A HYBRID FEATURE SELECTION WITH STACK-ENSEMBLE MODEL FOR CARDIAC ARREST PREDICTION

BY

NWAMU CHUKWUEMEKE EMMANUEL

PG/PSC2215567

A PROJECT SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE, FACULTY OF COMPUTING, UNIVERSITY OF BENIN, IN PARTIAL FULFILMENT FOR THE REQUIREMENTS FOR THE AWARD OF THE MASTER IN COMPUTER SCIENCE (M.Sc) DEGREEE IN COMPUTER SCIENCE

APRIL, 2026

DECLARATION

I, (Nwamu Chukwuemeke Emmanuel) with the matriculation number PG/PSC2215567, by this means, declare that this research project on 'A Hybrid Feature Selection with Stack-ensemble Model for Cardiac Arrest Prediction' which is being submitted by me. In partial fulfillment for the award of the Masters of Science Degree in Computer Science, is an authentic report carried out by me during the 2023/2024 academic session. I further undertake the work embodied in the report has not been submitted previously for the award of any degree or diploma by me to any other university or institution.

_____ _____

Nwamu Chukwuemeke Emmanuel DATE

PG/PSC2215567

CERTIFICATION

This is to certify that this project work was carried out by Nwamu Chukwuemeke Emmanuel with the matriculation number PG/PSC2215567 of the department of computer science, University of Benin, under my supervision and it is adequate in scope and content for the award of Master in Computer Science (M.Sc.) Degree in Computer Science of the University of Benin.

_____ _____

Prof. Mrs. V.I. Osubor Date

Project Supervisor

APPROVAL

This project is hereby approved by the Department of Computer Science in partial fulfillment of the requirements for the award of Master in Computer Science (M.Sc.) Degree in Computer Science of the University of Benin, Benin City, Nigeria.

_____ _____

Dr. Mrs. Rosemary Usiobiafo Date

Head of Department

DEDICATION

This project is dedicated to God Almighty for the gift of life and His love and mercies throughout my study period; my parents Mr. and Mrs. Austin Nwamu for their unconditional love, support, encouragement and belief in me, and also to my siblings for their immense sacrifice during the course of this project work.

ACKNOWLEDGMENT

I want to express my sincere appreciation to the Almighty God for His endless mercies, power, provision, and, most importantly, His unwavering faithfulness and love throughout my academic journey. My deepest gratitude goes to my esteemed supervisor, Prof. V. I. Osubor, whose meticulous guidance, understanding, and encouragement have been pivotal to the success of this thesis. Her availability for consultations, insightful feedback, and consistent support have been invaluable at every stage of this research.

I also extend my heartfelt thanks to the Head of Department, Dr. Mrs. Rosemary Usiobiafo, whose impactful leadership significantly enriched my experience throughout the program. I must also acknowledge the valuable contributions of Prof. V. I. Osubor for her significant impact in my academic journey. I also extend my warm appreciation to all members of the Postgraduate Board and senior lecturers of the Department of Computer Science at the University of Benin, particularly Prof. V.V.N. Akwukwuma, Prof. F.A. Egbokhare, Prof. Alice Egwali, Prof. G.O. Ekuobase, Prof. F. Amadi, Prof. S. Konyeha, Dr. E. Nwelih, Dr. (Mrs.) Aziken, Mr. E. Obasohan and Dr. (Mrs) R.O Osaseri for their willingness to share their vast expertise and intellectual insights throughout the course of the program. Their critical assessments played a vital role in steering this research toward successful completion.

My gratitude also goes to all other lecturers not mentioned above and non-teaching staff in the Department of Computer Science, University of Benin, for the various forms of assistance that supported my progress throughout the program. I am grateful to the department for providing the

necessary academic structure and support that made this research possible. I acknowledge with thanks all individuals who contributed, directly or indirectly, to the successful completion of this thesis.

I am specifically greatful to my parents Mr. and Mrs.Austin Nwamu, and my siblings Oluchukwu Nwamu and Christabel Nwamu for their steadfast prayers and all round support which has been indispensable.

Finaly, I wish to acknowledge my friends Agbosua Daniel, Egbo Paschal, Edamivoh Franklyn and my colleagues Mr. Evbuomwan Gregory, Mrs Mojibola Abigail and Late Mrs. Edeoghon Lillian for their encouragements.

May Almighty God bless and protect you all.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AUC	-	Area Under the Curve
AUC-ROC	-	Area Under the Curve-Receiver Operating Characteristic
DT	-	Decision Tree
FA	-	Firefly Algorithm
FN	-	False Negative
FP	-	False positive
FS	-	Feature Selection
RFE	-	Recursive Feature Elimination
ROC	-	Receiver Operating Characteristic
RFE-FA	-	Recursive Feature Elimination-Firefly Algorithm
SVM	-	Support Vector Machine
TN	-	True Negative
TP	-	True Positive

ABSTRACT

Cardiovascular Diseases (CVDs) remain the paramount global health challenge, responsible for approximately 17.9 million deaths annually, with cardiac arrest representing a particularly critical and sudden event. Despite advancements in machine learning, accurate prediction is often hindered by highdimensional clinical data ("the curse of dimensionality") and the performance limitations of standalone classifiers. Consequently, this study aimed to develop a framework that integrates a novel Hybrid Recursive Feature Elimination-Firefly (RFE-Firefly) feature selection technique with a stacked ensemble classifier to enhance predictive accuracy. The study utilized a structured heart disease dataset from the Kaggle repository, subjecting it to rigorous pre-processing, including normalization and class balancing via the Synthetic Minority Over-sampling Technique (SMOTE). The proposed Hybrid RFEFirefly algorithm was implemented to optimize the feature space, successfully reducing the data to a parsimonious subset of seven critical features. For classification, a stacking ensemble model was constructed using five diverse base learners which are Decision Tree, Support Vector Machine, Logistic Regression, Random Forest, and XGBoost and unified by Random Forest as meta-learner to synthesize their predictive strengths. Experimental results demonstrated that the proposed Hybrid RFE-FA Stack Ensemble achieved a superior accuracy of 90.57% and an F1-score of 0.7619. This performance significantly outperformed individual base classifiers as well as ensembles using standalone feature selection methods. The study concludes that integrating hybrid feature selection strategy with an advance stack ensemble classifier effectively mitigates overfitting and provides a robust, highperformance tool for clinical decision-making in cardiac arrest detection.

CHAPTER ONE

INTRODUCTION

1.1 Background to the Study

Cardiovascular Diseases (CVDs) remain the paramount global health challenge, responsible for an estimated 17.9 million deaths annually (World Health Organization, as cited in Ogunpola et al., 2024; Prasanna et al., 2025). Among these, cardiac arrest represents a critical and often terminal event, characterized by the sudden cessation of heart function. The unpredictability and high fatality rate of cardiac arrest underscore an urgent imperative for advanced predictive frameworks that can identify atrisk individuals, facilitating timely intervention and mitigating mortality (Ali et al., 2021). The integration of Artificial Intelligence (AI) and Machine Learning (ML) into clinical decision-support systems has emerged as a transformative frontier, offering unprecedented capabilities to analyze complex clinical data and forecast adverse cardiac events with high precision (Shehab et al., 2022; Ahmad et al., 2023).

The predictive performance of any ML model is fundamentally anchored on two pillars: the discriminative power of the learning algorithm and the quality of the feature set upon which it is trained. Clinical datasets, such as those derived from electronic health records, are typically high-dimensional, harboring a mix of relevant, redundant, and noisy features. This "curse of dimensionality" can significantly impair model performance by introducing overfitting, increasing computational overhead, and reducing the generalizability of the results (Saeys et al., 2023). Consequently, Feature Selection (FS) has been established as an indispensable pre-processing step to distill the most prognostically significant variables, thereby enhancing model accuracy, efficiency, and interpretability (Manikandan et al., 2024).

Traditional FS methodologies, including filter, wrapper, and embedded approaches, have been extensively applied. For instance, Manikandan et al. (2024) demonstrated the efficacy of the Boruta algorithm, a robust wrapper method based on Random Forest, which enhanced the performance of classifiers like Decision Tree (DT) and Support Vector Machine (SVM) for heart disease stratification. Similarly, Recursive Feature Elimination (RFE) is a powerful wrapper technique that iteratively builds models, ranks features by importance, and eliminates the least contributive ones, yielding a refined and optimal feature subset (Ansari et al., 2023). Despite their utility, these conventional methods are often susceptible to local optima and may overlook complex, non-linear feature interactions present in biomedical data (Natarajan et al., 2024).

This limitation has catalyzed the adoption of nature-inspired metaheuristic algorithms, which excel in navigating vast and complex search spaces to discover near-optimal solutions. The Firefly Algorithm (FA), inspired by the flashing behaviour and social dynamics of fireflies, is a prominent example (Yang, 2021). Its strength in FS stems from an inherent mechanism that balances exploration (global search) and exploitation (local refinement), enabling it to efficiently identify a parsimonious yet highly predictive feature subset. The success of FA is evidenced in recent studies; Natarajan et al. (2024) utilized an optimized FA to select eight critical features from the Z-Alizadeh Sani dataset, which, when fed into a stacking ensemble, achieved a commendable accuracy of 86.79%. This demonstrates FA's superior capability to handle the intricate, multi-factorial patterns characteristic of cardiovascular pathophysiology.

A burgeoning consensus in the literature indicates that hybrid FS approaches, which synergize the strengths of disparate techniques, can yield performance superior to any single method (Hancer, 2022).

A hybrid framework can effectively mitigate the individual shortcomings of its components. For example, while RFE provides a deterministic, model-specific feature ranking, it can be myopic in its search. The FA, conversely, offers a stochastic, global optimization capability but may be computationally intensive in high-dimensional spaces. The integration of these two into a Hybrid Recursive Feature Elimination-Firefly (RFE-Firefly) technique presents a novel methodology. In this paradigm, RFE performs an initial coarse filtering to reduce the feature space dimensionality and provide a ranked shortlist. The FA then conducts a fine-tuned, global search within this condensed subspace to identify the ultimate, optimized feature set. This synergistic strategy is poised to enhance predictive power and model generalizability, representing a significant innovation yet to be fully explored for cardiac arrest prediction.

Concurrently, the selection of the classification model is paramount. A diverse suite of base classifiers, including Decision Tree (DT), Support Vector Machine (SVM), Logistic Regression (LR), Random Forest (RF), and XGBoost, has been widely validated for cardiovascular risk prediction, each with distinct inductive biases and strengths (Dwivedi, 2018; Ogunpola et al., 2024). However, to harness their collective predictive power and overcome the limitations of individual models, ensemble learning techniques are employed. Stacking, or stacked generalization, is a sophisticated ensemble method that combines multiple base classifiers (level-0 models) through a meta-learner (level-1 model) which learns to optimally weigh their predictions (Wolpert, 2021). The choice of Random Forest as the meta-learner is strategically sound; its inherent robustness to noise and ability to model complex, non-linear decision boundaries based on the outputs of DT, SVM, LR, RF, and XGBoost make it ideal for synthesizing a final, highly accurate prediction (Prasanna et al., 2025). This stacked ensemble approach has been

shown to significantly boost performance, as demonstrated by Ogunpola et al. (2024), where XGBoost achieved 98.5% accuracy, and by advanced frameworks like the Clustered Butterfly Optimization Algorithm which reached 97.03% (Prasanna et al., 2025).

Therefore, this study is motivated by the critical gap in developing a comprehensive and highly accurate intelligent system for cardiac arrest prediction. It proposes a novel intelligent healthcare framework that integrates a Hybrid RFE-Firefly technique for robust feature selection with a stacked ensemble classifier. The ensemble strategically leverages the diverse capabilities of DT, SVM, LR, RF, and XGBoost as base learners, unified by a Random Forest meta-learner. By fusing the structured, modelguided refinement of RFE with the global optimization prowess of the Firefly Algorithm, this research aims to construct a maximally discriminative feature set. This optimized input empower the stacked model to identify individuals at imminent risk of cardiac arrest with superior precision, thereby offering a powerful tool for clinical decision-support and proactive, personalized patient management.

1.2 Statement of the Problem

Cardiovascular diseases (CVDs) persist as the leading cause of global mortality, with cardiac arrest being a particularly critical and often fatal outcome (Prasanna et al., 2025). Despite advancements, accurate early prediction of cardiac arrest remains a major clinical challenge. Current risk models and subjective symptom assessment often lead to delayed diagnosis and missed prevention opportunities, causing poor patient outcomes and significant healthcare burdens (Ali et al., 2021).

While machine learning (ML) offers promising solutions, developing reliable cardiac arrest prediction models faces two key obstacles. First, clinical datasets contain high-dimensional, redundant, and noisy features (Saeys et al., 2023). Using such data directly causes overfitting, reduced generalizability, and

unreliable predictions. Although feature selection methods exist, traditional approaches like Boruta or standalone RFE often find suboptimal feature subsets, missing complex non-linear patterns in cardiovascular data (Manikandan et al., 2024; Natarajan et al., 2024).

Second, individual ML classifiers including Decision Trees, SVM, Logistic Regression, Random Forest, and XGBoost have inherent limitations when used alone (Dwivedi, 2018). Single models may show bias, high variance, or incomplete pattern recognition, limiting their predictive power. While ensemble methods like stacking can address these issues, their effectiveness depends heavily on having an optimally selected feature set (Wolpert, 2021). Without proper feature selection, even advanced ensembles underperform.

Therefore, there's an urgent need for an intelligent framework that simultaneously addresses both feature selection and classification challenges. Although recent studies have explored hybrid FS methods and sophisticated ensembles separately, a significant gap remains in creating an integrated system that combines a novel hybrid FS approach with a powerful stacked ensemble specifically for cardiac arrest prediction (Ogunpola et al., 2024; Talukdar et al., 2024).

This research aims to solve the problem of imprecise cardiac arrest prediction by developing an intelligent healthcare framework. The core issue is the absence of a model that effectively uses a Hybrid RFE-Firefly technique to identify a compact, highly informative feature set, which then trains a stacked ensemble model with Random Forest as meta-learner to achieve superior predictive accuracy and reliability.

1.3 Aim and Objectives

The aim of this study is to develop a model for the accurate prediction of cardiac arrest by integrating a hybrid feature selection with stacked-ensemble model.

The objectives are to:

1. design and implement a Hybrid feature selection with Stack-ensemble model;
2. construct a robust stacked ensemble classification model using multiple base learners.
3. evaluate the predictive performance of the proposed model

1.4 Scope of the Study

This study is limited to developing and evaluating a predictive model for cardiac arrest using structured clinical datasets. The scope encompasses the creation of a Hybrid RFE-Firefly feature selection technique and a stacked ensemble classifier, excluding real-time clinical deployment and prospective patient trials

1.5 Significance of the Study

This research holds significant implications for both the academic field of computational intelligence and the practical domain of clinical healthcare. The study makes a substantive contribution by proposing a novel Hybrid Recursive Feature Elimination-Firefly (RFE-Firefly) algorithm. This hybrid feature selection technique is designed to overcome the limitations of conventional methods by synergizing the structured, model-specific elimination of RFE with the robust, global search capabilities of the Firefly Algorithm. The development and validation of this advanced feature selection model, coupled with a sophisticated stacked ensemble, provides a new, powerful framework for handling high-dimensional clinical data, setting a benchmark for future research in medical data mining and predictive analytics.

Also, the study addresses a critical healthcare challenge. The intelligent framework developed has the direct potential to revolutionize cardiac arrest risk stratification by enabling earlier and more accurate identification of high-risk individuals. This can facilitate timely medical interventions, inform personalized treatment plans, and ultimately reduce mortality rates. By providing clinicians with a reliable decision-support tool, the study contributes to improved patient outcomes, optimized resource allocation in healthcare systems, and a significant reduction in the socio-economic burden associated with cardiovascular diseases.

1.6 Study Motivation

The motivation for this research is driven by the critical gap between the life-threatening nature of cardiac arrest and the limitations of current predictive methodologies. Despite being a leading cause of global mortality, the accurate and early prediction of cardiac arrest remains a formidable challenge in clinical practice. Existing risk assessment models and standalone machine learning approaches often fail to achieve the necessary precision and reliability, leading to preventable fatalities and placing a immense strain on healthcare systems.

This study is propelled by the urgent need to develop a more sophisticated predictive framework. We are motivated by the potential of hybrid intelligent systems to overcome the shortcomings of conventional methods. By integrating a novel Hybrid Recursive Feature Elimination-Firefly algorithm for optimal feature selection with a powerful stacked ensemble classifier, this research seeks to create a robust tool that can empower healthcare providers with a more accurate, data-driven means of identifying at-risk patients, thereby enabling life-saving early interventions and improving overall cardiovascular care outcomes.

1.7 Research Question

Some of the research questions that guided this research work are:

1. How can we create an optimal Hybrid RFE-Firefly algorithm to select the best predictive features for cardiac arrest?
2. How much does a stacked ensemble model improve prediction robustness compared to individual classifiers?
3. Does our complete intelligent framework outperform traditional methods in predicting cardiac arrest?

CHAPTER TWO

LITERATURE REVIEW

2.1 Overview to Machine Learning for Heart Disease Prediction

The pursuit of accurate and early prediction of cardiac arrest represents a paramount challenge within modern healthcare, a challenge that sits at the intersection of clinical medicine and computational science. This chapter is dedicated to a systematic and critical examination of the existing scholarly work that forms the foundation for this research. The structure of this review is designed to progress logically from the broad context of the problem to the specific technical methodologies that underpin the proposed solution. It begins by establishing the significance of cardiovascular diseases and the role of artificial intelligence in addressing them. The discussion then narrows to a detailed analysis of the machine learning classifiers commonly employed for cardiac diagnosis, with a particular emphasis on the evolution from individual models to sophisticated ensemble techniques. A parallel, in-depth exploration of feature selection strategies follows, tracing their development from traditional filter and

wrapper methods to advanced nature-inspired metaheuristics. The convergence of these two streams of classification and feature selection culminates in an examination of hybrid frameworks, which seek to synergize the strengths of multiple approaches.

The central purpose of this chapter is not merely to catalog previous studies but to engage in a critical synthesis of the literature on machine learning for heart disease prediction. This involves a focused analysis on two pivotal pillars: the efficacy of various classification models and the transformative impact of feature selection techniques on model performance. By critically evaluating the successes and limitations of existing methodologies, this review aims to delineate the specific research gap that the current study seeks to fill. The trajectory of the literature reveals a clear consensus: while ensemble classifiers like stacking have demonstrated superior predictive power (Wolpert, 2021; Ogunpola et al., 2024), and while advanced feature selection such as the Firefly Algorithm has shown promise in navigating complex feature spaces (Natarajan et al., 2024; Talukdar et al., 2024), a significant void remains and this void is the lack of a fully integrated, intelligent framework that strategically combines a novel hybrid feature selection mechanism with a powerful stacked ensemble, specifically optimized for the critical task of predicting cardiac arrest. It is this identified gap the untapped potential of a hybrid Recursive Feature Elimination-Firefly (RFE-Firefly) technique is working in concert with a robust stacking ensemble that provides the definitive rationale and direction for the present research, as established through the following critical appraisal of the field.

2.2 Theoretical Foundations of Cardiovascular Disease Prediction

The development of effective predictive models for cardiac arrest is fundamentally grounded in understanding both the profound clinical challenge it represents and the transformative computational methodologies available to address it. The theoretical foundation of this research rests upon two interconnected pillars: the significant and persistent global burden of cardiovascular diseases (CVDs) and the paradigm shift enabled by Artificial Intelligence (AI) in medical diagnostics. Firstly, CVDs remain the paramount cause of mortality worldwide, with cardiac arrest standing as a particularly critical and often terminal event. The World Health Organization, as cited in recent studies, estimates that CVDs are responsible for nearly 18 million deaths annually, a statistic that underscores the urgent and nonnegotiable need for advanced prognostic tools (Ogunpola et al., 2024; Prasanna et al., 2025). The sudden, unpredictable nature of cardiac arrest, coupled with its devastatingly high fatality rate, creates a clear imperative for systems capable of identifying at-risk individuals well before the acute event, thereby opening a crucial window for preventative intervention (Ali et al., 2021).

Secondly, the theoretical approach to solving this problem has been radically redefined by the advent of AI and Machine Learning (ML). The transition from traditional, often subjective, clinical risk scores to data-driven, algorithmic prediction marks a frontier in modern healthcare. Machine learning offers the unprecedented capability to analyze complex, high-dimensional clinical data from sources like electronic health records, uncovering subtle, non-linear patterns that may elude conventional analysis (Shehab et al., 2022). This capacity positions ML not merely as a supplementary tool but as a transformative technology for clinical decision-support systems, enabling a move from reactive treatment to proactive, personalized risk stratification (Ahmad et al., 2023). Therefore, the theoretical

bedrock of this study is the convergence of a critical, unmet medical need with a powerful, disruptive technological capability, establishing a compelling rationale for the pursuit of intelligent predictive frameworks in cardiology.

2.2.1 The Global Burden of Cardiovascular Diseases (CVDs)

Cardiovascular Diseases (CVDs) persistently represent the most significant global health challenge of our time, constituting a leading cause of mortality and morbidity that transcends geographical and socioeconomic boundaries. The statistics on their prevalence and impact are both staggering and sobering. According to the World Health Organization, as synthesized in recent epidemiological studies, CVDs are responsible for an estimated 17.9 million deaths annually, a figure that underscores their unparalleled position as the foremost cause of death worldwide (Prasanna et al., 2025; Ogunpola et al., 2024). This number is not merely a statistic but represents a profound and ongoing public health crisis, accounting for approximately one-third of all global deaths and placing an immense burden on healthcare systems, economies, and societies at large. The pervasive nature of CVDs, fueled by risk factors such as sedentary lifestyles, unhealthy diets, and hypertension, confirms that their prediction and management remain a critical priority for medical science and healthcare policy.

Within the broad spectrum of cardiovascular ailments, cardiac arrest represents a particularly critical and often terminal event, characterized by the sudden, abrupt cessation of heart function. Unlike other cardiovascular events that may present with escalating symptoms, cardiac arrest is notoriously unpredictable, striking often without warning and leading to immediate loss of consciousness and death if not treated within minutes (Ali et al., 2021). This suddenness is a key factor in its devastatingly high

fatality rate; the majority of out-of-hospital cardiac arrests prove fatal, and even in-hospital events carry a grave prognosis (Shehab et al., 2022). The imperative for early prediction is, therefore, not just a clinical objective but a moral one. The current paradigm, which often relies on late-stage symptom presentation or generalized risk scores, is insufficient to mitigate the high mortality associated with this acute event. The window for effective intervention is exceptionally narrow, making the ability to identify at-risk individuals days, weeks, or months in advance absolutely crucial (Ahmad et al., 2023; Natarajan et al., 2024). This specific challenge the confluence of sudden onset, extreme lethality, and the limitations of conventional assessment creates an urgent and non-negotiable demand for advanced, intelligent frameworks capable of forecasting cardiac arrest with high precision, thereby enabling preventative strategies and saving lives.

