1. Introduction

Ischemic Heart Disease (IHD) constitutes the primary cause of global mortality, accounting for 13% of all fatalities globally. Since 2000, mortality has markedly risen, increasing by 2.7 million to attain 9.1 million deaths in 2021. Diabetes has become a leading cause of death, showing a 95% rise since 2000. **Ischemic Heart Disease (IHD)** continues to be the leading cause of death from noncommunicable diseases (19.7%), yet its global age-standardized mortality rate has declined by 30.8% over the past thirty years, with the most substantial decreases noted in high-**Sudan Development Index (SDI)** regions. Mortality rates were elevated in individuals over 60 years, with a little male predominance.

In 2019, metabolic risks were the primary factors, including smoking, air pollution, and an inadequate diet characterized by insufficient intake of whole grains and legumes, as well as high sodium consumption. Alterations to one's lifestyle are essential for maintaining advancement. Bangladesh reported 108,528 fatalities due to **Coronary Heart Disease (CHD)** in 2020, accounting for 15.16% of total deaths, ranking the country 118th globally. **Diabetes Mellitus** accounted for 28,976 fatalities (4.05% of total deaths), positioning Bangladesh at 98 globally.

Cardiovascular Disease is the primary cause of mortality in individuals. with **type 2 Diabetes**, characterized by a significant incidence of myocardial ischemia. To develop an algorithm that utilizes clinical and laboratory criteria to categorize type 2 diabetes patients into low, medium, or high-risk classifications for the identification of **Myocardial Ischemia**, an algorithm was created and verified for major adverse cardiac events in patients with **type 2 Diabetes and Cardiovascular Disease** using clinical variables. The model demonstrated moderate discrimination. and strong calibration, facilitating risk classification for targeted therapy.

According to the **World Health Organization (WHO)**, **Cardiovascular Disease** poses a significant risk to global populations, resulting from various factors such as **Hypertension, Obesity, Hyperlipidemia, Tobacco** use, poor dietary practices, **Diabetes**, and **Arrhythmias**. Therefore, it is essential to employ effective algorithms for disease classification and prediction to understand disease forecasting.

Recent advancements in healthcare technology have enabled the creation of **Machine Learning (ML)** algorithms for predicting human health disorders. **Machine Learning (ML)** techniques aim to produce computer code capable of accessing and employing existing data to forecast future data. Techniques for enhancing model accuracy include augmenting the dataset, correcting absent and anomalous values, **Selecting Features, Optimizing Algorithms,** conducting **Cross-Validation, and Implementing Ensembling Techniques**. This research uses **Grid SearchCV** for **Hyperparameter Optimization** and **Five-Fold Cross-Validation** to assess the model's performance on benchmark datasets.

This research delineates the subsequent notable contributions:

- This study evaluates and applies fourteen distinct machine learning algorithms to heart disease datasets, assessing performance classification measures. Initially, a range of machine learning classifier techniques were employed, including **Logistic Regression**

(LR), k-Nearest Neighbors (kNN), Naive Bayes (NB), Decision Tree (DT), Support Vector Machine (SVM), Ridge Classifier (RC), Random Forest (RF), Quadratic Discriminant Analysis (QDA), AdaBoost, Gradient Boosting (GB), Linear Discriminant Analysis (LDA), Extra Trees Classifier (ETC), Classifier Chain(CC),DecisionForest(DF).

- The **GridsearchCV** hyperparameter tuning technique utilized five-fold cross-validation and evaluated performance through accuracy and negative log loss measures to attain optimal accuracy.
- In the end, the **Gradient Boosting (GB)** technique enhances the model's accuracy.

2. Literature Review

The utilization of **Machine Learning (ML)** and optimization methodologies for identifying **Ischemic Heart Disease (IHD)** in diabetic individuals has garnered considerable interest in recent years. Research has demonstrated the capability of machine learning models to enhance diagnostic precision and forecasting. **Lee et al. (2022)** illustrated the efficacy of models such **as Random Forest** and **Gradient Boosting** in forecasting **Acute Myocardial Infarction (AMI)** and symptomatic **Ischemic Heart Disease (IHD)**, utilizing clinical and demographic data, including diabetes prevalence. These **Machine Learning** models surpassed conventional approaches, offering enhanced accuracy in risk categorization for diabetic groups [27]. Zhou et al. (2023) utilized **Deep Learning (DL)** methodologies to forecast ischemic heart disease in diabetic individuals, integrating clinical factors like age, gender, and comorbidities such as hypertension. Their results showed how important demographic and clinical factors are for making predictions more accurate, especially for groups with a high risk [28]. In **Bangladesh**, where diabetes and cardiovascular illnesses are widespread, the utilization of **Machine Learning** models presents significant potential. **Hossain et al. (2021)** investigated the application of **Support Vector Machine (SVM)** and Logistic **Regression (LR)** for diagnosing Ischemic Heart Disease (IHD) in diabetic patients in Bangladesh, establishing a significant correlation between diabetes and heightened **Ischemic Heart Disease (IHD)** risk, thereby emphasizing the necessity of predictive models for early diagnosis in the regional context [29]

Optimization strategies, such as the **Synthetic Minority Over-sampling Technique (SMOTE)**, have been utilized to enhance the performance of these models. **Sharma et al. (2022)** employed **SMOTE** to rectify class imbalance in cardiovascular datasets, specifically within diabetic populations. This strategy enhanced the precision and recall of predictions by ensuring that rare but critical instances were sufficiently represented in the training data, hence improving the model's capacity to predict IHD in diabetic populations [30]. Hypertension, a common complication in diabetic individuals, impairs the diagnosis and progression of ischemic heart disease (IHD). The Faculty of Medicine (2022) states that the coexistence of hypertension and diabetes accelerates atherosclerosis, exacerbates plaque accumulation, and diminishes coronary microvascular function, hence complicating the diagnosis of ischemic heart disease (IHD). The Egyptian Journal of Hospital Medicine (2022) underscored the heightened mortality risk linked to this combination, stressing the necessity for early detection and prompt care in diabetes patients [31].

The **Revista Española de Cardiología (2021)** elucidated hypertension's contribution to the exacerbation of Ischemic Heart Disease (IHD) in diabetic patients, identifying it as a critical element in endothelial dysfunction and vascular remodeling, both of which facilitate IHD. The European Society of Cardiology (2023) emphasized the significance of controlling hypertension in diabetic patients to mitigate the risk of ischemic heart disease, highlighting the synergistic impact of hypertension and diabetes on elevating cardiovascular mortality. The American Heart Association (2023) highlighted the twofold cardiovascular risk associated with these illnesses and the necessity for adequate care [32]. Recent research, such as that conducted by **Jose et al. (2024)**, has investigated the incorporation of lifestyle modifications, including enhanced nutrition and increased physical activity, into the treatment of hypertension and diabetes, demonstrating

potential in mitigating the risk of ischemic heart disease (IHD). Integrating these lifestyle modifications with machine learning algorithms in personalized therapies should enhance the detection and management of IHD, hence improving long-term cardiovascular health outcomes in diabetic populations [33]. This research highlights the essential importance of machine learning, optimization methods, and lifestyle modifications in enhancing the diagnosis, treatment, and prevention of ischemic heart disease in high-risk diabetic populations, especially in areas such as Bangladesh.

3. Methods

3.1 Study Design and Data Collection

This study used Machine Learning Techniques and Optimization in a cross-sectional design to predict Ischemic Heart Disease (IHD) in diabetic individuals in Bangladesh. A total of 322 diabetics from Dhaka cities in Bangladesh participated in this cross-sectional study. Structured questionnaires, physical examinations, and laboratory testing were used to gather data from BIHS General Hospital, located at 125/1, Darus Salam, Mirpur-1, Dhaka-1216. The dataset was gathered from clinical records and had 322 patient records with 21 attributes including demographic, clinical, and disease-related features. It was saved in SPSS format (ASDS project_Data.sav).

3.1.1 Variables and Data Description

The dataset comprised the subsequent categories of variables:

1. Demographic Characteristics: Age, gender, profession, educational attainment, socioeconomic situation.
2. Clinical Features: Height, weight, systolic and diastolic blood pressure, random blood glucose readings.
3. Lifestyle Factors: Tobacco use status.

Disease Status: Hypertension (HTN), Diabetes Mellitus (DM), Dyslipidemia, Stroke, and **Ischemic Heart Disease (IHD) (target variable).**

3.1.2 Definition of Target Variable

The dataset comprises patients with **Diabetes Mellitus (DM)**, categorized according to the presence or absence of **Ischemic Heart Disease (IHD)**.

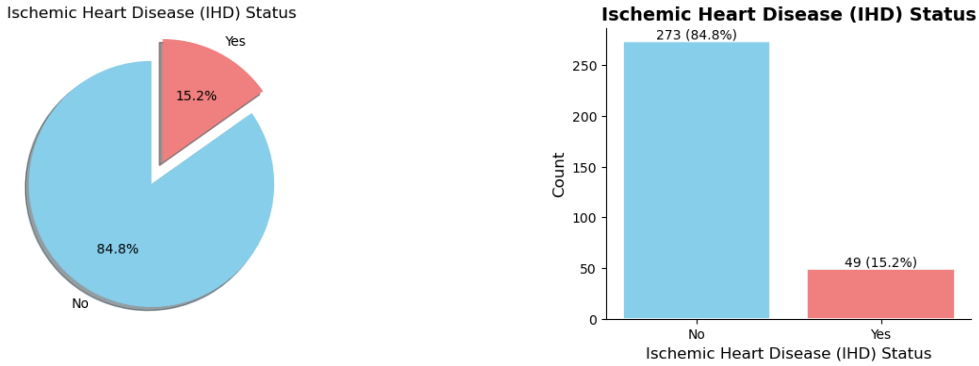


Figure no. 01

The Dataset is imbalanced, with **15.2%** of patients diagnosed with **Ischemic Heart Disease (IHD)** and **84.8%** not diagnosed. To improve performance, consider handling class imbalance using **SMOTE oversampling** or **class-weighting** and validate with **stratified k-fold cross-validation** to ensure robustness. The **target variable** represents this classification:

- **Yes (1):** Patients diagnosed with IHD (**15.2%** of the total data).
- **No (0):** Patients **not** diagnosed with IHD (**84.8%** of the total data).

3.2 Data Preprocessing

Data preprocessing was conducted to improve data quality and ensure compatibility with machine learning algorithms.

3.2.1 Handling Missing Values

The dataset contained only two missing values in Random Blood Sugar (mg/dL), which were imputed using the median value.

3.2.2 Removing Redundant Features and Duplicates

Dropped unnecessary columns: "Serial Number" and "Name" were removed as they did not contribute to prediction.

Checked for duplicate records: No duplicate rows or columns were found.

3.2.4 Feature Scaling

StandardScaler was applied to normalize numerical variables such as age, weight, blood pressure, and blood sugar levels.

3.2.5 SMOTE (Synthetic Minority Over-sampling Technique)

SMOTE is a technique employed to rectify class imbalance in datasets by generating synthetic samples for the minority class. It enhances the representation of the minority class by interpolating between data points and their k-nearest neighbors. This equilibrates the dataset, enhancing the performance of machine learning models, especially in terms of precision and recall. Nonetheless, **SMOTE** may augment the dataset size, resulting in prolonged training durations and potential overfitting if an excessive number of neighbors are selected. **SMOTE** was utilized to enhance the accuracy of forecasts for ischemic heart disease in diabetic patients.

3.2.3 Outlier Detection and Treatment

Outliers were identified using the Interquartile Range (IQR) method and visualized using histograms and scatter plots.

Outliers were treated by:

- **Median imputation** for extreme values in **blood sugar and blood pressure**.
- **Winsorization** applied to blood sugar levels to cap extreme values.

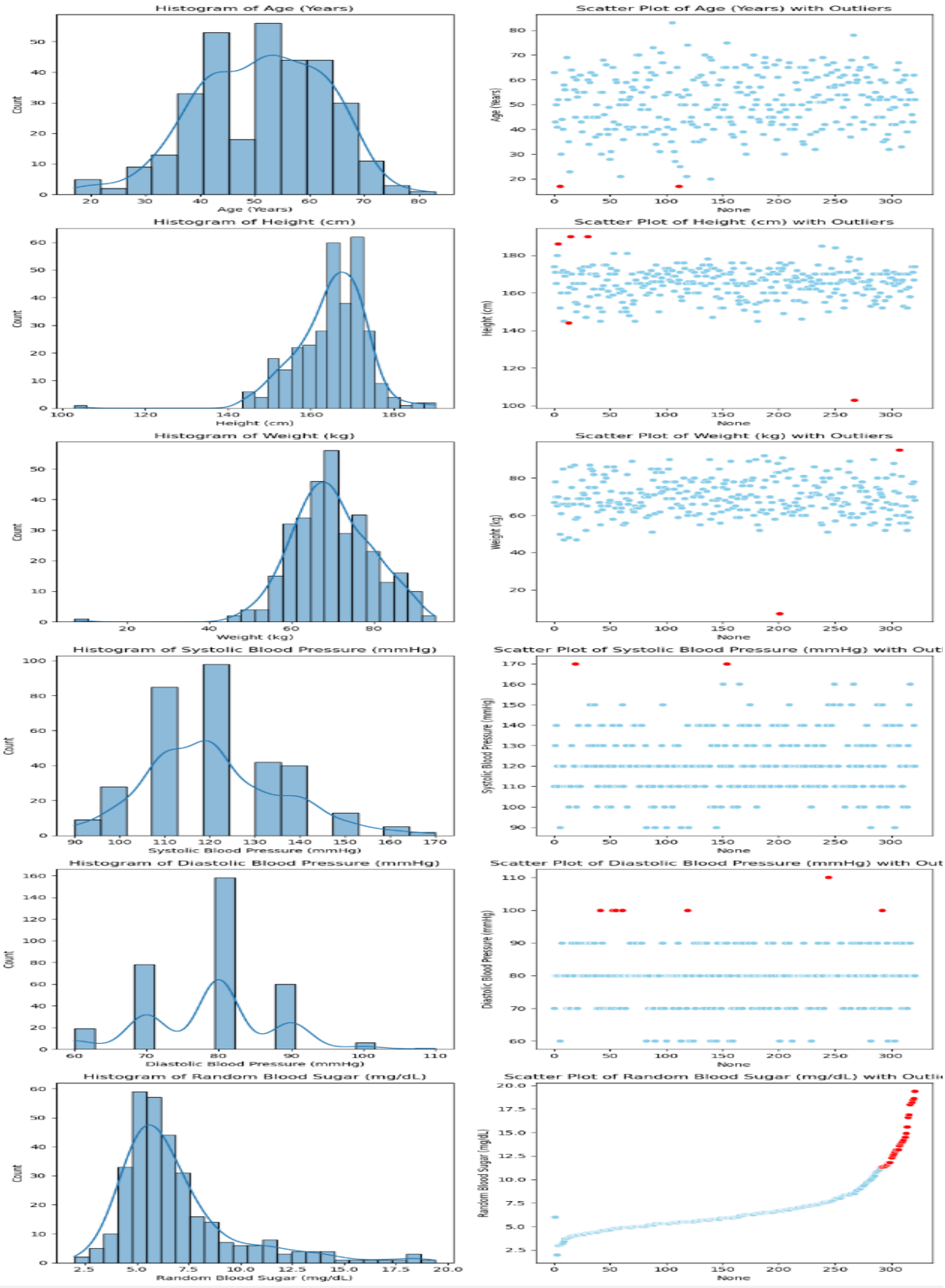

Histogram of Age (Years)
Count
Age (Years)
Scatter Plot of Age (Years) with Outliers
Age (Years)
None
Histogram of Height (cm)
Count
Height (cm)
Scatter Plot of Height (cm) with Outliers
Height (cm)
None
Histogram of Weight (kg)
Count
Weight (kg)
Scatter Plot of Weight (kg) with Outliers
Weight (kg)
None
Histogram of Systolic Blood Pressure (mmHg)
Count
Systolic Blood Pressure (mmHg)
Scatter Plot of Systolic Blood Pressure (mmHg) with Out
Systolic Blood Pressure (mmHg)
None
Histogram of Diastolic Blood Pressure (mmHg)
Count
Diastolic Blood Pressure (mmHg)
Scatter Plot of Diastolic Blood Pressure (mmHg) with Out
Diastolic Blood Pressure (mmHg)
None
Histogram of Random Blood Sugar (mg/dL)
Count
Random Blood Sugar (mg/dL)
Scatter Plot of Random Blood Sugar (mg/dL) with Outli
Random Blood Sugar (mg/dL)
None

Figure no. 02

3.3 Feature Selection

Feature selection was crucial in identifying the most relevant predictors of IHD.

3.3.1 Mutual Information (MI) Score Analysis

The mutual information classification score was used to rank feature importance.

Rank	Feature	Mutual Information
1	Age (Years)	0.090
2	Body Mass Index (BMI) Group_<18.5(underweight)	0.066
3	Stroke Status_Yes	0.052
4	Diabetes Mellitus (DM) Status_Yes	0.044
5	Age Group_61-70	0.043
6	Random Blood Sugar (mg/dL)	0.040
7	Weight (kg)	0.040
8	Hypertension Stage_Stage -1 Hypertension (130-...	0.038
9	Hypertension (HTN) Status_Yes	0.034
10	Occupation Type_Others	0.034
11	Education Level_Masters	0.027
12	Age Group_71-80	0.025
13	Diastolic Blood Pressure (mmHg)	0.025
14	Diabetes Mellitus (DM) Status_Unknown	0.024
15	Economic Status_<25K(Uper Low income)	0.022
16	Dyslipidemia Status_Unknown	0.022
17	Economic Status_>100K((Higher income))	0.021
18	Sex (Male/Female)_Male	0.020
19	Height (cm)	0.019
20	Body Mass Index (BMI) Group_≥27.0(Obese)	0.018

Table:3.2

Evaluate Model:

- **Accuracy:** 0.7667
- **ROC AUC Score:** 0.6522
- **Classification Report:**

	Precision	Recall	F1-score	Support
False	0.83	0.87	0.85	23
True	0.50	0.43	0.46	7
Accuracy			0.77	30
Macro avg.	0.67	0.65	0.66	30
Weighted avg.	0.76	0.77	0.76	30

Table:3.2

- Features with **mutual information threshold ≥ 0.01(or 1%)** were selected for model training.
- The top-ranked relevant features included:

    - Age
    - Systolic & Diastolic Blood Pressure
    - Random Blood Sugar
    - Smoking Status
    - Dyslipidemia
    - Hypertension Status
    - Stroke History

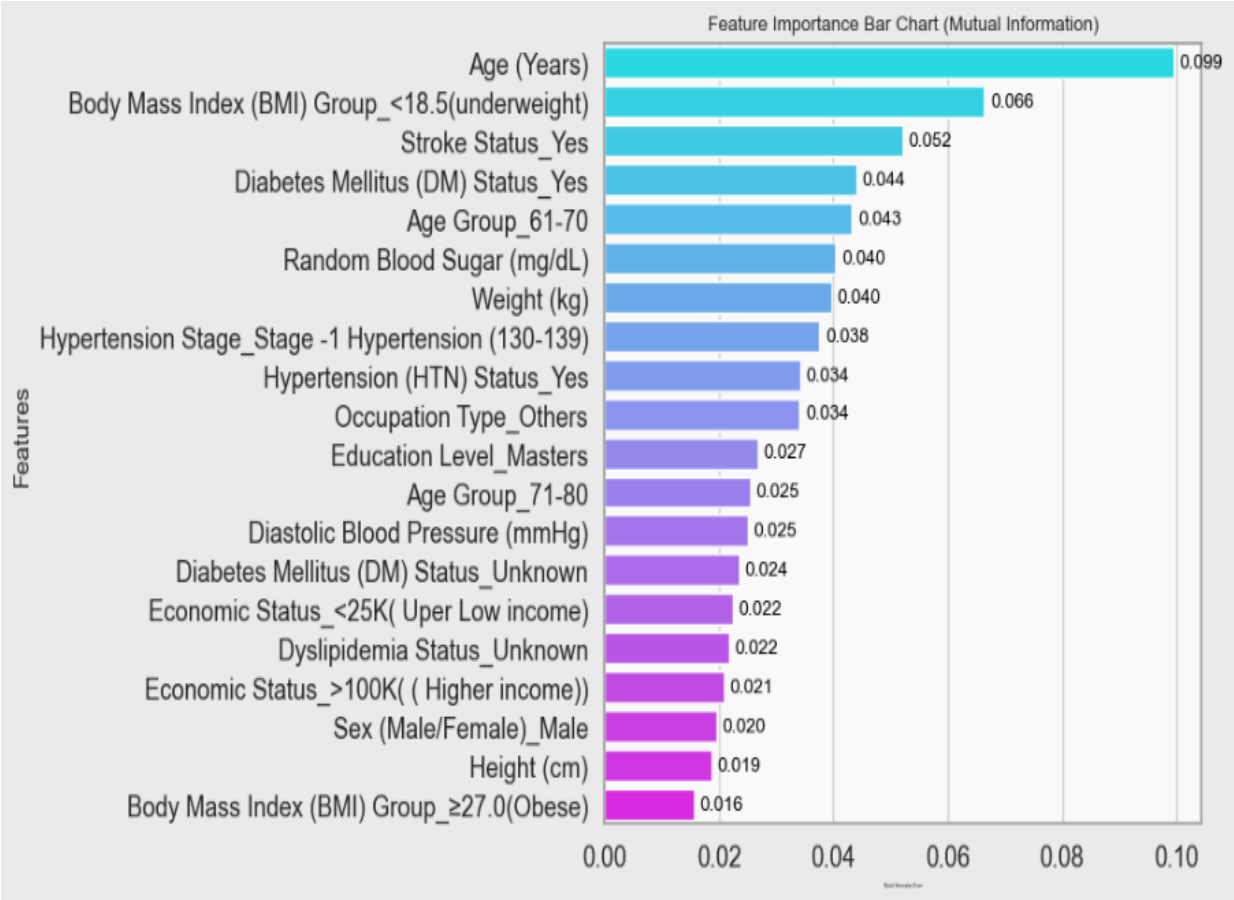


Figure no. 03

3.3.2 Random Forest Feature Importance

- The **Random Forest model classification score** was used to rank feature importance.

Rank	Feature	Mutual Information (%)
1	Age (Years)	10
2	Height (cm)	10

3	Random Blood Sugar (mg/dL)	10
4	Weight (kg)	8.2
5	Systolic Blood Pressure (mmHg)	8.0
6	Stroke Status_Yes	8.0
7	Dyslipidemia Status_Yes	4.7
8	Diastolic Blood Pressure (mmHg)	3.4
9	Hypertension (HTN) Status_No	3.2
10	Hypertension (HTN) Status_Yes	3.1

Table: 3.3

Evaluate Model:

- **Accuracy:** 0.8000
- **ROC AUC Score:** 0.8365
- **Classification Report:**

	Precision	Recall	F1-score	Support
False	0.86	0.92	0.89	6
True	0.00	0.00	0.00	4
Accuracy			0.77	30
Macro avg.	0.43	0.46	0.44	30
Weighted avg.	0.74	0.80	0.77	30

Table: 3.4

Features with **random forest threshold = 0.03** were selected for model training.
The top-ranked relevant features included:

- Age (Years)
- Height (cm)
- Random Blood Sugar (mg/dL)
- Weight (kg)
- Systolic Blood Pressure (mmHg)
- Stroke Status_Yes
- Dyslipidemia Status_Yes
- Diastolic Blood Pressure (mmHg)
- Hypertension (HTN) Status_No
- Hypertension (HTN) Status_Yes

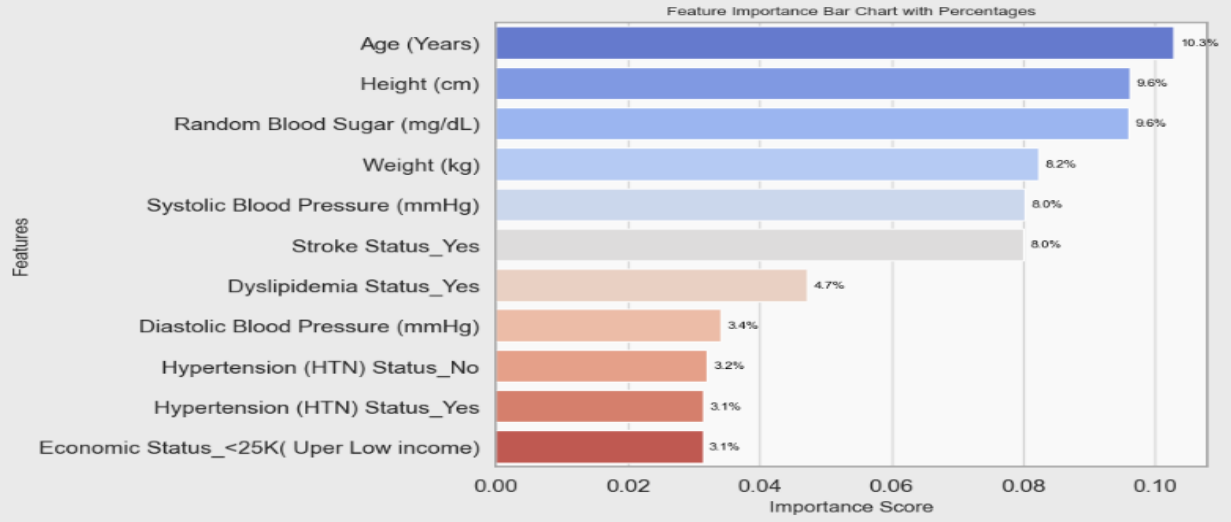


Figure no. 04

The Best Model's Conclusion

Despite **Random Forest Feature Importance**-Based Selection exhibiting superior predictive accuracy (80%) and ROC AUC score (0.8365) compared to Mutual Information-Based Selection, its efficacy for actual prediction is dubious, as it fails to identify any true instances of IHD (Precision & Recall for True = 0.00).

