusion matrix are:')
report_dt=classification_report(y_test, y_pred_dt, output_dict=True)
print(classification_report(y_test, y_pred_dt))

#training a Random Forest Model
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

#evaluation and accuracy
print('Accuracy of Random Forest model is =',
      accuracy_score(y_test, y_pred_rf))

#confusion Matrix for Random Forest
cm_rf = confusion_matrix(y_test, y_pred_rf)
conf_matrix_rf = pd.DataFrame(data=cm_rf,
                              columns=['Predicted:0',
'Predicted:1'],
                              index=['Actual:0', 'Actual:1'])

plt.figure(figsize=(8, 5))
sns.heatmap(conf_matrix_rf, annot=True, fmt='d', cmap="Greens")
plt.title("Confusion Matrix - Random Forest")
plt.show()

print('The details for confusion matrix are:')
report_rf=classification_report(y_test, y_pred_rf, output_dict=True)
print(classification_report(y_test, y_pred_rf))
```

```
#compare all the models
comparison = pd.DataFrame({
    "Model": ["Logistic Regression", "Decision Tree", "Random
Forest"],
    "Accuracy": [
        report_lr["accuracy"],
        report_dt["accuracy"],
        report_rf["accuracy"]
    ],
    "Precision (Class 1)": [
        report_lr["1"]["precision"],
        report_dt["1"]["precision"],
        report_rf["1"]["precision"]
    ],
    "Recall (Class 1)": [
        report_lr["1"]["recall"],
        report_dt["1"]["recall"],
        report_rf["1"]["recall"]
    ],
    "F1-Score (Class 1)": [
        report_lr["1"]["f1-score"],
        report_dt["1"]["f1-score"],
        report_rf["1"]["f1-score"]
    ]
})

#display comparison table
print("Comparison of Models:")
print(comparison)
```

Output and Analysis

The three models — **Logistic Regression**, **Decision Tree**, and **Random Forest** — were trained and evaluated to predict heart disease, represented as **Class 1** (positive class). The evaluation metrics include accuracy, precision, recall, and F1-score. The results of the evaluation are summarized in the following table:

Model	Accuracy	Precision (Class 1)	Recall (Class 1)	F1-Score (Class 1)
Logistic Regression	0.849843	0.625000	0.076142	0.135747
Decision Tree	0.750000	0.183246	0.177665	0.180412
Random Forest	0.844340	0.481481	0.065990	0.116071

Analysis of Model Performance

1. **Logistic Regression:**
    - **Accuracy**: Logistic Regression achieves a high accuracy of **84.98%**, which is the best among the three models. This suggests that the model correctly classifies a large portion of both positive and negative instances.
    - **Precision (Class 1)**: The precision of **62.5%** is the highest among all models, indicating that when Logistic Regression predicts heart disease, it is correct a good portion of the time.
    - **Recall (Class 1)**: However, the **recall** of **7.61%** is quite low, meaning the model misses a significant number of actual positive cases. The low recall could be attributed to the model being more conservative in predicting heart disease, as it prioritizes accuracy over identifying as many positives as possible. This leads to fewer false positives but more false negatives, which results in a lower recall.
    - **F1-Score**: The low **F1-score** of **0.136** reflects the imbalance between precision and recall. While the precision is relatively high, the low recall indicates that the model is not very effective at identifying heart disease cases.

2. **Decision Tree:**
    - **Accuracy**: The Decision Tree model has a lower accuracy of **75%** compared to Logistic Regression. This suggests that the Decision Tree may be overfitting to the training data, as it tends to create overly complex decision boundaries, leading to lower generalization to unseen data.
    - **Precision (Class 1)**: The **precision** of **18.32%** is the lowest among all models. This indicates that the Decision Tree makes many false positive predictions — predicting heart disease when it is not actually present. This is common in decision trees, especially when they are not well-pruned and overfit to the training data.
    - **Recall (Class 1)**: The **recall** of **17.77%** is slightly higher than Logistic Regression but still low. The Decision Tree is more likely to predict positive cases of heart disease than Logistic Regression, but it does so with many false positives, which affects its precision.
    - **F1-Score**: The **F1-score** of **0.18** is still low, reflecting the imbalance between precision and recall. Despite having a slightly better recall than Logistic Regression, the poor precision leads to a low overall F1-score.

3. **Random Forest:**
    - **Accuracy**: The **Random Forest** model achieves an accuracy of **84.43%**, which is similar to Logistic Regression. This suggests that the Random Forest model is effective in making overall predictions, but it doesn't significantly outperform Logistic Regression.
    - **Precision (Class 1)**: The **precision** of **48.15%** is better than Decision Tree but still lower than Logistic Regression. This means that when Random Forest predicts heart disease, it is correct about half the time. However, this still indicates that the model makes a notable number of false positive predictions.
    - **Recall (Class 1)**: The **recall** of **6.60%** is the lowest among the three models. Despite its good accuracy and reasonable precision, the Random Forest model struggles to correctly identify positive heart disease cases. This suggests that Random Forest may be underfitting the positive class or not capturing the relevant patterns in the data as effectively as the other models.
    - **F1-Score**: The **F1-score** of **0.116** reflects the poor trade-off between precision and recall. Although the accuracy is high, the model struggles

with identifying true positive cases of heart disease, resulting in a low F1-score.

Conclusion

1. **Logistic Regression**: Logistic Regression outperforms the other models in terms of accuracy and precision. The low recall and F1-score, however, indicate that while the model is good at predicting the absence of heart disease, it fails to identify many of the positive cases. This suggests that the model is prioritizing precision over recall. This may be acceptable in situations where minimizing false positives is more important, but it is not ideal for medical applications where detecting every positive case is crucial.
2. **Decision Tree**: The Decision Tree model shows the highest recall, meaning it identifies a larger proportion of actual heart disease cases. However, its low precision indicates that it also predicts many false positives. The overfitting issue may be a contributing factor, as Decision Trees tend to model the data too closely, capturing noise in the training set, which leads to poor generalization on unseen data.
3. **Random Forest**: The Random Forest model performs similarly to Logistic Regression in terms of accuracy, but its poor recall suggests that it is not as effective at identifying heart disease cases. Its poor recall can be attributed to class imbalance in the dataset, where negative cases (no heart disease) dominate. This causes the model to predict the negative class more often, leading to a lower recall for positive cases (heart disease).

References

- https://www.geeksforgeeks.org/ml-heart-disease-prediction-using-logistic-regression/
- https://www.kaggle.com/code/lauriandwu/machine-learning-heart-disease-framingham
- https://www.geeksforgeeks.org/data-preprocessing-machine-learning-python/
- https://en.wikipedia.org/wiki/Multinomial_logistic_regression
- https://scikit-learn.org/1.5/modules/generated/sklearn.linear_model.LogisticRegression.html
- https://www.geeksforgeeks.org/decision-tree/
- https://www.geeksforgeeks.org/random-forest-classifier-using-scikit-learn/
- https://www.geeksforgeeks.org/confusion-matrix-machine-learning/