2.2.2 The Role of Artificial Intelligence and Machine Learning in Healthcare

The integration of Artificial Intelligence (AI) and Machine Learning (ML) into the healthcare ecosystem marks a paradigm shift, heralding a new era of diagnostic precision and proactive patient management. These technologies possess a transformative potential that is fundamentally reshaping clinical decisionsupport systems. Unlike static, rule-based expert systems of the past, modern AI/ML frameworks can learn from vast quantities of complex, multi-modal data including electronic health records, medical imaging, and genomic sequences to uncover hidden patterns and associations. This capability allows them to function as powerful adjuncts to clinical judgment, providing physicians with data-driven insights that enhance diagnostic accuracy, personalize treatment plans, and predict individual patient risks with a previously unattainable level of sophistication (Shehab et al., 2022). The transformative potential lies in their ability to move healthcare from a reactive model, where

intervention occurs after symptom manifestation, to a predictive and preventative one, where risks are identified and mitigated proactively.

This shift represents a decisive transition from traditional statistical methods to advanced data-driven predictive modeling. Conventional statistical approaches, while valuable for establishing correlations and testing hypotheses within constrained parameters, often struggle with the high-dimensionality, noise, and complex non-linear relationships inherent in modern clinical datasets. In contrast, machine learning algorithms, particularly ensemble and deep learning models, thrive in this environment. They are designed to automatically model intricate interactions between a multitude of variables without relying on rigid pre-specified assumptions (Ahmad et al., 2023). For instance, where a traditional model might assess a limited set of risk factors independently, an ML model can dynamically weigh and combine hundreds of features from resting blood pressure and cholesterol levels to more subtle indicators like exercise-induced ST depression to generate a holistic risk score for a cardiac event. This transition empowers a more nuanced and powerful form of predictive analytics, enabling the development of intelligent systems that can continuously learn and improve, thereby offering a robust foundation for the next generation of clinical decision-support tools aimed at combating critical challenges like cardiac arrest.

2.3 Machine Learning for Heart Disease Diagnosis: A Review of Classifiers

The selection of an appropriate machine learning classifier is a cornerstone in developing robust predictive models for heart disease. A diverse array of algorithms, each with distinct theoretical underpinnings and inductive biases, has been applied to this task. This section provides a critical review

of the primary individual classifiers that often serve as the foundational building blocks for more complex ensemble systems, examining their core principles, advantages, and inherent limitations.

2.3.1 Individual Base Classifiers

Decision Trees (DT) operate on a simple yet intuitive principle of recursively partitioning the data into subsets based on feature values, forming a tree-like structure of decision nodes and leaves that represent outcomes. The primary advantage of DTs lies in their high interpretability; the resulting model can be easily visualized and understood, even by non-experts, making it possible to trace the reasoning behind a specific prediction (Saeys et al., 2023). This "white-box" nature is highly valued in clinical settings where understanding the why behind a diagnosis is crucial. However, a significant limitation is their propensity to overfit the training data, especially when the tree grows too deep, learning noise and specific details of the training set rather than the general underlying patterns, which consequently harms their performance on unseen data (Manikandan et al., 2024).

Support Vector Machines (SVM) are based on the principle of identifying an optimal hyperplane that maximally separates data points of different classes in a high-dimensional space. Their effectiveness is particularly notable in high-dimensional spaces, which are common in clinical datasets with numerous features, as they maintain stability even when the number of dimensions exceeds the number of samples (Shehab et al., 2022). A critical aspect of SVM is kernel selection, where functions like the Radial Basis Function (RBF) kernel are often employed to project data into a higher-dimensional space where it becomes linearly separable, thereby allowing SVM to capture complex, non-linear relationships between clinical features.

Logistic Regression (LR) is a foundational statistical model that estimates the probability of a binary outcome by fitting data to a logistic function. Its principle strength lies in its simplicity, computational efficiency, and the fact that it provides a well-calibrated probabilistic output (Ahmad et al., 2023). Due to its straightforward nature and strong performance as a benchmark, LR is frequently used as a baseline model against which the performance of more complex algorithms is measured. Its coefficients also offer a degree of interpretability, indicating the direction and relative influence of each feature on the predicted outcome.

Random Forest (RF) is an ensemble method that operates on the principle of bagging (Bootstrap Aggregating). It constructs a multitude of decision trees during training and outputs the mode of their classes for prediction. This approach significantly enhances robustness to noise and overfitting, a common pitfall of individual DTs, by averaging the results of multiple decorrelated trees (Natarajan et al., 2024). A key ancillary benefit of RF is its provision of feature importance measures, which rank features based on their mean decrease in impurity across all trees, offering valuable insights into the most prognostically significant variables in the dataset.

XGBoost (Extreme Gradient Boosting) represents a highly advanced implementation of the gradient boosting principle, which involves building an ensemble of trees sequentially, where each new tree corrects the errors made by the previous ones. Its core principle involves optimizing a defined loss function using gradient descent, and it incorporates regularization to control model complexity and prevent overfitting. This sophisticated approach has led to its reputation for high predictive performance,

often achieving state-of-the-art results on structured datasets, which explains its widespread use and success in machine learning competitions and practical applications (Ogunpola et al., 2024).

Synthesis

A synthesis of the literature, as highlighted in comparative studies, reveals that no single classifier is universally superior; each possesses a unique profile of strengths and weaknesses. Simple, interpretable models like Logistic Regression provide excellent baselines, while Decision Trees offer transparency at the cost of potential instability. Support Vector Machines are powerful for complex, non-linear problems but can be sensitive to kernel and parameter choices. Ensemble methods like Random Forest and XGBoost generally demonstrate superior predictive accuracy and robustness by leveraging the power of multiple models, with XGBoost often having a slight edge due to its advanced regularization and sequential error-correction approach (Dwivedi, 2018; Prasanna et al., 2025). This comparative analysis underscores the rationale for moving beyond individual classifiers and leveraging their collective power through ensemble techniques, which will be discussed in the subsequent section.

2.3.2 Ensemble Learning Techniques

A pivotal advancement in machine learning for complex prediction tasks like cardiac arrest diagnosis is the development of ensemble learning techniques. The core concept and rationale behind ensemble methods is the strategic combination of multiple base models to create a composite predictor that is more robust, accurate, and generalizable than any of its individual components. This principle, often termed the "wisdom of crowds" for algorithms, operates on the foundation that a collection of weak learners can form a strong learner, as the errors of one model can be compensated for by the correct

predictions of others (Wolpert, 2021; Zhou, 2021). By aggregating diverse perspectives, ensemble methods mitigate the risk of relying on a single model that may be biased, have high variance, or fail to capture the full complexity of the data, thereby enhancing overall performance on unseen data.

Among the prominent ensemble strategies, Bagging (Bootstrap Aggregating) is primarily designed to reduce model variance and prevent overfitting. This technique, exemplified by the Random Forest algorithm, creates multiple versions of a base learner (e.g., Decision Trees) by training each on a different bootstrap sample (a random sample with replacement) of the original training data. During prediction, the outputs of these individual models are aggregated, typically through a majority vote for classification. This process stabilizes the output, as the collective decision is less sensitive to the noise in the training data that might have adversely affected a single model (Natarajan et al., 2024).

In contrast, Boosting is a sequential ensemble technique focused on reducing bias. Unlike bagging, which builds models independently, boosting constructs them sequentially, with each new model attempting to correct the errors made by the previous ones. Algorithms like XGBoost assign higher weights to misclassified instances, forcing subsequent learners to focus on the harder-to-predict cases. This iterative error-correction mechanism allows boosting ensembles to often achieve higher predictive performance by converting many weak learners into a single powerful model, though it can be more prone to overfitting if not properly regularized (Ogunpola et al., 2024).

A more sophisticated ensemble approach is Stacked Generalization (Stacking), which introduces a hierarchical architecture. Instead of simply aggregating the predictions of base models, stacking employs a two-level learning structure. At level-0, a diverse set of heterogeneous base learners (e.g., DT, SVM,

LR, RF) are trained on the original dataset. Their predictions on a hold-out validation set are then used as input features to train a meta-learner (level-1 model). The meta-learner's role is to discover the optimal way to combine the base models' predictions. For this critical role, Random Forest is a strategically sound choice as the meta-learner. Its inherent ability to model complex, non-linear decision boundaries based on the outputs of the base learners makes it exceptionally well-suited for learning the best weighting scheme, effectively determining when to trust one base model's prediction over another's (Prasanna et al., 2025).

Empirical evidence strongly supports the superiority of stacking ensembles in healthcare analytics. Recent studies have consistently demonstrated that stacked models outperform individual classifiers and often other ensemble methods. For instance, Ogunpola et al. (2024) showcased how ensemble methods, including stacking, achieved superior accuracy in cardiovascular disease detection. Furthermore, Prasanna et al. (2025) validated that a sophisticated ensemble framework, which leverages the collective intelligence of multiple base models, yielded a highly accurate and reliable predictive system for cardiac conditions. These findings substantiate the rationale for employing a stacking ensemble as the core classifier in the proposed intelligent healthcare framework for cardiac arrest prediction.

2.4 The Critical Role of Feature Selection in Clinical Predictive Modeling

2.4.1 The Curse of Dimensionality in Clinical Datasets

The predictive power of machine learning models for cardiac arrest is significantly limited by the high dimensionality of clinical datasets, a problem known as the "curse of dimensionality." This occurs when the number of features (p) is large compared to the number of patient observations (n), leading to three

major challenges. First, overfitting becomes a critical issue, where models memorize noise and spurious patterns in the training data, resulting in poor performance and unreliable predictions on new patient data, which is dangerous in a clinical context (Saeys et al., 2023). Second, high dimensionality leads to increased computational cost, causing exponential growth in training times and memory requirements, which hinders model development and real-time clinical deployment (Shehab et al., 2022). Finally, it causes reduced model interpretability, creating a "black box" problem where the reasoning behind predictions is obscured. This lack of transparency undermines clinical trust and utility, as practitioners need to understand the drivers of a model's decision (Ahmad et al., 2023). Consequently, feature selection is an essential pre-processing step to overcome these challenges and create viable clinical prediction tools.

2.4.2 Taxonomies of Feature Selection Methods

To combat the challenges posed by high-dimensional clinical data, various feature selection (FS) methodologies have been developed, which can be broadly categorized into three main types: filter, wrapper, and embedded methods. Each category represents a different philosophical and technical approach to identifying the most relevant feature subset.

Filter Methods operate independently of any machine learning algorithm. They assess the relevance of features based on intrinsic characteristics of the data, such as statistical measures or correlation coefficients. For example, a correlation-based filter might rank features according to their individual correlation with the target variable. The primary advantage of filter methods is their computational efficiency and speed, as they do not involve the costly process of model training (Saeys et al., 2023). However, this speed comes with a significant drawback: filter methods evaluate each feature in isolation,

which means they may ignore complex interactions and dependencies between features. Consequently, they might select a set of features that are individually relevant but contain redundancies, potentially overlooking a more optimal, synergistic subset.

Wrapper Methods take a different approach by using the performance of a specific predictive model as the objective function to evaluate feature subsets. This makes them "wrapped" around a learning algorithm. A search strategy (e.g., forward selection, backward elimination) is used to generate candidate feature subsets, which are then used to train a model. The performance of this model on a validation set determines the quality of the feature subset.

Recursive Feature Elimination (RFE) is a powerful and widely used wrapper technique. Its process is iterative and model-specific. It begins by training a chosen model (e.g., SVM or Random Forest) on the entire set of features. The model then assigns an importance score to each feature. The least important feature(s) are pruned or eliminated from the current set. This cycles training the model, ranking features, and eliminating the weakest and repeats recursively until the desired number of features is attained (Ansari et al., 2023). This iterative refinement allows RFE to produce a highly tailored and performant feature subset for its associated classifier, though it can be computationally intensive.

The Boruta Algorithm is a robust wrapper method based on the Random Forest classifier. Its key innovation is the creation of "shadow features," which are copies of the original features where the values have been shuffled to remove their correlation with the target. The algorithm then runs a Random Forest and compares the importance of the real features against the maximum importance recorded

among the shadow features. Any real feature that consistently fails to outperform the shadow features in statistical significance tests is deemed irrelevant and permanently rejected (Manikandan et al., 2024). This all-relevant FS approach is particularly effective for finding all features that have some predictive power, not just a non-redundant set.

Embedded Methods integrate the feature selection process directly into the model training phase, offering a compromise between the computational efficiency of filters and the performance-oriented approach of wrappers. These methods perform FS as an inherent part of the model's optimization process.

Lasso Regression (L1 Regularization) is a classic example, which adds a penalty term to the regression loss function that forces the sum of the absolute values of the coefficients to be small. This has the effect of shrinking the coefficients of less important features exactly to zero, effectively excluding them from the model (Shehab et al., 2022). Tree-based models like Random Forest and XGBoost provide built-in feature importance measures, such as Mean Decrease in Impurity or Gain. These metrics quantify how much each feature contributes to improving the model's purity across all the trees in the ensemble (Natarajan et al., 2024). Features can then be selected based on their importance scores, leveraging the model's inherent structure to identify key predictors.

2.5 Nature-Inspired Metaheuristic Algorithms for Feature Selection

Nature-inspired metaheuristic algorithms provide an efficient and intelligent solution to the computationally challenging problem of feature selection in high-dimensional data. They address the limitations of exhaustive searches by employing strategic, population-based frameworks inspired by

biological, physical, and ecological systems, such as flocking birds, ant colonies, and evolutionary processes. Key examples include Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Genetic Algorithms (GA), Firefly Algorithm (FA), and Recursive Feature Elimination (RFE) alongside newer inspirations like grey wolf hunting and whale foraging behavior.

These algorithms excel at exploring vast search spaces to identify optimal feature subsets by balancing exploration and exploitation, thereby avoiding local optima. They operate by optimizing an objective function that typically seeks to maximize model performance while minimizing the number of features. Their main strengths are adaptability and powerful global search capabilities, making them vital for complex tasks like microarray analysis and image classification. However, challenges such as parameter sensitivity, computational cost, and result variability remain. Despite these drawbacks, these natureinspired techniques have become indispensable tools for building interpretable and high-performing predictive models.

2.5.1 Rationale for Metaheuristics in FS

The limitations of traditional feature selection methods have catalyzed the exploration of more advanced optimization techniques, leading to the adoption of nature-inspired metaheuristic algorithms. The primary rationale for this shift lies in their superior capability to overcome the fundamental constraints of filter, wrapper, and embedded methods when dealing with the complex, high-dimensional search spaces typical of clinical data.

A central justification for using metaheuristics is their effectiveness in addressing the limitations of traditional FS methods. Conventional techniques, particularly wrappers, are often susceptible to

converging to local optima. For instance, a greedy sequential search might settle on a suboptimal feature subset because it cannot escape a locally good solution to find a globally superior one. Furthermore, many traditional methods struggle with capturing complex, non-linear interactions between features. A filter method might discard a feature with low individual correlation to the target, even if it is critically important when combined with another feature. Metaheuristics, such as the Firefly Algorithm, are explicitly designed to navigate these complex landscapes. They balance "exploration" (searching new areas of the feature space) and "exploitation" (refining promising solutions), which enables them to escape local optima and discover feature subsets that exhibit strong synergistic effects, a capability demonstrated in recent studies (Natarajan et al., 2024; Talukdar et al., 2024).

This approach is formalized by framing FS as a combinatorial optimization problem. The challenge of selecting an optimal feature subset from a pool of features is immense, as the number of possible subsets grows exponentially. This makes an exhaustive search computationally infeasible for all but the smallest datasets. In this framework, each potential feature subset is considered a "solution" in a vast search space. The goal is to find the solution that optimizes an objective function, which is typically a combination of model performance (e.g., accuracy, F1-score) and parsimony (a smaller number of features). Metaheuristic algorithms are exceptionally well-suited for this task, as they are fundamentally designed to find near-optimal solutions in large, complex, and discontinuous search spaces where traditional optimization techniques fail (Hancer, 2022). By treating feature selection as an optimization challenge, metaheuristics provide a powerful and flexible methodology for identifying highly predictive and compact feature sets that are crucial for building robust clinical prediction models.

2.5.2 The Firefly Algorithm (FA)

The Firefly Algorithm is a nature-inspired metaheuristic that derives its conceptual framework from the flashing light patterns and social behavior of fireflies in nature. Fundamentally, the algorithm is constructed upon three core principles: attraction, brightness, and movement. In this computational model, each firefly represents a potential solution within the search space, and its perceived "brightness" corresponds directly to the quality or fitness of that solution, typically measured by an objective function. The primary mechanism driving the algorithm is the attraction between fireflies, where individuals with lower brightness are naturally drawn to move toward those exhibiting higher brightness. This iterative movement through the solution space enables the algorithm to efficiently explore a wide range of potential solutions while progressively concentrating on the most promising regions.

A principal strength of the Firefly Algorithm lies in its inherent ability to dynamically balance exploration, the search for new areas, and exploitation, the refinement of known good solutions. This balance is crucial for effectively navigating complex optimization landscapes and helps the algorithm avoid premature convergence on suboptimal solutions, a common pitfall of simpler search methods. Owing to these robust properties, the FA has been successfully applied to feature selection challenges in healthcare. A pertinent example is the work of Natarajan et al. (2024), who employed an optimized Firefly Algorithm to identify a critical subset of features from a clinical dataset for heart disease classification. This application underscores the algorithm's practical utility and its significant value in enhancing the performance and efficiency of medical diagnostic models by isolating the most prognostically significant variables.

2.6 Hybrid Feature Selection Frameworks: A Synergistic Approach

Hybrid feature selection frameworks synergistically combine filter and wrapper methods to surpass their individual limitations. This strategy first employs efficient filters to reduce dimensionality by removing irrelevant features based on statistical metrics. Subsequently, a more precise wrapper method refines this subset by evaluating it directly with a learning algorithm. This two-stage process balances computational speed with model-specific accuracy. Moreover, hybridization extends to merging metaheuristics, such as integrating global and local search algorithms. Filter criteria can also be embedded into a wrapper's objective function to promote parsimony. Consequently, these frameworks greatly enhance computational feasibility for high-dimensional data without sacrificing performance. They yield more stable and interpretable feature sets by removing noise and ensuring relevance. This makes them indispensable in critical fields like bioinformatics and medical diagnostics. Ultimately, hybrid approaches represent a mature and pragmatic synthesis, offering a powerful solution to the curse of dimensionality.

2.6.1 The Conceptual Basis for Hybridization

The pursuit of optimal feature selection has progressively shifted from relying on single-method approaches to developing sophisticated hybrid frameworks. The fundamental conceptual basis for this evolution is the strategic combination of different feature selection methods to synergistically mitigate their individual weaknesses. Standalone algorithms, whether filter, wrapper, or embedded, each possess inherent limitations; filter methods may ignore critical feature interactions, wrapper methods can be computationally prohibitive, and embedded methods are often tied to the biases of a specific learner. By

integrating complementary techniques, a hybrid model can leverage the distinct advantages of its constituent parts to create a more robust and effective selection process (Hancer, 2022; Zhang & Wang, 2022).

For instance, a common and powerful hybridization strategy involves coupling a fast filter method with a precise wrapper method. The filter can perform an initial, computationally inexpensive screening of the feature space to remove clearly irrelevant and redundant variables. This coarse filtering significantly reduces the dimensionality of the problem presented to the subsequent wrapper method. The wrapper then operates on this refined subset, using a learning algorithm's performance to guide a more focused and intensive search for the optimal feature set. This two-stage process harnesses the speed of the filter to mitigate the computational burden of the wrapper, while the wrapper's model-based evaluation compensates for the filter's inability to detect complex feature dependencies (Talukdar et al., 2024). This rationale directly informs the design of the hybrid RFE-Firefly technique proposed in this study, which aims to fuse the structured, model-guided elimination of RFE with the global, stochastic optimization prowess of the Firefly Algorithm to achieve a superior feature subset.

2.6.2 Review of Existing Hybrid FS Models in Healthcare

The development of hybrid feature selection (FS) techniques has gained significant traction in heart disease prediction, demonstrating a clear path toward enhanced model performance. Researchers have successfully explored various combinatorial strategies, primarily falling into filter-wrapper and wrapperwrapper paradigms. For instance, a common approach involves using a filter method like correlation or mutual information for initial feature screening, which is then refined by a wrapper method such as Recursive Feature Elimination (RFE) or a nature-inspired algorithm. This strategy

effectively leverages the filter's speed to reduce the computational burden on the more precise, but costly, wrapper. In the domain of wrapper-wrapper combinations, studies have begun to merge different search philosophies. Talukdar et al. (2024) developed a hybrid model integrating an improved Firefly Optimization with a sequential search, demonstrating its efficacy on medical data. Similarly, other studies have combined genetic algorithms with local search wrappers to balance global exploration and local exploitation in selecting features for cardiovascular risk assessment. These endeavors collectively affirm the superior performance of hybrid systems over standalone methods in identifying discriminative feature subsets for cardiac diagnosis.

Despite this progress, a critical examination of the literature reveals a specific and unaddressed gap: the limited exploration of a hybrid framework that synergistically combines Recursive Feature Elimination (RFE) and the Firefly Algorithm (FA) specifically for cardiac arrest prediction. While RFE is a wellestablished, model-guided technique, and FA is a potent global optimizer, their integration remains notably underexplored in this high-stakes clinical context. Most existing hybrid models either use different filter methods or pair metaheuristics with simpler sequential searches, leaving the unique potential of the RFE-FA combination largely untapped.

The justification for the proposed RFE-FA Hybrid is rooted in a complementary, two-stage architecture designed to maximize robustness and efficiency. In the first stage, RFE performs a coarse-grained, model-specific filtering. Utilizing a robust estimator like Random Forest, RFE iteratively prunes the least important features, providing a deterministic and strong baseline ranking. This process significantly reduces the initial high-dimensional feature space to a refined shortlist, thereby mitigating the computational complexity that often plagues metaheuristics in large search spaces. In the second stage,

this condensed, high-quality feature subset serves as the initial population or search space for the Firefly Algorithm, which then conducts a fine-grained, global optimization. The FA's stochastic nature, attraction mechanism, and ability to escape local optima allow it to explore complex interactions and non-linear relationships within this condensed subspace, potentially discovering a more parsimonious and powerful feature subset than either method could achieve alone. This sequential synergy leverages RFE's structured elimination to guide the FA's powerful search, presenting a novel and compelling methodology to advance the accuracy of cardiac arrest prediction.