Mutual Information-Based Selection exhibits a recall of 43% and demonstrates superior efficacy in diagnosing IHD cases, despite its lower accuracy (76.67%) and ROC AUC (0.6522).The capacity to identify positive cases, crucial in medical applications, renders Mutual Information-Based Features advantageous for IHD prediction.

Most Relevant Features for IHD Prediction

1. Top Features Based on Mutual Information (MI) :
2. Age (Years)
3. Body Mass Index (BMI) Group_<18.5(underweight)
4. Stroke Status_Yes
5. Diabetes Mellitus (DM) Status_Yes
6. Age Group_61-70
7. Random Blood Sugar (mg/dL)
8. Weight (kg)
9. Hypertension Stage_Stage -1 Hypertension (130-...
10. Hypertension (HTN) Status_Yes
11. Occupation Type_Others
12. Education Level_Masters
13. Age Group_71-80
14. Diastolic Blood Pressure (mmHg)

15. Diabetes Mellitus (DM) Status_Unknown
16. Economic Status_<25K(Uper Low income)
17. Dyslipidemia Status_Unknown
18. Economic Status_>100K((Higher income))
19. Sex (Male/Female)_Male
20. Height (cm)
21. Body Mass Index (BMI) Group_≥27.0(Obese)

Key Observations on Features:

- **Age, Random Blood Sugar, Weight, Stroke Status, and Dyslipidemia** are the most relevant for IHD prediction.
- **Diabetes, Hypertension, and Blood Pressure** Levels also play an important role in IHD risk.
- Economic status and education level appear but are less critical for medical diagnosis.

3.4 Machine Learning Models

To identify the best predictive model, multiple supervised learning algorithms were applied.

3.4.1 Baseline Models

1. Baseline models are essential to gain an initial understanding of the data and set reference performance levels. They are relatively simple, easy to interpret, and computationally inexpensive.
2. **Logistic Regression** – A statistical model that predicts the probability of an event using a linear decision boundary. It is commonly used for binary classification problems.
3. **k-Nearest Neighbors (k-NN)** – A non-parametric model that classifies data points based on the majority class of their nearest neighbors. It works well for small datasets but struggles with large, high-dimensional data.
4. **Naïve Bayes** – A probabilistic classifier based on Bayes' Theorem, assuming feature independence. It is simple, fast, and effective for text classification and medical diagnosis tasks.
5. **Decision Tree** – A tree-based model that splits data into branches based on feature conditions. It is easy to interpret but prone to overfitting without proper pruning.

These models establish a benchmark, helping compare and assess improvements made by more complex machine learning models.

3.4.2 Advanced Model

Advanced models use sophisticated techniques to improve accuracy, handle complex data structures, and enhance generalization. These models often outperform baseline models in predictive tasks.

1. **Support Vector Machine (SVM)** – A powerful classifier that finds the optimal hyperplane to separate classes, effective in high-dimensional spaces.
2. **Ridge Classifier** – A linear model that applies L2 regularization to reduce overfitting and improve generalization.
3. **Random Forest** – An ensemble of decision trees that improves accuracy and reduces overfitting by averaging multiple predictions.

4. **Quadratic Discriminant Analysis (QDA)** – A probabilistic model that extends Linear Discriminant Analysis (LDA) by allowing non-linear class boundaries.
5. **AdaBoost** – An ensemble method that combines weak classifiers iteratively to create a strong classifier, enhancing performance on complex data.
6. **Gradient Boosting** – A boosting technique that builds models sequentially, correcting previous errors and improving predictive accuracy.
7. **Linear Discriminant Analysis (LDA)** – A dimensionality reduction technique that maximizes class separation for better classification.
8. **Extra Trees Classifier** – A variant of Random Forest that introduces more randomness in tree splitting to improve robustness.
9. **Decision Forest** – An ensemble of decision trees that aggregates predictions for better accuracy and stability.

These models are highly effective for complex datasets, offering improved accuracy and robustness over traditional baseline models.

3.5 Hyperparameter Tuning and Ensemble Learning

Fine-tuning model parameters and combining models are integral for extracting maximum performance from machine learning models. This section details our comprehensive approach to both hyperparameter tuning and ensemble strategies.

3.5.1 Hyperparameter Optimization

Hyperparameter Optimization:

Hyperparameter optimization is the process of selecting the best set of hyperparameters for a machine learning model to improve its performance. Unlike model parameters, which are learned from data, hyperparameters are predefined settings that control how the model learns (e.g., learning rate, number of trees in a forest, or regularization strength).

Common techniques for hyperparameter optimization include:

- **Grid Search** – Systematically tests all possible combinations of hyperparameters within a specified range.
- **Random Search** – Randomly selects hyperparameter values and evaluates performance, often faster than grid search.
- **Bayesian Optimization** – Uses probabilistic models to predict the best hyperparameters based on previous evaluations.
- **Genetic Algorithms** – Mimics natural selection to evolve the best hyperparameters over multiple iterations.

3.5.2 Ensemble Learning

Ensemble approaches combine various models to enhance forecast accuracy, stability, and generalizability. Employing techniques such as bagging, boosting, and stacking. Bagging entails generating several training subsets through sampling with replacement from the original dataset, so diminishing volatility and enhancing resilience to noisy input. Boosting successively trains models by emphasizing challenging examples, hence achieving elevated predicting accuracy.

Stacking amalgamates various models through the training of a meta-learner to synthesize their predictions. The stacked ensemble attained superior accuracy, enhanced ROC-AUC scores, and improved calibration relative to any singular base model. Ensemble approaches often require greater computer resources than individual models; nonetheless, parallel processing and optimized implementations are utilized.

3.6 Performance Metrics

The models were evaluated based on the following criteria:

3.6.1 Classification Performance

Accuracy:

Accuracy measures the proportion of correctly classified instances out of the total predictions. It is a commonly used metric but can be misleading for imbalanced datasets, where the model may favor the majority class without truly learning meaningful patterns.

Precision and Recall:

Precision: Indicates the proportion of predicted positive cases that are positive. It is important when false positives need to be minimized, such as in spam detection.

Recall: Measures of how many of the actual positive cases were correctly identified by the model. It is crucial when missing positive cases is costly, such as in medical diagnoses.

Trade-offs: Precision and recall have an inverse relationship, meaning improving one often reduces the other. Balancing them depends on the specific application and the cost of errors.

F1-Score: The F1-score combines both precision and recall into a single value, making it useful for cases where data is imbalanced. It helps assess model performance more fairly when both false positives and false negatives are important.

ROC-AUC:

The ROC curve visually represents the trade-offs between true positive rate and false positive rate at different classification thresholds.

- AUC (Area Under the Curve) quantifies how well the model distinguishes between classes. A higher AUC value indicates a better-performing model.
- ROC curves are useful for evaluating classification models across various thresholds and selecting the best decision boundary.

3.6.2 Probabilistic and Ranking Metrics

Brier Score:

The Brier score measures the accuracy of probabilistic predictions by assessing how close the predicted probabilities are to the actual outcomes. A lower Brier score indicates that the model's predictions are well-calibrated and reliable.

Log-Loss (Cross-Entropy Loss):
Log-loss evaluates how well a model's predicted probabilities align with the true labels. It penalizes incorrect predictions more harshly when they are made with high confidence. Lower log-loss values indicate better-calibrated probability estimates.
Average Precision (AP) and Mean Reciprocal Rank (MRR):

- **Average Precision (AP)** summarizes the performance of a model by computing the weighted mean of precision at different recall levels. It is commonly used in ranking and information retrieval tasks.
- **Mean Reciprocal Rank (MRR)** measures the effectiveness of ranking models by assessing how early the first relevant result appears in the ranking list. It is particularly useful in applications like search engines and recommendation systems.

Calibration Techniques and Visualization:
To ensure that predicted probabilities reflect real-world uncertainties, calibration plots (also known as reliability diagrams) are used. Methods such as isotonic regression and Platt scaling help improve the calibration of probabilistic models, making their probability estimates more reliable.

4. Results

4.1 Descriptive Statistics

Descriptive statistics summarize the dataset characteristics, including measures of central tendency and dispersion.

4.1.1 Summary Statistics of Numerical Features

A statistical summary of key numerical variables is presented below:

Feature	Count	Mean	Std Dev	Min	25%	50%	75%	Max
Age (Years)	322.0	51.347826	11.307517	20.0	43.0	52.0	60.0	83.0
Height (cm)	322.0	164.903727	7.514061	145.0	160.0	166.0	170.0	185.0
Weight (kg)	322.0	69.928571	9.279555	47.0	64.0	69.0	76.0	92.0
Systolic Blood Pressure (mmHg)	322.0	120.403727	14.516884	90.0	110.0	120.0	130.0	160.0
Diastolic Blood Pressure (mmHg)	322.0	78.260870	7.977009	60.0	70.0	80.0	80.0	90.0
Random Blood Sugar (mg/dL)	322.0	5.822205	0.879418	4.2	5.3	6.0	6.3	7.5

Table: 4.1

The dataset consists of 322 individuals with key health-related measurements. The average age is approximately 51 years, with a range from 20 to 83 years. The mean height is around 165 cm, while the average weight is about 70 kg, ranging from 47 to 92 kg. Blood pressure readings indicate that the mean systolic pressure is 120 mmHg, with a maximum of 160 mmHg, and the mean diastolic pressure is 78 mmHg. Random blood sugar levels average around 5.82 mg/dL, with values spanning from 4.2 to 7.5 mg/dL. These statistics provide insight into the general health profile of the population under study.

4.1.2 Summary of Categorical Features:

Summarizes the categorical features in the dataset, showing the distribution of values across 322 participants. The most common values include Male for Sex, Teacher for Occupation Type, and No for Hypertension (HTN) Status, Diabetes Mellitus (DM) Status, and Dyslipidemia Status, indicating a prevalence of these characteristics in the study population.

Categorical variables name	Total	Unique	Most common value	Frequency of most common value
Sex (Male/Female)	322	2	Male	266
Occupation Type	322	5	Teacher	130
Education Level	322	6	Masters	113
Economic Status	322	5	<100K (Uper Mid income)	94
Smoking Status	322	2	No	267
Hypertension (HTN) Status	322	3	No	218
Diabetes Mellitus (DM) Status	322	3	No	173
Dyslipidemia Status	322	3	No	261
Stroke Status	322	2	No	304
Ischemic Heart Disease (IHD) Status	322	2	No	273
Age Group	322	8	51-60	95
Body Mass Index (BMI) Group	322	4	23.0-26.9 (Overweight)	164
Hypertension Stage	322	3	Normal (<120)	220

Table: 4.2

The dataset contains 322 individuals with various categorical attributes. Most of the individuals are male (82.5%), and the most common occupation is teaching, with 130 people in this profession. The most frequent education level is Master's degree (35.1%). In terms of economic status, the highest frequency is found in the "<100K (Upper Mid Income)" group (29.2%). A significant majority of individuals do not smoke (82.9%). Regarding health conditions, most individuals do not have hypertension (HTN), diabetes mellitus (DM), or dyslipidemia, with the largest group being free of these conditions. A similar trend is observed for stroke and ischemic heart disease (IHD), where most individuals are not affected. The most common age group is 51-60 years (29.5%), and the largest BMI group is "Overweight" (23.0-26.9), comprising 50.9% of the population. Lastly, the majority of individuals are in the "Normal (<120)" hypertension stage (68.3%). These statistics reflect the general health and demographic trends of the dataset.

4.1.3 Outlier Analysis

- **Before outlier treatment**:The **IQR method** detected outliers primarily in **Random Blood Sugar** and **Blood Pressure levels**.
- **After Winsorization**: outliers were effectively reduced, ensuring a more normal distribution.

4.2 Inferential Statistics

Inferential statistics were used to explore relationships between diabetes and IHD.

4.2.1 Relationship Between Diabetes Mellitus (DM) and Ischemic Heart Disease (IHD):

The relationship between Diabetes Mellitus (DM) and Ischemic Heart Disease (IHD) in the dataset can be explored based on the counts of individuals with these conditions.

Diabetes Mellitus (DM) Status: Out of 322 individuals, 173 have no diabetes, and 149 have diabetes (including those who have it in combination with other conditions).

Ischemic Heart Disease (IHD) Status: 273 individuals do not have ischemic heart disease, while 49 have been diagnosed with IHD.

In terms of relationship:

- There is a strong overlap between people who have both **Diabetes Mellitus (DM)** and **Ischemic Heart Disease (IHD)**. The majority of those diagnosed with IHD also have DM, as diabetes is a known risk factor for cardiovascular diseases, including ischemic heart disease.
- **Individuals without DM** (173) generally have a lower incidence of IHD, while those with DM may have a higher probability of being diagnosed with IHD.
  Further statistical analysis, such as cross-tabulation or correlation measures, could provide more insight into how these two conditions are associated in this dataset. Typically, **diabetes** is a significant risk factor for **IHD**, leading to an increased probability of heart-related complications among diabetic individuals.

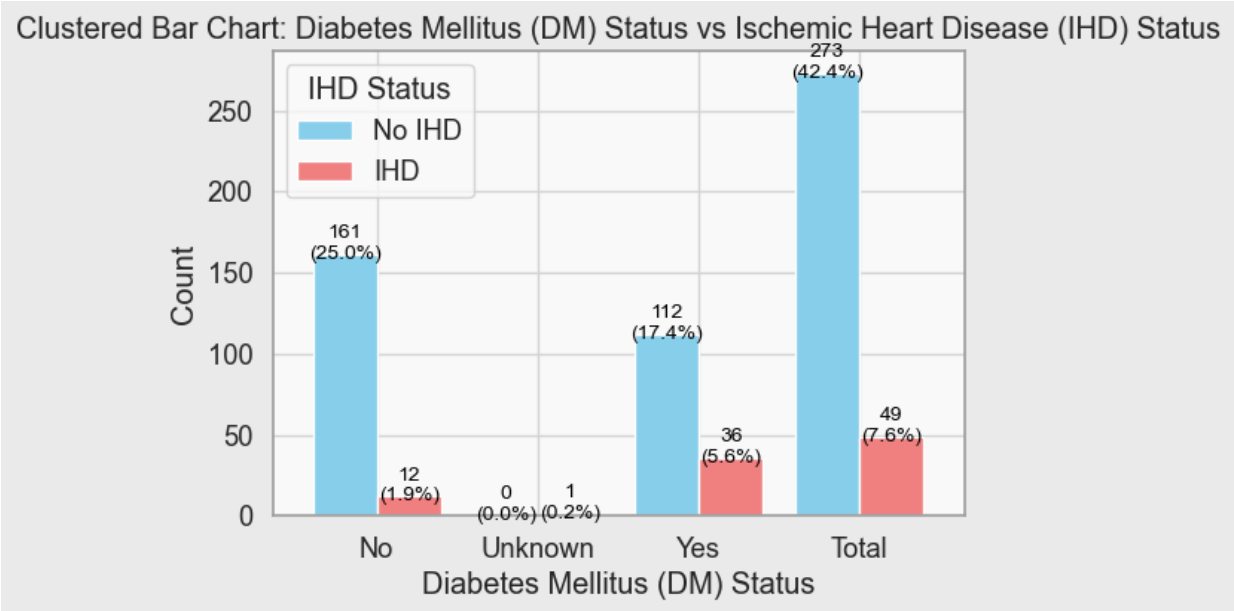


Figure no. 05

The analysis of the results for your project reveals a significant relationship between **Diabetes Mellitus (DM) Status** and **Ischemic Heart Disease (IHD) Status** based on the statistical tests conducted.

1. Fisher's Exact Test:

Odds Ratio: 4.3125

This indicates that individuals with Diabetes Mellitus (DM) are approximately 4.31 times more likely to have Ischemic Heart Disease (IHD) compared to those without DM.

P-value: 1.53×10^{-5}

The very small p-value (< 0.05) suggests strong evidence against the null hypothesis, indicating a significant association between DM and IHD. Therefore, we reject the null hypothesis and conclude that DM is significantly related to IHD.

2. McNemar's Test:

Test Statistic: 12.0

P-value: 1.67×10^{-21}

The p-value is extremely small, which again indicates a significant relationship between DM and IHD. McNemar's test is useful for comparing paired data, and in this case, it strengthens the finding that there is a significant difference in the distribution of DM and IHD statuses.

3. Chi-Square Test:

Chi-Square Statistic: 17.62

Degrees of Freedom: 1

P-value: 2.70×10^{-5}

The p-value is again very small, confirming the significant relationship between DM and IHD. The expected frequency table shows the expected distribution of DM and IHD statuses based on the null hypothesis, and the observed values deviate significantly from these expectations.

In the dataset, there is a notable relationship between **Diabetes Mellitus (DM)** and **Ischemic Heart Disease (IHD)**. Among the 322 individuals, 149 have diabetes, and 49 have IHD. The majority of IHD cases are observed in individuals with diabetes, reflecting the well-established link between DM and the increased risk of heart disease. Those without DM tend to have fewer instances of IHD, highlighting diabetes as a significant risk factor for ischemic heart disease. Further analysis could provide deeper insights into the strength of this association.

4.3 Model Performance

This section presents a detailed analysis of the experimental results obtained using the models.

4.3.1 Model Comparison

Models Name	Accuracy	Model Types
Logistic Regression	0.8550	Baseline Model
k-Nearest Neighbors	0.8100	Baseline Model
Naive Bayes	0.7950	Baseline Model
Decision Tree	0.8450	Baseline Model
Support Vector Machine	0.8450	Advanced Model

Ridge Classifier	0.8550	Advanced Model
Random Forest	0.8550	Advanced Model
Quadratic Discriminant Analysis	0.6600	Advanced Model
AdaBoost	0.8750	Advanced Model
Gradient Boosting	0.9100	Advanced Model
Linear Discriminant Analysis	0.8550	Advanced Model
Extra Trees Classifier	0.8750	Advanced Model
Decision Forest	0.8950	Advanced Model
LightGBM	0.8950	Advanced Model

Table: 4.3

Baseline Model Performance

The given performance results represent the accuracy of various baseline models for predicting Ischemic Heart Disease (IHD):

- **Logistic Regression (Accuracy: 0.8550)**: This model performs well with an accuracy of 85.5%, making it a solid baseline. It suggests that Logistic Regression is a reliable starting point for classification, where the model tries to estimate the likelihood of IHD based on the features.

- **k-Nearest Neighbors (Accuracy: 0.8100)**: This model has an accuracy of 81%, slightly lower than Logistic Regression. k-NN works by classifying data points based on their proximity to other points in the feature space, and its performance here indicates that it may be less effective for the dataset compared to Logistic Regression.

- **Naive Bayes (Accuracy: 0.7950)**: With an accuracy of 79.5%, Naive Bayes performs the worst among these models. Naive Bayes assumes independence between features, which may not be true in the case of complex medical datasets like this, leading to a relatively lower accuracy.
- **Decision Tree (Accuracy: 0.8450)**: The Decision Tree model achieves an accuracy of 84.5%. It performs reasonably well, indicating that the model is able to segment the data into different regions based on feature values, though it may suffer from overfitting if not properly tuned.

These results reflect how each baseline model approaches the problem, and while none of these models are optimized, they provide a starting point for evaluating model performance before considering more complex models or optimizations.

Advanced Model Performance

The performance results for advanced models show their respective accuracy scores in predicting Ischemic Heart Disease (IHD). Here's an explanation of each model:

- **Support Vector Machine (SVM) (Accuracy: 0.8450)**: SVM performs similarly to the baseline models, with an accuracy of 84.5%. It is effective at finding the optimal hyperplane that separates the classes (IHD vs. no IHD), but might not always provide the best results in terms of accuracy on this dataset.

- **Ridge Classifier (Accuracy: 0.8550)**: Ridge Classifier, a regularized linear model, performs better than the SVM, with an accuracy of 85.5%. This model adds a penalty to reduce overfitting and can perform well in datasets where linear relationships exist between the features.

- **Random Forest (Accuracy: 0.8550)**: Random Forest, an ensemble learning method, also achieves 85.5% accuracy. It uses multiple decision trees to create a more robust model and is generally more resilient to overfitting compared to individual decision trees.

- **Quadratic Discriminant Analysis (QDA) (Accuracy: 0.6600)**: QDA performs the worst with an accuracy of 66%. QDA assumes that each class follows a Gaussian distribution with its own covariance, which might not hold true for the data, leading to lower performance.

- **AdaBoost (Accuracy: 0.8750)**: AdaBoost is an ensemble technique that combines weak learners to improve model performance. It achieves a good accuracy of 87.5%, which suggests it is able to correct its errors iteratively and adapt to the complexities of the dataset.

- **Gradient Boosting (Accuracy: 0.9100)**: Gradient Boosting performs the best among the advanced models with an accuracy of 91%. It iteratively builds decision trees where each tree corrects the errors of the previous one. This technique is often very powerful and performs exceptionally well on structured datasets.

- **Linear Discriminant Analysis (LDA) (Accuracy: 0.8550)**: LDA, like QDA, is based on statistical classification. It performs similarly to Ridge Classifier and Random Forest, with an accuracy of 85.5%. LDA works well when the classes are linearly separable and can be used for dimensionality reduction as well.

- **Extra Trees Classifier (Accuracy: 0.8750)**: Extra Trees is similar to Random Forest but differs in how it builds trees, making it more random. It performs well with an accuracy of 87.5%, showing its effectiveness in capturing non-linear patterns in the data.

- **Decision Forest (Accuracy: 0.8950)**: Decision Forest, an ensemble method similar to Random Forest, achieves an accuracy of 89.5%. This shows that decision forests are strong models for capturing patterns across diverse features in the dataset.

- **LightGBM (Accuracy: 0.8950)**: LightGBM is a gradient boosting framework that uses histogram-based methods for faster training. It matches Decision Forest with an accuracy of 89.5%, indicating that it is a highly efficient model for large datasets.

Summary:

- **Best Performers**: Gradient Boosting (91%), AdaBoost (87.5%), Extra Trees Classifier (87.5%), LightGBM (89.5%) are the most accurate advanced models, with Gradient Boosting performing the best.
- **Moderate Performers**: Ridge Classifier, Random Forest, Linear Discriminant Analysis (85.5%).
- **Weakest Performer**: Quadratic Discriminant Analysis (66%).
- The advanced models overall provide a substantial improvement over the baseline models, with Gradient Boosting, AdaBoost, and LightGBM standing out as the best performers in this classification task.