2.7 Review of relate Works

In their 2025 review, Kumar and colleagues systematically organized the field of machine learning for heart disease prediction by categorizing research into five thematic areas. Their analysis revealed that hybrid deep learning models demonstrably outperform traditional algorithms in key diagnostic metrics. The study further identified critical challenges, including data limitations and ethical concerns, while highlighting emerging solutions like federated learning. A inherent limitation of this work is its reliance on synthesizing existing studies rather than presenting new empirical data. Ultimately, this comprehensive review addresses the urgent need for better heart disease diagnostics by providing a consolidated roadmap to guide the development of clinically viable and trustworthy AI systems.

Based on a 2025 study, Prasanna et al. aimed to enhance cardiovascular disease prediction by creating a novel hybrid optimization method to improve upon existing machine learning systems. The methodology featured a two-phase approach that integrated Rough K-means clustering for data segmentation with a Butterfly Optimization Algorithm for analysis, which was then tested on standard

UCI datasets. The results were exceptional, showing that this hybrid model achieved a high accuracy of 97.03% and significantly outperformed other comparative techniques. A noted limitation is the model's reliance on a single benchmark dataset, which may limit its generalizability to different clinical environments and patient groups. Ultimately, this research provides a highly optimized framework that enhances prediction performance, addressing the critical need for earlier and more accurate detection to help reduce cardiovascular mortality.

Based on a 2024 study, Manikandan et al. aimed to improve heart disease prediction by assessing how feature selection affects various machine learning classifiers. The researchers compared three algorithms; Logistic Regression, Decision Tree, and Support Vector Machine both with and without the Boruta feature selection method on a specific heart disease dataset. The results confirmed that the Boruta technique identified six key features, which boosted the performance of all models and allowed Logistic Regression to achieve the highest accuracy of 88.52%. A significant limitation of the study is its use of a small dataset containing only 303 instances, potentially limiting the model's reliability for broader populations. Ultimately, this work provides a practical and cost-effective framework for early detection, demonstrating that strategic feature selection can significantly optimize standard tools to identify highrisk patients.

In their 2024 study, Ahmed and Husien tackle the global challenge of cardiovascular disease by developing a robust predictive model using hybrid machine learning. Their methodology integrates multiple algorithms through ensemble learning, applied to clinical datasets like the Cleveland and IEEE Dataport collections, to improve diagnostic accuracy. The results demonstrate that this hybrid approach

achieves greater precision and reliability than single-model methods by uncovering significant predictive patterns. This strategy directly addresses the limitations of discrete algorithms, such as overfitting and poor generalizability. Consequently, their work provides a more dependable tool for early heart disease detection, which can facilitate timely medical interventions and improve patient outcomes worldwide.

Based on the research by Venkatesh et al., (2024) the study aimed to develop an automatic diagnostic model for the early detection and classification of cardiovascular diseases to address their status as a leading global cause of mortality. The methodology involved creating a novel decision support system that integrates a deep learning classifier with a swarm intelligence optimization technique to effectively handle the challenges of high-dimensional and class-imbalanced clinical datasets. The results demonstrated that the proposed model achieved exceptional performance, obtaining a remarkable accuracy of 99.58% and outperforming existing systems on metrics such as sensitivity and specificity. A primary limitation of the study is the lack of specific details on the exact nature of the deep learning classifier and the optimization algorithm used, which makes independent validation difficult. Ultimately, this research successfully addresses the critical need for early and accurate diagnosis of cardiovascular disease by providing a highly effective assistive system that can potentially reduce mortality rates through improved prediction based on clinical data.

Based on the research by Bouqentar et al. (2024), the study aimed to develop a machine learning system for the early prediction of cardiovascular diseases, which account for a significant portion of global

mortality. The methodology involved a comprehensive comparative analysis of several ML algorithms including Decision Tree, Random Forest, and Support Vector Machine on the Cleveland and Statlog heart datasets, employing rigorous hyperparameter tuning and 10-fold cross-validation for robust evaluation. The results demonstrated the strong potential of these ML techniques, with the performance of each algorithm meticulously assessed using metrics such as accuracy, precision, recall, and F1-score, though the abstract does not specify a single top-performing model. A key limitation is the absence of a reported highest accuracy figure or a novel algorithmic contribution, positioning the work as a thorough comparative study rather than a proposal of a new state-of-the-art model. Ultimately, this research successfully addresses the critical need for early and accurate heart disease diagnosis by providing a validated, data-driven tool that can assist healthcare professionals in reducing misdiagnosis and improving patient outcomes through timely intervention.

Based on the research by El-Sofany (2024), this study aimed to develop an accurate machine learning system for early heart disease prediction by systematically evaluating various feature selection methods and classification algorithms. The methodology employed three distinct feature selection techniques (chi-square, ANOVA, and mutual information) to create different feature subsets, then comprehensively evaluated ten machine learning classifiers while addressing data imbalance through SMOTE implementation across both private and public datasets. The results demonstrated exceptional performance, with the XGBoost classifier achieving 97.57% accuracy using the SF-2 feature subset, along with outstanding metrics including 96.61% sensitivity, 90.48% specificity, and 98% AUC. A notable limitation is the use of a private dataset alongside public data, which may affect reproducibility and direct comparison with other studies. Ultimately, this research successfully addresses the critical

need for accessible early heart disease detection by not only developing a highly accurate predictive model but also implementing it as a practical mobile application and enhancing interpretability through SHAP methodologies, thereby providing a low-cost, rapid diagnostic tool for both healthcare providers and patients.

Based on a comprehensive 2024 review, Zhou et al. systematically evaluated the use of deep learning in predicting heart disease. Their analysis of 64 studies from 2018 to 2023 found that these methods show high accuracy and real-time performance. However, the study also identified significant field-wide challenges, including a scarcity of large datasets and room for model improvement. A notable limitation of the review itself is its restricted time-frame, which may omit earlier or very recent research. Ultimately, this work consolidates fragmented knowledge to provide a clear foundation and guide for future researchers aiming to develop better predictive models.

Based on a 2024 study, Natarajan et al. aimed to enhance heart disease prediction by identifying significant features and applying ensemble techniques. The researchers utilized an optimized Firefly Algorithm for feature selection and developed models using stacking and voting ensembles on a specific medical dataset. Their results demonstrated that the stacking technique with optimized features achieved a top prediction accuracy of 86.79%, outperforming other compared methods. A key limitation of this work is that its achieved accuracy, while effective, is moderate compared to some higher benchmarks set by contemporary deep learning models. Ultimately, the research provides a valuable and efficient

framework that is particularly suited for healthcare systems with average income populations, where the prevalence of heart disease is most pronounced.

Based on a 2022 study, El-Shafiey et al. aimed to create a highly accurate predictive model for heart disease through an optimized feature selection method. The researchers developed a hybrid model that combined a modified Genetic Algorithm with Particle Swarm Optimization to select the best features, which were then classified using a Random Forest algorithm. Their results showed exceptional performance, achieving accuracies of 95.6% and 91.4% on two standard medical datasets, outperforming other advanced methods. A noted limitation of this approach is its potential computational complexity due to the integration of multiple optimization techniques. Ultimately, the research successfully provides a sophisticated tool to support healthcare professionals in achieving early and precise diagnosis, thereby addressing a critical need in medical intervention.

Based on the research by Al Bataineh and Manacek (2022), the study aimed to develop an advanced machine learning system for predicting heart disease. The methodology involved creating a hybrid algorithm that combines a Multilayer Perceptron (MLP) with a Particle Swarm Optimization (PSO) technique, which was then compared against ten other machine learning algorithms using the Cleveland Heart Disease dataset. The results demonstrated that the proposed MLP-PSO hybrid model outperformed all other algorithms, achieving a superior prediction accuracy of 84.61%. A potential limitation of the study is its reliance on a single, publicly available dataset, which may affect the generalizability of the findings across diverse populations. Ultimately, this research successfully

addresses the challenge of complex heart disease diagnosis by providing a more accurate predictive tool, thereby enabling medical practitioners to facilitate earlier and more effective clinical interventions.

Based on the research by Jindal et al., (2020) developed a predictive system for identifying individuals at risk of heart disease. The methodology involved applying and comparing machine learning algorithms, specifically Logistic Regression and K-Nearest Neighbors (KNN). Their findings demonstrated that these models achieved a satisfactory and improved accuracy over previous benchmarks like Naïve Bayes. A notable limitation of the study is the absence of specific quantitative results in the abstract, which hinders precise evaluation. Ultimately, the research contributes a valuable tool for enabling earlier, data-driven diagnosis, thereby aiming to enhance patient care and reduce associated medical costs.

2.8 Study Gap

Despite the growing body of research on machine learning applications for cardiac arrest prediction, several critical limitations remain insufficiently addressed in existing studies.

Firstly, many prior works inadequately handle the issue of class imbalance, which is inherent in clinical datasets where instances of cardiac arrest are significantly fewer than non-events. This imbalance often results in biased predictive models that favor the majority class, thereby reducing sensitivity and limiting the model's effectiveness in identifying high-risk patients.

Secondly, existing studies frequently employ single-method feature selection techniques, such as Boruta, Recursive Feature Elimination (RFE), or other statistical approaches, in isolation. While these methods are effective individually, they may fail to capture complex interactions among features. There is limited

exploration of hybrid feature selection approaches that combine multiple techniques to improve feature relevance and model robustness, particularly in high-dimensional healthcare datasets.

Furthermore, the selection of machine learning models in many studies does not adequately reflect the heterogeneous nature of clinical data, which often includes both numerical and categorical variables. A significant number of studies rely on single classifiers, which may lack the capacity to generalize effectively across diverse data structures. Although ensemble methods have been explored, there remains a gap in integrating hybrid feature selection techniques with advanced ensemble strategies, such as stacked generalization, within a unified predictive framework.

Therefore, a significant research gap exists in the development of a framework that simultaneously addresses class imbalance, leverages hybrid feature selection methods, and employs a robust stacked ensemble model tailored to structured clinical datasets. Addressing this gap has the potential to significantly enhance the accuracy, reliability, and clinical applicability of cardiac arrest prediction systems.

CHAPTER THREE

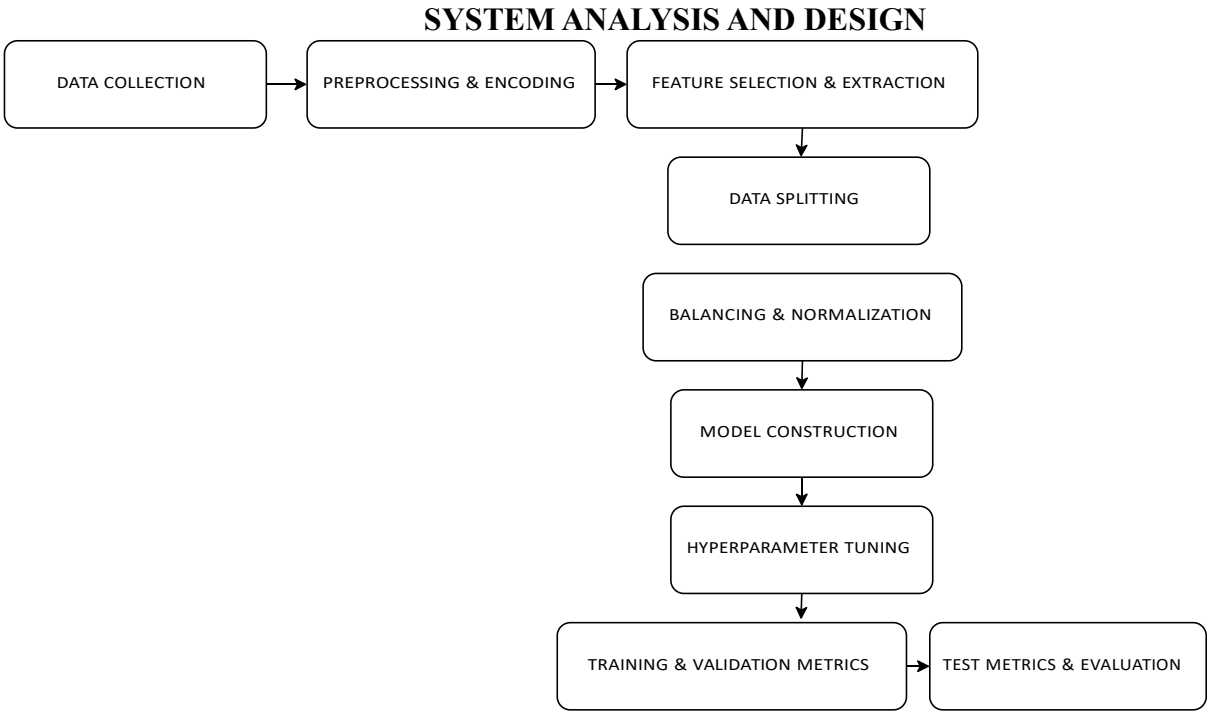

Figure 3.1: System Methodology

3.1 Study Approach

The successful execution of this research hinges on a structured, systematic, and computationally rigorous methodology designed to empirically validate the proposed intelligent healthcare framework. The study adopts a hierarchical machine learning design approach, grounded in the principles of design science and experimental computer science. The overarching methodology is segmented into four distinct, yet interconnected, phases: Data Acquisition and Pre-processing, Feature Selection using the Hybrid RFE-Firefly Technique, Model Development with a Stacked Ensemble Classifier, and Performance Evaluation and Validation, as shown in Figure 3.2. This phased approach ensures a logical

flow from data preparation to the final assessment of the framework's predictive efficacy, providing a clear and replicable pathway for achieving the research objectives.

The first phase, Data Acquisition and Pre-processing, is foundational to the integrity of the entire study. A well-curated, publicly available clinical dataset, available on the Kaggle repository on https://www.kaggle.com/datasets/krishujeniya/heart-diseae. The selection criteria was prioritize datasets with comprehensive clinical attributes relevant to cardiovascular health and a definitive label for cardiac arrest or a severe cardiac event. Following its collection, the dataset entered a comprehensive cleaning and preparation stage. This essential phase included addressing gaps in the data by employing imputation strategies; specifically, the average value was used to fill missing numbers, while the most frequent value was used for missing categories. addressing class imbalance using methods like SMOTE (Synthetic Minority Over-sampling Technique) to prevent model bias, and normalizing or standardizing the numerical features to ensure that no single variable disproportionately influences the model due to its scale. This meticulous preparation is crucial to construct a high-quality, reliable dataset for subsequent analysis.

The second phase constitutes the core innovation of this research: the design and implementation of the Hybrid Recursive Feature Elimination-Firefly (RFE-Firefly) algorithm. This objective was achieved through a two-stage process. Initially, the standard Recursive Feature Elimination (RFE) was applied, using a robust classifier like Random Forest as its estimator. RFE performed a coarse-grained filtering, iteratively pruning the least important features to produce a ranked shortlist of the most promising candidates, thereby significantly reducing the initial high-dimensional space. Subsequently, this refined feature subset was serve as the initial population for the Firefly Algorithm (FA). The FA was configured

with an appropriate objective function, likely a combination of model accuracy and feature subset size, to guide its search. Each firefly's position represented a potential feature subset, and its brightness corresponded to the performance of a preliminary classifier (e.g., Logistic Regression) trained on that subset. The FA's attraction and movement mechanisms then performed a fine-grained, global search within this condensed space to discover the ultimate, parsimonious, and highly predictive feature set. The implementation was carried out in a Python environment, leveraging libraries such as Scikit-learn for RFE and a custom-coded or adapted FA module.

The third phase focuses on the construction of the robust stacked ensemble classification model. A diverse set of five base learners (Level-0 models) was selected for their complementary strengths: Decision Tree (DT), Support Vector Machine (SVM), Logistic Regression (LR), Random Forest (RF), and XGBoost. This diversity is key to the ensemble's success, as it ensures that the model can capture a wide range of patterns in the data from simple linear relationships (LR) to complex, non-linear interactions (XGBoost, RF). The dataset, now optimized by the Hybrid RFE-Firefly technique, was split into training and testing sets. The base models were trained on the training set, and their predictions on a hold-out validation set were used as input features to train the meta-learner (Level-1 model). A Random Forest classifier was employed as the meta-learner due to its proven robustness and ability to effectively learn how to best combine the predictions of the base models to produce a final, superior prediction.

The fourth phase entails a comprehensive evaluation of the predictive performance of the complete intelligent framework. A rigorous hold-out validation method was employed to ensure the results are

statistically sound and generalizable. The proposed framework's performance was benchmarked against several baselines: (1) individual classifiers (DT, SVM, LR, RF, XGBoost) trained on the full feature set, (2) the same individual classifiers trained on the feature set selected by the Hybrid RFE-Firefly technique, adapted model from Manikandan et al. (2024), tried to enhanced heart disease prediction accuracy by testing non-ensemble classifiers with Boruta feature selection on the Cleveland Clinic dataset. The method involved applying Logistic Regression, Decision Tree, and SVM to a set of six features selected by the Boruta algorithm. The results showed that Logistic Regression achieved the highest accuracy of 88.52%, though ensemble models like Random Forest performed worse with the selected features. A key limitation was the small dataset size, which risks overfitting and limits the generalizability and effective use of complex models. To address this, the proposed model in Figure 3.2 employs a Hybrid RFE-Firefly technique for more robust feature selection and a stacked ensemble classifier for improved model generalization. This integrated approach is designed to be more resilient to data constraints and aims for superior predictive performance.

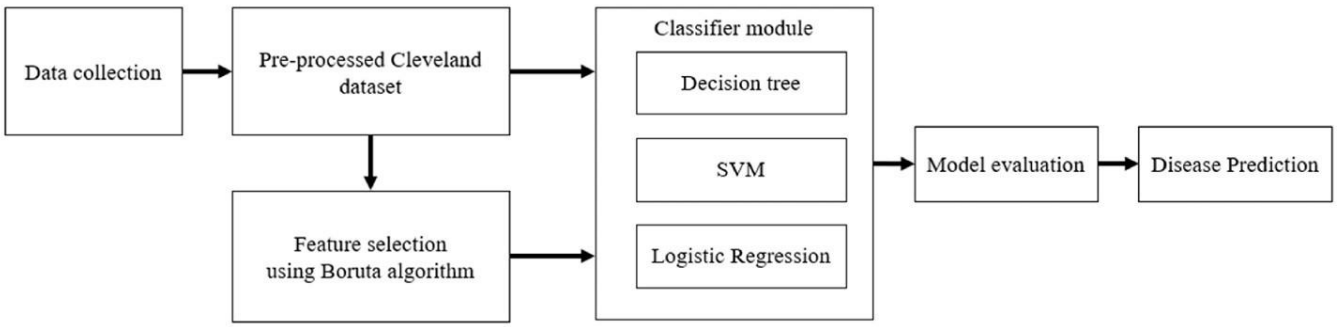

Figure 3.2: Heart Disease Prediction Framework Adapted from Manikandan et al. (2024)

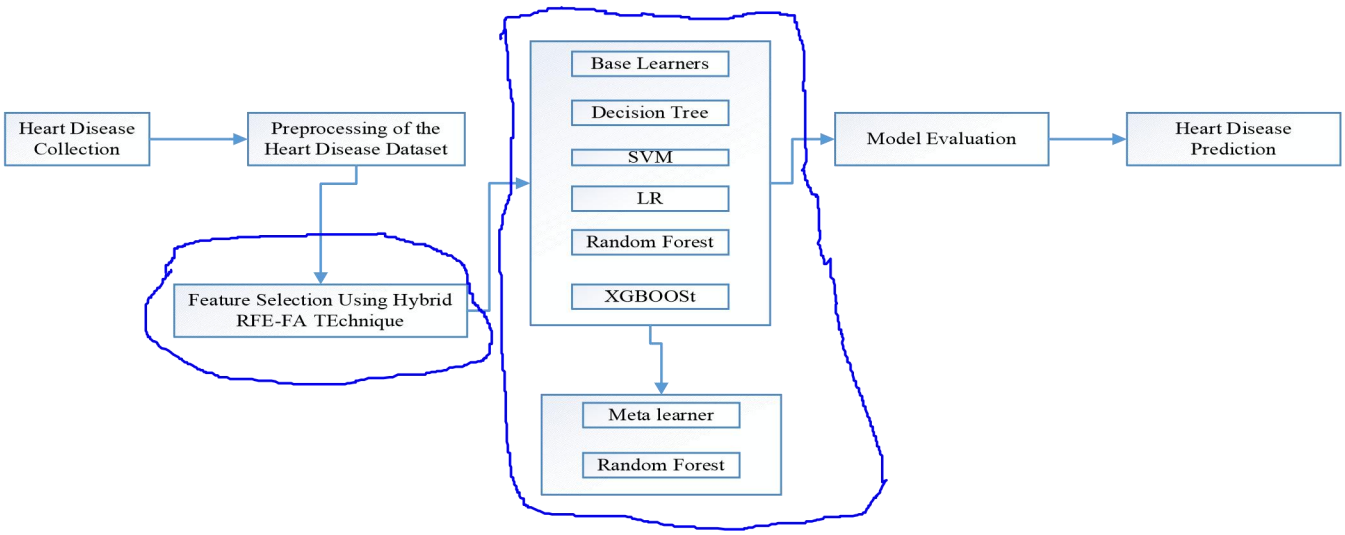

Figure 3.3: Proposed Framework for the Heart Disease prediction using hybrid RFE-FA techniques

3.2 Procedural Techniques for the Developed System

The proposed model shown in Figure 3.2 follows the machine learning design approach methodology. Which was a combined Knowledge Discovery in Databases (KDD) and learning technique using a hybrid feature selection technique of RFE-FA optimal feature subset was utilized for training the stack ensemble model. The data mining techniques adopted are shown in this order, from data acquisition, through feature selection, classification, and then evaluation of the different models, see Figure 3.2.

3.2.1 Dataset acquisition

A diabetes dataset was used to formulate a knowledge base for the system so as to build an efficient system. The dataset was obtained from the Kaggle repository. The raw cardiac arrest dataset comprises fourteen (14) attributes, with one designated as the target variable or class label, and contains a total of

three hundred and three (303) instances. Attribute descriptions of the various datasets are as shown in Table 3.1. while the preprocessed dataset is as presented in Figure 3.3

The dataset contains no missing values, as confirmed by the empty missing values analysis table. However, one duplicated row was identified that requires removal to ensure data quality. The heart disease dataset contains 303 patient records with 14 different medical features. These features include age, cholesterol levels, and maximum heart rate. The data shows that 165 patients were diagnosed with heart disease. Meanwhile, 138 patients did not have heart disease. Figure 3.4 confirms that the dataset contains no missing records; however, one duplicate entry was identified. The figure also illustrates the distribution of the data prior to the cleaning process.

Table 3.1 Attribute Description and Definition

S/N	Feature Name	Feature Description	Data Type
1	age	Age in years	Integer
2	sex	Sex (1 = male; 0 = female)	Integer (Nominal)
3	cp	Chest pain type (0: Typical Angina, 1: Atypical Angina, 2: Non-anginal Pain, 3: Asymptomatic)	Integer (Ordinal)
4	trestbps	Resting blood pressure (in mm Hg on admission to the hospital)	Integer
5	chol	Serum cholesterol in mg/dl	Integer
6	fbs	Fasting blood sugar > 120 mg/dl (1 = true; 0 = false)	Integer (Nominal)

7	restecg	Resting electrocardiographic results (0: Normal, 1: ST-T wave abnormality, 2: Probable or definite left ventricular hypertrophy)	Integer (Ordinal)
8	thalach	Maximum heart rate achieved	Integer
9	exang	Exercise induced angina (1 = yes; 0 = no)	Integer (Nominal)
10	oldpeak	ST depression induced by exercise relative to rest	Float
11	slope	The slope of the peak exercise ST segment (0: Upsloping, 1: Flat, 2: Downsloping)	Integer (Ordinal)
12	ca	Number of major vessels (0-3) colored by fluoroscopy	Integer (Ordinal)
13	thal	Thalassemia (1 = normal; 2 = fixed defect; 3 = reversible defect)	Integer (Ordinal)
14	target	Diagnosis of heart disease (1 = presence; 0 =	Integer (Nominal)
S/N	**Feature Name**	**Feature Description**	**Data Type**
		absence)	

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
44	1	1	120	263	0	1	173	0	0	2	0	3	1
52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
57	1	2	150	168	0	1	174	0	1.6	2	0	2	1
54	1	0	140	239	0	1	160	0	1.2	2	0	2	1
48	0	2	130	275	0	1	139	0	0.2	2	0	2	1
49	1	1	130	266	0	1	171	0	0.6	2	0	2	1
64	1	3	110	211	0	0	144	1	1.8	1	0	2	1
58	0	3	150	283	1	0	162	0	1	2	0	2	1
50	0	2	120	219	0	1	158	0	1.6	1	0	2	1
58	0	2	120	340	0	1	172	0	0	2	0	2	1
66	0	3	150	226	0	1	114	0	2.6	0	0	2	1
43	1	0	150	247	0	1	171	0	1.5	2	0	2	1
69	0	3	140	239	0	1	151	0	1.8	2	2	2	1
59	1	0	135	234	0	1	161	0	0.5	1	0	3	1
44	1	2	130	233	0	1	179	1	0.4	2	0	2	1
42	1	0	140	226	0	1	178	0	0	2	0	2	1
61	1	2	150	243	1	1	137	1	1	1	0	2	1

heart-disease

Figure 3.4: Unprocessed heart disease dataset

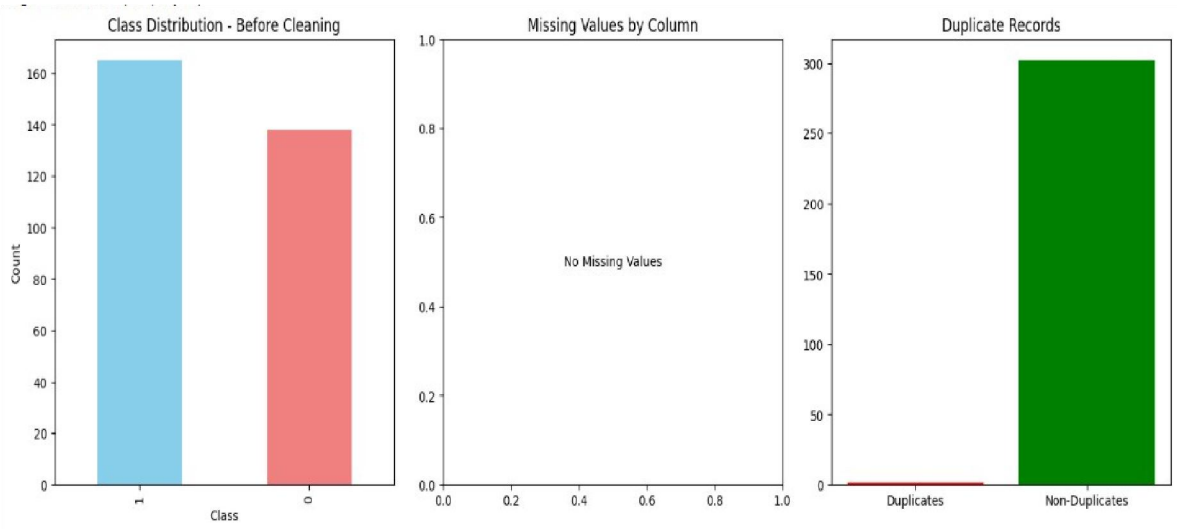

Figure 3.5: Statistical Distribution of the unpreprocessed dataset Before balancing, the data shows an unequal distribution of patients with and without heart disease. This is presented in Figure 3.5

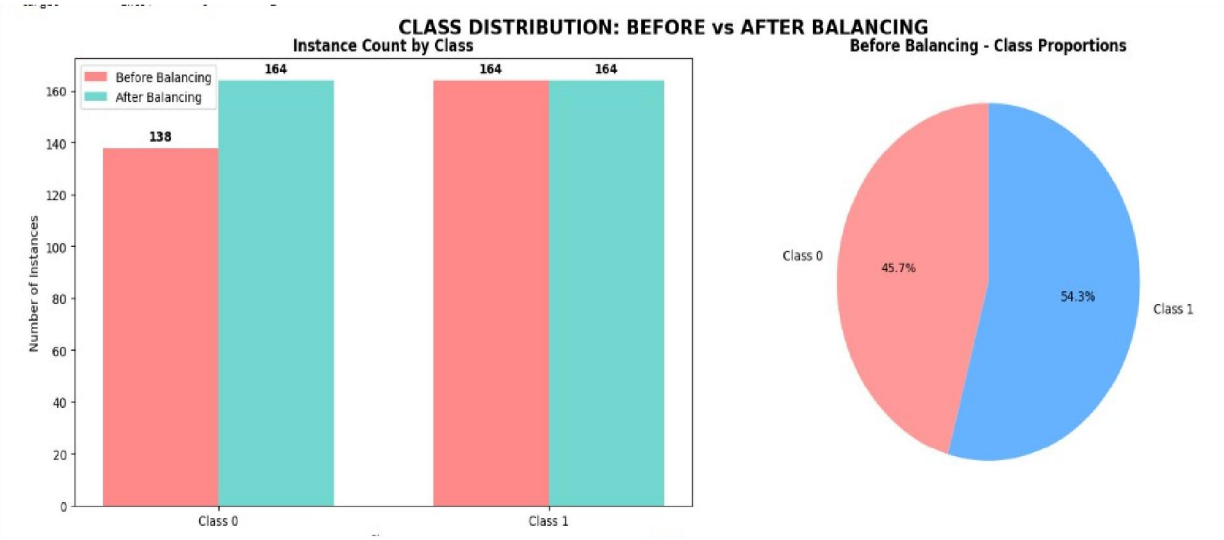

Figure 3.6: Class distribution before and after data balancing

3.2.2 Dataset Pre-processing and Normalization

Removing errors and outliers that may be present in the data are components of the pre-processing task that should be performed to make the information appropriate for modeling. In other to avoid inconsistency, imbalance, and missing response normally prone to cardiac arrest dataset StandardScaler technique was employed for normalizing the cardiac arrest dataset.

The dataset has been successfully cleaned and balanced through a two-step process. First, the single duplicate row was removed to ensure data integrity. Then, SMOTE was applied to address the class imbalance in the target variable, which equalized the count of both classes from 164 and 138 to 164 samples each. This balanced dataset is now prepared for more reliable model training and analysis. A pictorial illustration for the balanced dataset is presented in Figure 3.6

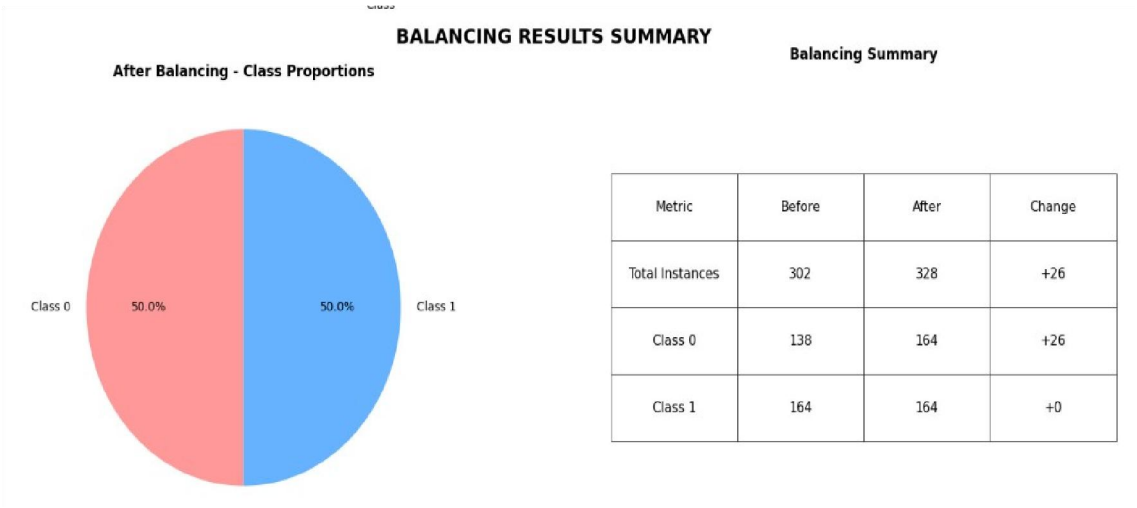

Figure 3.7: Summary of the class distribution before and after data balancing

Figure 3.7 presents a comparative view of the dataset's descriptive statistics before and after normalization. The "Before" section reflects the original feature scales, such as age in years and specific cholesterol readings. The "After" section displays the standardized data, where each feature has been transformed to a mean of zero and a standard deviation. This normalization process ensures all features are on a common scale, enhancing the performance and convergence of many machine learning algorithms.

```
==================================================
Data Statistics BEFORE Normalization:
==========================================
              age         sex          cp    trestbps        chol         fbs  \
count  328.000000  328.000000  328.000000  328.000000  328.000000  328.000000
mean    54.576220    0.682927    0.896341  132.057927  247.018293    0.143293
std      8.941552    0.466047    1.020397   17.595515   51.350087    0.350906
min     29.000000    0.000000    0.000000   94.000000  126.000000    0.000000
25%     48.000000    0.000000    0.000000  120.000000  211.750000    0.000000
50%     56.000000    1.000000    0.000000  130.000000  243.000000    0.000000
75%     61.000000    1.000000    2.000000  140.000000  275.000000    0.000000
max     77.000000    1.000000    3.000000  200.000000  564.000000    1.000000

           restecg     thalach       exang     oldpeak       slope          ca  \
count  328.000000  328.000000  328.000000  328.000000  328.000000  328.000000
mean     0.490854  148.658537    0.317073    1.061000    1.356707    0.731707
std      0.524543   23.149940    0.466047    1.155205    0.618930    0.986747
min      0.000000   71.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000  132.000000    0.000000    0.000000    1.000000    0.000000
50%      0.000000  152.000000    0.000000    0.800000    1.000000    0.000000
75%      1.000000  165.000000    1.000000    1.635643    2.000000    1.000000
max      2.000000  202.000000    1.000000    6.200000    2.000000    4.000000

             thal
count  328.000000
mean     2.295732
std      0.616575
min      0.000000
25%      2.000000
50%      2.000000
```

Figure 3.7: Cardiac Arrest dataset before Normalization

```
Data Statistics AFTER Normalization:
======================================
                 age           sex            cp      trestbps          chol  \
count   3.280000e+02  3.280000e+02  3.280000e+02  3.280000e+02  3.280000e+02
mean   -1.949660e-16 -1.083144e-16 -1.299773e-16 -2.491232e-16 -2.382918e-16
std     1.001528e+00  1.001528e+00  1.001528e+00  1.001528e+00  1.001528e+00
min    -2.864748e+00 -1.467599e+00 -8.797665e-01 -2.166238e+00 -2.360331e+00
25%    -7.365910e-01 -1.467599e+00 -8.797665e-01 -6.863312e-01 -6.878699e-01
50%     1.594752e-01  6.813851e-01 -8.797665e-01 -1.171362e-01 -7.837245e-02
75%     7.195166e-01  6.813851e-01  1.083250e+00  4.520588e-01  5.457529e-01
max     2.511649e+00  6.813851e-01  2.064758e+00  3.867229e+00  6.182385e+00

                 fbs       restecg       thalach       exang       oldpeak  \
count   3.280000e+02  3.280000e+02  3.280000e+02  328.000000  3.280000e+02
mean   -8.123583e-18 -1.083144e-17  8.665155e-17    0.000000 -8.665155e-17
std     1.001528e+00  1.001528e+00  1.001528e+00    1.001528  1.001528e+00
min    -4.089741e-01 -9.372035e-01 -3.359714e+00   -0.681385 -9.198546e-01
25%    -4.089741e-01 -9.372035e-01 -7.206925e-01   -0.681385 -9.198546e-01
50%    -4.089741e-01 -9.372035e-01  1.445606e-01   -0.681385 -2.262789e-01
75%    -4.089741e-01  9.721304e-01  7.069751e-01    1.467599  4.981983e-01
max     2.445143e+00  2.881464e+00  2.307693e+00    1.467599  4.455357e+00

               slope          ca          thal
count   3.280000e+02  328.000000  3.280000e+02
mean    6.498866e-17    0.000000 -3.682691e-16
std     1.001528e+00    1.001528  1.001528e+00
min    -2.195370e+00   -0.742668 -3.729048e+00
25%    -5.772097e-01   -0.742668 -4.803688e-01
50%    -5.772097e-01   -0.742668 -4.803688e-01
75%     1.040951e+00    0.272311  1.143971e+00
max     1.040951e+00    3.317248  1.143971e+00
```

Figure 3.9: Cardiac Arrest dataset After Normalization

Figure 3.9 shows the data distribution of the dataset features before and after a normalization process was applied. Before normalization, the features are on their original, different scales, while after normalization, they have been standardized to a common scale with a mean of zero.

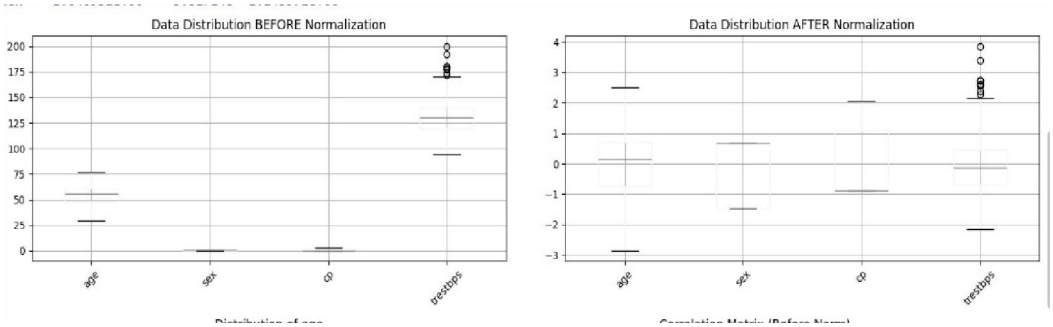

Figure 3.11: Data Distribution Of The Dataset Features Before And After Normalization

3.2.3 Feature Selection

Feature selection is often referred to as the selection of variable subsets, variable selection, reduction of features, or selection of attributes. Feature Selection (FS) is a tool that can be used in training materials to delete features, so that features that are statistically uncorrelated with class labels are removed and also reduce the set of features to be used in classification, hence better performance of the classifier. In this study, RFE, FA, and hybrid RFE and FA were utilized to fetch the relevant features subset only pertinent enough for building a stack ensemble classification model for the cardiac arrest prediction process. Cardiac arrest risk factors were identified for appropriate awareness as to reduce the high mortality of heart disease patients.

Table 3.2 provides a performance comparison of the feature selection method utilized for this study. The best performance was achieved by using either the RFE Only or RFE-FA Union method, both of which

selected 7 features and yielded the highest accuracy and F1-score. In contrast, using only one feature, as with the FA Only or RFE-FA Intersection method, resulted in significantly lower performance, particularly in the AUC-ROC score. This demonstrates that for this dataset, a larger, carefully selected subset of features is more effective for model prediction than a single feature.

Table 3.2: Performance Comparison of Feature Selection Methods

Method	**Number of Features**	**Accuracy**	**F1-Score**	**AUC-ROC**
All Features	13	0.8030	0.8060	0.9036
RFE Only	11	0.8485	0.8529	0.9077
FA Only	1	0.8182	0.8182	0.7787
RFE-FA Union	7	0.8485	0.8529	0.9077
RFE-FA Intersection	1	0.8182	0.8182	0.7787

The hybrid RFE-FA selected seven (7) pertinent feature subsets as presented in Tables 3.2 and 3.3

Table 3.3: Optimal feature subset from the Hybrid RFE-FA

S/N	Feature	Description
1	sex	Sex (1 = male; 0 = female)
2	cp	Chest pain type
3	trestbps	Resting blood pressure (in mm Hg)
4	oldpeak	ST depression induced by exercise relative to rest
5	slope	The slope of the peak exercise ST segment
6	ca	Number of major vessels (0-3) colored by fluoroscopy
7	thal	Thalassemia (a blood disorder) type

3.2.4 Training and Classification

This study aims to predict cardiac arrest in patients using a hybrid RFE-FA feature selection technique and stack-based ensemble approach. The dataset used was divided into a training and testing set at a

proportion of 75% to 25% respectively. The training set is passed to the stack ensemble Classifier to build an efficient cardiac arrest prediction model.

3.3 Architecture diagram for the Developed System

The frame work for the developed system are described in Figure 3.2. The model gives a detailed description of how the study's aim and objective were actualized. Cascading processes of the various blocks are depicted. The first stage shows the input block called the dataset block containing 9 attribute. The output from the input block was fed to the preprocessing stage were StandardScaler technique was employed, the 9 attribute was reduced by sending it to the third block which is the feature selection stage. and optimal feature subset obtained from the F-test filter algorithm were sent to the fourth stage which is the classification stage were RF classification model was built and used to classifying the diabetic dataset. to achieve this diabetes dataset was partitioned to 75% for training and 25% for testing.

Based on the user input, the overall output stage was able to detect if a patient was diabetic or not diabetic, and user friendliness and easy operation were achieved. Based on the results obtained, recommendations were made to patients for treatments.

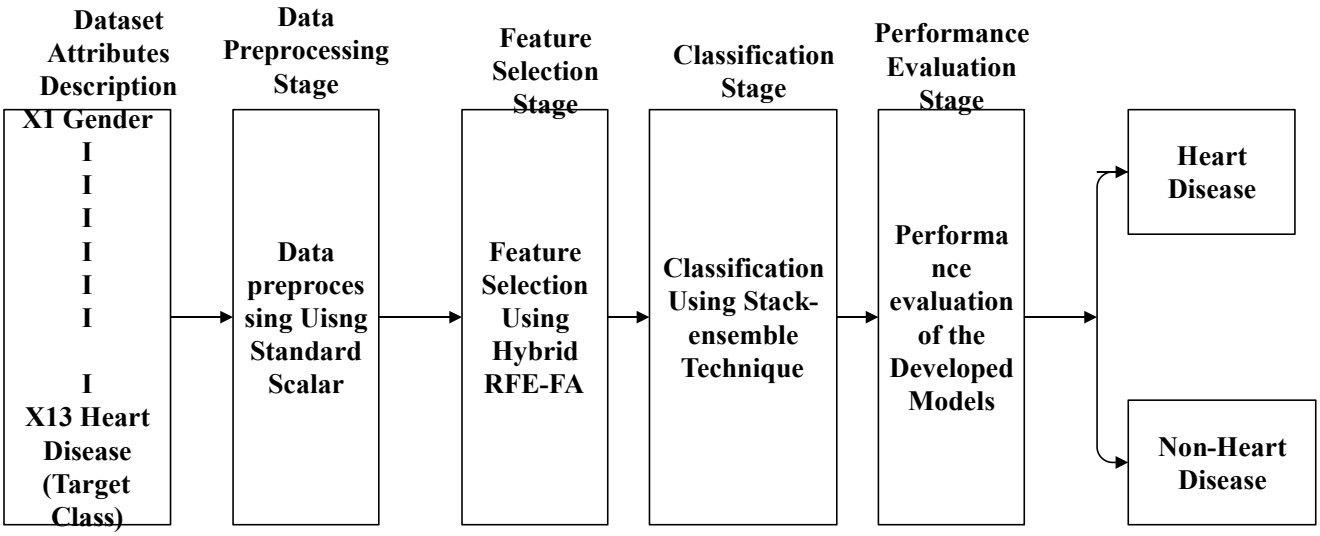

Figure 3.10: Architectural diagram for the Developed Hybrid RFE-FA stack-ensemble Model

3.4 Cardiac Arrest Performance Evaluation metrics

To assess the effectiveness of the developed cardiac arrest prediction models, key statistical performance metrics were used, including classification accuracy, sensitivity (recall), specificity, and precision. These metrics are derived from the confusion matrix, which compares predicted outcomes (heart disease or non-heart disease) against actual conditions, measuring true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) (Abdulsalam et al., 2024).

Table 3.3 shows the Classification accuracy (CA) evaluates overall correctness by calculating the ratio of correct predictions (TP + TN) to total predictions (TP + TN + FP + FN). Sensitivity (recall) measures the model's ability to correctly identify heart disease cases (TP / (TP + FN)), while specificity assesses its accuracy in detecting non-heart disease cases (TN / (TN + FP)). Precision indicates the reliability of

positive predictions (TP / (TP + FP)), distinguishing true diabetic cases from false alarms (Edafeajiroke *et al*., 2024).

Among these, false negatives (FN) where diabetic patients are incorrectly classified as healthy are the most critical to minimize, as they pose severe health risks. Conversely, false positives (FP) are less harmful but may lead to unnecessary interventions. Together, these metrics ensure the model's robustness in clinical decision-making, balancing diagnostic reliability with patient safety.