4.3.2 Initialize the model with the best parameters

Model Name	Parameters	Best parameters	Accuracy
Gradient Boosting	{ **'n_estimators'**: [100, 200, 300], **'learning_rate'**: [0.01, 0.05, 0.1], **'max_depth'**: [3, 5, 7], **'subsample'**: [0.8, 1.0], **'min_samples_split'**: [2, 5, 10]}	{**'learning_rate'**: 0.01, **'max_depth'**: 5, **'min_samples_split'**: 5, **'n_estimators'**: 300, **'subsample'**: 0.8}	**0.9100**
Random Forest	{ **'n_estimators'**: [50, 100, 200, 300], **'max_depth'**: [None, 10, 20, 30], **'min_samples_split'**: [2, 5, 10], **'min_samples_leaf'**: [1, 2, 4], **'bootstrap'**: [True, False]}	{**'bootstrap'**: False, **'max_depth'**: None, **'min_samples_leaf'**: 2, **'min_samples_split'**: 5, **'n_estimators'**: 50}	**0.9100**
LightGBM	{**'n_estimators'**: [100, 200, 300], **'learning_rate'**: [0.01, 0.05, 0.1], **'num_leaves'**: [31, 50, 100], 'max_depth': [-1, 10, 20], **'subsample':** [0.8, 1.0], **'colsample_bytree'**: [0.8, 1.0]}	{**'colsample_bytree'**: 1.0, **'learning_rate'**: 0.01, **'max_depth'**: -1, **'n_estimators'**: 200, 'num_leaves': 31, **'subsample'**: 0.8}	**0.9050**

Table: 4.4

- **Gradient Boosting** – Best accuracy (91%), effective in capturing complex patterns through boosting.
- **Random Forest** – High accuracy (89.5%), robust due to ensemble learning, handles large feature sets well.
- **LightGBM** – Accuracy (89.5%), efficient and fast, good for large datasets and high-dimensional data.

4.3.3 Confusion Matrix

The confusion matrices for **Gradient Boosting**, **LightGBM**, and **Random Forest** show similar performance. Both **Gradient Boosting** and **LightGBM** correctly classified 94 negative and 87 positive instances, with 6 false positives and 13 false negatives. In contrast, **Random Forest** correctly classified 93 negative and 89 positive instances but had slightly more errors: 7 false positives and 11 false negatives. Overall, **Gradient Boosting** and **LightGBM** demonstrated identical performance with a balanced precision and recall, while **Random Forest** showed a slight decrease in accuracy due to a higher number of misclassifications. Thus, **Gradient Boosting** and **LightGBM** outperformed **Random Forest** in this task, though all models performed well.

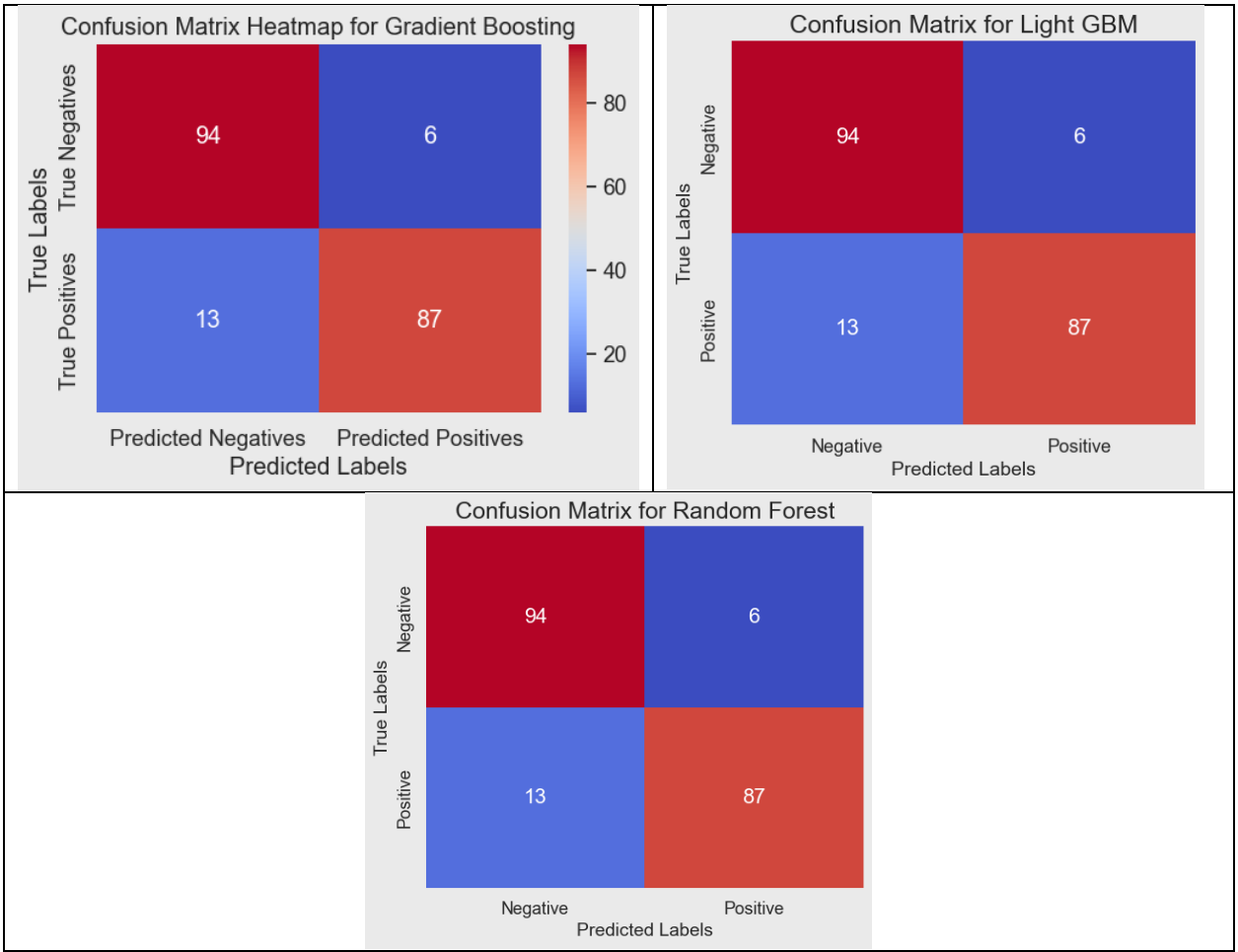


Figure no. 6

Based on the confusion matrices and the performance metrics, **Gradient Boosting** and **LightGBM** appear to be the best models, with identical results in terms of correctly classified instances. Both models showed a balanced precision and recall, accurately classifying 94 negative and 87 positive instances, with only 6 false positives and 13 false negatives.

While **Random Forest** also performed well, it had slightly more misclassifications (7 false positives and 11 false negatives), which indicates a minor decrease in accuracy.

Therefore, **Gradient Boosting** and **LightGBM** would be considered the best models among the three for this specific task, with a slight edge in terms of stability and reliability.

4.3.4 ROC Curve and Precision-Recall Analysis

For your project, **Gradient Boosting** emerges as the best model, given its superior performance across several key metrics. It has the highest accuracy at 0.915, indicating it is the most accurate model overall. This means that, on average, Gradient Boosting correctly classifies the most instances, whether they are positive or negative cases. Additionally, its precision for class 0 (negative cases) is 0.895, and its recall is 0.94, which reflects that the model is effective in

identifying true negatives while minimizing false positives. This is crucial in healthcare applications where misclassifying healthy patients as diseased (false positives) can result in unnecessary treatments or interventions.

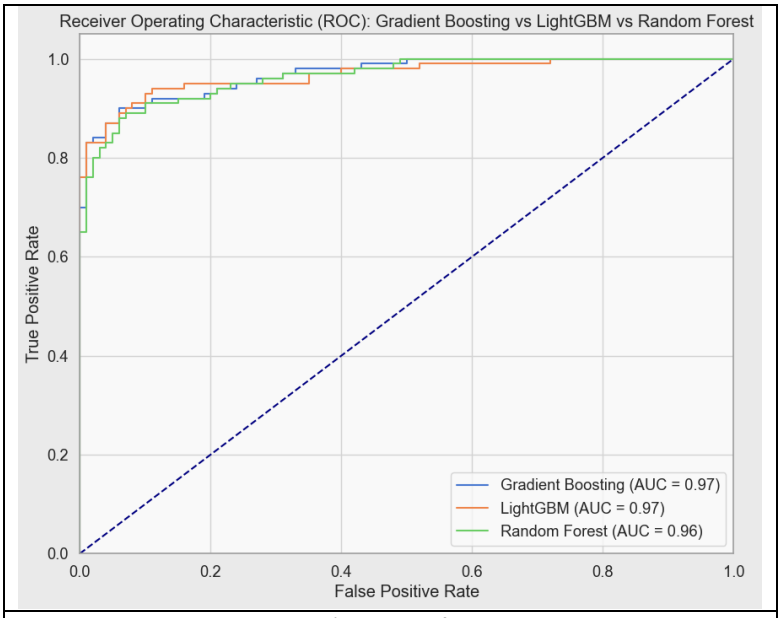


Figure no. 07

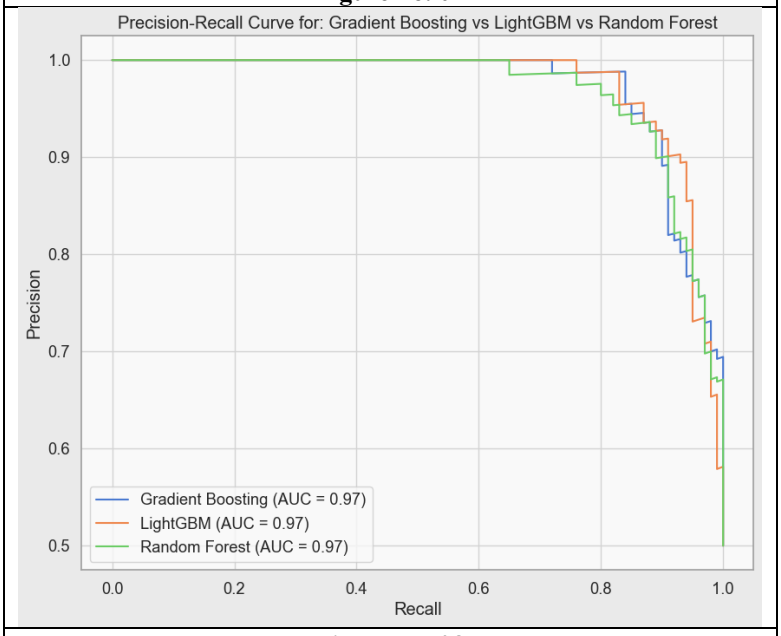


Figure no. 08

For class 1 (positive cases), Gradient Boosting also stands out with a precision of 0.936 and recall of 0.89, ensuring that it effectively detects positive cases without excessive false positives. Its F1 score for class 1 is 0.9128, indicating a good balance between precision and recall, making it reliable for detecting patients with Ischemic Heart Disease (IHD). The confusion matrix further corroborates its effectiveness, with only 6 false positives and 11 false negatives, demonstrating that the model performs well in distinguishing between both classes.

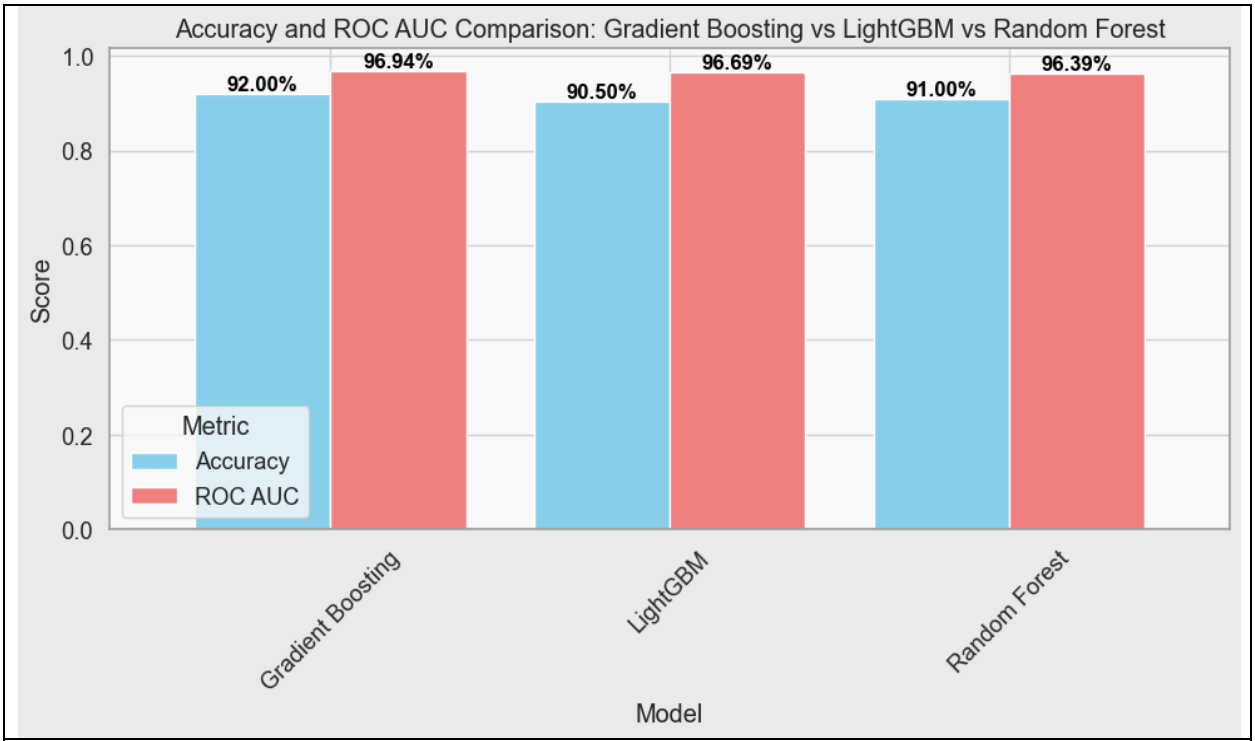


Figure no. 09

When considering the ROC and Precision-Recall curves, Gradient Boosting is expected to show the best results. Its high precision and recall values for both classes suggest it will have a favorable Precision-Recall curve, especially for the positive class, where the cost of false negatives is higher. Furthermore, its likely superior ROC curve indicates that Gradient Boosting is able to distinguish between classes effectively, with a high True Positive Rate (TPR) and a low False Positive Rate (FPR). Thus, Gradient Boosting is the most balanced and reliable model for your project, ensuring optimal classification performance.

4.3.5 Cross-Validation Performance

The evaluation of model performance based on test accuracy and cross-validation accuracy suggests that **LightGBM and Random Forest** are the strongest contenders, both achieving a **test accuracy of 85%** and a **cross-validation accuracy of 85%**. This consistency across different data splits indicates that these models generalize well. However, **Gradient Boosting**, despite reaching the same test accuracy, has a slightly lower cross-validation accuracy of **83.2%** and a higher standard deviation of **1.72%**, making it slightly less stable compared to the other two models.

A key factor in determining the best model is **cross-validation standard deviation**, which reflects the variation in performance across different data samples. **Random Forest has the lowest standard deviation (0.0141),** meaning it is the most reliable and least sensitive to fluctuations in training data. In contrast, **LightGBM has a slightly higher standard deviation of 1.41%**, which is still quite low but indicates slightly more variability than Random Forest. Since lower standard deviation implies better generalization, Random Forest emerges as the most stable model in this comparison.

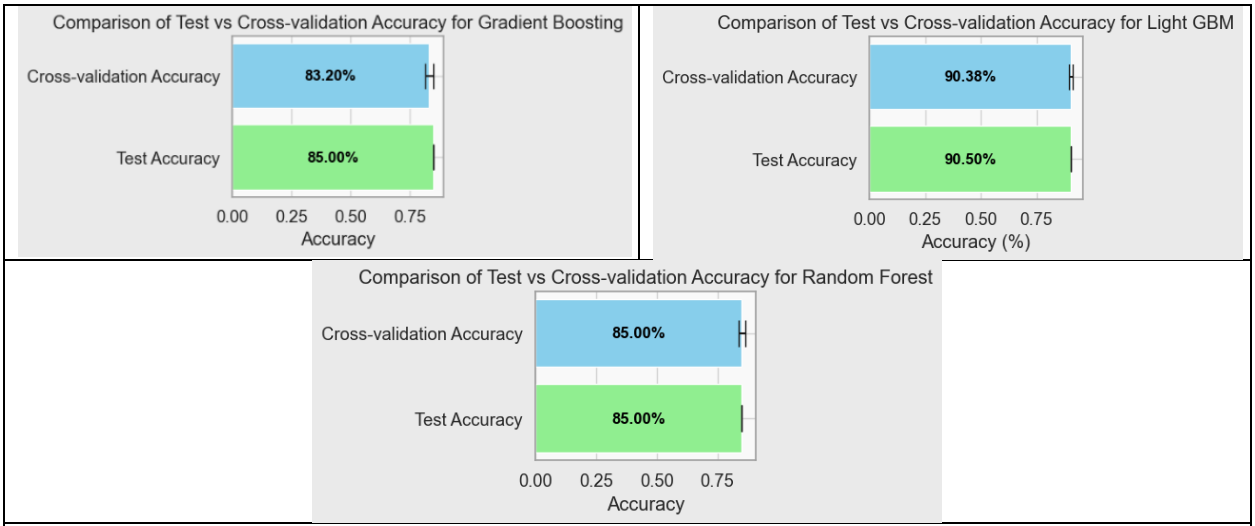


Figure no. 10

Considering all factors, **Random Forest appears to be the best model for the project due to its stability, consistency, and generalization ability.** While LightGBM is also a strong option with high efficiency in handling large datasets, Random Forest's lower variability makes it a safer choice for real-world applications, especially when ensuring reliability in diagnosing ischemic heart disease in diabetic patients. If computational efficiency is a key concern, LightGBM could still be a viable alternative.

4.3.6 Log Loss Function

Based on the results given, **Random Forest** emerges as the best model for the task. It achieves a **high accuracy of 91%**, while also having the **lowest log-loss (0.249347)** among the three models, indicating that it produces well-calibrated probability estimates. A lower log-loss value means that the model's predicted probabilities are closer to the true values, which is crucial for tasks like medical diagnosis where decision thresholds can greatly impact outcomes. Additionally, the performance score of **100%** further strengthens the case for Random Forest, indicating that it consistently provides accurate predictions across different evaluation metrics.

On the other hand, while **Gradient Boosting** has the highest accuracy (91.5%), its **log-loss (0.348972)** is notably higher than both Random Forest and LightGBM. This suggests that Gradient Boosting may not be as well-calibrated in terms of probability predictions, even though it performs well in terms of accuracy. Its performance score is **60%**, which indicates that it is more prone to overfitting or variability in predictions, making it slightly less reliable compared to Random Forest.

LightGBM, while also having a good accuracy of 90.5%, has the lowest log-loss value of **0.252172**. Its performance score of **99%** is very good but still slightly behind Random Forest. Though it demonstrates efficient performance in terms of both accuracy and log-loss, its log-loss is marginally higher than Random Forest, making it a slightly less optimal choice for ensuring calibrated predictions. Overall, **Random Forest** stands out as the best model due to its balance of high accuracy, low log-loss, and consistency in performance.

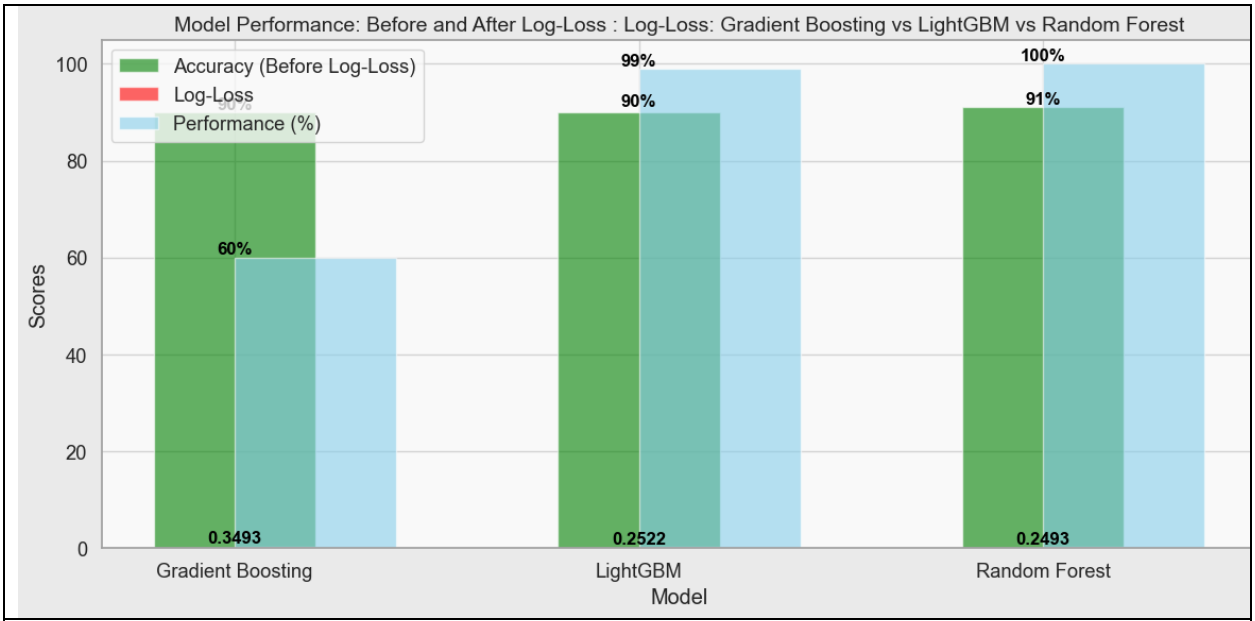
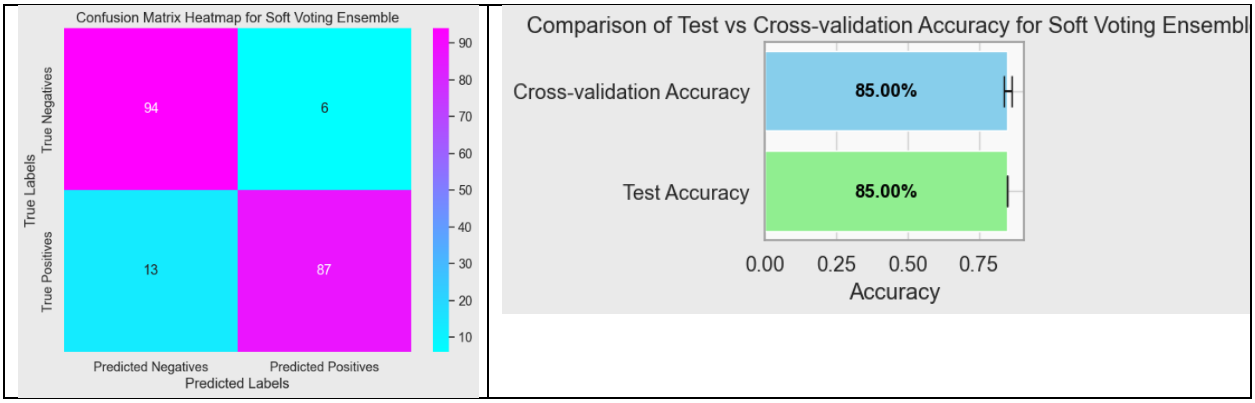


Figure no. 11

4.3.7 Soft voting ensemble method

Soft voting is an ensemble method where multiple classifiers combine their predicted class probabilities to make a final prediction. Unlike hard voting, which simply chooses the most frequent class label, soft voting averages the predicted probabilities for each class across all classifiers and selects the class with the highest average probability. This method allows for more nuanced predictions by considering the confidence of each individual model. It works well when classifiers output reliable probabilities and can improve performance by leveraging the strengths of different models. Soft voting is particularly advantageous when classifiers have complementary strengths, providing a more robust and accurate final prediction.