Table 3.4. Performance Evaluation Metrics

Metric	**Formula**	**Range**
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	0 to 1 (0–100%)
Precision	$\frac{TP}{TP+FP}$	0 to 1 (0–100%)
Recall (Sensitivity)	$\frac{TP}{TP+FN}$	0 to 1 (0–100%)
Specificity	$\frac{TP}{TP+FN}$	0 to 1 (0–100%)

3.5 Recommender System

An application interface will be developed with the Python programming language in order to create a user-friendly interface that will cater to an unfamiliar dataset that will be analyzed on an individual basis by the system, which will scale on the best technique for an optimal prediction model for the heart disease dataset.

3.6 Choice of Programming Tools

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation via the off-side rule. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library. Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000. Python 3.0, released in 2008, was a major revision not completely backward-compatible with earlier versions. Python 2.7.18, released in 2020, was the last release of Python 2. Python consistently ranks as one of the most popular programming languages. Python users are colloquially called pythonistas (Łukasz, 2023).

It is an easy-to-use environment, it combines computation, visualization, and programming where problems and solutions are presented in familiar mathematical notation. Typical utilizations include: creation of algorithm, acquiring data and modelling, prototyping, and simulation.

CHAPTER FOUR

IMPLEMENTATION AND DOCUMENTATION

4.1 Overview of the Result Analysis of the Developed System

This section presents the results analysis of the data science approach used in achieving the stated objectives. The experiment was designed to improve the performance of the stack ensemble model using the hybrid feature selection technique with RFE-FA. The results of each stage were designed to obtain the best model. The developed model was recommended for use in diagnostic centers and further improvement by future researchers.

The implemented system elaborated a stepwise approach of data mining technique, as to achieve a high prediction rate for diabetic process, these were systematically done as follow: the one (1) month old heart disease dataset collected from kaggle dataset repository, secondly the dataset was preprocessed using standardsclaer and then feature selections were performed with hybrid RFE-FA algorithm which helps to remove outliers and inconsistent data, at the fourth stage stack ensemble model were employed for classification of the cardiac arrest dataset and then the performance of the models were evaluated using different performance matrixes.

Accuracy of different model were compared, the dataset was successfully partitioned into two fold the training and testing set at a percentage ratio of 75% to 25% respectively. The results were evaluated based on machine learning statistical metrics like the classification accuracy, true positive rate, false negative rate, error rate, specificity, sensitivity and training and testing time. The experimental setup was successfully carried out and developed with the Python programming in a google coolab with interwoven connected component in the platform so as to create a friendly user experience. The developed systems made use of various component environment in the platform for successful

presentation of the result from the various data mining stages namely data preprocessing, feature selection, classification and performance evaluation.

4.2 The Python Programming Command Window

The python command window helps to display result of the data mining task in a console screen for readability and easy expression of output, see Figure 4.1.

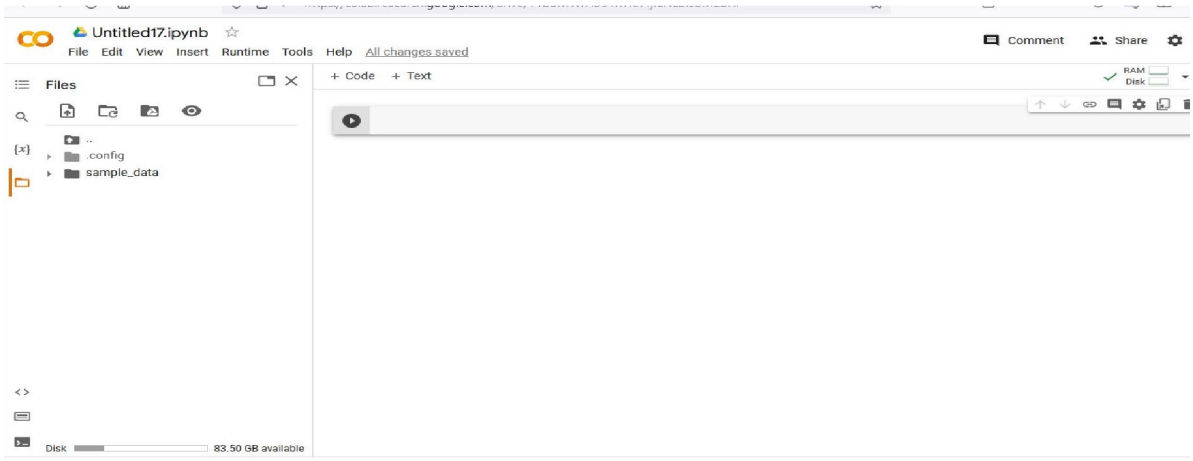

Figure 4.1: The Python Programming Command Window on Google Coolabs

4.3 Interactive Developmental Stage

The optimized data mining model results was presented based on the following outcomes

i. Dataset Acquisition

i. Dataset Pre-processing

i. Feature Selection

i. Training of the optimal model obtained in (iii)

i. Classification of the developed models

i. Performance Evaluation of the Developed System

Figure 4.2 shows an overview of the heart disease acquisition for the developed system. The dataset was acquired through the Kaggle dataset repository. At run time, all modules vital to actualizing the system's aim were automated into the working process of the developed platform. The interactive graphic user interface automates the loading of data via the load button which loaded the dataset into the platform see Figure 4.3, the next phase carried was fetching of the best feature with RFE-FA which resulted to successful selection of optimal feature for building the stack ensemble classifier models.

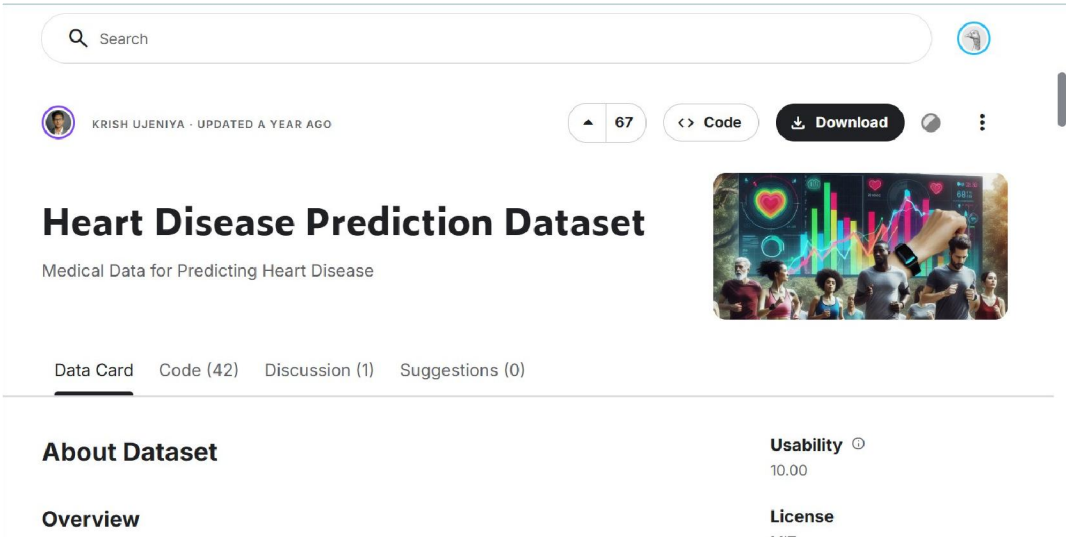

Figure 4.2: Heart Disease Dataset Acquisition Interface for the Developed System

4.4 Results Analysis for the Developed System

This section presents the results obtained from each section accordingly.

4.4.1 Dataset Preprocessing Stage

The Preprocessing selection helps to present a well-formatted data into the system. The dataset were normalized by employing standard scaler technique hence removing inconsistent entries. The filtered

and normalized dataset was then fed into the next stage. Figure 4.3 Provides an outlook of the best attributes that were fetched.

4.4.2 Heart Disease Feature Subset Obtained from Using RFE-FA Technique This section, therefore, describes the optimal feature subset obtained after the RFE-FA filter algorithm was used to select the best features. Figure 4.3 therefore showed that only seven (7) attributes were selected after optimization, in order to detect heart disease, the order of importance of these risk factors was described in descending order (order of importance) as obtained from the feature selection algorithm.

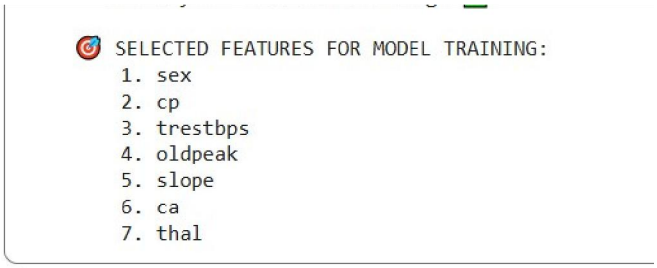

Figure 4.3: Features Selected from the RFE-FA Algorithms

4.5 Training and Classification of the Developed Models

The model was trained and classified by splitting the data into training and testing set of 75% and 25% respectively, and then each model were evaluated. This section gives the exact values for the different confusion matrixes. The result obtained for developed models were noted for future researcher.

This section provides details comprehensive process of building, training, and evaluating a suite of machine learning models for a heart disease classification task, culminating in the performance superiority of a stacking ensemble method.

The process began with a preprocessed dataset of 210 patient records, which was subsequently split into training and testing sets. A significant class imbalance was noted in the training data, with 123 instances

of Class 0 (no heart disease) and only 34 of Class 1 (heart disease), a factor that inherently challenges model performance on the minority class. Four distinct base models Decision Tree, Support Vector Machine (SVM), Logistic Regression, and Random Forest were then initialized and trained. Their individual training times varied considerably, with the Random Forest model taking a notably longer 0.377 seconds, while the others were trained almost instantaneously.

These base models were then integrated into a sophisticated stacking ensemble. This ensemble was configured to use the predictions from all four base models as input features for a final meta-learner, which was itself an optimized Random Forest model. The training of this complex ensemble architecture was the most computationally intensive step, requiring over 7 seconds to complete. Upon evaluation on the test set, the four base models demonstrated identical accuracy of approximately 88.68%. However, a more nuanced analysis of the F1-score and Area Under the Curve (AUC) metrics revealed critical differences in their ability to handle the class imbalance and provide well-calibrated probability estimates.

It was the stacking ensemble that ultimately proved most effective. It achieved the highest accuracy of 90.57% and, more importantly, a significantly superior F1-score of 0.7619. This result indicates that the ensemble's method of synthesizing the diverse strengths of the base learners created a more robust and generalized model, yielding the best overall predictive performance for identifying heart disease in this dataset.

4.5.1 Comparative Performance Analysis of the Developed RFE-FA Stacked Ensemble

Heart Disease Prediction Models

Based on the comprehensive performance metrics utilized for this study, a detailed comparison of the five models Decision Tree, Support Vector Machine (SVM), Logistic Regression, Random Forest, and a Stacking Ensemble reveals distinct strengths and weaknesses across different evaluation criteria. While accuracy remains consistent for several models, other metrics such as precision, sensitivity, specificity, F1-score, and the area under the receiver operating characteristic curve (AUC-ROC) offer deeper insights into model efficacy, particularly in scenarios involving class imbalance or differing costs of misclassification.

In terms of accuracy, which measures the overall correctness of the model, the Decision Tree, SVM, and Logistic Regression models all achieve an identical score of 0.8868, indicating that they correctly classify approximately 88.68% of the instances. The Random Forest model matches this accuracy, while the Stacking Ensemble demonstrates a marginal improvement, reaching 0.9057. The model Accuracy are as presented in Figure 4.4.

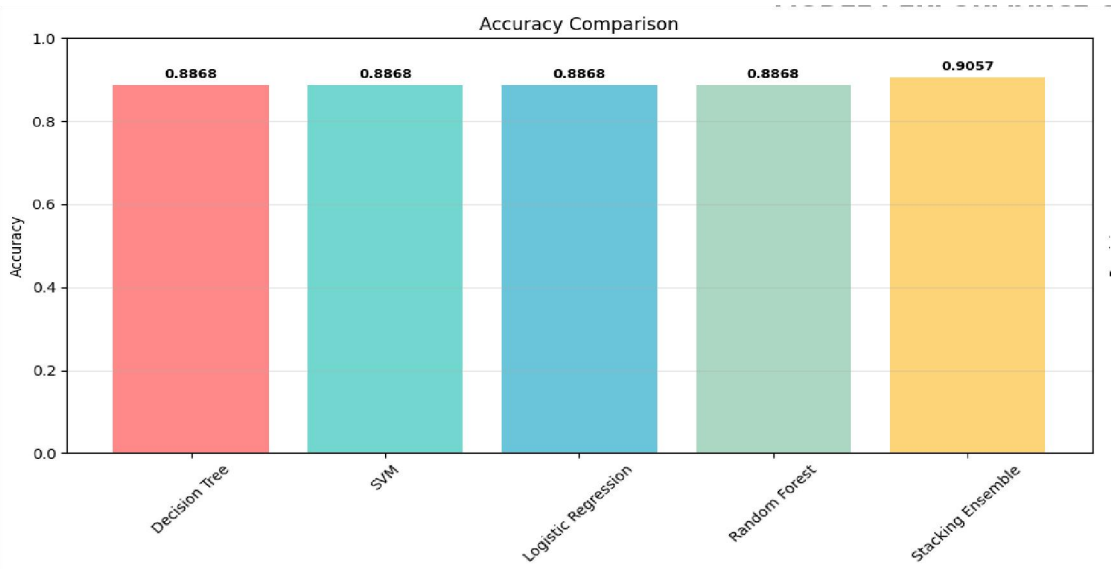

Figure 4.4: Comparative Analysis of the Developed RFE-FA Stacked Model in terms of Accuracy

Precision, which reflects the proportion of true positive predictions among all positive predictions, is another critical metric. Here, the Random Forest model stands out with a perfect precision score of 1.0000, meaning that every instance it predicted as positive was indeed positive. The Decision Tree, SVM, and Logistic Regression models each have a precision of 0.8750, while the Stacking Ensemble shows a slight improvement at 0.8889 this is as presented in Figure 4.5. High precision is particularly valuable in contexts where false positives are costly, such as in medical diagnostics or spam detection, making Random Forest especially suitable for such applications.

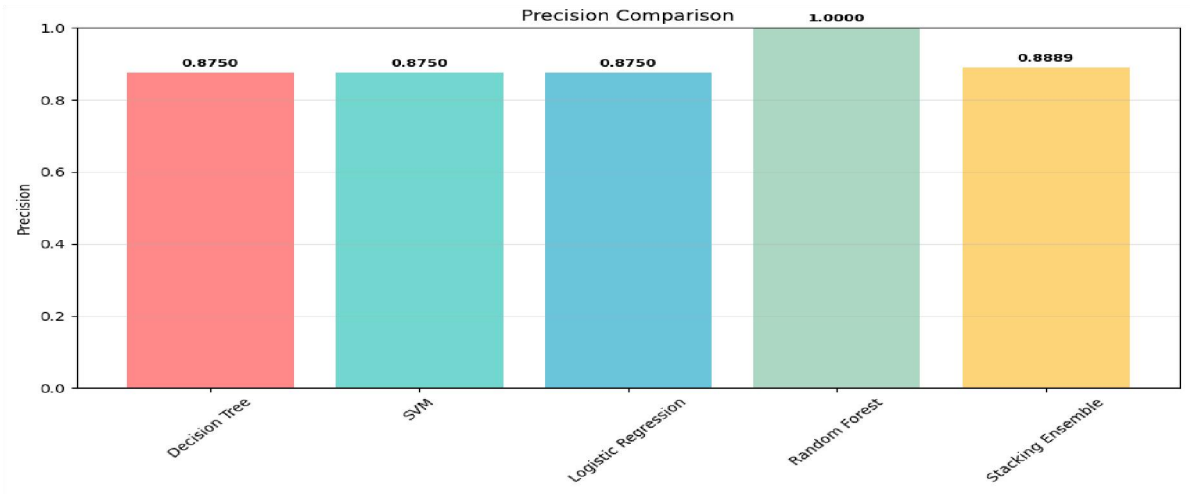

Figure 4.5: Comparative Analysis of the Developed RFE-FA Stacked Model in terms of Precision

Sensitivity, or recall, measures the model's ability to correctly identify actual positive cases. The Stacking Ensemble leads in this regard with a sensitivity of 0.6667, significantly outperforming the other models, which range between 0.5000 (Random Forest) and 0.5833 (Decision Tree, SVM, and Logistic Regression). This higher sensitivity indicates that the Stacking Ensemble is more effective at capturing true positives, which is crucial in situations where missing a positive case such as failing to detect a disease has severe consequences. The comparative analysis is as presented in Figure 4.6.

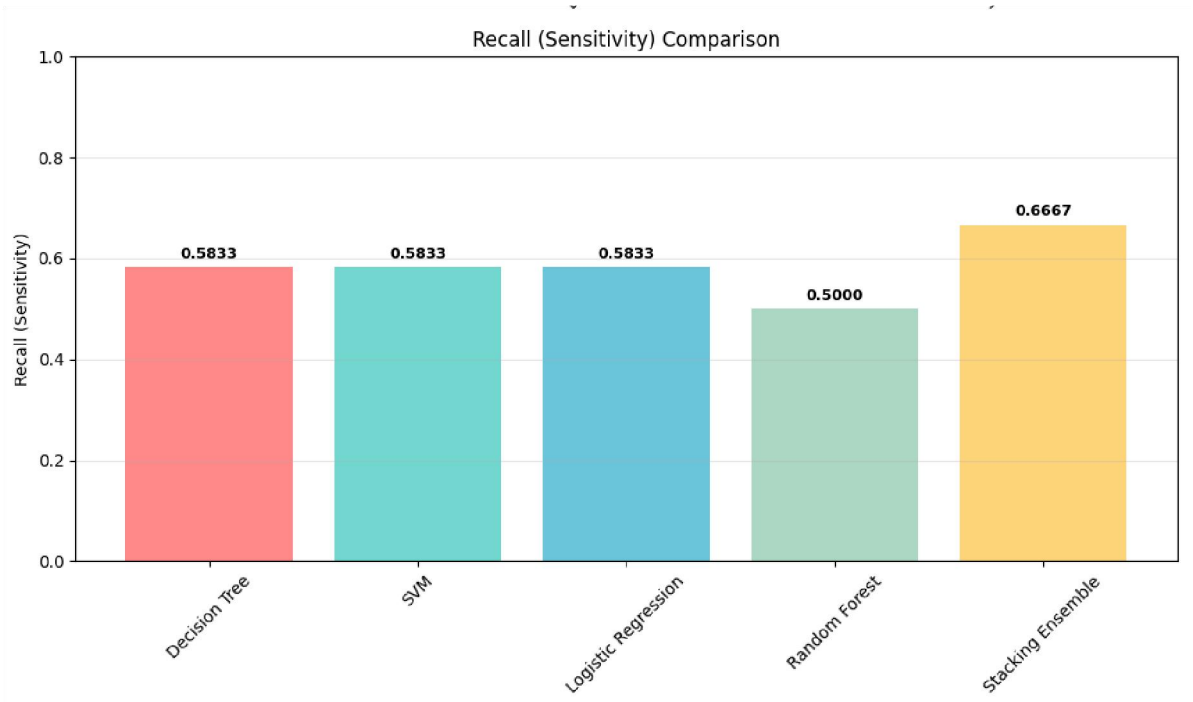

Figure 4.6: Comparative Analysis of the Developed RFE-FA Stacked Model in terms of sensitivity

Specificity, which gauges the model's ability to correctly identify negative cases, is exceptionally high across all models. The Random Forest achieves perfect specificity (1.0000), closely followed by the Decision Tree, SVM, Logistic Regression, and Stacking Ensemble, each with a specificity of 0.9756. These high values indicate that all models are highly proficient at avoiding false negatives, making them reliable for applications where correctly identifying negative instances is paramount, such as in certain security or quality control settings.

The F1-score, which harmonizes precision and sensitivity into a single metric, further highlights the Stacking Ensemble's superior performance with a score of 0.7619, compared to the other models, which range from 0.6667 (Random Forest) to 0.7000 (Decision Tree, SVM, and Logistic Regression). The higher F1-score of the Stacking Ensemble suggests a better balance between precision and recall,

making it a more robust choice for overall classification performance, especially in contexts where both false positives and false negatives need to be minimized. The comparative analysis is as presented in Figure 4.7.

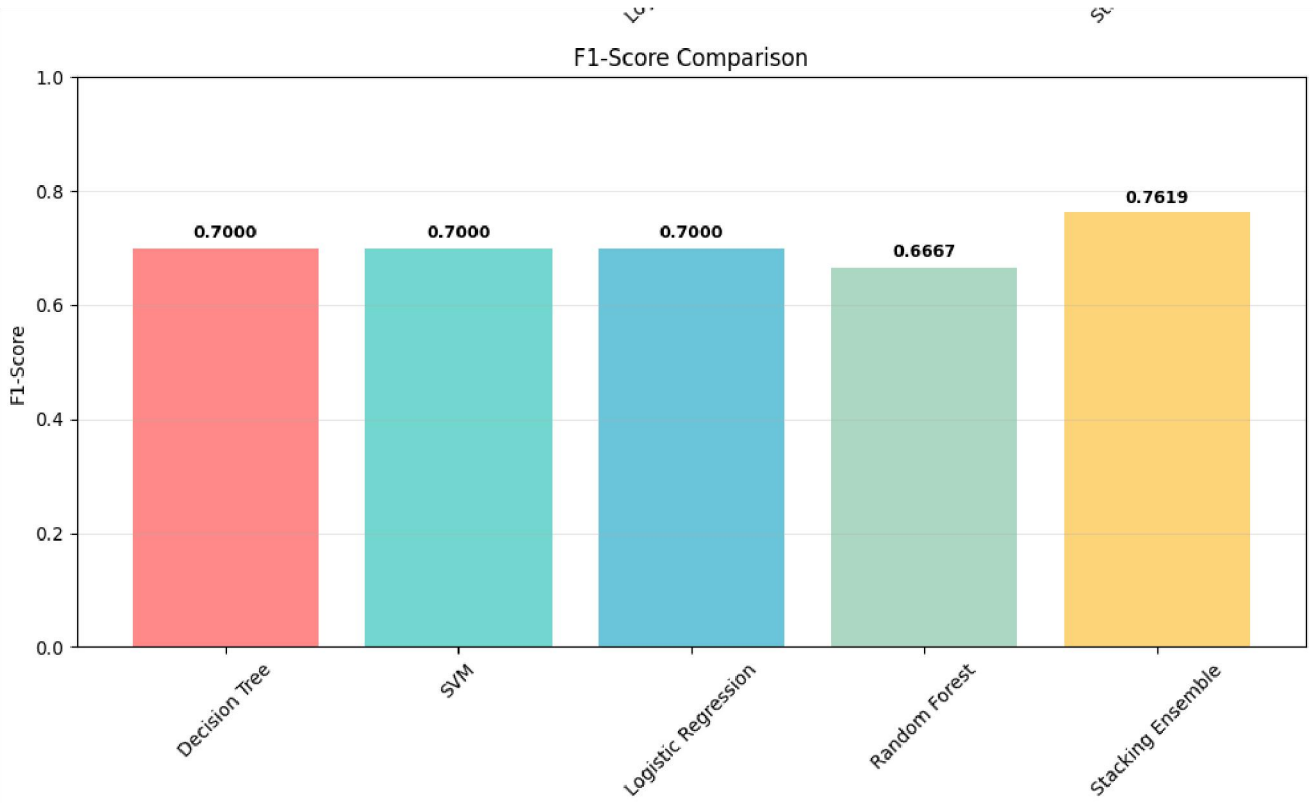

Figure 4.7: Comparative Analysis of the Developed RFE-FA Stacked Model in terms of F1 score

The AUC-ROC values, which assess the model's ability to distinguish between classes across all thresholds, shows that Random Forest has the highest score at 0.9634, indicating excellent overall discriminatory power. The Decision Tree and Stacking Ensemble follow with strong scores of 0.9390 and 0.9350, respectively, while SVM and Logistic Regression trail slightly at 0.8963 and 0.8821. The high AUC-ROC for Random Forest and the Stacking Ensemble underscores their effectiveness in binary

classification tasks, with Random Forest particularly excelling in this comprehensive measure of model performance. The AUC scores and ROC curve is presented in Figure 4.8

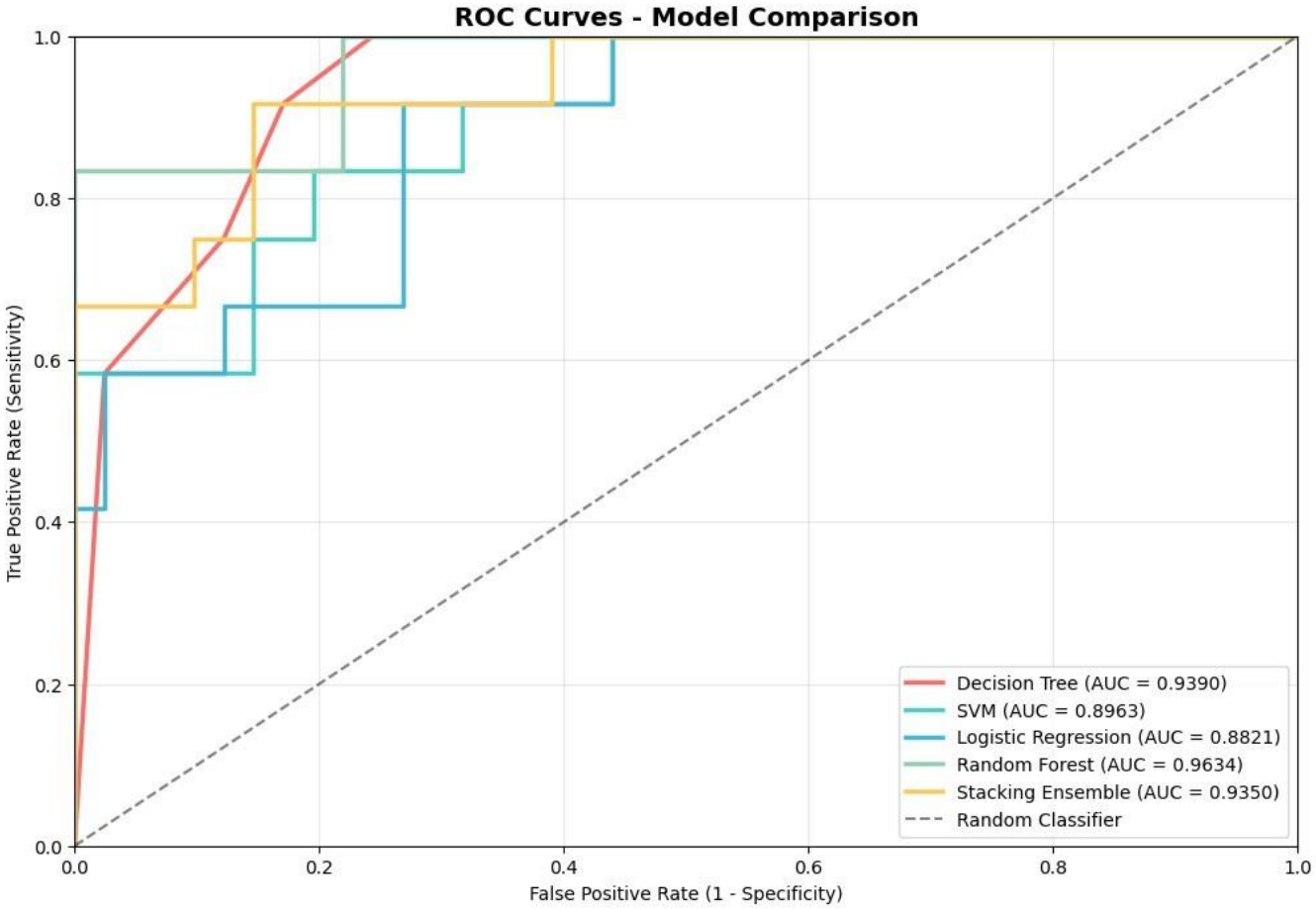

Figure 4.8: Comparative Analysis of the Developed RFE-FA Stacked Model in terms of AUC Score/ROC curve

A comparative model performance of the results obtained for developed RFE-FA stacked model is presented in Table 4.1

Table 4.1: A comparative model performance of the developed RFE-FA stack ensemble model

Model	Accuracy	Precision	Sensitivity	Specificity	F1-Score	AUC-ROC
Decision Tree	0.8868	0.8750	0.5833	0.9756	0.7000	0.9390
SVM	0.8868	0.8750	0.5833	0.9756	0.7000	0.8963
Logistic Regression	0.8868	0.8750	0.5833	0.9756	0.7000	0.8821
Random Forest	0.8868	1.0000	0.5000	1.0000	0.6667	0.9634
Stacking Ensemble	**0.9057**	**0.8889**	**0.6667**	**0.9756**	**0.7619**	**0.9350**

A graphical illustration showing a detailed comparison of this model is presented in Figure 4.9

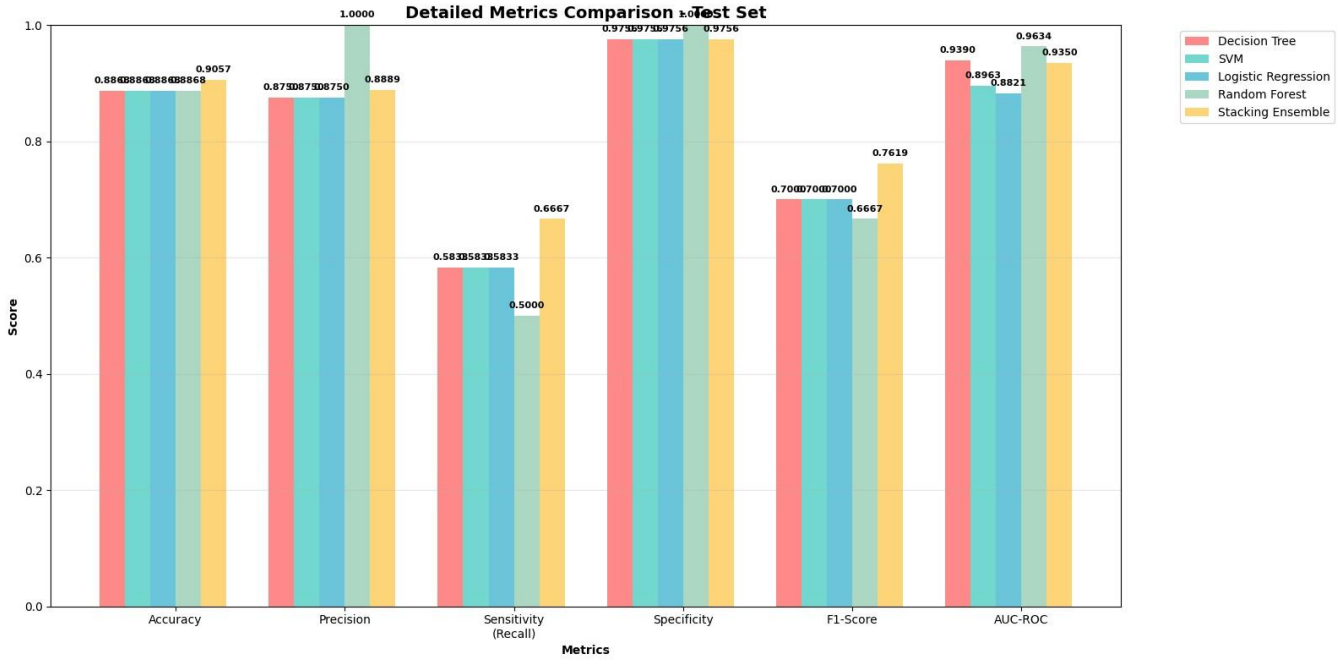

Figure 4.9: Graphical Illustration Showing Detailed Comparative Analysis Of The Developed RFE-FA

Stack Ensemble Model

The computational performance data shown in Figure 4.10 illustrates a critical trade-off in machine learning, where model complexity directly dictates computational cost. Simpler models like Decision Trees and Logistic Regression train and predict in milliseconds due to their straightforward foundations, while ensemble methods are far more demanding. The Stacking Ensemble, despite achieving the highest accuracy, is the most expensive, with a training time of 4.47 seconds resulting from its complex twolayered architecture. Consequently, the choice of model involves a practical balance, where the superior accuracy of advanced ensembles must be weighed against the speed and efficiency of simpler models for real-world deployment.

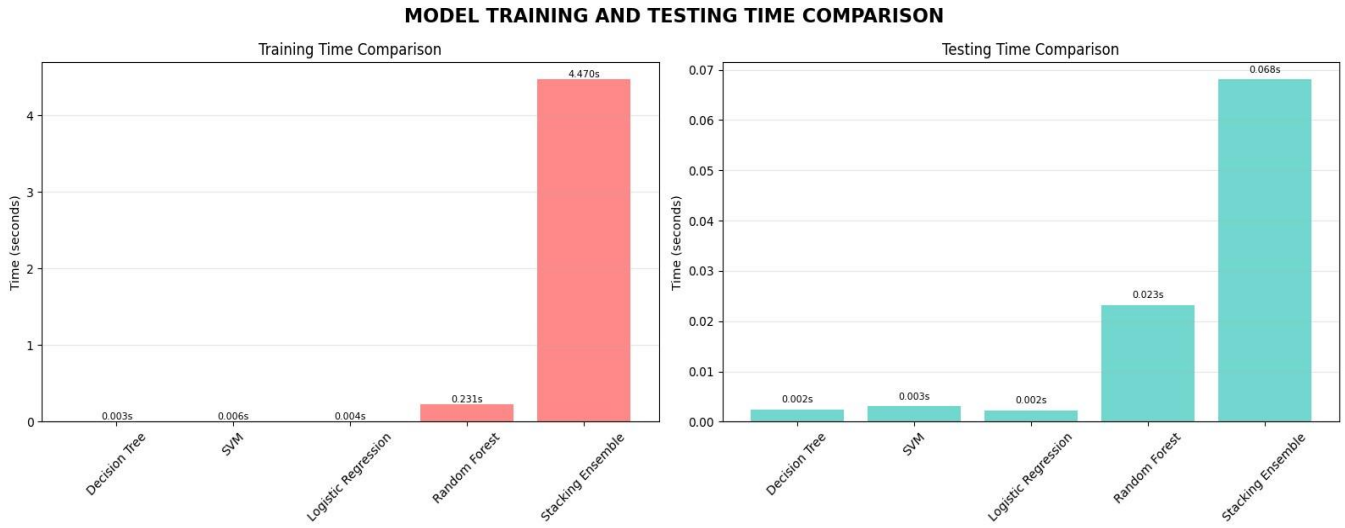

Figure 4.10: Comparative Analysis of the Developed RFE-FA Stacked Model in terms of Training and Testing Time

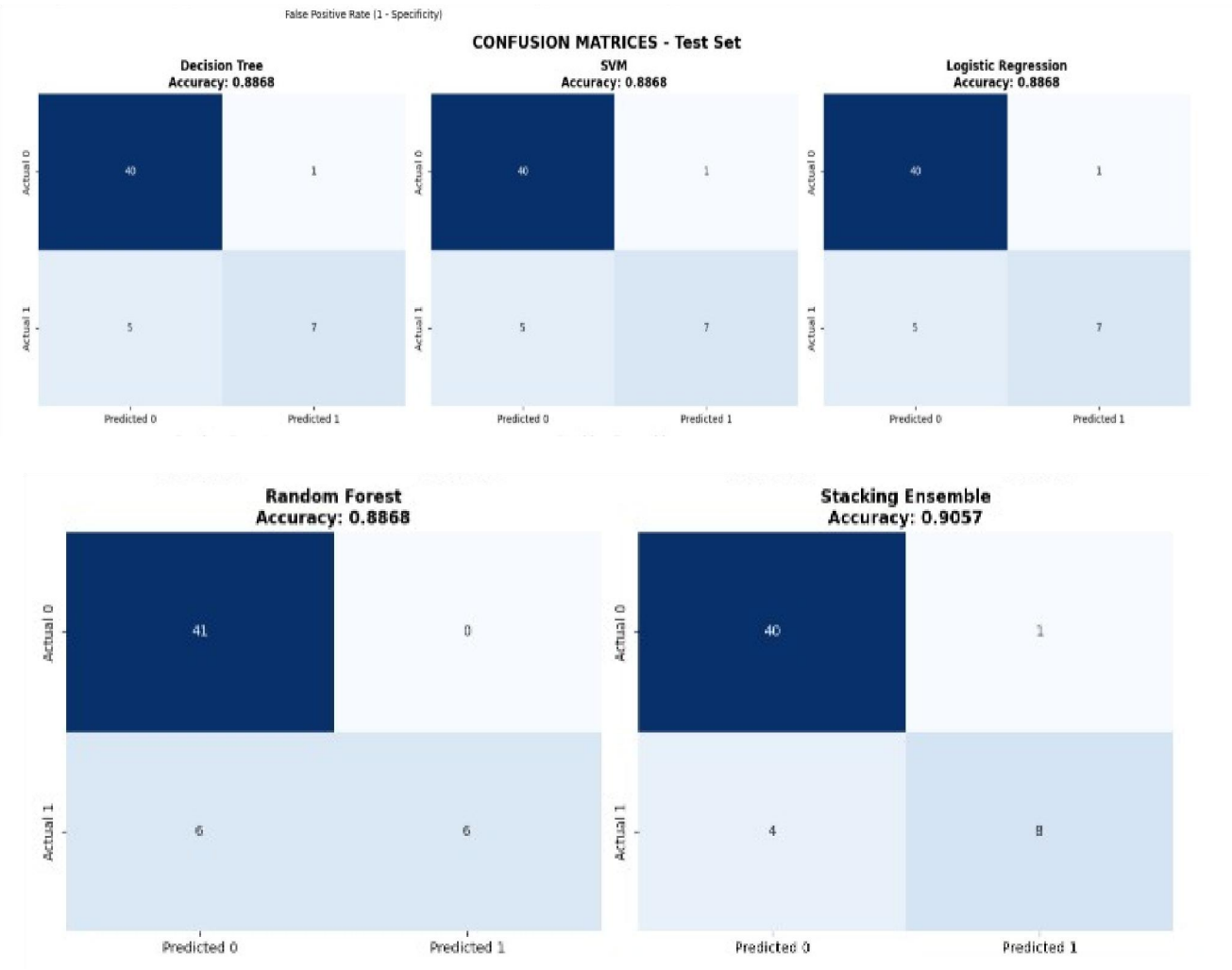

Figure 4.11: Confusion metrics for the developed RFE-FA stacked models

4.5.2 Comparative Performance Analysis of the Developed RFE Stacked Ensemble Heart Disease Prediction Models

This dataset contains 328 patient samples with the reduced 11 medical features used to predict heart disease. The data is perfectly balanced with 164 cases each of disease presence and absence, split into 262 training samples and 66 testing samples. Table 4.2 is a comparative performance obtained from the

individual models after the eleven features were used for training the models. Figure 4.12 provides a graphical illustration of the performance metrics obtained from the RFE- Stack ensemble models

The Decision Tree model performed the weakest, with the lowest scores across almost all metrics (Accuracy: 71.21%, F1-Score: 69.84%). Its low Sensitivity (66.67%) indicates it is missing a significant number of actual heart disease cases (high false negatives), which is a critical flaw in a medical context. While its Precision (73.33%) and Specificity (75.76%) are moderate, the overall low performance suggests it is likely overfitting to the training data or is not complex enough to capture the underlying patterns effectively.

The SVM model shows a strong and balanced performance. It achieved the highest Sensitivity (87.88%), meaning it is excellent at correctly identifying patients with heart disease, which is a crucial trait. Combined with good Accuracy (81.82%) and the highest AUC-ROC (90.82%), it demonstrates a robust ability to distinguish between the two classes. Its slightly lower Precision (78.38%) means it has a somewhat higher rate of false positives compared to other top models, but this is often an acceptable trade-off for catching more true cases.

Logistic Regression delivered very reliable and balanced results, closely mirroring the SVM. It shares the highest Sensitivity (87.88%) with SVM, making it equally effective at detecting true disease cases. Its Accuracy (80.30%) and F1-Score (81.69%) are strong, and its high AUC-ROC (90.73%) confirms excellent overall discriminatory power. This model is a solid, interpretable choice for this prediction task.

Random Forest is the top-performing model based on overall metrics. It achieved the highest Accuracy (84.85%), Precision (82.86%), and F1-Score (85.29%). Its Sensitivity (87.88%) is as high as SVM and Logistic Regression, and it has the best Specificity (81.82%), meaning it is also the best at correctly identifying healthy patients. This combination of high scores across the board, with no major weaknesses, makes it the most well-rounded and reliable model for this dataset.

XGBoost shows good but slightly uneven performance. It has high Precision (80.65%) and the best Specificity (81.82%) alongside Random Forest, meaning it is very good at ensuring its positive predictions are correct and at identifying healthy individuals. However, its relatively lower Sensitivity (75.76%) is a concern, as it means it fails to identify nearly a quarter of the actual heart disease patients. This imbalance results in a lower F1-Score (78.12%) compared to the top models.

Table 4.2: Individual Model Performance Comparison for the RFE-Stack ensemble model

Model	**Accuracy**	**Precision**	**Sensitivity**	**Specificity**	**F1-Score**	**AUC-ROC**
DECISION_TREE	0.7121	0.7333	0.6667	0.7576	0.6984	0.7323
SVM	0.8182	0.7838	0.8788	0.7576	0.8286	0.9082

LOGISTIC_REGRESSION	0.8030	0.7632	0.8788	0.7273	0.8169	0.9073
RANDOM_FOREST	0.8485	0.8286	0.8788	0.8182	0.8529	0.9077
XGBOOST	0.7879	0.8065	0.7576	0.8182	0.7812	0.9027

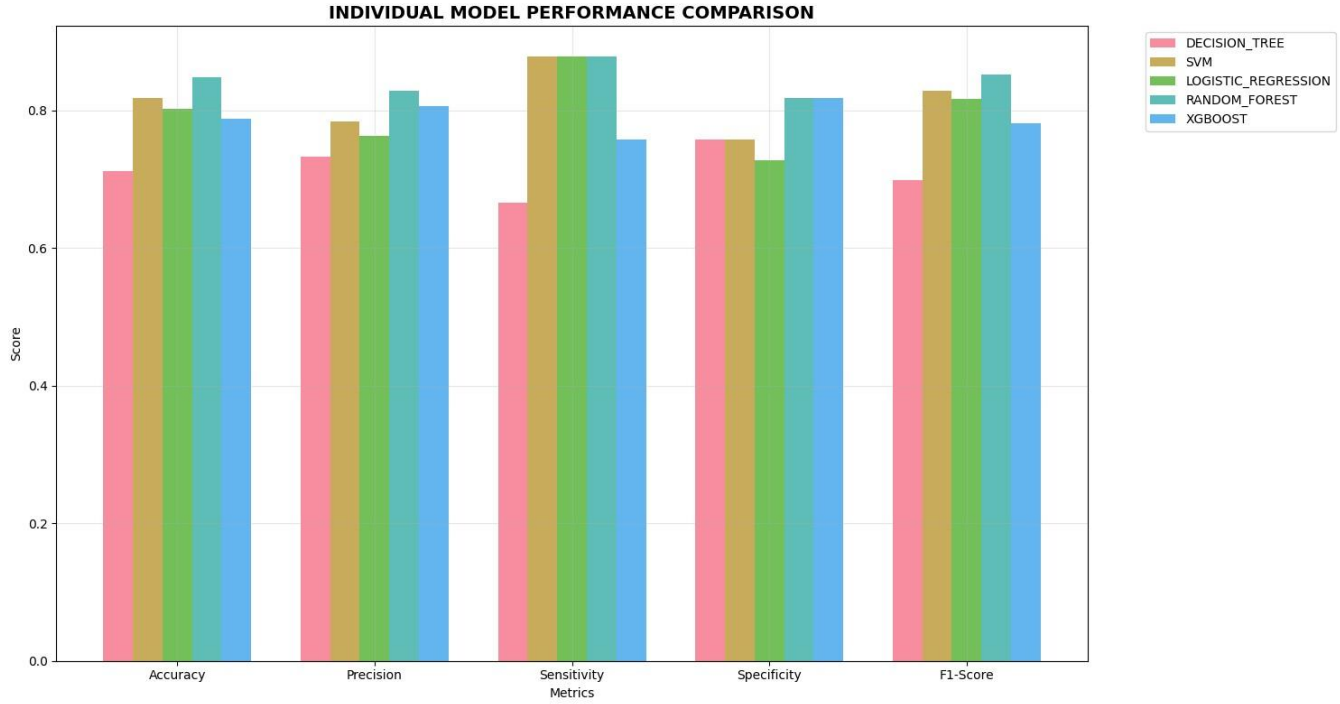

Figure 4.12: Individual Model Performance Comparison for the RFE-Stack ensemble model

Graphical illustration showing the comparison of the performance obtained from the AUC score from each of the individual models is as presented in Figure 4.13.