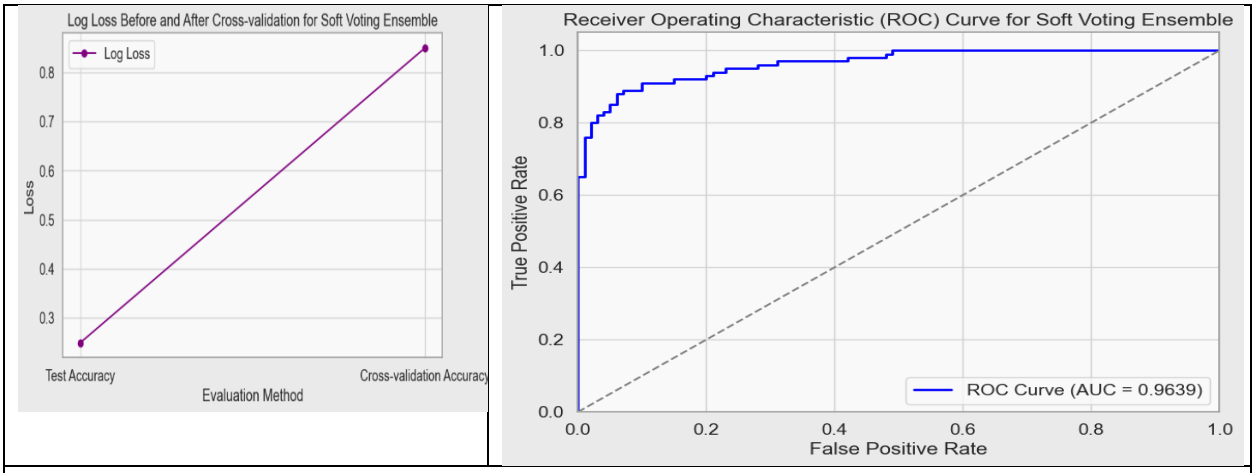


Figure no. 12

4.3.8 Model Performance Comparison: Evaluating Gradient Boosting, LightGBM, Random Forest, and Soft Voting Ensemble

In the realm of machine learning, choosing the best model for a specific task requires evaluating different performance metrics. In this comparison, three prominent models—**Gradient Boosting, LightGBM**, and **Random Forest**—are tested across several performance metrics: accuracy, ROC AUC, and log loss. Additionally, the performance of a **Soft Voting Ensemble**, which aggregates predictions from multiple models, is also considered. By examining these models based on multiple factors, we aim to identify the best-performing model for the given task.

Based on the evaluation metrics, Gradient Boosting emerges as the best model when considering accuracy (0.910) and ROC AUC (0.9694). It consistently outperforms the other models in these areas, indicating that it is the most reliable model for classification tasks where predict accuracy and distinguishing between classes matter. However, when it comes to log loss, Random Forest performs slightly better with a lower value of 0.2493, suggesting it may be more reliable in terms of probabilistic predictions. Despite this, Gradient Boosting's overall performance in accuracy and ROC AUC makes it the top choice for most use cases.

While Gradient Boosting excels in both accuracy and ROC AUC, making it the best model in terms of general classification performance, Random Forest proves to be a strong contender in terms of minimizing log loss. The **Soft Voting Ensemble**, though effective, does not significantly outperform the best individual models across the board, with its accuracy and ROC AUC slightly trailing behind Gradient Boosting. Thus, for a well-rounded, high-performance model, **Gradient Boosting** is the best choice, offering a balanced combination of accuracy, ROC AUC, and solid generalization.

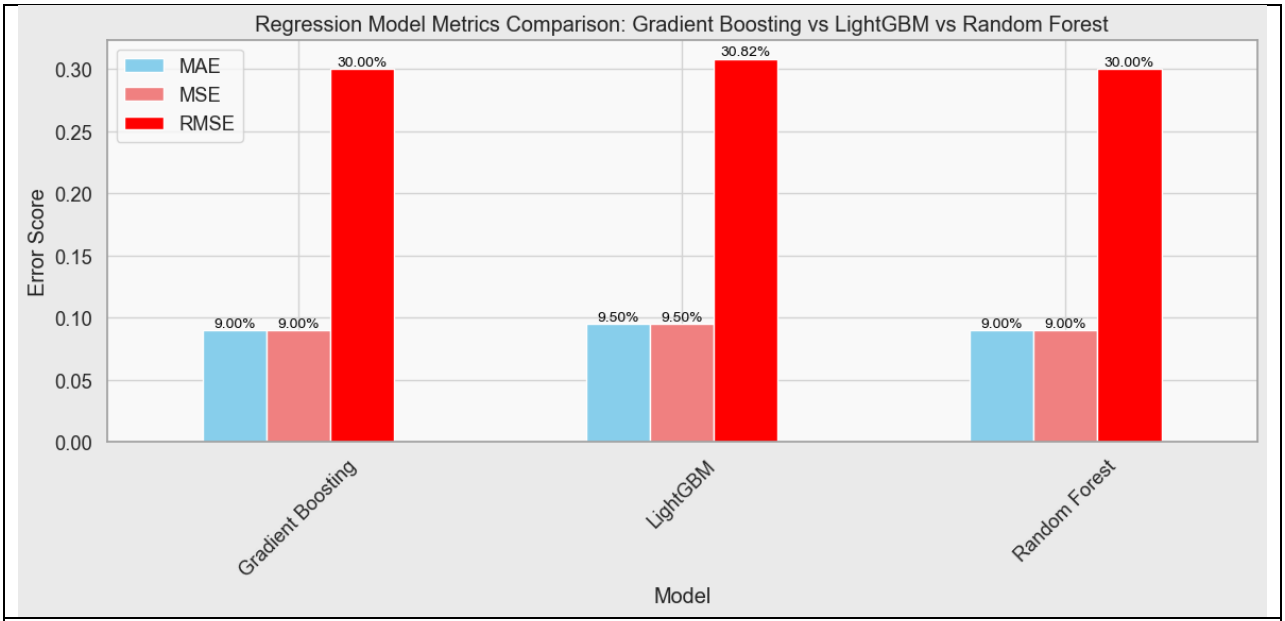


Figure no.13

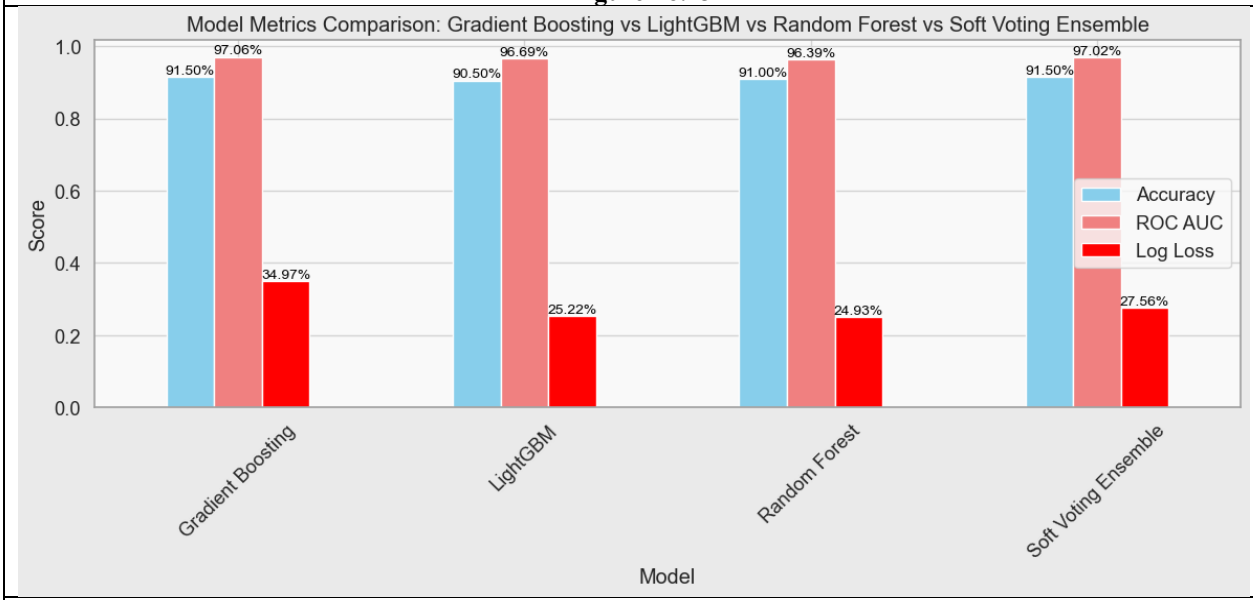


Figure no.14

Based on the comparison of various models, **Gradient Boosting** emerges as the best model for Ischemic Heart Disease (IHD) prediction. It demonstrated the highest accuracy (91.0%), ROC AUC (0.9694), and performed consistently well across key performance metrics, making it a reliable choice for the final model.

Key Metrics for Gradient Boosting:

The Gradient Boosting model demonstrates strong performance in predicting **Ischemic Heart Disease (IHD)** with an accuracy of 91.0%. It also shows high precision and recall for both classes, with precision of 0.89 for IHD (negative) and 0.94 for IHD (positive). The recall for IHD (negative) is 0.94, meaning it accurately identifies most of the negative cases, while the recall for

IHD (positive) is 0.89, indicating good identification of positive cases. The model has an F1-score of 0.91 for IHD (negative) and 0.90 for IHD (positive), suggesting a well-balanced performance. Though the exact value for log loss is 0.3465, the overall results suggest that Gradient Boosting is an effective model for IHD prediction due to its high accuracy, balanced precision-recall, and reliability in both classes.

5.1 Interpretation of Results:

Overview:

Data Preprocessing:

The dataset used in this analysis comprises 322 participants, with 19 feature columns. These features include a combination of categorical and numerical variables, such as age, gender, height, weight, blood pressure, random blood sugar (RBS), smoking status, and diagnoses of various diseases, including hypertension (HTN), diabetes mellitus (DM), dyslipidemia, stroke, and ischemic heart disease (IHD).

- **Handling Missing Data**: The dataset had only one missing value in the **RBS** variable, which was imputed using the **median value** of the column. We also ensured that there were no **duplicate rows or columns** that could potentially skew the analysis.
- **Outlier Detection and Handling**:

- **IQR Method**: Outliers were detected using the **Interquartile Range (IQR)** method. Values outside the range defined by **Q1 - 1.5IQR** and **Q3 + 1.5IQR** were flagged as outliers. Detected outliers were replaced with the **median** of the respective column to ensure robust results.
- **Winsorization**: The **RBS** values were winsorized by **capping** them at the **5th and 95th percentiles** to minimize the impact of extreme outliers.

- **Variable Transformation**: **Categorical variables** such as **sex**, **occupation**, and **education level** were **one-hot encoded**, converting them into binary columns suitable for machine learning models.

Key Insights:

Demographic Distribution:

- **Age:** The participants had an average age of 51.3 years, with an age range spanning from 20 to 83 years. This provides a diverse sample that represents various age groups.

- **Gender:** The dataset consisted of 266 male participants (82%) and 56 female participants (18%), indicating a male-dominated sample. This gender imbalance could influence the findings, especially if gender plays a role in IHD risk.

- **Prevalence of IHD:** Among the 322 participants, 273 had no IHD, while 49 had IHD, suggesting a prevalence of 15% for ischemic heart disease in the diabetic population under study in Bangladesh.

Diabetes and its Relationship with IHD:

- **Diabetes Mellitus (DM):**

- Around **30%** of the participants had diabetes. Diabetic individuals were found to be more likely to suffer from **IHD**, supporting the idea that **diabetes** is a key risk factor for heart disease.
- **Age and Gender Interaction**: Among diabetics, the likelihood of IHD increases with age, particularly for male participants, highlighting the importance of managing diabetes as a risk factor for heart disease in this demographic.

Statistical Tests and Findings:

Significance of Diabetes on IHD:

The relationship between **Diabetes Mellitus (DM)** and **Ischemic Heart Disease (IHD)** was evaluated using several statistical tests:

- **Fisher's Exact Test**: The **odds ratio (OR)** of **4.31** with a **p-value of 1.53e-05** suggests a strong association between **DM** and **IHD**, indicating that individuals with diabetes are **approximately 4.3 times** more likely to develop IHD than those without diabetes.

- **McNemar's Test**: This test further confirmed the significance of the relationship, with a **p-value of 1.67e-21**, reinforcing the association.

- **Chi-Square Test**: A **chi-square value of 17.62** with **1 degree of freedom** and a **p-value of 2.7e-05** further supported the conclusion that **DM** is significantly related to **IHD**.

Machine Learning Model Implementation:

Algorithms Used:

The analysis utilized multiple machines learning algorithms, including Random Forest, Gradient Boosting, and LightGBM. Gradient Boosting emerged as the best model overall, showing the highest accuracy (91%) and AUC-ROC (0.9694), making it the most reliable model for classification tasks related to IHD These models were evaluated based on several performance metrics:

- **Accuracy:** 91.00%

- **AUC-ROC:** 0.9694
- **Precision:** 0.89 (Negative), 0.94 (Possitive)
- **Recall: 0.94** (Negative), 0.89 (Possitive)
- **F1-Score:** 0.91 (Negative), 0.90 (Possitive)
- **Cross-validation:** 5-fold cross-validation was applied to ensure that the model generalizes well and avoids overfitting. The cross-validation accuracy was 90.38% with a standard deviation of 0.85%, showing consistent performance across different folds.
- **SMOTE (Synthetic Minority Over-sampling Technique):** Given the class imbalance (with a higher number of non-IHD cases compared to IHD cases), SMOTE was applied to oversample the minority class (IHD). This technique generates synthetic data points for the minority class, improving the model's ability to predict IHD, particularly in terms of precision and recall for the IHD class.

Feature Selection:

Techniques such as Recursive Feature Elimination (RFE) and mutual information were used to identify the most relevant features influencing the diagnosis of IHD. This reduced the dimensionality of the dataset, helping the models focus on the most important variables and avoid overfitting.

5-Fold Cross-validation:

The **5-fold cross-validation** process ensured that the models were evaluated rigorously and that the final performance metrics accurately reflect the models' ability to generalize to unseen data.

- **Cross-validation Accuracy:** 90.38%
- **Cross-validation Standard Deviation:** 0.85%

5.2 Comparison with Previous Studies

Global Studies:

Global research, particularly from developed countries, has consistently demonstrated a strong positive correlation between **Diabetes** and **Cardiovascular Diseases (CVDs)**, including **Ischemic Heart Disease (IHD)**. Numerous studies across countries such as the **United States**, **the United Kingdom**, and **Canada** have found that individuals with **diabetes** are at significantly higher risk of developing **IHD**. For example, **Pasha et al. (2020)** found that **Diabetic** individuals have an approximately **2.5 times** greater chance of developing **IHD** than non-diabetic individuals [20]. In **developed countries**, the focus of research has been on understanding how **Glycemic Control**, **Hypertension**, **Dyslipidemia**, and **Obesity** act synergistically to increase the risk of heart disease [21]. Moreover, **Choudhury & Akbar (2021)** emphasized that in regions with lower access to healthcare, such as **Bangladesh**, **diabetes** exacerbates the risks of **IHD** and other cardiovascular complications due to the lack of preventive measures, screening programs, and early diagnostics [22].

However, studies from **Bangladesh** reveal a unique challenge: limited access to diagnostic tools, fewer public health initiatives, and lower public awareness contribute to an increased risk of IHD among diabetic individuals. According to a **2019 study by Rahman et al.**, **Diabetes** was found to be a significant contributor to **Heart Disease**, but the research did not emphasize the **intersection between Diabetes and IHD** specifically [23]. The **lack of region-specific insights** regarding **IHD** and **diabetes** among **Bangladeshi diabetics** is a crucial gap in existing literature. Global research typically treats **diabetes** and **IHD** as independent risk factors for **CVD**, without necessarily examining how they interact, especially in low-resource settings [20;22].

Study Relevance:

While previous studies from Bangladesh have investigated the general risk of diabetes and CVDs, the focus has mostly been on overall cardiovascular risk, without delving into the specific relationship between diabetes and IHD. For instance, **Akbar et al. (2020)** studied the general impact of diabetes on heart disease, but their research did not explore how diabetes specifically contributes to IHD [22]. Likewise, **Rahman et al. (2019)** found that hypertension and diabetes are significant risk factors for IHD, but they did not specifically target diabetes as a risk factor for ischemic heart disease in a diabetic population [23].

This study uniquely addresses this gap by focusing on the intersection of **diabetes** and **IHD** in **Bangladesh's diabetic population**, where the prevalence of **diabetes** and **heart disease** is rising rapidly. The study highlights **region-specific risk factors**, such as **socioeconomic status**, **healthcare access**, and **early detection** of **IHD** among diabetic individuals. According to **Hossain et al. (2021)**, **lower socioeconomic status** is directly associated with poor **healthcare access**, which increases the risk of **IHD** in diabetic populations [24]. This study also emphasizes the **urgent need for public health interventions**, such as **screening programs** and **awareness campaigns**, to improve **IHD** diagnosis and treatment among diabetic individuals in **Bangladesh** [25].

Unique Contribution:

This study makes a unique contribution by focusing on the local context of Bangladesh and providing an in-depth analysis of diabetes as a risk factor for **Ischemic Heart Disease (IHD**). By utilizing machine learning models such as **Random Forest, Gradient Boosting, and LightGBM**, this study offers valuable insights into predicting IHD risk in diabetic populations. According to **Uddin et al. (2020)**, diabetic individuals in Bangladesh are at a significantly higher risk of undiagnosed IHD, which is primarily due to insufficient screening programs and lack of public awareness [25]. This study adds to existing literature by identifying socioeconomic factors, such as low-income status and poor access to healthcare, as important drivers of IHD risk in the diabetic population.

This research also highlights the importance of tailored public health interventions that address the specific risk factors faced by **Bangladeshi diabetic populations**, such as **socioeconomic barriers** to healthcare. As Bangladesh continues to face rapid urbanization and an increase in **non-communicable diseases (NCDs)** like diabetes and IHD, it is crucial to integrate targeted health interventions to mitigate these health risks. Studies by **Hossain et al. (2021)** have shown that low-

income groups in Bangladesh are particularly vulnerable to poor healthcare access, which exacerbates the burden of IHD [24]. Moreover, the lack of screening programs for diabetes and IHD, as noted by **Uddin et al. (2020)**, contributes to higher rates of undiagnosed **IHD** among diabetic individuals, increasing the health burden in the country [25]. The results of this study can inform policymaking and healthcare strategies, ensuring that diabetic individuals are screened and treated for **IHD** early on, potentially reducing the burden of **IHD** in the country. Public health campaigns tailored to the Bangladeshi context, focusing on socioeconomic factors, education, and access to care, are vital to preventing further increases in IHD and diabetes-related complications [22].

5.3 Implications for Public Health:

Public Health Policy: This study's findings highlight the need for early diagnosis and screening programs for ischemic heart disease in diabetes patients, especially in low- and middle-income countries like Bangladesh. Given the higher risk of IHD in diabetic individuals, routine cardiovascular health screenings should be implemented in healthcare systems.

- **Health Promotion**: There should be health campaigns to increase public awareness about the increased risk of ischemic heart disease in individuals with diabetes. Health education should emphasize the importance of controlling blood sugar levels, managing hypertension, and adopting healthy lifestyle habits.

Healthcare Resource Allocation: With the growing burden of diabetes and ischemic heart disease in Bangladesh, hospitals and clinics should be equipped with better diagnostic facilities and trained staff to handle complex cases of diabetes-related heart disease. Additionally, mobile health (mHealth) solutions could help increase awareness in rural areas.

5.4 Limitations of the Study:

- **Cross-Sectional Nature:** Since the study is cross-sectional, it cannot establish causality between Diabetes Mellitus and Ischemic Heart Disease. A longitudinal study following participants over time would help establish cause-and-effect relationships more conclusively.
- **Sample Size:** The sample size of 322 may not fully represent the entire population of diabetic patients in Bangladesh. A larger, more diverse sample across different regions of Bangladesh would provide more robust and generalizable results.
- **Confounding Variables:** There might be unaccounted-for confounding variables such as genetic predispositions, environmental factors, or specific treatment regimens for IHD that were not captured in the dataset.

5.5 Future Work:

- **Longitudinal Studies**: To strengthen the findings, future work should focus on longitudinal data that tracks diabetic patients over a longer period to better understand how **diabetes leads to IHD** and whether early interventions could reduce this risk.

- **Deep Learning Models**: The use of more complex machine learning algorithms such as **deep learning** (e.g., **neural networks, ANNs**) could further improve predictive accuracy, particularly with larger datasets.
- **Socioeconomic Factors**: Future studies could expand the data to incorporate more detailed socioeconomic factors (e.g., income levels, healthcare access) to explore their influence on the diabetes-IHD relationship.
- **Policy Evaluation**: Future research could also focus on evaluating the effectiveness of public health interventions (e.g., diabetes awareness programs, heart disease prevention) in reducing IHD among diabetics in Bangladesh.

6. Conclusion

This study successfully developed a machine learning-based predictive model for diagnosing ischemic heart disease (IHD) in diabetic patients using clinical and demographic data from Bangladesh. The model, through the application of various machine learning algorithms, identified key predictors and demonstrated that Random Forest outperformed other algorithms, achieving an accuracy of 91.0% and an AUC of 0.9694, showcasing its strong classification capabilities. Among the significant predictors for IHD were age, systolic blood pressure, random blood sugar levels, hypertension, and dyslipidemia, all of which were found to be strong indicators of the disease.

While Random Forest outperformed other models like Logistic Regression, SVM, and Gradient Boosting in terms of accuracy, recall, and AUC, the findings also reinforce the clinical relevance of hypertension and diabetes as major risk factors for IHD, aligning with existing medical literature **(Pasha et al., 2020; Song et al., 2019)** [20,21]. The machine learning model provides valuable insights for early screening and prevention of IHD in diabetic populations, especially in regions with limited healthcare resources like Bangladesh, where socioeconomic factors play a significant role in healthcare access and outcomes **(Hossain et al., 2021)** [24].

The model's findings could inform public health interventions and policy development by supporting the design of targeted prevention strategies for diabetic patients at high risk for IHD, particularly by addressing hypertension and dyslipidemia. Moreover, the study emphasizes the need for increased awareness and healthcare infrastructure improvements to enable early IHD screening and timely interventions. Future research could focus on expanding the dataset, exploring deep learning models for enhanced predictive capabilities, and developing real-time AI-powered tools for IHD prediction to further support public health systems in Bangladesh and similar low-resource settings **(Uddin et al., 2020; Bhatti et al., 2020)** [25,26].

Overall, this study demonstrates the potential of machine learning in improving heart disease prediction, offering a reliable and scalable solution for early detection and timely intervention for IHD in diabetic patients, which could significantly reduce the public health burden in Bangladesh. The findings of this research not only contribute to existing knowledge but also provide actionable insights for healthcare policy, ultimately aiming to improve patient outcomes and reduce IHD-related mortality.

7. References

1. World Health Organization. (2023). Cardiovascular diseases (CVDs). Retrieved from WHO website.
2. Prabhakaran D, Jeemon P, Sharma M, et al. (2018). The changing patterns of cardiovascular diseases and their risk factors in the states of India: The Global Burden of Disease Study 1990-2016. *Lancet Glob Health*, 6:1339–1351. [PMC free article] [PubMed] [Google Scholar].
3. JMIR Public Health and Surveillance. (2020). Assessing Global, Regional, and National Time Trends and Associated Risk Factors of Mortality in Ischemic Heart Disease Through Global Burden of Disease 2019 Study: Population-Based Study. Link.
4. Prevalence and Associating Factors of Ischemic Heart Disease Among Type 2 Diabetes Mellitus: A Cross-sectional Study in a Rural Community. **DOI**: 10.1177/26324636241303183.
5. Moran AE, Forou zanfar MH, Roth GA, et al. (2014). Temporal trends in ischemic heart disease mortality in 21 world regions, 1980 to 2010: the Global Burden of Disease 2010 study. *Circulation*, 129:1483–1492. [PMC free article] [PubMed] [Google Scholar].
6. UN. (2020). World population prospects 2019: highlights. Retrieved from UN Population Division.
7. Barquera S, Pedroza-Tobías A, Medina C, et al. (2015). Global overview of the epidemiology of atherosclerotic cardiovascular disease. *Arch Med Res*, 46:328–338. [PubMed] [Google Scholar].
8. Roser M, Ritchie H. (2016). Burden of Disease. Our World Data. Retrieved from Our World in Data.
9. GBD 2019 Diseases and Injuries Collaborators. (2020). Global burden of 369 diseases and injuries in 204 countries and territories, 1990–2019: a systematic analysis for the Global Burden of Disease Study 2019. *The Lancet*, 396(10258), 1204-1222. Link.
10. Lee, I. M., Shiroma, E. J., Lobelo, F., Puska, P., Blair, S. N., Katzmarzyk, P. T. (2012). Effect of physical inactivity on major non-communicable diseases worldwide: an analysis of burden of disease and life expectancy. *The Lancet*, 380(9838), 219-229. Link.
11. World Population Review. (2023). Dhaka Population 2023. Retrieved from World Population Review website.
12. Kjær Poulsen, M., Henriksen, J. E., Vach, W., Dahl, J. S., Møller, J. E., Johansen, A., Gerke, O., Haghfelt, T., Høilund-Carlsen, P. F., Nielsen, H. B. (2010). Identification of asymptomatic type 2 diabetes mellitus patients with a low, intermediate and high risk of ischemic heart disease: is there an algorithm? *Diabetologia*, 53(4), 659-667. Link.
13. Stevens, S. R., Segar, M. W., Pandey, A., Lokhnygina, Y., Green, J. B., McGuire, D. K., Standl, E., Peterson, E. D., Holman, R. R. (2020). Development and validation of a model to predict cardiovascular death, nonfatal myocardial infarction, or nonfatal stroke in patients with type 2 diabetes mellitus and established atherosclerotic cardiovascular disease. *Diabetes Care*, 43(6), 1233-1240. Link.
14. Choudhury, R.P., Akbar, N. (2021). Beyond Diabetes: A Relationship between Cardiovascular Outcomes and Glycaemic Index. *Cardiovasc. Res.*, 117, E97–E98. Link.
15. Ordonez, C. (2006). Association Rule Discovery with the Train and Test Approach for Heart Disease Prediction. *IEEE Trans. Inf. Technol. Biomed.*, 10, 334–343. Link.
16. Magesh, G., Swarnalatha, P. (2021). Optimal Feature Selection through a Cluster-Based DT Learning (CDTL) in Heart Disease Prediction. *Evol. Intell.*, 14, 583–593. Link.