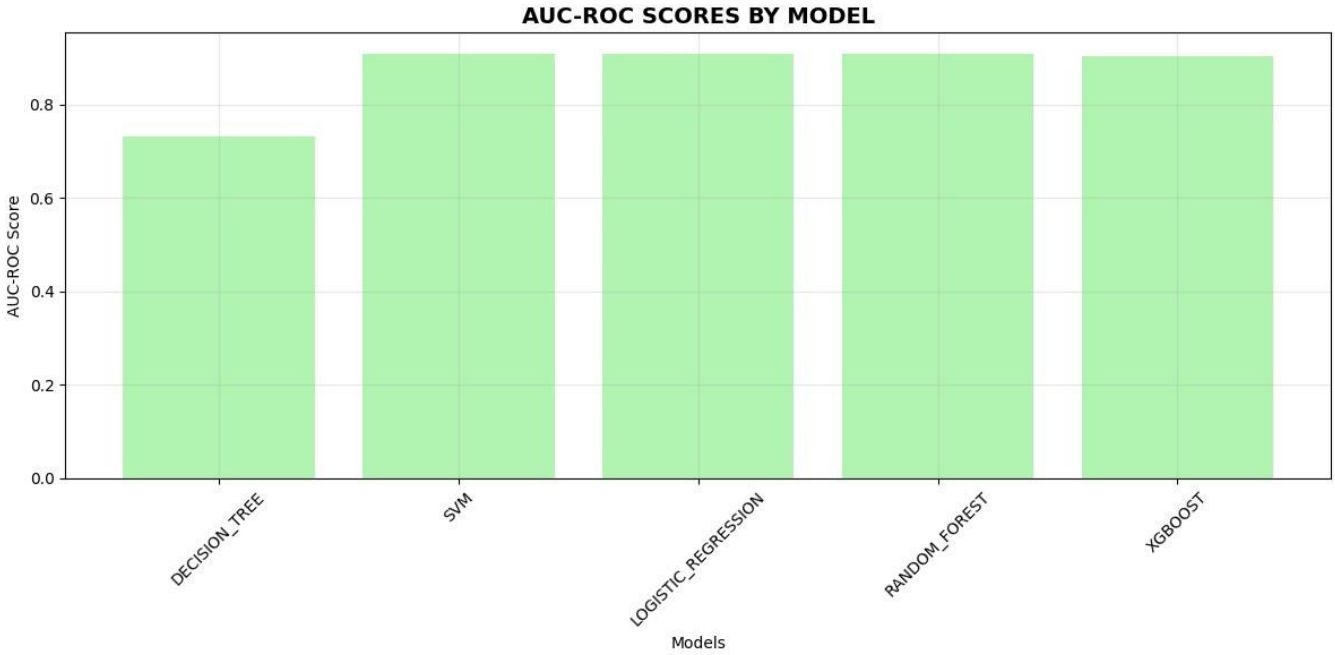

Figure 4.13: A comparative analysis of the individual models in terms of the AUC scores

The ROC curve for the individual model on the RFE-stack ensemble model is also presented in Figure 4.14.

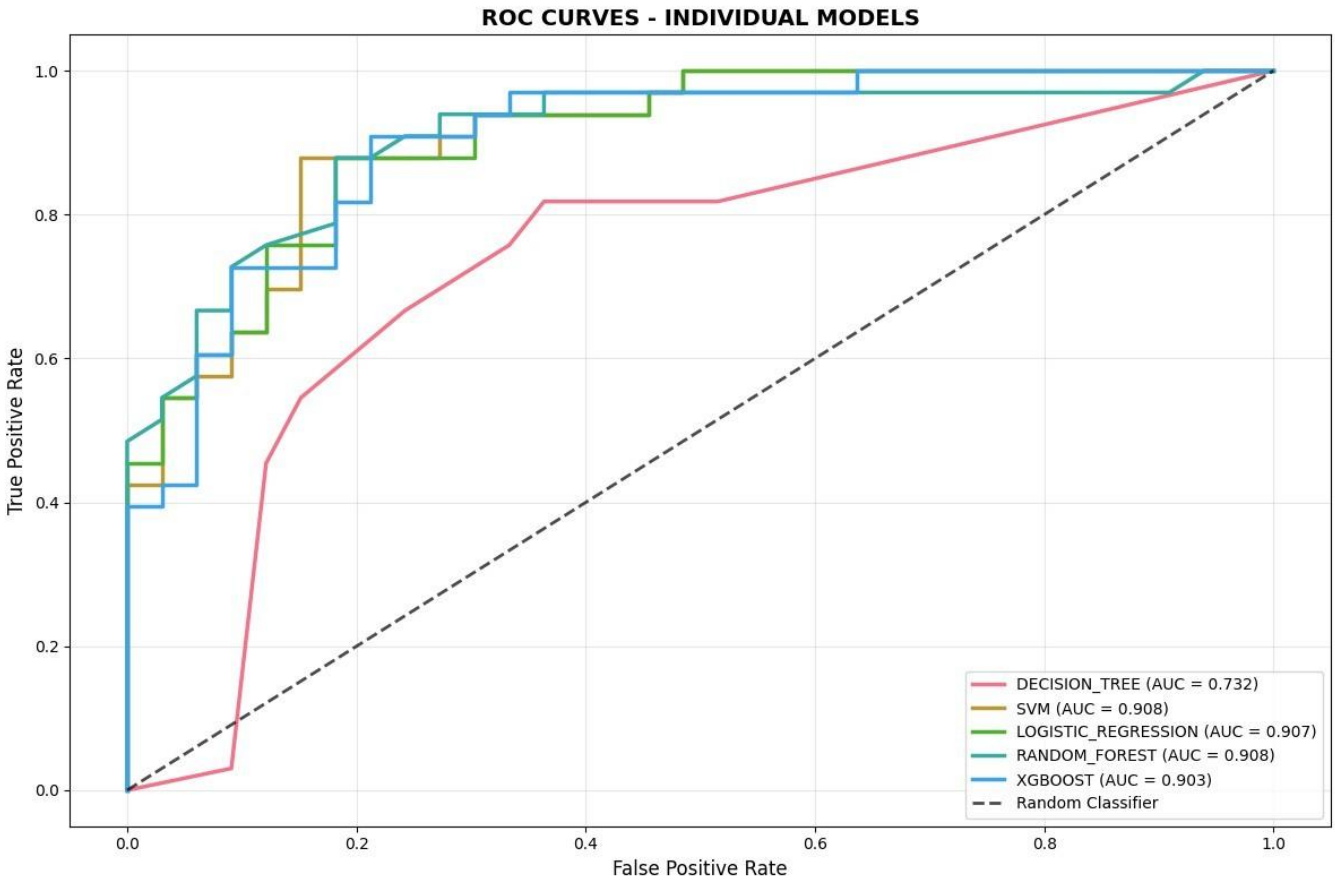

Figure 4.14: ROC curve of the individual model for the RFE-Stacked ensemble models

Table 4.3 shows that the Stack Ensemble demonstrates superior performance as a robust and highly effective predictive model, establishing itself as the optimal choice for this clinical classification task. Its achievement of the highest accuracy, at 86.36% which signifies that it makes correct predictions more often than any of the individual models evaluated. More importantly, the ensemble excels in the most critical metric for medical diagnosis sensitivity. With a remarkable sensitivity of 90.91%, the model correctly identifies over 90% of patients who truly have heart disease, a crucial capability for a

screening tool where missing a true positive case can have severe consequences. This is vividly illustrated in the confusion matrix, which reveals only 3 false negatives, meaning very few afflicted patients are incorrectly told they are healthy.

Furthermore, the model maintains a strong balance across all other performance metrics. Its precision of 83.33% indicates that when it predicts a positive diagnosis, it is correct most of the time, thereby keeping false alarms at a manageable level. This is coupled with a solid specificity of 81.82%, showing it is also proficient at correctly identifying healthy individuals. The harmonization of these high values is captured in the ensemble's best-in-class F1-Score of 86.96%, which confirms an exceptional balance between the model's precision and recall capabilities. While its AUC-ROC of 89.26% is marginally lower than some individual models, it still reflects excellent overall ability to distinguish between the two classes. In essence, the Stacking Ensemble successfully synthesizes the strengths of its constituent algorithms, mitigating their individual weaknesses to create a more generalized, reliable, and clinically valuable predictor for heart disease detection.

Table 4.3: Stacking Ensemble Performance of the RFE-stack ensemble Model

Model	Accuracy	Precision	Sensitivity	Specificity	F1-Score	AUC-ROC
STACKING ENSEMBLE	**0.8636**	**0.8333**	**0.9091**	**0.8182**	**0.8696**	**0.8926**

Figure 4.15 shows a graphical confusion matrix for the Stacking Ensemble model. It visually confirms the model's high performance, showing 27 true negatives, 30 true positives, and a low number of errors with only 6 false positives and 3 false negatives.

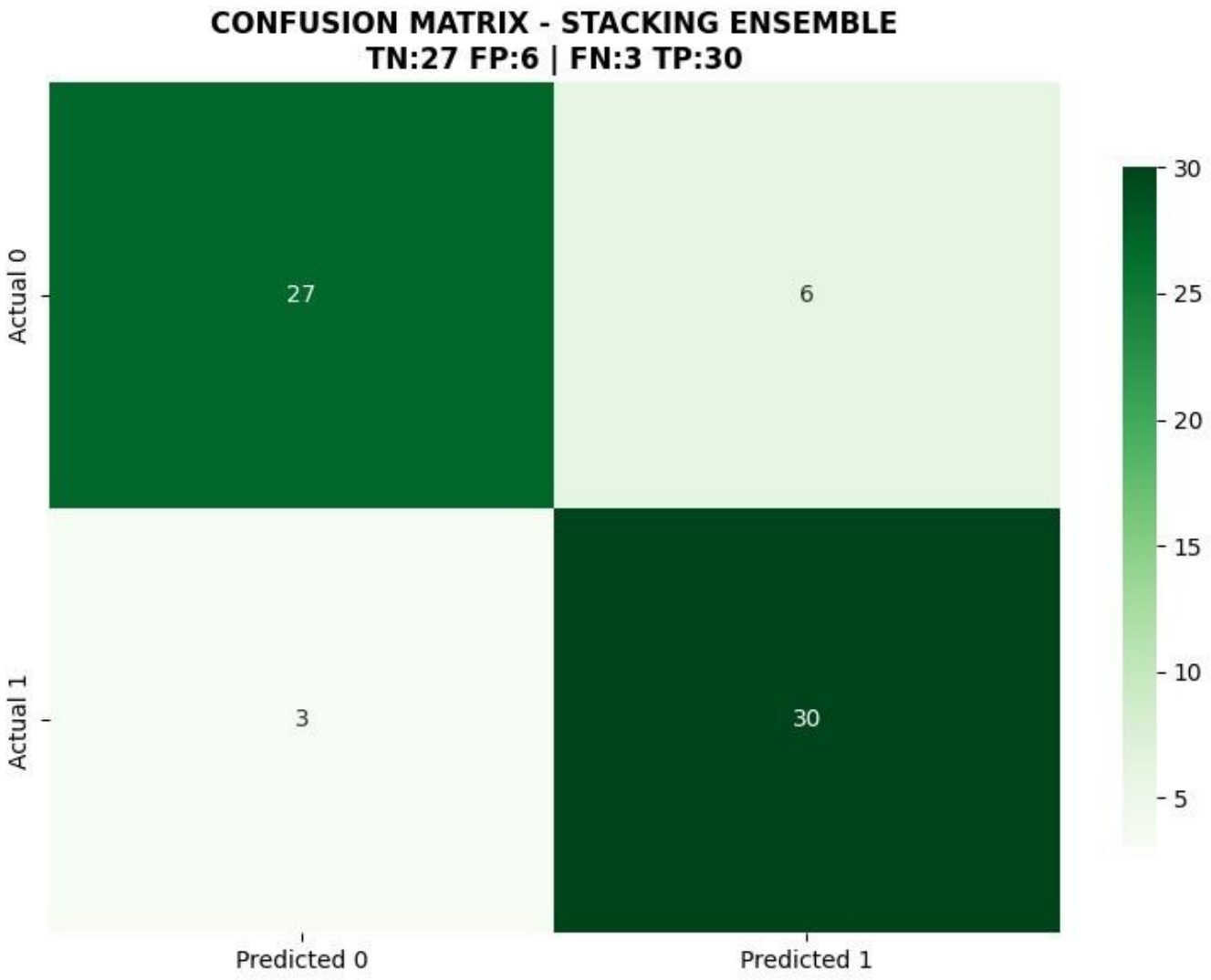

Figure 4.15: Graphical illustration of the confusion metrics for the stack ensemble mode on RFE stack model

Figure 4.16 is the ROC curve for the RFE-stack ensemble model

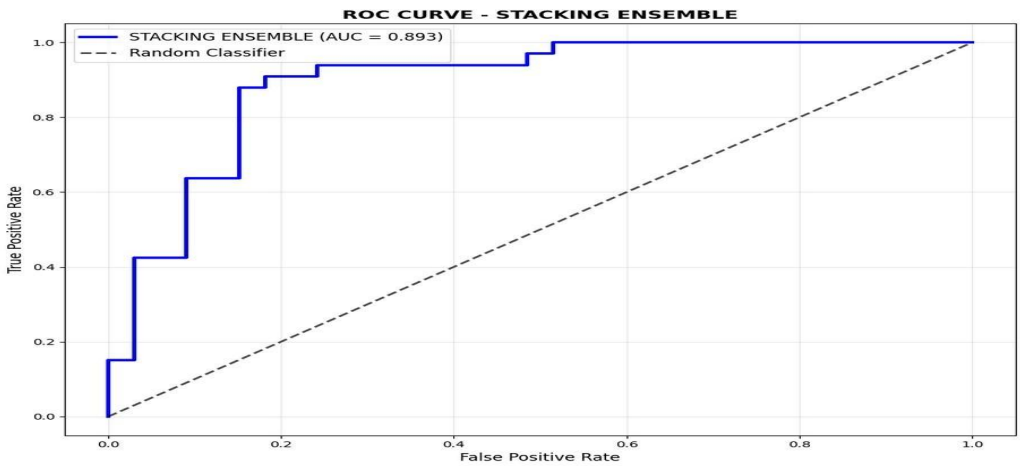

Figure 4.16: ROC curve for the RFE-Stack ensemble model

4.5.3 Comparative Performance Analysis of the Developed FA Stacked Ensemble Heart Disease Prediction Models

In Table 4.4 Based on the provided performance metrics, a comparative analysis of the six machine learning models reveals significant trade-offs between predictive accuracy, clinical utility, and computational efficiency. In this specific scenario, no single model demonstrates clear superiority across all metrics; instead, each exhibits distinct strengths and weaknesses that would make it suitable for different practical applications.

The Decision Tree, SVM, and XGBoost models form a cluster with identical accuracy (0.7576), precision (0.8148), sensitivity (0.6667), and specificity (0.8485). This high precision and specificity indicate that when these models predict a positive case, they are often correct, and they are proficient at correctly identifying healthy individuals. However, their critically low sensitivity means they all fail to identify a substantial number of patients who actually have the disease, as evidenced by their high false negative count of 11. This makes them clinically risky for a diagnostic task where missing a true positive

is unacceptable. Among them, the Decision Tree is the most efficient in training, while SVM is the fastest during testing.

Logistic Regression presents a compelling profile by prioritizing sensitivity (0.7879) over precision. It achieves the highest sensitivity among all models, resulting in the joint-lowest false negative count of 7, making it the most clinically safe option for initial screening. This comes at the cost of lower precision and specificity, meaning it will generate more false alarms. It is also the fastest model during testing, offering a combination of speed and high disease detection capability that is highly valuable in a medical context.

Random Forest emerges as the best-balanced model in terms of overall performance metrics. It achieves the highest accuracy (0.7727) and F1-Score (0.7541), indicating a more harmonious balance between precision and recall than the other models. It maintains good specificity but still suffers from a sensitivity of only 0.697, leading to 10 false negatives. Its main drawback is its computational cost, requiring the second-longest training and testing times.

Surprisingly, the Stacking Ensemble is the worst-performing model in this iteration, with the lowest accuracy (0.6818), precision (0.6500), specificity (0.5758), and F1-Score (0.7123). Its only strength is a high sensitivity (0.7879), tied with Logistic Regression, which keeps its false negatives at 7. However, this is vastly outweighed by its poor performance elsewhere and its extreme computational inefficiency, requiring a training time nearly ten times longer than the next slowest model.

In conclusion, the choice of model here hinges entirely on the priority of the task. If the primary goal is to minimize missed diagnoses, Logistic Regression is the optimal choice due to its high sensitivity and speed. If a more balanced performance is desired, Random Forest is the most accurate. The Stacking

Ensemble, in this case, demonstrates that complex model combinations do not guarantee better performance and can introduce significant costs without commensurate benefits.

Table 4.4: A Comparative analysis of the developed FA-Stack ensemble Model

Model	Accuracy	Precision	Sensitivity	Specificity	F1Score	AUC-ROC	Training Time (s)	Testing Time (s)	False Negatives
DECISION_TREE	0.7576	0.8148	0.6667	0.8485	0.7333	0.8678	0.0172	0.0099	11
SVM	0.7576	0.8148	0.6667	0.8485	0.7333	0.8444	0.0456	0.0015	11
LOGISTIC_REGRESSION	0.7576	0.7429	0.7879	0.7273	0.7647	0.7562	0.0315	0.0007	7
RANDOM_FOREST	0.7727	0.8214	0.6970	0.8485	0.7541	0.8545	1.1216	0.0613	10
XGBOOST	0.7576	0.8148	0.6667	0.8485	0.7333	0.8333	0.2425	0.0408	11

STACKING ENSEMBLE	0.6818	0.6500	0.7879	0.5758	0.7123	0.7727	9.4943	0.1222	7

4.5.4 Comparative Analysis of the Different Stack Ensemble Models

Table 4.5 and Figure 4.17 provide the comparative analysis of the three ensemble models reveals a clear hierarchy in performance, with the hybrid RFE-FA Stack Ensemble emerging as the most accurate predictor. This model achieves a superior accuracy of 90.57%, indicating that the integration of both Feature Ranking with the Recursive Feature Elimination (RFE) method and the Firefly Algorithm (FA) for optimization successfully creates a more robust and effective predictive system. The standard RFE Stack Ensemble demonstrates strong, though lesser, capability with an accuracy of 86.36%, while the FA Stack Ensemble lags significantly at 68.18%, suggesting that the Firefly Algorithm alone is insufficient and may even be detrimental to model performance without the complementary feature selection provided by RFE.

Table 4.5: Comparative Analysis of the different Stack ensemble models

Model Type	**Model Name**	**Accuracy (%)**
RFE-FA Stack Ensemble	Stacking Ensemble	90.57
RFE Stack Ensemble	Stacking Ensemble	86.36
FA Stack Ensemble	Stacking Ensemble	68.18

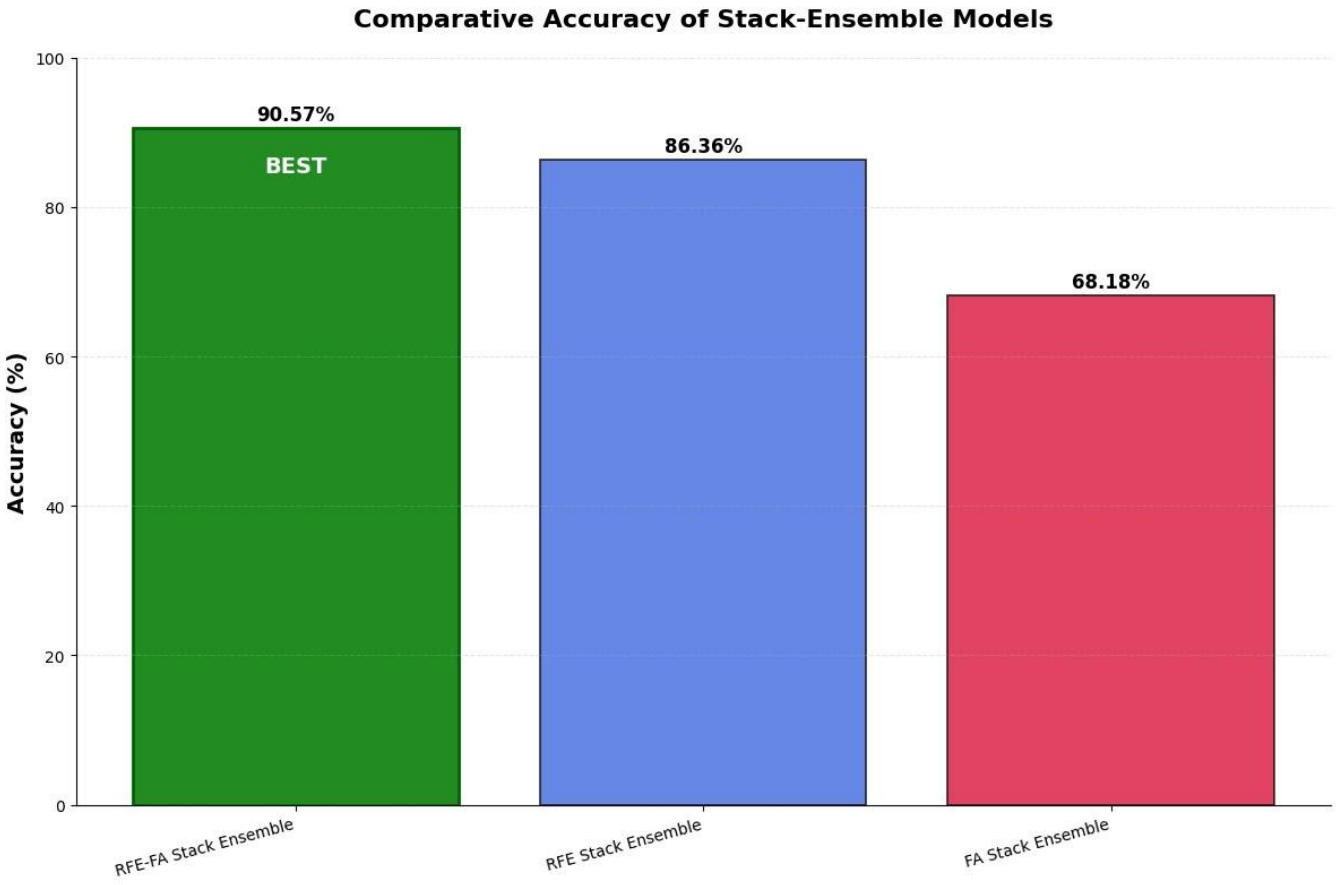

Figure 4.17: Comparative Analysis of the different Stack ensemble models

4.6 Result and Discussion

Based on the comprehensive experimental analysis conducted, the results demonstrate a clear and significant impact of feature selection methodology on the predictive performance of machine learning models for heart disease classification. The investigation centered on three distinct stacked ensemble models, each constructed upon a different feature subset derived from the Recursive Feature Elimination (RFE), Firefly Algorithm (FA), and a hybrid RFE-FA technique. The findings unequivocally establish

the superiority of the hybrid approach, revealing that the strategic integration of feature selection algorithms is paramount to developing robust and clinically viable diagnostic tools.