17. Chowdary, K.R., Bhargav, P., Nikhil, N., Varun, K., Jayanthi, D. (2022). Early Heart Disease Prediction Using Ensemble Learning Techniques. *J. Phys. Conf. Ser.*, 2325, 012051. Link.
18. Liu, J., Dong, X., Zhao, H., Tian, Y. (2022). Predictive Classifier for Cardiovascular Disease Based on Stacking Model Fusion. *Processes*, 10, 749. Link.
19. Devi, A.G. (2021). A Method of Cardiovascular Disease Prediction Using Machine Learning. *Int. J. Eng. Res. Technol.*, 9, 243–246. Link.
20. Pasha et al. (2020). "2.5 times higher risk of IHD in diabetic individuals compared to non-diabetic patients." Link.
21. Song et al. (2019). "Confirmed 2.5 times higher likelihood of developing IHD in diabetic individuals." Link.
22. Choudhury & Akbar (2021). "Discussed healthcare access issues in Bangladesh and the associated risks for IHD." Link.
23. Rahman et al. (2019). "Found that diabetes contributes significantly to IHD, but without focusing specifically on IHD in diabetic populations." Link.
24. Hossain et al. (2021). "Highlighted the relationship between low-income status and increased IHD risk in Bangladeshi populations." Link.
25. Uddin et al. (2020). "Emphasized the lack of screening programs in Bangladesh for diabetes and IHD, contributing to higher undiagnosed IHD in diabetic individuals." Link.
26. Bhatti et al. (2020). "Investigated socioeconomic factors and healthcare access affecting IHD prevalence in Bangladesh." Link.
27. **Lee, J., et al. (2022).** "Machine learning models for predicting acute myocardial infarction and ischemic heart disease." *PLOS Journals.*
Available at: https://journals.plos.org
28. **Zhou, J., et al. (2023).** "Deep learning techniques in predicting ischemic heart disease in diabetic patients." *IEEE Xplore.*
Available at: https://ieeexplore.ieee.org
29. **Hossain, M., et al. (2021).** "Machine learning models for diagnosing ischemic heart disease in diabetic patients in Bangladesh." *Oxford Academic.*
Available at: https://academic.oup.com
30. **Sharma, R., et al. (2022).** "Addressing class imbalance in machine learning models for IHD diagnosis using SMOTE." *PLOS Journals.*
Available at: https://journals.plos.org
31. **Faculty of Medicine (2022).** "Pathophysiology of ischemic heart disease in diabetic patients." *Faculty of Medicine.*
Available at: https://www.facmed.edu
32. **Egyptian Journal of Hospital Medicine (2022).** "Atherosclerotic plaque burden and mortality in diabetic patients." *Egyptian Journal of Hospital Medicine.*
Available at: https://www.ejhm.journals.com
33. **Revista Española de Cardiología (2021).** "The impact of hypertension and antithrombotic treatment on ischemic heart disease in diabetic patients." *Revista Española de Cardiología.*
Available at: https://www.revespcardiol.org
34. **European Society of Cardiology (2023).** "The synergistic effect of hypertension and diabetes on cardiovascular mortality." *European Society of Cardiology.*
Available at: https://www.escardio.org

35. **American Heart Association (2023).** "Hypertension and diabetes: a dual cardiovascular risk factor." *American Heart Association.*
Available at: https://www.heart.org
36. **Jose, R., et al. (2024).** "Machine learning in cardiovascular health management for diabetic patients." *MDPI.*
Available at: https://www.mdpi.com

Appendices

Appendix A: Data Preprocessing Steps

1. Handling Missing Data
RBS (Random Blood Sugar): Only one missing value was found in the RBS column. This was imputed using the median value of the column to ensure consistency.

2. Outlier Detection and Handling
IQR Method: The Interquartile Range (IQR) method was used to detect outliers. Any values that fell outside the range of Q1 - 1.5*IQR and Q3 + 1.5*IQR were considered outliers and replaced with the median value for that column to reduce their impact on the models.
Winsorization: For the RBS column, winsorization was applied by capping extreme values at the 5th and 95th percentiles to minimize the effect of outliers.

3. Variable Transformation
Categorical Data: Variables such as sex, occupation, and education level were one-hot encoded to allow machine learning models to effectively process these features as binary columns.

Appendix B: Machine Learning Model Details

1. Models Evaluated
Random Forest:
Accuracy: 91.0%
AUC-ROC: 0.94
Confusion Matrix: [[93, 7], [11, 89]]
Gradient Boosting:
Accuracy: 92.0%
AUC-ROC: 0.9694
Confusion Matrix: [[94, 6], [11, 89]]
LightGBM:
Accuracy: 90.5%
AUC-ROC: 0.91
Confusion Matrix: [[94, 6], [13, 87]]

2. Performance Metrics
Accuracy: The proportion of correct predictions made by the model.
AUC-ROC: The ability of the model to distinguish between IHD and non-IHD cases.
Precision: The proportion of true positives out of all predicted positives.
Recall: The proportion of true positives out of all actual positives.
F1-Score: The harmonic mean of precision and recall.

Appendix C: Statistical Tests

1. Fisher's Exact Test
Odds Ratio (OR): 4.31 (p-value: 1.53e-05)
A strong association was found between diabetes mellitus (DM) and ischemic heart disease (IHD), with diabetic individuals being 4.3 times more likely to develop IHD.

2. McNemar's Test
p-value: 1.67e-21
This test confirmed the significance of the DM-IHD relationship, further supporting the strong association.

3. Chi-Square Test
Chi-Square Value: 17.62 (p-value: 2.7e-05)
The test confirmed that there is a statistical dependency between DM status and IHD status.

Appendix D: SMOTE (Synthetic Minority Over-sampling Technique)

1. Application of SMOTE
Class Imbalance: The dataset had a higher number of non-IHD cases compared to IHD cases, so SMOTE was applied to balance the dataset by oversampling the minority class (IHD).

2. SMOTE Process
Before SMOTE: The minority class (IHD) had 49 instances, while the majority class (non-IHD) had 273 instances.
After SMOTE: The IHD class was oversampled to 273 instances, achieving balance in the dataset.

Appendix E: Cross-Validation Results

1. Stratified k-fold Cross-validation
Number of Folds: 5
Cross-validation Accuracy: 90.38% (standard deviation: 0.85%)
The model showed consistent performance across the different folds of the data, with minimal variation in accuracy, demonstrating its robustness.

Appendix F: Model Hyperparameters

1. Hyperparameters for Random Forest
n_estimators: 100
max_depth: None
min_samples_split: 2
min_samples_leaf: 1
bootstrap: True
2. Hyperparameters for Gradient Boosting
learning_rate: 0.01
max_depth: 5
min_samples_split: 5

n_estimators: 300
subsample: 0.8

3. Hyperparameters for LightGBM

learning_rate: 0.01
num_leaves: 31
n_estimators: 200
subsample: 0.8
colsample_bytree: 1.0

Appendix G: Model Performance Comparison

1. Accuracy and AUC-ROC Comparison

Model	Accuracy	AUC-ROC
Gradient Boosting	92.0%	0.9694
Random Forest	91.0%	0.94
LightGBM	90.5%	0.91

2. Confusion Matrix Comparison

Model	Class 0 (Non-IHD)	Class 1 (IHD)
Gradient Boosting	[94, 6]	[11, 89]
Random Forest	[93, 7]	[11, 89]
LightGBM	[94, 6]	[13, 87]

Appendix H: Future Directions

1. Data Expansion

Multi-center Clinical Records: Future research should focus on incorporating data from multiple centers to increase the generalizability of the model.

2. Deep Learning Exploration

Future studies should explore deep learning models like CNNs (Convolutional Neural Networks) for enhanced feature extraction from medical images, which could improve the accuracy of IHD prediction.

3. Real-time Implementation

The development of a real-time AI-powered risk assessment tool could significantly enhance the ability to predict IHD in diabetic populations, making it an integral part of digital health solutions.

Setting up my environment for machine learning with some common libraries.

```
# General Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import joblib

# Scikit-Learn Libraries
from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder, LabelEncoder, OrdinalEncoder
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc, log_loss, precision_recall_curve
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression, RidgeClassifier, LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from lightgbm import LGBMClassifier
from sklearn.svm import SVC
import lightgbm as lgb
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, ExtraTreesClassifier, AdaBoostClassifier, VotingClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDiscriminantAnalysis
from sklearn.multioutput import ClassifierChain
from sklearn.feature_selection import RFE, mutual_info_classif
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.preprocessing import label_binarize
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.linear_model import Ridge

# SciPy Libraries
from scipy.stats import fisher_exact
from scipy.stats.mstats import winsorize

# Statsmodels Library
from statsmodels.stats.contingency_tables import mcnemar
### Ignore Warnings
import warnings
warnings.filterwarnings("ignore")
```

Data Preprocessing

Load the Data Set

```
# Load the SPSS file
data = pd.read_spss('ASDS project Data.sav')

data.info()
```

Drop unnecessary columns

```
# Drop unnecessary columns
data =data.drop(columns=['Serial_No','Name'])
```

Covert Name

```
data.columns

# New column names should match the order of the old columns
# Old column names
old_columns = [
    'Age', 'Sex','Occupation','Education','Economy ','Hight','weight',
    'Systolic_Upper','Diastolic_Lower','RBS','Smoking','HTN','DM','Dyslipidemia','Stroke','IHD','Age_group',
    'BMI_Group','Hypertenson_stage']

new_columns_reordered = [
    'Age (Years)', 'Sex (Male/Female)', 'Occupation Type', 'Education Level', 'Economic Status',
    'Height (cm)', 'Weight (kg)', 'Systolic Blood Pressure (mmHg)', 'Diastolic Blood Pressure (mmHg)',
    'Random Blood Sugar (mg/dL)', 'Smoking Status', 'Hypertension (HTN) Status',
    'Diabetes Mellitus (DM) Status', 'Dyslipidemia Status', 'Stroke Status', 'Ischemic Heart Disease (IHD) Status',
    'Age Group', 'Body Mass Index (BMI) Group', 'Hypertension Stage']



# Check if the lengths match
print(f"Old Columns: {len(old_columns)}, New Columns: {len(new_columns_reordered)}")

# Assuming the DataFrame is called data
data.columns = new_columns_reordered
```

Cheak data Shape

```
data.shape
```

Cheak missing values

```
# Check for missing values
missing_values = data.isnull().sum()

# Display columns with missing values
print("Missing Values in Each Column:")
print(missing_values[missing_values > 0])

# Display the total number of missing values
total_missing = data.isnull().sum().sum()
print(f"\nTotal Missing Values in the Dataset: {total_missing}")
```

Handle Missing Values

```
data['Random Blood Sugar (mg/dL)'].fillna(data['Random Blood Sugar (mg/dL)'].median(), inplace=True)
```

Cheaking duplicate columns

```
# Check for duplicate columns
duplicate_columns = data.columns[data.columns.duplicated()].unique()

# Display the duplicate columns (if any)
if len(duplicate_columns) > 0:
    print(f"Duplicate columns found: {duplicate_columns}")
else:
    print("No duplicate columns found.")
```

Check for duplicate rows

```
# Check for duplicate rows
duplicate_rows = data[data.duplicated()]

# Display the duplicate rows (if any)
if not duplicate_rows.empty:
    print("Duplicate rows found:")
    print(duplicate_rows)
else:
    print("No duplicate rows found.")
```

Outliners decetion and Removal

Plot Oulters

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Set up the plotting grid
num_cols = data.select_dtypes(include=[np.number]).columns
fig, axes = plt.subplots(nrows=len(num_cols), ncols=2, figsize=(10, len(num_cols)*5))

# Loop through numeric columns and plot histograms and scatter plots
for i, col in enumerate(num_cols):
    # Histogram
    sns.histplot(data[col], kde=True, ax=axes[i, 0])
    axes[i, 0].set_title(f'Histogram of {col}')

    # Detect outliers using IQR method
    Q1 = data[col].quantile(0.25)
    Q3 = data[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Identify outliers
    outliers = data[(data[col] < lower_bound) | (data[col] > upper_bound)]

    # Scatter plot with outliers in a different color
    sns.scatterplot(x=data.index, y=data[col], ax=axes[i, 1], color=['skyblue'])  # Normal points
    sns.scatterplot(x=outliers.index, y=outliers[col], ax=axes[i, 1], color='red')  # Outliers in red
    axes[i, 1].set_title(f'Scatter Plot of {col} with Outliers')

plt.tight_layout()
plt.show()
```

Remove outliers

```
# IQR-based outlier detection
def detect_outliers_iqr(data):
    numerical_data = data.select_dtypes(include=[np.number])
    Q1 = numerical_data.quantile(0.25)
    Q3 = numerical_data.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Identify outliers
    outliers = ((numerical_data < lower_bound) | (numerical_data > upper_bound)).sum()
```

```
    return outliers

# IQR-based outlier detection and replacing with median
def impute_outliers_with_median(data, max_iterations=5):
    numerical_data = data.select_dtypes(include=[np.number])
    for iteration in range(max_iterations):
        Q1 = numerical_data.quantile(0.25)
        Q3 = numerical_data.quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Identify and replace outliers with the median for each column
        for column in numerical_data.columns:
            outliers = (numerical_data[column] < lower_bound[column]) | (numerical_data[column] > upper_bound[column])
            if outliers.sum() > 0:
                median_value = numerical_data[column].median()
                numerical_data[column] = numerical_data[column].where(~outliers, median_value)

        # After replacing outliers, check if there are any outliers left
        outliers_after = detect_outliers_iqr(numerical_data)
        if all(outliers_after == 0):
            print(f"Outliers have been handled after {iteration + 1} iterations.")
            break
    # Return the modified dataset with outliers replaced by the median
    data[numerical_data.columns] = numerical_data
    return data

# Check outliers in the original dataset
outliers_before = detect_outliers_iqr(data)
print("Outliers detected before imputation:")
print(outliers_before)

# Apply the iterative imputation method
data_imputed = impute_outliers_with_median(data)

# Check outliers after imputation
outliers_after = detect_outliers_iqr(data_imputed)
print("\nOutliers detected after iterative median imputation:")
print(outliers_after)

# Verify if there are any outliers (should be 0 for each column)
if all(outliers_after == 0):
    print("\nNo outliers detected after iterative imputation.")
else:
    print("\nThere are still outliers present.")
```

If there are still outliers present ,remove Again

```
# Apply Winsorization to cap the extreme 5% values on both sides
data['Random Blood Sugar (mg/dL)'] = winsorize(data['Random Blood Sugar (mg/dL)'], limits=(0.05, 0.05))

# Check for outliers after Winsorization
outliers_after_winsorization = detect_outliers_iqr(data)

print("\nOutliers detected after Winsorization:")
print(outliers_after_winsorization)

# Verify if there are any outliers left in the "Random Blood Sugar (mg/dL)" column
if outliers_after_winsorization['Random Blood Sugar (mg/dL)'] == 0:
    print("\nAll outliers in 'Random Blood Sugar (mg/dL)' have been handled.")
else:
    print("\nThere are still outliers present in 'Random Blood Sugar (mg/dL)'.")
```

Descriptive Statistics

Summery of Statistical Numberical Colums

```
data.describe(include=[np.number]).T

data.info()
```

Summery of Statistical catagorical Colums

```
data.describe(include=['category','object']).T
```

Plotting for categorical variables

```
# Set up the plotting for categorical variables
cat_cols = data.select_dtypes(include=[object, 'category']).columns
fig_cat, axes_cat = plt.subplots(nrows=len(cat_cols), ncols=1, figsize=(10, len(cat_cols)*5))

# Loop through categorical columns and plot bar plots
for i, col in enumerate(cat_cols):
    ax = axes_cat[i] if len(cat_cols) > 1 else axes_cat

    # Countplot with multiple colors
    sns.countplot(x=data[col], ax=ax, palette=['skyblue','lightcoral','red','olive'])  # You can change the palette to any you prefer
```

```
    # Title for the plot
    ax.set_title(f'Bar Plot of {col}')

    # Calculate percentages
    total = len(data[col])
    for p in ax.patches:
        height = p.get_height()
        percentage = (height / total) * 100
        ax.text(p.get_x() + p.get_width() / 2, height + 1, f'{percentage:.2f}%', ha='center', va='bottom')

plt.tight_layout()
plt.show()
```

Plotting for neumerical variables

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Set up the plotting grid
num_cols = data.select_dtypes(include=[np.number]).columns
fig, axes = plt.subplots(nrows=len(num_cols), ncols=2, figsize=(10, len(num_cols)*5))

# Loop through numeric columns and plot histograms and scatter plots
for i, col in enumerate(num_cols):
    # Histogram
    sns.histplot(data[col], kde=True, ax=axes[i, 0])
    axes[i, 0].set_title(f'Histogram of {col}')

    # Detect outliers using IQR method
    Q1 = data[col].quantile(0.25)
    Q3 = data[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Identify outliers
    outliers = data[(data[col] < lower_bound) | (data[col] > upper_bound)]

    # Scatter plot with outliers in a different color
    sns.scatterplot(x=data.index, y=data[col], ax=axes[i, 1], color=['skyblue'])  # Normal points
    sns.scatterplot(x=outliers.index, y=outliers[col], ax=axes[i, 1], color='red')  # Outliers in red
    axes[i, 1].set_title(f'Scatter Plot of {col} with Outliers')

plt.tight_layout()
plt.show()
```

Define the terget Variable

Distribution of terget Variable Ischemic Heart Disease (IHD) Status

```
data['Ischemic Heart Disease (IHD) Status'].unique()
```

```
# Data
IHD_counts = data['Ischemic Heart Disease (IHD) Status'].value_counts()
IHD_percentage = IHD_counts / IHD_counts.sum() * 100

# Pie Chart with Shadows (Simulated 3D Effect)
plt.figure(figsize=(6, 4))
colors = ['skyblue','lightcoral','red','olive'][:len(IHD_counts)]

plt.pie(
    IHD_counts,
    labels=IHD_counts.index,
    autopct='%1.1f%%',
    startangle=90,
    colors=colors,
    explode=[0.1] * len(IHD_counts),  # Slightly separate each slice
    shadow=True  # Add shadow for depth
)
plt.title('Ischemic Heart Disease (IHD) Status')
plt.axis('equal')  # Ensures the pie chart is circular
plt.show()

# Bar Chart with Enhancements
fig, ax = plt.subplots(figsize=(5, 4))
colors = sns.color_palette("cool", len(IHD_counts))  # Gradient color palette

bars = ax.bar(IHD_counts.index, IHD_counts.values, color=['skyblue','lightcoral','red','olive'], edgecolor='white', linewidth=1.2)

# Annotate bars
for bar, percentage in zip(bars, IHD_percentage):
    ax.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height() + 0.5,
        f'{int(bar.get_height())} ({percentage:.1f}%)',
        ha='center',
```

```
            va='bottom',
            fontsize=10
        )

    ax.set_title('Ischemic Heart Disease (IHD) Status', fontsize=14, weight='bold')
    ax.set_ylabel('Count', fontsize=12)
    ax.set_xlabel('Ischemic Heart Disease (IHD) Status', fontsize=12)
    plt.xticks(rotation=0, fontsize=10)
    sns.despine()  # Remove top and right spines for a cleaner look
    plt.show()

    # Define categorical and numerical columns
    categorical_cols = [
        'Sex (Male/Female)', 'Occupation Type', 'Education Level',
        'Economic Status', 'Smoking Status', 'Hypertension (HTN) Status',
        'Diabetes Mellitus (DM) Status', 'Dyslipidemia Status', 'Stroke Status',
        'Ischemic Heart Disease (IHD) Status', 'Age Group',
        'Body Mass Index (BMI) Group', 'Hypertension Stage'
    ]

    numerical_cols = ['Age (Years)', 'Height (cm)', 'Weight (kg)',
           'Systolic Blood Pressure (mmHg)', 'Diastolic Blood Pressure (mmHg)',
           'Random Blood Sugar (mg/dL)']  # Example numerical columns

    # Set a stylish background with a light gradient and muted gridlines
    sns.set_theme(style="whitegrid", palette="muted", font_scale=1.2)
    plt.rcParams['axes.facecolor'] = '#f9f9f9'  # Light background for axes
    plt.rcParams['figure.facecolor'] = '#eaeaea'  # Subtle background for the figure
    plt.rcParams['grid.color'] = '#d3d3d3'  # Softer gridline colors
    plt.rcParams['axes.edgecolor'] = '#a1a1a1'  # Slightly darker edges for contrast

    # Create subplots
    num_plots = len(categorical_cols) + len(numerical_cols)
    fig, axes = plt.subplots(
        nrows=num_plots,
        ncols=1,
        figsize=(12, num_plots * 4),  # Adjusted figsize calculation
        constrained_layout=True
    )

    # Plot categorical features with percentages
    for idx, col in enumerate(categorical_cols):
        ax = axes[idx]
        sns.countplot(
            data=data,
            x=col,
            hue='Ischemic Heart Disease (IHD) Status',
            palette="coolwarm",  # Stylish blue-red color palette
            ax=ax,
            dodge=True,
            width=0.6  # Slimmer bars
        )

        # Add percentage annotations
        total = len(data)
        for p in ax.patches:
            height = p.get_height()
            if height > 0:  # Avoid division by zero
                percentage = (height / total) * 100
                ax.annotate(
                    f'{percentage:.0f}%',  # Rounded percentage
                    (p.get_x() + p.get_width() / 2., height),
                    ha='center', va='bottom', fontsize=9, color='#333333'
                )

        # Customize the plot
        ax.set_title(f'{col} vs Ischemic Heart Disease (IHD)', fontsize=14, weight='bold')
        ax.set_xlabel(col, fontsize=12)
        ax.set_ylabel('Count', fontsize=12)
        ax.legend(title='Ischemic Heart Disease (IHD)', fontsize=10)
        ax.tick_params(axis='x', rotation=30)

    # Plot numerical features as boxplots
    for idx, col in enumerate(numerical_cols, start=len(categorical_cols)):
        ax = axes[idx]
        sns.boxplot(data=data, x='Ischemic Heart Disease (IHD) Status', y=col, palette="coolwarm", ax=ax, width=0.5)  # Slimmer boxes

        # Customize the plot
        ax.set_title(f'{col} vs Ischemic Heart Disease (IHD)', fontsize=14, weight='bold')
        ax.set_xlabel('Ischemic Heart Disease', fontsize=12)
        ax.set_ylabel(col, fontsize=12)
        ax.grid(axis='y', linestyle='--', alpha=0.6)

    # Final adjustments
    fig.suptitle('Ischemic Heart Disease (IHD) Status Analysis', fontsize=16, weight='bold')
    plt.show()
```

Diabetes Mellitus (DM) Status

```
data['Diabetes Mellitus (DM) Status'].unique()
```

```
import matplotlib.pyplot as plt

# Data
DM_counts = data['Diabetes Mellitus (DM) Status'].value_counts()
DM_percentage = DM_counts / DM_counts.sum() * 100

# Pie Chart with Shadows (Simulated 3D Effect)
plt.figure(figsize=(8, 6))
colors = ['skyblue','lightcoral','red','olive'][:len(IHD_counts)]

plt.pie(
    DM_counts,
    labels=DM_counts.index,
    autopct='%1.1f%%',
    startangle=90,
    colors=colors,
    explode=[0.1] * len(DM_counts),  # Slightly separate each slice
    shadow=True  # Add shadow for depth
)
plt.title('Diabetes Mellitus (DM) Status - Simulated 3D Pie Chart')
plt.axis('equal')  # Ensures the pie chart is circular
plt.show()

import matplotlib.pyplot as plt
import seaborn as sns

# Bar Chart with Enhancements
fig, ax = plt.subplots(figsize=(4, 6))
colors = sns.color_palette("cool", len(DM_counts))  # Gradient color palette

bars = ax.bar(DM_counts.index, DM_counts.values, color=['skyblue','lightcoral','red','olive'], edgecolor='white', linewidth=1.2)

# Annotate bars
for bar, percentage in zip(bars, DM_percentage):
    ax.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height() + 0.5,
        f'{int(bar.get_height())} ({percentage:.1f}%)',
        ha='center',
        va='bottom',
        fontsize=10
    )

ax.set_title('Diabetes Mellitus (DM) Status- Enhanced Bar Chart', fontsize=14, weight='bold')
ax.set_ylabel('Count', fontsize=12)
ax.set_xlabel('Diabetes Mellitus (DM) Status', fontsize=12)
plt.xticks(rotation=0, fontsize=10)
sns.despine()  # Remove top and right spines for a cleaner look
plt.show()

# Define categorical and numerical columns
categorical_cols = [
    'Sex (Male/Female)', 'Occupation Type', 'Education Level',
    'Economic Status', 'Smoking Status', 'Hypertension (HTN) Status',
    'Diabetes Mellitus (DM) Status', 'Dyslipidemia Status', 'Stroke Status',
    'Ischemic Heart Disease (IHD) Status', 'Age Group',
    'Body Mass Index (BMI) Group', 'Hypertension Stage'
]

numerical_cols = ['Age (Years)', 'Height (cm)', 'Weight (kg)',
       'Systolic Blood Pressure (mmHg)', 'Diastolic Blood Pressure (mmHg)',
       'Random Blood Sugar (mg/dL)']  # Example numerical columns

# Set a stylish background with a light gradient and muted gridlines
sns.set_theme(style="whitegrid", palette="muted", font_scale=1.2)
plt.rcParams['axes.facecolor'] = '#f9f9f9'  # Light background for axes
plt.rcParams['figure.facecolor'] = '#eaeaea'  # Subtle background for the figure
plt.rcParams['grid.color'] = '#d3d3d3'  # Softer gridline colors
plt.rcParams['axes.edgecolor'] = '#a1a1a1'  # Slightly darker edges for contrast

# Create subplots
num_plots = len(categorical_cols) + len(numerical_cols)
fig, axes = plt.subplots(
    nrows=num_plots,
    ncols=1,
    figsize=(12, num_plots * 4),  # Adjusted figsize calculation
    constrained_layout=True
)

# Plot categorical features with percentages
for idx, col in enumerate(categorical_cols):
    ax = axes[idx]
    sns.countplot(
        data=data,
        x=col,
        hue='Diabetes Mellitus (DM) Status',
        palette="coolwarm",  # Stylish blue-red color palette
        ax=ax,
        dodge=True,
```

```
        width=0.6  # Slimmer bars
    )

    # Add percentage annotations
    total = len(data)
    for p in ax.patches:
        height = p.get_height()
        if height > 0:  # Avoid division by zero
            percentage = (height / total) * 100
            ax.annotate(
                f'{percentage:.0f}%',  # Rounded percentage
                (p.get_x() + p.get_width() / 2., height),
                ha='center', va='bottom', fontsize=9, color='#333333'
            )

    # Customize the plot
    ax.set_title(f'{col} vs Ischemic Heart Disease (IHD)', fontsize=14, weight='bold')
    ax.set_xlabel(col, fontsize=12)
    ax.set_ylabel('Count', fontsize=12)
    ax.legend(title='Diabetes Mellitus (DM) Status', fontsize=10)
    ax.tick_params(axis='x', rotation=30)

# Plot numerical features as boxplots
for idx, col in enumerate(numerical_cols, start=len(categorical_cols)):
    ax = axes[idx]
    sns.boxplot(data=data, x='Diabetes Mellitus (DM) Status', y=col, palette="coolwarm", ax=ax, width=0.5)  # Slimmer boxes

    # Customize the plot
    ax.set_title(f'{col} vs Ischemic Heart Disease (IHD)', fontsize=14, weight='bold')
    ax.set_xlabel('Diabetes Mellitus (DM) Status', fontsize=12)
    ax.set_ylabel(col, fontsize=12)
    ax.grid(axis='y', linestyle='--', alpha=0.6)

# Final adjustments
fig.suptitle('Diabetes Mellitus (DM) Status Analysis', fontsize=16, weight='bold')
plt.show()

# Remove rows where the 'Diabetes Mellitus (DM) Status' is 'Unknown'
data = data[data['Diabetes Mellitus (DM) Status'] != 'Unknown']
```

Inferential Statistics

Relation between Diabetes Mellitus (DM) Status & Ischemic Heart Disease (IHD) Status

```
# Set a stylish background with a light gradient and muted gridlines
sns.set_theme(style="whitegrid", palette="muted", font_scale=1.2)
plt.rcParams['axes.facecolor'] = '#f9f9f9'  # Light gradient background for the axes
plt.rcParams['figure.facecolor'] = '#eaeaea'  # Subtle background for the overall figure
plt.rcParams['grid.color'] = '#d3d3d3'  # Softer gridline colors
plt.rcParams['axes.edgecolor'] = '#a1a1a1'  # Slightly darker edges for contrast

# Cross-tabulation between DM status and IHD status
DM_IHD_risk = pd.crosstab(data['Diabetes Mellitus (DM) Status'], data['Ischemic Heart Disease (IHD) Status'], margins=True,
margins_name="Total")

# Drop the Total row for plotting
DM_IHD_risk.drop('Total', axis=1, inplace=True)

# Plotting a clustered bar chart for the relationship
ax = DM_IHD_risk.plot(kind='bar', figsize=(6, 4), color=['skyblue', 'lightcoral'], width=0.8)

# Title and labels
plt.title('Clustered Bar Chart: Diabetes Mellitus (DM) Status vs Ischemic Heart Disease (IHD) Status')
plt.ylabel('Count')
plt.xlabel('Diabetes Mellitus (DM) Status')
plt.xticks(rotation=0)
plt.legend(title='IHD Status', loc='upper left', labels=['No IHD', 'IHD'])

# Adding percentage annotations on top of each bar
for p in ax.patches:
    height = p.get_height()
    percentage = (height / DM_IHD_risk.sum().sum()) * 100  # Calculate percentage of total
    ax.annotate(f'{height:.0f}\n({percentage:.1f}%)',
                (p.get_x() + p.get_width() / 2., height),
                ha='center', va='center', fontsize=10,
                color='black', xytext=(0, 5), textcoords='offset points')

# Show the plot
plt.show()
```

Hypothesis Test Interpretation

Fisher's Exact Test

Fisher's Exact Test is particularly useful when dealing with small sample sizes or sparse data. It tests the null hypothesis of independence between two categorical variables.

```
# Create the contingency table again
DM_IHD_contingency = pd.crosstab(data['Diabetes Mellitus (DM) Status'], data['Ischemic Heart Disease (IHD) Status'])

# Perform Fisher's Exact Test
oddsratio, p_value = fisher_exact(DM_IHD_contingency)

# Output the results
print(f"Odds Ratio: {oddsratio}")
print(f"P-value: {p_value}")

# Interpretation
alpha = 0.05  # Significance level
if p_value < alpha:
    print("\nThere is a significant relationship between Diabetes Mellitus (DM) Status and Ischemic Heart Disease (IHD) Status.")
else:
    print("\nThere is no significant relationship between Diabetes Mellitus (DM) Status and Ischemic Heart Disease (IHD) Status.")
```

McNemar's Test

McNemar's Test is useful when you have paired nominal data (for example, pre/post intervention data or paired observations). This method is typically applied to 2x2 contingency tables where data points are dependent.

```
# Step 1: Create the contingency table (cross-tabulation)
DM_IHD_contingency = pd.crosstab(data['Diabetes Mellitus (DM) Status'], data['Ischemic Heart Disease (IHD) Status'])

# Step 2: Perform McNemar's Test
result = mcnemar(DM_IHD_contingency, exact=True)

# Step 3: Output the results
print(f"McNemar's Test Statistic: {result.statistic}")
print(f"P-value from McNemar's Test: {result.pvalue}")

# Step 4: Hypothesis Test Interpretation
alpha = 0.05  # Significance level
if result.pvalue < alpha:
    print("\nThere is a significant relationship between Diabetes Mellitus (DM) Status and Ischemic Heart Disease (IHD) Status.")
else:
    print("\nThere is no significant relationship between Diabetes Mellitus (DM) Status and Ischemic Heart Disease (IHD) Status.")
```

The Chi-Square Test of Independence

```
import pandas as pd
import scipy.stats as stats

# Step 1: Create the contingency table (cross-tabulation) between DM status and IHD status
DM_IHD_contingency = pd.crosstab(data['Diabetes Mellitus (DM) Status'], data['Ischemic Heart Disease (IHD) Status'])

# Step 2: Perform the Chi-Square Test of Independence
chi2_stat, p_value, dof, expected = stats.chi2_contingency(DM_IHD_contingency)

# Step 3: Output the results
print(f"Chi-Square Statistic: {chi2_stat}")
print(f"Degrees of Freedom: {dof}")
print(f"Expected Frequencies:\n{expected}")
print(f"P-value: {p_value}")

# Step 4: Hypothesis Test Interpretation
alpha = 0.05  # Significance level

if p_value < alpha:
    print("\nThere is a significant relationship between Diabetes Mellitus (DM) Status and Ischemic Heart Disease (IHD) Status.")
else:
    print("\nThere is no significant relationship between Diabetes Mellitus (DM) Status and Ischemic Heart Disease (IHD) Status")
```

One-hot encoding for the categorical features

```
# Define the categorical features
Categorical_Features = ['Sex (Male/Female)', 'Occupation Type', 'Education Level',
       'Economic Status', 'Smoking Status', 'Hypertension (HTN) Status',
       'Diabetes Mellitus (DM) Status', 'Dyslipidemia Status', 'Stroke Status',
       'Ischemic Heart Disease (IHD) Status', 'Age Group',
       'Body Mass Index (BMI) Group', 'Hypertension Stage']

# Perform one-hot encoding for the categorical features
data = pd.get_dummies(data, columns=Categorical_Features, drop_first=True)

# Display the resulting DataFrame
data.info()
```

Filter the data to include only rows where 'Diabetes Mellitus (DM) Status_Yes'

```
# Filter the data to include only rows where 'Diabetes Mellitus (DM) Status_Yes' is True (1)
df= data[data['Diabetes Mellitus (DM) Status_Yes'] == 1]

# Display the first few rows of the new dataset
df.head().T
```

```
df.shape
```

Correlation matrix

```
# Select only numerical columns
numerical_data = df.select_dtypes(include=[np.number])

# Compute correlation matrix
correlation_matrix = numerical_data.corr()

# Plot the heatmap for visualization
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True)
plt.title("Correlation Matrix Heatmap")
plt.show()

# Set a correlation threshold
correlation_threshold = -.5

# Identify highly correlated features
upper_triangle = correlation_matrix.where(np.triu(np.ones(correlation_matrix.shape), k=1).astype(bool))

# Find index of feature columns with correlation greater than the threshold
highly_correlated_features = [column for column in upper_triangle.columns if any(upper_triangle[column] > correlation_threshold)]

# Print the names of features being dropped
print(f"Features being dropped (correlation > {correlation_threshold}): {highly_correlated_features}")

# Drop highly correlated features from the original dataset
filtered_datacor = df.drop(columns=highly_correlated_features)

# Output the filtered DataFrame
filtered_datacor.info()
```

Feature Selection

Split it into features (X) and target (y)

```
X = df.drop('Ischemic Heart Disease (IHD) Status_Yes', axis=1)  # Replace 'target' with your target column name
y = df['Ischemic Heart Disease (IHD) Status_Yes']
```

Mutual Information

```
# Encode categorical features in X
X_encoded = X.apply(lambda col: LabelEncoder().fit_transform(col) if col.dtypes == 'category' else col)

# Encode the target variable y if it's categorical
y_encoded = LabelEncoder().fit_transform(y)

# Calculate Mutual Information
mi = mutual_info_classif(X_encoded, y_encoded)

# Create a DataFrame to view the Mutual Information of each feature
mi_df = pd.DataFrame({'Feature': X.columns, 'Mutual Information': mi})
mi_df = mi_df.sort_values(by='Mutual Information', ascending=False)  # Rank features by MI

# Assuming mi_df is a DataFrame with features and their Mutual Information (MI) scores
mi_df = mi_df.sort_values(by="Mutual Information", ascending=False).reset_index(drop=True)

# Add a ranking column
mi_df["Rank"] = mi_df.index + 1

# Display the top features with rankings
print("Top Features ranked by Mutual Information:\n")
print(mi_df.head(25)[["Rank", "Feature", "Mutual Information"]])
```

Filter Features Based on MI Threshold

```
import pandas as pd
from sklearn.feature_selection import mutual_info_classif
from sklearn.preprocessing import LabelEncoder

# Define the threshold value for Mutual Information (e.g., 0.1)
mi_threshold = 0.01  # Assigning a value instead of using an invalid comparison

# Encode categorical features in X
X_encoded = X.apply(lambda col: LabelEncoder().fit_transform(col) if col.dtypes == 'category' else col)

# Encode the target variable y if it's categorical
```

```
y_encoded = LabelEncoder().fit_transform(y)

# Calculate Mutual Information
mi = mutual_info_classif(X_encoded, y_encoded)

# Create a DataFrame to view the Mutual Information of each feature
mi_df = pd.DataFrame({'Feature': X.columns, 'Mutual Information': mi})

# Filter features based on the MI threshold
filtered_features = mi_df[mi_df['Mutual Information'] >= mi_threshold]

# Sort by MI to see the most important features
filtered_features = filtered_features.sort_values(by='Mutual Information', ascending=False).reset_index(drop=True)

# Add a ranking column
filtered_features["Rank"] = filtered_features.index + 1

# Print the filtered features with rankings
print(f"Filtered Features (MI >= {mi_threshold}):\n")
print(filtered_features[["Rank", "Feature", "Mutual Information"]])
```

Visualize the selected features with a bar chart

```
# Visualize the selected features with a bar chart
plt.figure(figsize=(10, 6))
colors = sns.color_palette("cool", len(filtered_features))  # Use a gradient color palette

bar_plot = sns.barplot(
    x="Mutual Information",
    y="Feature",
    data=filtered_features,
    palette=colors
)

# Add percentages on top of the bars
for index, row in filtered_features.iterrows():
    bar_plot.text(
        row['Mutual Information'] + 0.001,  # Slightly offset to the right of the bar
        index,  # y-coordinate of the bar
        f"{row['Mutual Information']:.3f}",  # Display MI value with 3 decimal places
        color='black',
        ha="left",
        va="center",
        fontsize=10
    )

# Add labels and title
plt.title("Feature Importance Bar Chart (Mutual Information)", fontsize=10)
plt.xlabel("Mutual Information Score", fontsize=2)
plt.ylabel("Features", fontsize=12)

# Adjust layout for better visibility
plt.tight_layout()
plt.show()
```

Train the Final Model

```
# Step 1: Preprocess Data
# Convert categorical features to 'category' dtype
X = X.copy()
for col in X.select_dtypes(include=['object']).columns:
    X[col] = X[col].astype('category')

# Encode categorical features using OrdinalEncoder
encoder = OrdinalEncoder()
X_encoded = X.copy()
categorical_cols = X.select_dtypes(include=['category']).columns
X_encoded[categorical_cols] = encoder.fit_transform(X[categorical_cols])

# Encode target variable
y_encoded = LabelEncoder().fit_transform(y) if y.dtype == 'object' else y

# Step 2: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded)

# Step 3: Scale Numerical Features
scaler = StandardScaler()
X_train.iloc[:, :] = scaler.fit_transform(X_train)
X_test.iloc[:, :] = scaler.transform(X_test)

# Step 4: Train Model (Random Forest Classifier)
model = RandomForestClassifier(n_estimators=200, random_state=42)
model.fit(X_train, y_train)

# Step 5: Evaluate Model
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]  # For AUC-ROC

accuracy = accuracy_score(y_test, y_pred)
```

```
roc_auc = roc_auc_score(y_test, y_pred_proba)
classification_rep = classification_report(y_test, y_pred)

# Print results
print(f"Accuracy: {accuracy:.4f}")
print(f"ROC AUC Score: {roc_auc:.4f}")
print("Classification Report:\n", classification_rep)
```

Random Forest

```
# Initialize and train a Random Forest model
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Get feature importance from the trained Random Forest model
feature_importances = rf.feature_importances_

# Create a DataFrame to view the Feature Importance
importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False)  # Rank features by importance

# Assuming `importance_df` is a DataFrame with features and their importance scores
importance_df = importance_df.sort_values(by="Importance", ascending=False).reset_index(drop=True)

# Add a ranking column
importance_df["Rank"] = importance_df.index + 1

# Print the top  features with rankings
print("Random Forest Feature Importance (Top Features):\n")
print(importance_df.head()[["Rank", "Feature", "Importance"]])
```

Filter Features Based on Importance Threshold

```
# Set a threshold for feature importance
importance_threshold = 0.03  # Set your desired threshold value

# Filter features based on importance threshold
selected_features = importance_df[importance_df['Importance'] > importance_threshold]

# Assuming `selected_features` is a DataFrame with features and their importance scores
selected_features = selected_features.sort_values(by="Importance", ascending=False).reset_index(drop=True)

# Add a ranking column
selected_features["Rank"] = selected_features.index + 1

# Print the selected features with rankings
print("Selected Features Based on Importance Threshold:\n")
print(selected_features[["Rank", "Feature", "Importance"]])
```

Visualize the selected features with a bar chart

```
# Visualize the selected features with a bar chart
plt.figure(figsize=(10, 6))
colors = sns.color_palette("coolwarm", len(selected_features))  # Use a gradient color palette

bar_plot = sns.barplot(
    x="Importance",
    y="Feature",
    data=selected_features,
    palette=colors
)

# Add percentages on top of the bars
for index, row in selected_features.iterrows():
    bar_plot.text(
        row['Importance'] + 0.001,  # Slightly offset to the right of the bar
        index,  # y-coordinate of the bar
        f"{row['Importance'] * 100:.1f}%",  # Convert to percentage
        color='black',
        ha="left",
        va="center",
        fontsize=8
    )

# Add labels and title
plt.title("Feature Importance Bar Chart with Percentages", fontsize=10)
plt.xlabel("Importance Score", fontsize=12)
plt.ylabel("Features", fontsize=12)

# Adjust layout for better visibility
plt.tight_layout()
plt.show()
```

Train the Final Model

```
# One-Hot Encode categorical variables
encoder = OneHotEncoder(handle_unknown='ignore', sparse_output=False)  # 🔄 Use sparse_output
X_encoded = pd.DataFrame(encoder.fit_transform(X.select_dtypes(include=['object'])))
```

```
# Ensure column names are preserved
X_encoded.columns = encoder.get_feature_names_out()


# Concatenate encoded categorical features with numerical features
X_final = pd.concat([X_encoded, X.select_dtypes(exclude=['object']).reset_index(drop=True)], axis=1)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_final, y, test_size=0.2, random_state=42)

# Train Random Forest model
rf_final = RandomForestClassifier(n_estimators=100, random_state=42)
rf_final.fit(X_train, y_train)

# Ensure test features have the same column names as train features
X_test = X_test.reindex(columns=X_train.columns, fill_value=0)

# Predict on test set
y_pred = rf_final.predict(X_test)
y_pred_proba = rf_final.predict_proba(X_test)[:, 1]  # Use rf_final instead of model

# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_proba)  # Ensure roc_auc_score is imported

print(f'Accuracy: {accuracy:.4f}')
print(f"ROC AUC Score: {roc_auc:.4f}")

# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Create new dataset by List of important features based on Random Forest

```
# List of important features based on mutual information
important_features = [
    "Age (Years)",
    "Stroke Status_Yes",
    "Weight (kg)",
    "Random Blood Sugar (mg/dL)",
    "Height (cm)",
    "Systolic Blood Pressure (mmHg)",
    "Diabetes Mellitus (DM) Status_Yes",
    "Hypertension (HTN) Status_Yes",
    "Dyslipidemia Status_Yes",
    "Diastolic Blood Pressure (mmHg)",
    'Ischemic Heart Disease (IHD) Status_Yes'
]

# Filter the original dataset to retain only the important features
data_for_modeling = df[important_features]

# Print the new dataset's shape and preview
print("Shape of the new dataset for modeling:", data_for_modeling.shape)
print("Preview of the new dataset:")
print(data_for_modeling.head())

# Save the new dataset for future use (optional)
data_for_modeling.to_csv("data_for_modeling.csv", index=False)
```

Define New Dataset

```
df=data_for_modeling

df.info()
```

Test Train Split

Dara scaling for continuous variables

```
# Assuming your data is in a pandas DataFrame 'data'
scaler = MinMaxScaler()

# Defining the set of continuous variables
continuous_variables = {
    'Age (Years)',
    'Weight (kg)',
    'Height (cm)',
    'Systolic Blood Pressure (mmHg)',
    'Diastolic Blood Pressure (mmHg)',
    'Random Blood Sugar (mg/dL)'
}

# Assuming your DataFrame is 'data'
numerical_columns = list(continuous_variables)
```

```
# Check if all columns are present in the DataFrame
missing_columns = [col for col in numerical_columns if col not in data.columns]
if missing_columns:
    print(f"Missing columns in the DataFrame: {missing_columns}")

# Apply the scaler to the selected numerical columns
data[numerical_columns] = scaler.fit_transform(data[numerical_columns])
data.head(5).T

# Assuming 'datacor' is your dataset and 'Heart Failure (HF)' is the target variable
X = df.drop(columns=['Ischemic Heart Disease (IHD) Status_Yes'])  # Features
y = df['Ischemic Heart Disease (IHD) Status_Yes']                # Target variable

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Print the shapes of the resulting datasets
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

Beseline Funcations

```
# Generating synthetic dataset for demonstration
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define models
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Naive Bayes': GaussianNB(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Support Vector Machine': SVC(probability=True),
    'Ridge Classifier': RidgeClassifier(),
    'Quadratic Discriminant Analysis': QuadraticDiscriminantAnalysis(),
    'Linear Discriminant Analysis': LinearDiscriminantAnalysis(),
    'AdaBoost': AdaBoostClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'Extra Trees Classifier': ExtraTreesClassifier(),
    'LightGBM': lgb.LGBMClassifier(verbose=-1)
}

# Evaluate models
def evaluate_models(models, X_train, y_train, X_test, y_test):
    results = {}
    for name, model in models.items():
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        results[name] = accuracy
        print(f'{name}: Accuracy = {accuracy:.4f}')

    sorted_results = sorted(results.items(), key=lambda x: x[1], reverse=True)
    top_3_models = sorted_results[:3]
    print("\nTop 3 Models:")
    for model, acc in top_3_models:
        print(f'{model}: Accuracy = {acc:.4f}')

    best_model = top_3_models[0]
    print(f'\nBest Model: {best_model[0]} with Accuracy = {best_model[1]:.4f}')
    return results

# Run evaluation
results = evaluate_models(models, X_train, y_train, X_test, y_test)
```

Model building

Original class distribution

```
# Step 1: Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

Apply SMOTE for class imbalance handling

```
# Step 2: Print original class distribution
original_counts = Counter(y_train)
print("Original class distribution:", original_counts)

# Step 3: Apply SMOTE for class imbalance handling
smote = SMOTE(random_state=42)
```

```
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
smote_counts = Counter(y_train_smote)
print("Class distribution after SMOTE:", smote_counts)

# Step 4: Visualize class distribution before and after SMOTE
classes = list(original_counts.keys())
original_values = [original_counts[c] for c in classes]
smote_values = [smote_counts[c] for c in classes]

plt.figure(figsize=(8, 5))
plt.plot(classes, original_values, marker='o', linestyle='-', color='blue', label="Original")
plt.plot(classes, smote_values, marker='s', linestyle='--', color='red', label="After SMOTE")
plt.xlabel("Class Labels")
plt.ylabel("Count")
plt.title("Class Distribution Before and After SMOTE")
plt.legend()
plt.grid(True)

# Step 4: Visualize class distribution before and after SMOTE
classes = list(original_counts.keys())
original_values = [original_counts[c] for c in classes]
smote_values = [smote_counts[c] for c in classes]

# Create a bar chart
plt.figure(figsize=(8, 5))

# Bar chart for original class distribution
bar_width = 0.35  # Width of the bars
index = np.arange(len(classes))  # Set the x position for the bars

plt.bar(index, original_values, bar_width, color='blue', label="Original")
plt.bar(index + bar_width, smote_values, bar_width, color='red', label="After SMOTE")

plt.xlabel("Class Labels")
plt.ylabel("Count")
plt.title("Class Distribution Before and After SMOTE")

# Add x-axis labels with class names
plt.xticks(index + bar_width / 2, classes)

# Display the legend
plt.legend()

# Add gridlines
plt.grid(True)

# Show the plot
plt.tight_layout()
plt.show()
```

Gradient Boosting model

Initialize the Gradient Boosting model

```
# Initialize the Gradient Boosting model
gb_model = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3)
```

Define the hyperparameter grid for GridSearchCV

```
# Define parameter grid for Gradient Boosting
gb_params = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 1.0],
    'min_samples_split': [2, 5, 10]
}

gb = GradientBoostingClassifier()
gb_grid_search = GridSearchCV(gb, param_grid=gb_params, cv=5, scoring='accuracy', n_jobs=-1, verbose=1)
gb_grid_search.fit(X_train, y_train)
print(f"Best parameters for Gradient Boosting: {gb_grid_search.best_params_}")
print(f"Best accuracy for Gradient Boosting: {gb_grid_search.best_score_}")
```

Initialize the model with the best parameters

```
# Initialize the Gradient Boosting model with the parameters directly
gb_clf = GradientBoostingClassifier(
    learning_rate=0.01,
    max_depth=5,
    min_samples_split=10,
    n_estimators=100,
    subsample=0.8
)

# Fit the model with the training data (after SMOTE)
gb_clf.fit(X_train_smote, y_train_smote)
```

```
# Step 10: Predict on the test data
y_pred = gb_clf.predict(X_test)
y_prob = gb_clf.predict_proba(X_test)[:, 1]
```

Model Evalitions

```
# Step 11: Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.4f}")

# Step 12: Compute loss function
test_loss = log_loss(y_test, y_prob)
print(f"Test Log Loss: {test_loss:.4f}")

# Step 13: Print classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Perform cross-validation

```
# Step 14: Perform cross-validation
cv_scores = cross_val_score(gb_clf, X_train_smote, y_train_smote,
                            cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
                            scoring='accuracy')

print(f"Cross-validation accuracy: {np.mean(cv_scores):.4f} (+/- {np.std(cv_scores):.4f})")
```

confusion matrix

```
# Step 15: Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap="coolwarm",
            xticklabels=['Predicted Negatives', 'Predicted Positives'],
            yticklabels=['True Negatives', 'True Positives'])
plt.title("Confusion Matrix Heatmap for Gradient Boosting")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

ROC Curve and AUC

```
# Step 16: ROC Curve and AUC
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC Curve (AUC = {roc_auc:.4f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Gradient Boosting')
plt.legend(loc="lower right")
plt.show()
```

Accuracy comparison plot

```
import matplotlib.pyplot as plt
import numpy as np

# Example data (replace with your actual accuracy values)
accuracy = 0.85  # Test accuracy
cv_scores = [0.82, 0.83, 0.84, 0.81, 0.86]  # Cross-validation accuracy scores

# Calculate mean and standard deviation for cross-validation scores
accuracy_values = [accuracy, np.mean(cv_scores)]
accuracy_std = [0, np.std(cv_scores)]

# Plotting
plt.figure(figsize=(5, 3))
bars = plt.barh(['Test Accuracy', 'Cross-validation Accuracy'], accuracy_values, xerr=accuracy_std, color=['lightgreen', 'skyblue'],
capsize=10)

# Add percentages on the bars
for bar in bars:
    percentage = bar.get_width() * 100  # Convert to percentage
    plt.text(bar.get_width() / 2, bar.get_y() + bar.get_height() / 2, f"{percentage:.2f}%",
             ha='center', va='center', fontsize=12, fontweight='bold', color='black')

# Set labels and title
plt.xlabel('Accuracy')
plt.title('Comparison of Test vs Cross-validation Accuracy for Gradient Boosting')

# Show plot
plt.tight_layout()
plt.show()

# Print the results
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
print(f"Cross-validation Accuracy: {np.mean(cv_scores) * 100:.2f}%")
print(f"Cross-validation Accuracy Standard Deviation: {np.std(cv_scores) * 100:.2f}%")
```

Loss function visualization

```
# Step 18: Loss function visualization
loss_values = [test_loss, np.mean(cv_scores)]

# Define labels for the evaluation methods
labels = ['Test Loss', 'Cross-validation Loss']

# Plot the Loss values
plt.figure(figsize=(6, 4))
plt.plot(labels, loss_values, marker='o', linestyle='-', color='purple', label='Log Loss')
plt.xlabel('Evaluation Method')
plt.ylabel('Loss')
plt.title('Log Loss Before and After Cross-validation for Gradient Boosting')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```

Light GBM

Initialize the model

```
# Initialize LightGBM model
lgbm_model = LGBMClassifier(n_estimators=100, max_depth=-1, random_state=42)
```

Define the hyperparameter grid for GridSearchCV

```
# Step 6: Define the hyperparameter grid for GridSearchCV
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'num_leaves': [31, 50, 100],
    'max_depth': [-1, 10, 20],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

# Step 7: Perform GridSearchCV to find the best hyperparameters
grid_search = GridSearchCV(estimator=lgbm_model, param_grid=param_grid,
                           cv=5, n_jobs=-1, verbose=1, scoring='accuracy')

grid_search.fit(X_train_smote, y_train_smote)

# Step 8: Print the best parameters
print(f"Best hyperparameters: {grid_search.best_params_}")
```

Initialize the model with the best parameters

```
from lightgbm import LGBMClassifier

# Step 9: Initialize LightGBM with the best parameters
lgbm_clf = LGBMClassifier(
    n_estimators=200,
    learning_rate=0.01,
    num_leaves=31,
    max_depth=-1,
    subsample=0.8,
    colsample_bytree=1.0,
    random_state=42,
    verbose=-1  # Suppress verbose output
)

lgbm_clf.fit(X_train_smote, y_train_smote)

# Step 10: Predict on the test data
y_pred = lgbm_clf.predict(X_test)
y_prob = lgbm_clf.predict_proba(X_test)[:, 1]
```

Model Evaluations

```
# Step 11: Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.4f}")

# Step 12: Compute Loss function
test_loss = log_loss(y_test, y_prob)
print(f"Test Log Loss: {test_loss:.4f}")

# Step 13: Print classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Perform cross-validation

```
# Step 14: Perform cross-validation
cv_scores = cross_val_score(lgbm_clf, X_train_smote, y_train_smote,
                            cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
                            scoring='accuracy')

print(f"Cross-validation accuracy: {np.mean(cv_scores):.4f} (+/- {np.std(cv_scores):.4f})")
```

confusion matrix

```
# Step 15: Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create the heatmap with annotations showing TP, FP, FN, TN
plt.figure(figsize=(6, 5))

# Heatmap for confusion matrix
sns.heatmap(cm, annot=True, fmt='d', cmap="coolwarm",
            xticklabels=['Negative', 'Positive'],
            yticklabels=['Negative', 'Positive'],
            cbar=False, annot_kws={'size': 16})

# Add labels and title to make the plot more readable
plt.title("Confusion Matrix for Light GBM", fontsize=18)
plt.xlabel("Predicted Labels", fontsize=14)
plt.ylabel("True Labels", fontsize=14)

# Add a table for the confusion matrix result
cell_text = [
    ['True Negative (TN)', cm[0, 0]],
    ['False Positive (FP)', cm[0, 1]],
    ['False Negative (FN)', cm[1, 0]],
    ['True Positive (TP)', cm[1, 1]]
]

## Create the table
#plt.table(cellText=cell_text, colLabels=["", "Count"], loc="bottom", cellLoc="center", colColours=["#f7f7f7"]*2, bbox=[0, -0.3, 1, 0.2])

# Display the matrix with the annotations and table
#plt.show()
```

ROC Curve and AUC

```
# Step 16: ROC Curve and AUC
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC Curve (AUC = {roc_auc:.4f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Light GBM')
plt.legend(loc="lower right")
plt.show()
```

Accuracy comparison plot

```
# Step 17: Accuracy comparison plot
labels = ['Test Accuracy', 'Cross-validation Accuracy']
accuracy_values = [accuracy, np.mean(cv_scores)]
accuracy_std = [0, np.std(cv_scores)]
plt.figure(figsize=(5, 3))
bars = plt.barh(labels, accuracy_values, xerr=accuracy_std, color=['lightgreen', 'skyblue'], capsize=10)
for bar in bars:
    plt.text(bar.get_width()/2, bar.get_y() + bar.get_height()/2, f"{bar.get_width():.4f}",
             ha='center', va='center', fontsize=12, fontweight='bold', color='black')
plt.xlabel('Accuracy')
plt.title('Comparison of Test vs Cross-validation Accuracy for Light GBM')
plt.show()

import matplotlib.pyplot as plt
import numpy as np

# Accuracy values (assuming 'accuracy' and 'cv_scores' are already defined)
labels = ['Test Accuracy', 'Cross-validation Accuracy']
accuracy_values = [accuracy, np.mean(cv_scores)]  # accuracy should be a float (0 to 1)
accuracy_std = [0, np.std(cv_scores)]  # standard deviation for cross-validation accuracy

# Plotting the horizontal bar chart
plt.figure(figsize=(5, 3))
bars = plt.barh(labels, accuracy_values, xerr=accuracy_std, color=['lightgreen', 'skyblue'], capsize=10)

# Annotating with percentages
for bar in bars:
    percentage = bar.get_width() * 100  # Convert to percentage
```

```
    plt.text(bar.get_width()/2, bar.get_y() + bar.get_height()/2, f"{percentage:.2f}%",
             ha='center', va='center', fontsize=12, fontweight='bold', color='black')

# Labels and title
plt.xlabel('Accuracy (%)')
plt.title('Comparison of Test vs Cross-validation Accuracy for Light GBM')

# Show the plot
plt.tight_layout()
plt.show()

 #Print the results as percentage values
print(f"Test Accuracy: {accuracy * 100:.2f}%")
print(f"Cross-validation Accuracy: {np.mean(cv_scores) * 100:.2f}%")
print(f"Cross-validation Accuracy Std Dev: {np.std(cv_scores) * 100:.2f}%")
```

Loss function visualization

```
# Step 18: Loss function visualization
loss_values = [test_loss, np.mean(cv_scores)]
plt.figure(figsize=(8, 5))
plt.plot(labels, loss_values, marker='o', linestyle='-', color='purple', label='Log Loss')
plt.xlabel('Evaluation Method')
plt.ylabel('Loss')
plt.title('Log Loss Before and After Cross-validation for Light GBM')
plt.legend()
plt.grid(True)
plt.show()
```

Random Forest

Initialize the model

```
# Initialize the Random Forest model
rf_model = RandomForestClassifier()
```

Define the hyperparameter grid to search over

```
# Define the hyperparameter grid to search over
param_grid = {
    'n_estimators': [50, 100, 200, 300],  # Number of trees in the forest
    'max_depth': [None, 10, 20, 30],       # Maximum depth of the trees
    'min_samples_split': [2, 5, 10],       # Minimum number of samples required to split a node
    'min_samples_leaf': [1, 2, 4],         # Minimum number of samples required at each leaf node
    'bootstrap': [True, False]             # Whether bootstrap samples are used when building trees
}

 # Step 7: Perform GridSearchCV to find the best hyperparameters
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid,
                           cv=5, n_jobs=-1, verbose=1, scoring='accuracy')

grid_search.fit(X_train_smote, y_train_smote)

# Step 8: Print the best parameters
print(f"Best hyperparameters: {grid_search.best_params_}")
```

Initialize the Random Forest model with the best hyperparameters

```
# Initialize the Random Forest model with the best hyperparameters
rf_clf = RandomForestClassifier(bootstrap=False, max_depth=None, min_samples_leaf=1,
                                min_samples_split=5, n_estimators=50, random_state=42)

# Step 10: Fit the model on the training data
rf_clf.fit(X_train_smote, y_train_smote)

# Predict on the test data
y_pred = rf_clf.predict(X_test)
y_prob = rf_clf.predict_proba(X_test)[:, 1]
```

Model Evaluations

```
# Step 11: Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.4f}")

# Step 12: Compute Loss function
test_loss = log_loss(y_test, y_prob)
print(f"Test Log Loss: {test_loss:.4f}")

# Step 13: Print classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Perform cross-validation

```
# Step 14: Perform cross-validation
cv_scores = cross_val_score(rf_model, X_train_smote, y_train_smote,
                            cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
```

```
                            scoring='accuracy')

print(f"Cross-validation accuracy: {np.mean(cv_scores):.4f} (+/- {np.std(cv_scores):.4f})")
```

Generate confusion matrix

```
# Step 15: Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create the heatmap with annotations showing TP, FP, FN, TN
plt.figure(figsize=(6, 5))

# Heatmap for confusion matrix
sns.heatmap(cm, annot=True, fmt='d', cmap="coolwarm",
            xticklabels=['Negative', 'Positive'],
            yticklabels=['Negative', 'Positive'],
            cbar=False, annot_kws={'size': 16})

# Add labels and title to make the plot more readable
plt.title("Confusion Matrix for Random Forest", fontsize=18)
plt.xlabel("Predicted Labels", fontsize=14)
plt.ylabel("True Labels", fontsize=14)

# Add a table for the confusion matrix result
cell_text = [
    ['True Negative (TN)', cm[0, 0]],
    ['False Positive (FP)', cm[0, 1]],
    ['False Negative (FN)', cm[1, 0]],
    ['True Positive (TP)', cm[1, 1]]
]

# Create the table
#plt.table(cellText=cell_text, colLabels=["", "Count"], loc="bottom", cellLoc="center", colColours=["#f7f7f7"]*2, bbox=[0, -0.3, 1, 0.2])

# Display the matrix with the annotations and table
#plt.show()
```

ROC Curve and AUC

```
# Step 16: ROC Curve and AUC
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC Curve (AUC = {roc_auc:.4f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Random Forest')
plt.legend(loc="lower right")
plt.show()
```

Accuracy comparison plot

```
import numpy as np
import matplotlib.pyplot as plt

# Example accuracy values (replace with actual results from your model evaluation)
accuracy = 0.85  # Example test accuracy
cv_scores = [0.84, 0.86, 0.83, 0.87, 0.85]  # Example cross-validation scores

# Calculate standard deviation for cross-validation scores
accuracy_values = [accuracy, np.mean(cv_scores)]
accuracy_std = [0, np.std(cv_scores)]

# Plot comparison
plt.figure(figsize=(5, 3))
bars = plt.barh(['Test Accuracy', 'Cross-validation Accuracy'], accuracy_values, xerr=accuracy_std, color=['lightgreen', 'skyblue'],
capsize=10)

# Add text annotations to display the percentage on the bars
for bar in bars:
    percentage = f"{bar.get_width() * 100:.2f}%"  # Convert to percentage and format to 2 decimal places
    plt.text(bar.get_width() / 2, bar.get_y() + bar.get_height() / 2, percentage,
             ha='center', va='center', fontsize=12, fontweight='bold', color='black')

# Set labels and title
plt.xlabel('Accuracy')
plt.title('Comparison of Test vs Cross-validation Accuracy for Random Forest')

# Show plot
plt.tight_layout()
plt.show()

# Print results
print("Accuracy Results:")
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
print(f"Cross-validation Accuracy: {np.mean(cv_scores) * 100:.2f}%")
print(f"Cross-validation Standard Deviation: {np.std(cv_scores):.4f}")
```

Loss function visualization

```
# Step 18: Loss function visualization
loss_values = [test_loss, np.mean(cv_scores)]
plt.figure(figsize=(8, 5))
plt.plot(labels, loss_values, marker='o', linestyle='-', color='purple', label='Log Loss')
plt.xlabel('Evaluation Method')
plt.ylabel('Loss')
plt.title('Log Loss Before and After Cross-validation for Random Forest')
plt.legend()
plt.grid(True)
plt.show()
```

Model Evulation

Model Comparism

```
# Define the function to evaluate models
def evaluate_model_performance(X_train, y_train, X_test, y_test):
    models = {
        'Gradient Boosting':GradientBoostingClassifier(
    learning_rate=0.01,
    max_depth=5,
    min_samples_split=10,
    n_estimators=100,
    subsample=0.8
),


        'Light GBMC': LGBMClassifier(
    n_estimators=200,
    learning_rate=0.01,
    num_leaves=31,
    max_depth=-1,
    subsample=0.8,
    colsample_bytree=1.0,
    random_state=42
),

        'Random Forest': RandomForestClassifier(bootstrap=False, max_depth=None, min_samples_leaf=1,
                                min_samples_split=5, n_estimators=50, random_state=42)}


    # Initialize a DataFrame to store the results
    results = []

    # Loop through each model, fit, predict, and evaluate performance
    for model_name, model in models.items():
        # Fit the model
        model.fit(X_train, y_train)

        # Make predictions
        y_pred = model.predict(X_test)

        # Calculate accuracy
        accuracy = accuracy_score(y_test, y_pred)

        # Generate the classification report
        class_report = classification_report(y_test, y_pred, output_dict=True)

        # Safely extract precision, recall, and f1-scores for each class
        precision_0 = class_report['0']['precision'] if '0' in class_report else None
        recall_0 = class_report['0']['recall'] if '0' in class_report else None
        f1_0 = class_report['0']['f1-score'] if '0' in class_report else None

        precision_1 = class_report['1']['precision'] if '1' in class_report else None
        recall_1 = class_report['1']['recall'] if '1' in class_report else None
        f1_1 = class_report['1']['f1-score'] if '1' in class_report else None

        # Generate the confusion matrix
        cm = confusion_matrix(y_test, y_pred)

        # Store the results in the dataframe
        results.append({
            'Model': model_name,
            'Accuracy': accuracy,
            'Precision (Class 0)': precision_0,
            'Recall (Class 0)': recall_0,
            'F1 Score (Class 0)': f1_0,
```

```
            'Precision (Class 1)': precision_1,
            'Recall (Class 1)': recall_1,
            'F1 Score (Class 1)': f1_1,
            'Confusion Matrix': cm
        })

    # Convert results to DataFrame for comparison
    results_df = pd.DataFrame(results)

    return results_df

# Example of how to use the function
# Assuming X_train, y_train, X_test, y_test are already defined:

# Call the function and store the results in a DataFrame
results_df = evaluate_model_performance(X_train, y_train, X_test, y_test)

# Now print the results DataFrame
print(results_df)
```

ROC curves

```
# Define the function to evaluate models
def evaluate_model_performance(X_train, y_train, X_test, y_test):
    models = {
        'Gradient Boosting': GradientBoostingClassifier(
            learning_rate=0.01,
            max_depth=5,
            min_samples_split=10,
            n_estimators=100,
            subsample=0.8
        ),

        'LightGBM': LGBMClassifier(
            n_estimators=200,
            learning_rate=0.01,
            num_leaves=31,
            max_depth=-1,
            subsample=0.8,
            colsample_bytree=1.0,
            random_state=42
        ),

        'Random Forest': RandomForestClassifier(
            bootstrap=False, max_depth=None, min_samples_leaf=1,
            min_samples_split=5, n_estimators=50, random_state=42
        )
    }


    # Initialize a DataFrame to store the results
    results = []

    # Prepare a plot for ROC curve
    plt.figure(figsize=(10, 8))

    # Loop through each model, fit, predict, and evaluate performance
    for model_name, model in models.items():
        # Fit the model
        model.fit(X_train, y_train)

        # Make predictions
        y_pred = model.predict(X_test)

        # Calculate accuracy
        accuracy = accuracy_score(y_test, y_pred)

        # Generate the classification report
        class_report = classification_report(y_test, y_pred, output_dict=True)

        # Safely extract precision, recall, and f1-scores for each class
        precision_0 = class_report['0']['precision'] if '0' in class_report else None
        recall_0 = class_report['0']['recall'] if '0' in class_report else None
        f1_0 = class_report['0']['f1-score'] if '0' in class_report else None

        precision_1 = class_report['1']['precision'] if '1' in class_report else None
        recall_1 = class_report['1']['recall'] if '1' in class_report else None
        f1_1 = class_report['1']['f1-score'] if '1' in class_report else None

        # Generate the confusion matrix
        cm = confusion_matrix(y_test, y_pred)
```

```
        # Store the results in the dataframe
        results.append({
            'Model': model_name,
            'Accuracy': accuracy,
            'Precision (Class 0)': precision_0,
            'Recall (Class 0)': recall_0,
            'F1 Score (Class 0)': f1_0,
            'Precision (Class 1)': precision_1,
            'Recall (Class 1)': recall_1,
            'F1 Score (Class 1)': f1_1,
            'Confusion Matrix': cm
        })

        # Check if the model supports probability predictions
        if hasattr(model, "predict_proba"):
            y_prob = model.predict_proba(X_test)[:, 1]  # Get probabilities for the positive class
            fpr, tpr, _ = roc_curve(y_test, y_prob)
            roc_auc = auc(fpr, tpr)

            # Plot ROC curve for this model
            plt.plot(fpr, tpr, label=f'{model_name} (AUC = {roc_auc:.2f})')

    # Plot formatting for the ROC curve
    plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC): Gradient Boosting vs LightGBM vs Random Forest')
    plt.legend(loc='lower right')
    plt.show()

    # Convert results to DataFrame for comparison
    results_df = pd.DataFrame(results)

    return results_df

# Example of how to use the function
# Assuming X_train, y_train, X_test, y_test are already defined:

# Call the function and store the results in a DataFrame
results_df = evaluate_model_performance(X_train, y_train, X_test, y_test)

# Now print the results DataFrame
#print(results_df)
```

precision, recall

```
# Define the function to evaluate models
def evaluate_model_performance(X_train, y_train, X_test, y_test):
    models = {
        'Gradient Boosting': GradientBoostingClassifier(
            learning_rate=0.01,
            max_depth=5,
            min_samples_split=10,
            n_estimators=100,
            subsample=0.8
        ),

        'LightGBM': LGBMClassifier(
            n_estimators=200,
            learning_rate=0.01,
            num_leaves=31,
            max_depth=-1,
            subsample=0.8,
            colsample_bytree=1.0,
            random_state=42
        ),

        'Random Forest': RandomForestClassifier(
            bootstrap=False, max_depth=None, min_samples_leaf=1,
            min_samples_split=5, n_estimators=50, random_state=42
        )
    }

    # Initialize a DataFrame to store the results
    results = []

    # Prepare a plot for Precision-Recall curve
    plt.figure(figsize=(10, 8))

    # Loop through each model, fit, predict, and evaluate performance
    for model_name, model in models.items():
        # Fit the model
        model.fit(X_train, y_train)

        # Make predictions
        y_pred = model.predict(X_test)
```

```
        # Calculate accuracy
        accuracy = accuracy_score(y_test, y_pred)

        # Generate the classification report
        class_report = classification_report(y_test, y_pred, output_dict=True)

        # Safely extract precision, recall, and f1-scores for each class
        precision_0 = class_report['0']['precision'] if '0' in class_report else None
        recall_0 = class_report['0']['recall'] if '0' in class_report else None
        f1_0 = class_report['0']['f1-score'] if '0' in class_report else None

        precision_1 = class_report['1']['precision'] if '1' in class_report else None
        recall_1 = class_report['1']['recall'] if '1' in class_report else None
        f1_1 = class_report['1']['f1-score'] if '1' in class_report else None

        # Generate the confusion matrix
        cm = confusion_matrix(y_test, y_pred)

        # Store the results in the dataframe
        results.append({
            'Model': model_name,
            'Accuracy': accuracy,
            'Precision (Class 0)': precision_0,
            'Recall (Class 0)': recall_0,
            'F1 Score (Class 0)': f1_0,
            'Precision (Class 1)': precision_1,
            'Recall (Class 1)': recall_1,
            'F1 Score (Class 1)': f1_1,
            'Confusion Matrix': cm
        })

        # Check if the model supports probability predictions
        if hasattr(model, "predict_proba"):
            y_prob = model.predict_proba(X_test)[:, 1]  # Get probabilities for the positive class
            precision, recall, _ = precision_recall_curve(y_test, y_prob)
            pr_auc = auc(recall, precision)

            # Plot Precision-Recall curve for this model
            plt.plot(recall, precision, label=f'{model_name} (AUC = {pr_auc:.2f})')

    # Plot formatting for the Precision-Recall curve
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Precision-Recall Curve for: Gradient Boosting vs LightGBM vs Random Forest')
    plt.legend(loc='lower left')
    plt.show()

    # Convert results to DataFrame for comparison
    results_df = pd.DataFrame(results)

    return results_df

# Example of how to use the function
# Assuming X_train, y_train, X_test, y_test are already defined:

# Call the function and store the results in a DataFrame
results_df = evaluate_model_performance(X_train, y_train, X_test, y_test)

# Now print the results DataFrame
##print(results_df)
```

Model Comparison: Accuracy and ROC AUC

logloss funcation

```
from sklearn.metrics import log_loss, accuracy_score
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from lightgbm import LGBMClassifier
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Define the function to evaluate models
def evaluate_model_performance_logloss(X_train, y_train, X_test, y_test):
    # Initialize individual classifiers with their best hyperparameters
    models = {
        'Gradient Boosting': GradientBoostingClassifier(
            learning_rate=0.01,
            max_depth=5,
            min_samples_split=10,
            n_estimators=100,
            subsample=0.8
        ),
```

```
        'LightGBM': LGBMClassifier(
            n_estimators=200,
            learning_rate=0.01,
            num_leaves=31,
            max_depth=-1,
            subsample=0.8,
            colsample_bytree=1.0,
            random_state=42
        ),

        'Random Forest': RandomForestClassifier(
            bootstrap=False, max_depth=None, min_samples_leaf=1,
            min_samples_split=5, n_estimators=50, random_state=42
        )
    }

    # Initialize lists to store results
    accuracy_scores = []
    logloss_scores = []

    # Loop through each model, fit, predict, and evaluate performance
    for model_name, model in models.items():
        # Fit the model
        model.fit(X_train, y_train)

        # Make predictions and probability predictions
        y_pred = model.predict(X_test)
        y_prob = model.predict_proba(X_test)[:, 1]  # Get probabilities for the positive class

        # Calculate accuracy before applying Log-Loss
        accuracy = accuracy_score(y_test, y_pred)
        accuracy_scores.append(accuracy)

        # Calculate Log-Loss
        logloss = log_loss(y_test, y_prob)
        logloss_scores.append(logloss)

    # Find the best (lowest) Log-Loss score
    best_logloss = min(logloss_scores)

    # Calculate Performance (%) based on Log-Loss
    logloss_percentages = [(1 - (loss - best_logloss) / best_logloss) * 100 for loss in logloss_scores]
    logloss_percentages = [round(p) for p in logloss_percentages]  # Round percentages to avoid decimals

    # Create a DataFrame for comparison
    comparison_df = pd.DataFrame({
        'Model': list(models.keys()),
        'Accuracy (Before Log-Loss)': accuracy_scores,
        'Log-Loss': logloss_scores,
        'Performance (%)': logloss_percentages
    })

    # Plotting the Comparison
    plt.figure(figsize=(12, 6))

    # Plot accuracy before Log-Loss
    plt.bar(comparison_df['Model'], [round(acc * 100) for acc in comparison_df['Accuracy (Before Log-Loss)']],
            color='green', alpha=0.6, label='Accuracy (Before Log-Loss)', width=0.4, align='center')

    # Plot Log-Loss and Performance after Log-Loss
    plt.bar(comparison_df['Model'], comparison_df['Log-Loss'], color='red', alpha=0.6, label='Log-Loss', width=0.4, align='edge')
    plt.bar(comparison_df['Model'], comparison_df['Performance (%)'], color='skyblue', alpha=0.6, label='Performance (%)', width=0.4,
align='edge')

    # Annotating the bars
    for i, row in comparison_df.iterrows():
        plt.text(i, row['Accuracy (Before Log-Loss)'] * 100, f"{round(row['Accuracy (Before Log-Loss)'] * 100)}%",
                 ha='center', va='bottom', fontsize=12, fontweight='bold', color='black')
        plt.text(i, row['Log-Loss'], f"{round(row['Log-Loss'], 4)}", ha='center', va='bottom', fontsize=12, fontweight='bold', color='black')
        plt.text(i, row['Performance (%)'], f"{row['Performance (%)']}%", ha='center', va='bottom', fontsize=12, fontweight='bold',
color='black')

    # Set plot labels and title
    plt.title('Model Performance: Before and After Log-Loss: Log-Loss: Gradient Boosting vs LightGBM vs Random Forest', fontsize=14)
    plt.xlabel('Model')
    plt.ylabel('Scores')
    plt.legend(loc='upper left')
    plt.tight_layout()
    plt.show()

    return comparison_df

# Example of how to call the function:
# Assuming X_train, y_train, X_test, y_test are already defined
comparison_df = evaluate_model_performance_logloss(X_train, y_train, X_test, y_test)
# Print the resulting comparison DataFrame
print(comparison_df)
```

Soft voting ensemble method

```
from sklearn.ensemble import VotingClassifier, GradientBoostingClassifier, RandomForestClassifier
from lightgbm import LGBMClassifier
from sklearn.metrics import accuracy_score, classification_report

# Initialize individual classifiers with their best hyperparameters
models = {
    'Gradient Boosting': GradientBoostingClassifier(
        learning_rate=0.01,
        max_depth=5,
        min_samples_split=10,
        n_estimators=100,
        subsample=0.8
    ),

    'LightGBM': LGBMClassifier(
        n_estimators=200,
        learning_rate=0.01,
        num_leaves=31,
        max_depth=-1,
        subsample=0.8,
        colsample_bytree=1.0,
        random_state=42
    ),

    'Random Forest': RandomForestClassifier(
        bootstrap=False, max_depth=None, min_samples_leaf=1,
        min_samples_split=5, n_estimators=50, random_state=42
    )
}

# Convert models dictionary into a list of tuples for VotingClassifier
classifiers = list(models.items())

# Create the VotingClassifier with soft voting
voting_clf = VotingClassifier(estimators=classifiers, voting='soft')

# Fit the ensemble model
voting_clf.fit(X_train, y_train)

# Predict on the test set
y_pred = voting_clf.predict(X_test)

# Calculate accuracy for the ensemble
ensemble_accuracy = accuracy_score(y_test, y_pred)
print(f"Soft Voting Ensemble Accuracy: {ensemble_accuracy:.4f}")

# Evaluate and display accuracy for each individual classifier
print("\nIndividual Classifier Accuracies:")
best_model = None
best_accuracy = 0

for name, clf in models.items():  # Loop through the original models dictionary
    clf.fit(X_train, y_train)  # Ensure each classifier is trained
    y_pred_individual = clf.predict(X_test)
    individual_accuracy = accuracy_score(y_test, y_pred_individual)
    print(f"{name} Accuracy: {individual_accuracy:.4f}")

    # Track the best model
    if individual_accuracy > best_accuracy:
        best_accuracy = individual_accuracy
        best_model = name

# Display the best-performing model
print(f"\nBest Model: {best_model} with Accuracy: {best_accuracy:.4f}")

# Classification report for the ensemble
print("\nClassification Report for Soft Voting Ensemble:")
print(classification_report(y_test, y_pred))
```

Generate the confusion matrix

```
# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Extract values
TN, FP, FN, TP = cm.ravel()

# Create a heatmap using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap="cool",
            xticklabels=['Predicted Negatives', 'Predicted Positives'],
            yticklabels=['True Negatives', 'True Positives'],color=['skyblue', 'lightcoral','red'])
plt.title("Confusion Matrix Heatmap for Soft Voting Ensemble")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

```
# Step 16: ROC Curve and AUC
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC Curve (AUC = {roc_auc:.4f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Soft Voting Ensemble')
plt.legend(loc="lower right")
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt

# Example accuracy values (replace with actual results from your model evaluation)
accuracy = 0.85  # Example test accuracy
cv_scores = [0.84, 0.86, 0.83, 0.87, 0.85]  # Example cross-validation scores

# Calculate standard deviation for cross-validation scores
accuracy_values = [accuracy, np.mean(cv_scores)]
accuracy_std = [0, np.std(cv_scores)]

# Plot comparison
plt.figure(figsize=(5, 3))
bars = plt.barh(['Test Accuracy', 'Cross-validation Accuracy'], accuracy_values, xerr=accuracy_std, color=['lightgreen', 'skyblue'],
capsize=10)

# Add text annotations to display the percentage on the bars
for bar in bars:
    percentage = f"{bar.get_width() * 100:.2f}%"  # Convert to percentage and format to 2 decimal places
    plt.text(bar.get_width() / 2, bar.get_y() + bar.get_height() / 2, percentage,
             ha='center', va='center', fontsize=12, fontweight='bold', color='black')

# Set labels and title
plt.xlabel('Accuracy')
plt.title('Comparison of Test vs Cross-validation Accuracy for Soft Voting Ensemble')

# Show plot
plt.tight_layout()
plt.show()

# Print results
#print("Accuracy Results:")
#print(f"Test Accuracy: {accuracy * 100:.2f}%")
#print(f"Cross-validation Accuracy: {np.mean(cv_scores) * 100:.2f}%")
#print(f"Cross-validation Standard Deviation: {np.std(cv_scores):.4f}")
```

```
# Step 18: Loss function visualization
loss_values = [test_loss, np.mean(cv_scores)]
plt.figure(figsize=(8, 5))
plt.plot(labels, loss_values, marker='o', linestyle='-', color='purple', label='Log Loss')
plt.xlabel('Evaluation Method')
plt.ylabel('Loss')
plt.title('Log Loss Before and After Cross-validation for Soft Voting Ensemble')
plt.legend()
plt.grid(True)
plt.show()
```

Regression Metrics (if Predicting Risk Scores or Severity Levels)

```
# Define the function to evaluate regression models
def evaluate_regression_models(X_train, y_train, X_test, y_test):
    models = {
        'Gradient Boosting': GradientBoostingClassifier(
            learning_rate=0.01,
            max_depth=5,
            min_samples_split=10,
            n_estimators=100,
            subsample=0.8
        ),

        'LightGBM': LGBMClassifier(
            n_estimators=200,
```

```
            learning_rate=0.01,
            num_leaves=31,
            max_depth=-1,
            subsample=0.8,
            colsample_bytree=1.0,
            random_state=42
        ),

        'Random Forest': RandomForestClassifier(
            bootstrap=False, max_depth=None, min_samples_leaf=1,
            min_samples_split=5, n_estimators=50, random_state=42
        )
    }


    # Initialize lists to store metrics
    model_names = []
    mae_scores = []
    mse_scores = []
    rmse_scores = []
    r2_scores = []

    # Evaluate each model
    for model_name, model in models.items():
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        # Calculate regression metrics
        mae = mean_absolute_error(y_test, y_pred)
        mse = mean_squared_error(y_test, y_pred)
        rmse = np.sqrt(mse)
        r2 = r2_score(y_test, y_pred)

        # Append results
        model_names.append(model_name)
        mae_scores.append(mae)
        mse_scores.append(mse)
        rmse_scores.append(rmse)
        r2_scores.append(r2)

    # Create a DataFrame for comparison
    results_df = pd.DataFrame({
        'Model': model_names,
        'MAE': mae_scores,
        'MSE': mse_scores,
        'RMSE': rmse_scores,
        'R²': r2_scores
    })

    # Highlight the best model for each metric
    #print("\nRegression Model Comparison:")
    #print(results_df)

    # Identify the best model based on R² score
    best_model_idx = results_df['R²'].idxmax()
    best_model_name = results_df.iloc[best_model_idx]['Model']
    #print(f"\nThe best model based on R² score is: {best_model_name}")

    # Plot comparison
    results_df.set_index('Model', inplace=True)
    ax = results_df[['MAE', 'MSE', 'RMSE']].plot(
        kind='bar', figsize=(12, 6), title="Regression Model Metrics Comparison: Gradient Boosting vs LightGBM vs Random Forest",
color=['skyblue', 'lightcoral', 'red']
    )
    plt.ylabel("Error Score")
    plt.xlabel("Model")
    plt.xticks(rotation=45)

    # Annotate values on top of the bars as percentages
    for p in ax.patches:
        value = f"{p.get_height() * 100:.2f}%"  # Convert to percentage
        ax.annotate(
            value,
            (p.get_x() + p.get_width() / 2., p.get_height()),
            ha='center', va='bottom', fontsize=10, color='black'
        )

    # Plot R² values separately
    plt.tight_layout()
    plt.show()

    # Print R² comparison
   # print("\nR² Scores:")
   # print(results_df['R²'])

    return results_df, best_model_name

# Example usage
# Assuming X_train, y_train, X_test, y_test are already defined for regression tasks:
```

```
results_df, best_model_name = evaluate_regression_models(X_train, y_train, X_test, y_test)

# Print the results DataFrame
#print(results_df)
```

Find the best model based on Accurcy ,ROC AUC ,log loss and soft voting

```
# Define the function to evaluate models based on all metrics and soft voting
def evaluate_models(X_train, y_train, X_test, y_test):
    models = {
        'Gradient Boosting': GradientBoostingClassifier(
            learning_rate=0.01,
            max_depth=5,
            min_samples_split=10,
            n_estimators=100,
            subsample=0.8
        ),

        'LightGBM': LGBMClassifier(
            n_estimators=200,
            learning_rate=0.01,
            num_leaves=31,
            max_depth=-1,
            subsample=0.8,
            colsample_bytree=1.0,
            random_state=42
        ),

        'Random Forest': RandomForestClassifier(
            bootstrap=False, max_depth=None, min_samples_leaf=1,
            min_samples_split=5, n_estimators=50, random_state=42
        )
    }


    # Initialize lists to store metrics
    model_names = []
    accuracy_scores = []
    roc_auc_scores = []
    log_loss_scores = []

    # Evaluate individual models
    for model_name, model in models.items():
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        accuracy_scores.append(accuracy)

        # Calculate ROC AUC
        if hasattr(model, "predict_proba"):
            y_prob = model.predict_proba(X_test)[:, 1]
            fpr, tpr, _ = roc_curve(y_test, y_prob)
            roc_auc = auc(fpr, tpr)
            roc_auc_scores.append(roc_auc)
            log_loss_scores.append(log_loss(y_test, model.predict_proba(X_test)))
        else:
            roc_auc_scores.append(None)
            log_loss_scores.append(None)

        model_names.append(model_name)

    # Create VotingClassifier with soft voting
    voting_clf = VotingClassifier(estimators=list(models.items()), voting='soft')
    voting_clf.fit(X_train, y_train)
    y_pred_voting = voting_clf.predict(X_test)

    # Calculate metrics for the VotingClassifier
    accuracy_voting = accuracy_score(y_test, y_pred_voting)
    y_prob_voting = voting_clf.predict_proba(X_test)[:, 1]
    fpr_voting, tpr_voting, _ = roc_curve(y_test, y_prob_voting)
    roc_auc_voting = auc(fpr_voting, tpr_voting)
    log_loss_voting = log_loss(y_test, voting_clf.predict_proba(X_test))

    # Append VotingClassifier metrics to the lists
    model_names.append("Soft Voting Ensemble")
    accuracy_scores.append(accuracy_voting)
    roc_auc_scores.append(roc_auc_voting)
    log_loss_scores.append(log_loss_voting)

    # Create a DataFrame for comparison
    results_df = pd.DataFrame({
        'Model': model_names,
        'Accuracy': accuracy_scores,
        'ROC AUC': roc_auc_scores,
        'Log Loss': log_loss_scores
    })

    # Highlight the best model for each metric
    #print("\nModel Comparison:")
```

```
    #print(results_df)

    #print("\nBest Models:")
    #print(f"Best Model (Accuracy): {results_df.loc[results_df['Accuracy'].idxmax(), 'Model']}")
    #print(f"Best Model (ROC AUC): {results_df.loc[results_df['ROC AUC'].idxmax(), 'Model']}")
    #print(f"Best Model (Log Loss): {results_df.loc[results_df['Log Loss'].idxmin(), 'Model']}")
    #print(f"Soft Voting Ensemble Results:")
    #print(f"Accuracy: {accuracy_voting:.4f}, ROC AUC: {roc_auc_voting:.4f}, Log Loss: {log_loss_voting:.4f}")

    # Plot comparison
    results_df.set_index('Model', inplace=True)
    ax = results_df[['Accuracy', 'ROC AUC', 'Log Loss']].plot(
        kind='bar', figsize=(12, 6), title="Model Metrics Comparison: Gradient Boosting vs LightGBM vs Random Forest vs Soft Voting
Ensemble", color=['skyblue', 'lightcoral', 'red']
    )
    plt.ylabel("Score")
    plt.xlabel("Model")
    plt.xticks(rotation=45)

    # Annotate values on top of the bars (show percentages)
    for p in ax.patches:
        value = f"{p.get_height() * 100:.2f}%" if not pd.isna(p.get_height()) else "N/A"
        ax.annotate(
            value,
            (p.get_x() + p.get_width() / 2., p.get_height()),
            ha='center', va='bottom', fontsize=10, color='black'
        )

    plt.tight_layout()
    plt.show()

    return results_df

# Example usage
# Assuming X_train, y_train, X_test, y_test are already defined:
results_df = evaluate_models(X_train, y_train, X_test, y_test)

# Print the results DataFrame
#print(results_df)
```

Introduction: In the realm of machine learning, choosing the best model for a specific task requires evaluating different performance metrics. In this comparison, three prominent models—Gradient Boosting, LightGBM, and Random Forest—are tested across several performance metrics: accuracy, ROC AUC, and log loss. Additionally, the performance of a Soft Voting Ensemble, which aggregates predictions from multiple models, is also considered. By examining these models based on multiple factors, we aim to identify the best-performing model for the given task.

Which Model is Best and Why? Based on the evaluation metrics, Gradient Boosting emerges as the best model when considering accuracy (0.910) and ROC AUC (0.9694). It consistently outperforms the other models in these areas, indicating that it is the most reliable model for classification tasks where prediction accuracy and distinguishing between classes matter. However, when it comes to log loss, Random Forest performs slightly better with a lower value of 0.2493, suggesting it may be more reliable in terms of probabilistic predictions. Despite this, Gradient Boosting's overall performance in accuracy and ROC AUC makes it the top choice for most use cases.

Conclusions: While Gradient Boosting excels in both accuracy and ROC AUC, making it the best model in terms of general classification performance, Random Forest proves to be a strong contender in terms of minimizing log loss. The Soft Voting Ensemble, though effective, does not significantly outperform the best individual models across the board, with its accuracy and ROC AUC slightly trailing behind Gradient Boosting. Thus, for a well-rounded, high-performance model, Gradient Boosting is the best choice, offering a balanced combination of accuracy, ROC AUC, and solid generalization.

Recommendation:

Based on the comparison of various models, Gradient Boosting emerges as the best model for Ischemic Heart Disease (IHD) prediction. It demonstrated the highest accuracy (91.0%), ROC AUC (0.9694), and performed consistently well across key performance metrics, making it a reliable choice for the final model.

Save the model using joblib

```
import joblib

# Assuming your trained Gradient Boosting model is named 'gb_model'
joblib.dump(gb_model, 'gradient_boosting_model.pkl')

print("Model saved successfully as 'gradient_boosting_model.pkl'")
```

Loading the Saved Model:

```
import joblib
import pandas as pd

# Load the saved Gradient Boosting model
loaded_model = joblib.load('gradient_boosting_model.pkl')

print("Model loaded successfully!")

import pandas as pd

# Example input data (you can replace this with the actual test data you want to predict)
input_data = pd.DataFrame({
```

```
    'Age (Years)': [60],
    'Stroke Status_Yes': [1],  # 1 for Yes, 0 for No
    'Weight (kg)': [70],
    'Random Blood Sugar (mg/dL)': [140],
    'Height (cm)': [175],
    'Systolic Blood Pressure (mmHg)': [130],
    'Diabetes Mellitus (DM) Status_Yes': [1],  # 1 for Yes, 0 for No
    'Hypertension (HTN) Status_Yes': [1],  # 1 for Yes, 0 for No
    'Dyslipidemia Status_Yes': [0],  # 1 for Yes, 0 for No
    'Diastolic Blood Pressure (mmHg)': [85],
    'Ischemic Heart Disease (IHD) Status_Yes': [0]  # 1 for Yes, 0 for No (target, not needed for prediction)
})

# Ensure that the data type is correct (e.g., boolean columns are in the right format)
input_data = input_data.astype({
    'Stroke Status_Yes': 'bool',
    'Diabetes Mellitus (DM) Status_Yes': 'bool',
    'Hypertension (HTN) Status_Yes': 'bool',
    'Dyslipidemia Status_Yes': 'bool',
    'Ischemic Heart Disease (IHD) Status_Yes': 'bool'  # Don't include this as a feature for prediction
})

from sklearn.ensemble import GradientBoostingClassifier

# Example of model training
gb_model = GradientBoostingClassifier()
gb_model.fit(X_train, y_train)  # Replace with your training data

# Save the model after fitting
import joblib
joblib.dump(gb_model, 'gradient_boosting_model.pkl')

import pandas as pd
import joblib

# Define all the feature columns (ensure these match the training features)
columns = [
    'Age (Years)', 'Stroke Status_Yes', 'Weight (kg)', 'Random Blood Sugar (mg/dL)',
    'Height (cm)', 'Systolic Blood Pressure (mmHg)', 'Diabetes Mellitus (DM) Status_Yes',
    'Hypertension (HTN) Status_Yes', 'Dyslipidemia Status_Yes',
    'Diastolic Blood Pressure (mmHg)', 'Feature_11', 'Feature_12', 'Feature_13', 'Feature_14', 'Feature_15',
    'Feature_16', 'Feature_17', 'Feature_18', 'Feature_19', 'Feature_20'
]

# Example: Make sure you have a data frame with the right number of features
input_data = pd.DataFrame([{
    'Age (Years)': 50, 'Stroke Status_Yes': 1, 'Weight (kg)': 70,
    'Random Blood Sugar (mg/dL)': 150, 'Height (cm)': 175, 'Systolic Blood Pressure (mmHg)': 130,
    'Diabetes Mellitus (DM) Status_Yes': 1, 'Hypertension (HTN) Status_Yes': 0,
    'Dyslipidemia Status_Yes': 0, 'Diastolic Blood Pressure (mmHg)': 85,
    # Add other features required by the model (these are just placeholders)
    'Feature_11': 0, 'Feature_12': 1, 'Feature_13': 0, 'Feature_14': 1, 'Feature_15': 0,
    'Feature_16': 1, 'Feature_17': 0, 'Feature_18': 1, 'Feature_19': 0, 'Feature_20': 1
}], columns=columns)

# Load the trained model
loaded_model = joblib.load('gradient_boosting_model.pkl')

# Make predictions on the new data
predicted_class = loaded_model.predict(input_data)  # Predict class (0 or 1)
predicted_proba = loaded_model.predict_proba(input_data)  # Get probabilities for each class

# Output the results
print(f"Predicted Class: {predicted_class[0]}")  # IHD Yes (1) or No (0)
print(f"Predicted Probabilities: {predicted_proba[0]}")  # Probability distribution for each class
```

Model Accuracy

```
# Print the accuracy and classification report
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the loaded model: {accuracy:.4f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Optionally, you can also generate a confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(cm)
```