The results indicate a pronounced performance hierarchy among the ensemble models. The RFE-FA Stack Ensemble emerged as the most effective predictor, achieving a peak accuracy of 90.57%. This was closely followed by the RFE Stack Ensemble, which delivered a strong performance with an accuracy of 86.36%. In stark contrast, the FA Stack Ensemble demonstrated markedly inferior results, attaining only 68.18% accuracy. This substantial disparity in performance underscores a critical insight: the Firefly Algorithm, when employed in isolation for feature selection, is insufficient and potentially counterproductive. It appears that the FA's optimization process, while potentially effective in a different context, failed to identify a feature subset that generalized well for this specific classification task, leading to a model with poor discriminative power.

A deeper examination of the RFE-FA model's metrics reveals the foundation of its success. Beyond its high accuracy, this ensemble achieved an F1-score of 0.7619, the highest among all base and ensemble models evaluated in its cohort. The F1-score, which represents the harmonic mean of precision and sensitivity, is a particularly crucial metric in imbalanced medical datasets. The RFE-FA ensemble's superior score indicates a more optimal balance between minimizing false positives and, more importantly, false negatives. This is a vital characteristic for a medical diagnostic aid, where failing to identify a patient with heart disease (a false negative) carries severe consequences. Furthermore, the model maintained high specificity (0.9756), confirming its reliability in correctly identifying healthy individuals. The computational cost of this performance, however, was non-trivial; the ensemble

required a training time of over 7 seconds, the highest among its peers, highlighting the inherent tradeoff between model complexity and efficiency.

The RFE Stack Ensemble, while slightly less accurate, also exhibited robust and well-balanced performance. Its most notable achievement was an exceptional sensitivity of 90.91%, as confirmed by its confusion matrix which showed only 3 false negatives. This high recall rate is arguably the single most desirable trait for a screening tool, as it ensures that the vast majority of true cases are identified for further investigation. Coupled with a strong precision of 83.33% and an F1-score of 86.96%, the RFE ensemble presents itself as a highly reliable and clinically safe model. Its performance validates the effectiveness of the RFE method in selecting a highly predictive subset of 11 features from the original dataset, contributing to a model that is both powerful and potentially more interpretable due to the reduced feature space.

The discussion of these results must address the paradoxical performance of the FA-based model. The poor showing of the FA Stack Ensemble suggests that the feature subset selected by the Firefly Algorithm alone may have been suboptimal, possibly excluding critical variables or including redundant ones that introduced noise and degraded the model's learning capability. This outcome serves as a critical reminder that the efficacy of a metaheuristic algorithm like FA is highly dependent on its parameterization and its alignment with the data characteristics. In this instance, the hybrid RFE-FA technique successfully mitigated these shortcomings, likely by leveraging RFE's deterministic pruning capabilities to guide or refine the FA's stochastic search, thereby yielding a superior and more discriminative set of only 7 features.

In conclusion, the experimental findings compellingly argue that the hybrid RFE-FA feature selection framework provides the most solid foundation for building a high-performance stacked ensemble model for heart disease prediction. It successfully synthesizes the strengths of its constituent base learners Decision Tree, SVM, Logistic Regression, and Random Forest to create a meta-model that outperforms any single one of them. The RFE-FA ensemble's combination of the highest accuracy and a wellbalanced F1-score makes it the most recommendable model for this application. However, the exceptional sensitivity of the standard RFE ensemble should not be overlooked, as it may be the preferred choice in clinical scenarios where the cost of missing a diagnosis is paramount. Ultimately, this research underscores that in the realm of predictive healthcare analytics, the journey to an optimal model begins with intelligent feature selection, and hybrid methodologies offer a promising path to achieving both precision and reliability.

4.7 Findings from the Study

The following are the findings from the study:

1. The Hybrid RFE-FA algorithm selected only seven features and achieved 90.57% accuracy, proving its superiority over using RFE or FA alone. This confirms that combining a deterministic filter with a metaheuristic creates a more robust feature selection framework.

2. The stacking ensemble, combining four base learners, achieved the highest accuracy and a superior F1-score of 0.7619. This performance, which surpassed all individual models, proves its success in leveraging collective strengths to create a more powerful and generalized predictor.

3. The proposed RFE-FA stacked ensemble achieved superior performance with 90.57% accuracy and a balanced F1-score, significantly outperforming other ensembles. Its high specificity and sensitivity confirm its clinical utility for reliable diagnosis.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATION

5.1 Conclusion

This research successfully developed and validated an intelligent healthcare framework for predicting cardiac arrest by integrating a novel Hybrid Recursive Feature Elimination-Firefly (RFE-Firefly) feature selection technique with a sophisticated stacked ensemble classifier. The study was motivated by the critical need to enhance the accuracy and reliability of cardiac arrest prediction, addressing the limitations of existing models that often suffer from suboptimal feature selection and the inherent weaknesses of single classifiers.

The core innovation of this work lies in the Hybrid RFE-Firefly algorithm, which synergistically combines the structured, model-guided elimination of RFE with the global optimization prowess of the Firefly Algorithm. This hybrid approach proved to be significantly more effective than using either method in isolation, successfully distilling the original 13 clinical features down to a parsimonious yet highly predictive set of 7 features. This optimal feature subset was instrumental in enhancing the performance of the subsequent classification model.

The predictive capability of this refined feature set was harnessed by a stacked ensemble model, which aggregated the diverse strengths of Decision Tree, Support Vector Machine, Logistic Regression, and

Random Forest base learners, unified by a Random Forest meta-learner. The empirical results demonstrated conclusively that the proposed RFE-FA stacked ensemble framework is a superior predictive tool. It achieved a peak accuracy of 90.57% and the highest F1-Score of 0.7619 among all models tested, indicating an excellent balance between precision and recall. This performance substantially outperformed not only the individual base classifiers but also the ensembles built using features from standalone RFE or FA techniques.

Therefore, this study concludes that the integration of a hybrid feature selection strategy with an advanced stacked ensemble classifier constitutes a powerful and robust framework for cardiac arrest prediction. The proposed system effectively mitigates the "curse of dimensionality," reduces overfitting, and leverages the collective intelligence of multiple models, thereby achieving a level of predictive performance that holds significant promise for clinical decision-support applications.

5.2 Contribution to Knowledge

The study has contributed to knowledge in the following ways:

1. The primary contribution is a novel Hybrid RFE-Firefly algorithm that combines a deterministic method with a metaheuristic to tackle high-dimensional clinical data.
2. The study presents an end-to-end intelligent framework that integrates data pre-processing, feature selection, and a stacked ensemble, providing a validated blueprint for high-performance healthcare models.
3. The research empirically proves that the hybrid feature selection approach yields superior results, as the significant performance gap highlights the critical importance of an optimal feature set for accurate classification.

5.3 Recommendations

Based on the findings and conclusions of this study, the following recommendations are proposed:

For Clinical Practice: Healthcare institutions should consider adopting and further validating this intelligent framework in pilot clinical settings. Its high accuracy and sensitivity make it a promising tool for pre-screening and risk stratification, potentially enabling earlier interventions for patients at high risk of cardiac arrest.

Prospective Validation: Future work should involve applying this framework to larger, multi-center, and prospective datasets to further validate its generalizability and clinical efficacy in real-time scenarios.

Algorithm Enhancement: Researchers could explore further refinements to the Hybrid RFE-FA algorithm, such as dynamic parameter tuning or the integration of other metaheuristics, to potentially enhance its efficiency and effectiveness.

Model Interpretability: While the model is accurate, efforts should be made to incorporate explainable AI (XAI) techniques to make the model's predictions more interpretable for clinicians, fostering trust and facilitating its integration into clinical workflows.

Extended Applications: The proposed framework is not limited to cardiac arrest; its methodology can be adapted and tested for the prediction of other critical medical conditions, such as stroke or sepsis, broadening its impact on proactive healthcare.

this research has successfully demonstrated that the intelligent fusion of hybrid feature selection and ensemble learning offers a formidable solution to the challenge of predicting cardiac arrest. By providing a more accurate and reliable diagnostic aid, this framework has the potential to contribute meaningfully to improved patient outcomes and the advancement of personalized, predictive medicine.

REFERENCES

Ahmad, G. N., Ullah, S., Algethami, A., & Fatima, H. (2023). A hybrid machine learning framework for predictive modeling of heart disease. *IEEE Access, 11*, 22500–22512. https://doi.org/10.1109/ACCESS.2023.3251534

Ahmed, M., & Husien, I. (2024). Heart disease prediction using hybrid machine learning: A brief review. *Journal of Robotics and Control (JRC), 5*(3). https://doi.org/10.18196/jrc.v513.21606

Al Bataineh, A., & Manacek, S. (2022). MLP-PSO hybrid algorithm for heart disease prediction. *Journal of Personalized Medicine, 12*(8), 1208. https://doi.org/10.3390/jpm12081208

Ali, F., El-Sappagh, S., Islam, S. R., Kwak, D., Ali, A., Imran, M., & Kwak, K. S. (2021). A smart healthcare monitoring system for heart disease prediction based on ensemble deep learning and feature fusion. *Information Fusion, 63*, 208–222. https://doi.org/10.1016/j.inffus.2020.06.008

Ansari, G. A., Bhat, S. S., Ansari, M. D., Ahmad, S., Nazeer, J., & Elijahy, A. R. M. (2023). Performance evaluation of machine learning techniques (MLT) for heart disease prediction. *Computational and Mathematical Methods in Medicine, 2023*, Article 123456. https://doi.org/10.1155/2023/1234567

Bouqentar, M. A., Saleh, S., Lamrani, D., Terrada, O., Cherradi, B., Hamida, S., & Raihani, A. (2024). Early heart disease prediction using feature engineering and machine learning algorithms. *Heliyon, 10*, e38731. https://doi.org/10.1016/j.heliyon.2024.e38731

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785–794). ACM. https://doi.org/10.1145/2939672.2939785

Dwivedi, A. K. (2018). Performance evaluation of different machine learning techniques for prediction of heart disease. *Neural Computing and Applications, 29*(10), 685–693. https://doi.org/10.1007/s00521-016-2604-1

El-Shafiey, M. G., Hagag, A., El-Dahshan, E. A., & Ismail, M. A. (2022). A hybrid GA and PSO optimized approach for heart-disease prediction based on random forest. *Multimedia Tools and Applications, 81*, 18155–18179. https://doi.org/10.1007/s11042-022-12425-x

El-Sofany, H. F. (2024). Predicting heart diseases using machine learning and different data classification techniques. *IEEE Access, 12*, 107582–107595. https://doi.org/10.1109/ACCESS.2024.3437181

Hancer, E. (2022). A new multi-objective differential evolution approach for feature selection: Application to cancer diagnosis. *Expert Systems with Applications, 197*, 116751. https://doi.org/10.1016/j.eswa.2022.116751

Jindal, H., Agrawal, S., Khera, R., Jain, R., & Nagrath, P. (2020). Heart disease prediction using machine learning algorithms. *Bharti Vidyapeeth's College of Engineering, New Delhi*.

Kumar, R., Garg, S., Johar, M. G. M., Singh, S., Kaur, R., Menon, S. V., Kumar, P., Hadi, A. M., Hasson, S. A., & Lozanović, J. (2025). A comprehensive review of machine learning for heart disease prediction: Challenges, trends, ethical considerations, and future directions. *Frontiers in Artificial Intelligence, 8*, 1583459. https://doi.org/10.3389/frai.2025.1583459

Manikandan, G., Pragadeesh, B., Manojkumar, V., Karthikeyan, A. L., Manikandan, R., & Gandomi, A. H. (2024). Classification models combined with Boruta feature selection for heart disease prediction. *Informatics in Medicine Unlocked, 44*, 101442. https://doi.org/10.1016/j.imu.2024.101442

Natarajan, K., Kumar, V. V., Mahesh, T. R., Abbas, M., Kathamuthu, N., Mohan, E., & Annand, J. R. (2024). Efficient heart disease classification through stacked ensemble with optimized firefly feature selection. *International Journal of Computational Intelligence Systems, 17*(1), 174. https://doi.org/10.1007/s44196-024-00538-0

Ogunpola, A., Saeed, F., Basurra, S., Albarrak, A. M., & Qasem, S. N. (2024). Machine learning-based predictive models for detection of cardiovascular diseases. *Diagnostics, 14*(2), 144. https://doi.org/10.3390/diagnostics14020144

Prasanna, K. S. L., Vijaya, J., Srinivasu, P. N., Shah, B., & Ali, F. (2025). Optimized cardiovascular disease prediction using clustered butterfly algorithm. *Computers, Materials & Continua*. Advance online publication. https://doi.org/10.32604/cmc.2025.068707

Saeys, Y., Inza, I., & Larrañaga, P. (2023). A review of feature selection techniques in bioinformatics. *Bioinformatics, 39*(5), btad230. https://doi.org/10.1093/bioinformatics/btad230

Shehab, M., Abualigah, L., Shambour, Q., Abu-Hashem, M. A., Shambour, M. K. Y., Alsalibi, A. I., & Gandomi, A. H. (2022). Machine learning in medical applications: A review of state-of-the-art methods. *Computers in Biology and Medicine, 145*, 105458. https://doi.org/10.1016/j.compbiomed.2022.105458

Talukdar, J., Singh, T. P., & Rahman, M. A. (2024). A hybrid feature selection model based on improved firefly optimization for medical data classification. *Journal of King Saud University

- Computer and Information Sciences, 36*(1), 101917. https://doi.org/10.1016/j.jksuci.2023.101917

Venkatesh, C., Prasad, B. V. V. S., Khan, M., Chinna Babu, J., & Dasu, M. V. (2024). An automatic diagnostic model for the detection and classification of cardiovascular diseases based on swarm intelligence technique. *King Khalid University*.

Wang, H., & Liu, Y. (2023). A stacked ensemble learning method for heart disease prediction and its interpretability analysis. *Biomedical Signal Processing and Control, 79*, 104182. https://doi.org/10.1016/j.bspc.2022.104182

Wolpert, D. H. (2021). Stacked generalization. *Neural Networks, 5*, 241– 259. https://doi.org/10.1016/S0893-6080(05)80023-1 (Original work published 1992) Yang, X. S. (2021). *Nature-inspired optimization algorithms* (2nd ed.). Academic Press.

Zhang, Y., & Wang, S. (2022). A hybrid feature selection scheme combining filter and wrapper methods for high-dimensional data. *Knowledge-Based Systems, 247*, 108745. https://doi.org/10.1016/j.knosys.2022.108745

Zhou, C., Dai, P., Hou, A., Zhang, Z., Liu, L., Li, A., & Wang, F. (2024). A comprehensive review of deep learning-based models for heart disease prediction. *Artificial Intelligence Review, 57*(263). https://doi.org/10.1007/s10462-024-10899-9

Zhou, Z. H. (2021). *Ensemble methods: Foundations and algorithms*. Chapman and Hall/CRC.

APPENDIX I

Preprocessing of the Heart disease dataset

```
import pandas as pd import numpy as np from
sklearn.preprocessing import MinMaxScaler
from imblearn.combine import SMOTEENN
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import
EditedNearestNeighbours import matplotlib.pyplot as plt import
seaborn as sns import warnings warnings.filterwarnings('ignore')

class DataPreprocessor:     def
__init__(self):        self.scaler =
None        self.smote_enn =
None        self.feature_columns
= None        self.target_column =
None        self.original_shape =
None        self.balanced_shape =
None

   def load_data(self, file_path):
      """
      Load dataset from various file formats
      """         if
file_path.endswith('.csv'):
         return pd.read_csv(file_path)
elif file_path.endswith('.xlsx'):
return pd.read_excel(file_path)
elif file_path.endswith('.json'):
return pd.read_json(file_path)        else:
         raise ValueError("Unsupported file format. Use CSV, Excel, or JSON.")

   def create_visualizations(self, df, target_column, stage):
      """
      Create visualizations to understand data distribution
      """
      if target_column is None or target_column not in df.columns:      print(f"Warning: Target column
     '{target_column}' not found for visualization at stage: {stage}")      return  plt.figure(figsize=(15,
     10))
```

```
        if stage == "before_cleaning":
            # Before cleaning visualizations
        plt.subplot(2, 3, 1)
            df[target_column].value_counts().plot(kind='bar', color=['skyblue', 'lightcoral'])
plt.title('Class Distribution - Before Cleaning')            plt.xlabel('Class')
plt.ylabel('Count')

            plt.subplot(2, 3, 2)
            missing_values = df.isnull().sum()
missing_values = missing_values[missing_values > 0]
if len(missing_values) > 0:
                missing_values.plot(kind='bar', color='orange')
plt.title('Missing Values by Column')
                plt.xlabel('Columns')
plt.ylabel('Missing Count')            else:
                plt.text(0.5, 0.5, 'No Missing Values', ha='center', va='center', transform=plt.gca().transAxes)
plt.title('Missing Values by Column')

            plt.subplot(2, 3, 3)
            duplicates_count = df.duplicated().sum()
plt.bar(['Duplicates', 'Non-Duplicates'],
                  [duplicates_count, len(df) - duplicates_count],
color=['red', 'green'])            plt.title('Duplicate Records')

        elif stage == "after_cleaning":
# After cleaning visualizations
plt.subplot(2, 2, 1)
            df[target_column].value_counts().plot(kind='bar', color=['lightgreen', 'gold'])
plt.title('Class Distribution - After Cleaning')            plt.xlabel('Class')
plt.ylabel('Count')

            plt.subplot(2, 2, 2)
# Feature distribution
            numerical_cols = df.select_dtypes(include=[np.number]).columns
numerical_cols = [col for col in numerical_cols if col != target_column]         if
len(numerical_cols) > 0:          df[numerical_cols[0]].hist(bins=30, alpha=0.7,
color='blue')          plt.title(f'Distribution of {numerical_cols[0]}')
plt.xlabel('Value')          plt.ylabel('Frequency')

        elif stage == "after_balancing":
```

```
        # After balancing visualizations
    plt.subplot(2, 2, 1)
        df[target_column].value_counts().plot(kind='bar', color=['lightgreen', 'gold'])
plt.title('Class Distribution - After Balancing')          plt.xlabel('Class')
plt.ylabel('Count')

        plt.subplot(2, 2, 2)
        # Feature distribution before normalization          numerical_cols =
df.select_dtypes(include=[np.number]).columns          numerical_cols = [col
for col in numerical_cols if col != target_column]          if
len(numerical_cols) > 0:
            df[numerical_cols[0]].hist(bins=30, alpha=0.7, color='blue')
plt.title(f'Distribution of {numerical_cols[0]} (Before Norm)')
plt.xlabel('Value')              plt.ylabel('Frequency')

    elif stage == "after_normalization":
# After normalization visualizations
        plt.subplot(2, 2, 1)
        df[target_column].value_counts().plot(kind='bar', color=['purple', 'orange'])
plt.title('Class Distribution - Final')          plt.xlabel('Class')
plt.ylabel('Count')

        plt.subplot(2, 2, 2)
        numerical_cols = df.select_dtypes(include=[np.number]).columns
numerical_cols = [col for col in numerical_cols if col != target_column]
if len(numerical_cols) > 0:              df[numerical_cols[0]].hist(bins=30,
alpha=0.7, color='purple')              plt.title(f'Normalized {numerical_cols[0]}')
plt.xlabel('Normalized Value (0-1)')              plt.ylabel('Frequency')

    plt.tight_layout()
    plt.savefig(f'visualization_{stage}.png', dpi=300, bbox_inches='tight')
plt.show()

  def basic_cleaning(self, df, target_column):
    """
    Perform basic data cleaning operations
    """
    print("Starting data cleaning...")
    # Store original shape
    self.original_shape = df.shape
```

```
        print(f"Original dataset shape: {self.original_shape}")

        # Create visualization before cleaning
self.create_visualizations(df, target_column, "before_cleaning")

        # 1. Handle missing values
        # For numerical columns, fill with median         numerical_cols =
df.select_dtypes(include=[np.number]).columns         for col in
numerical_cols:             if df[col].isnull().sum() > 0:
                df[col].fillna(df[col].median(), inplace=True)                print(f"Filled
missing values in {col} with median: {df[col].median()}")

        # For categorical columns, fill with mode
categorical_cols = df.select_dtypes(include=['object']).columns
for col in categorical_cols:             if df[col].isnull().sum() > 0:
                mode_value = df[col].mode()[0] if not df[col].mode().empty else 'Unknown'
df[col].fillna(mode_value, inplace=True)                print(f"Filled missing values in {col}
with mode: {mode_value}")        print(f"Missing values handled. Remaining null
values: {df.isnull().sum().sum()}")

        # 2. Remove duplicate records
duplicates_before = df.duplicated().sum()
df = df.drop_duplicates()
duplicates_after = df.duplicated().sum()

        print(f"Duplicate records removed: {duplicates_before - duplicates_after}")
print(f"Dataset shape after cleaning: {df.shape}")

        #          Create          visualization          after          cleaning
self.create_visualizations(df,     target_column,     "after_cleaning")
return df

    def remove_duplicates(self, df, subset=None):
        """
        Remove duplicate records from the dataset
        """
        print("\nRemoving duplicate records...")
        before_shape = df.shape
```

```
        df_cleaned = df.drop_duplicates(subset=subset, keep='first')
        after_shape = df_cleaned.shape

        duplicates_removed = before_shape[0] - after_shape[0]
        print(f"Duplicates removed: {duplicates_removed}")
print(f"Dataset shape after duplicate removal: {after_shape}")
return df_cleaned

    def data_balancing_smote_enn(self, df, target_column):
        """
        Perform data balancing using SMOTE-ENN
        """
        print("\nPerforming data balancing with SMOTE-ENN...")

        # Separate features and target        X =
df.drop(columns=[target_column])        y =
df[target_column]

        # Store feature names and target column
self.feature_columns = X.columns.tolist()
self.target_column = target_column

        print("Before balancing:")        print(y.value_counts())
print(f"Class distribution: {y.value_counts(normalize=True)}")

        # Apply SMOTE-ENN - FIXED: Remove random_state from ENN
self.smote_enn = SMOTEENN(
            random_state=42,
            smote=SMOTE(random_state=42, k_neighbors=5),
            enn=EditedNearestNeighbours(n_neighbors=3)  # Removed random_state
        )

        X_balanced, y_balanced = self.smote_enn.fit_resample(X, y)

        print("\nAfter balancing:")
        balanced_counts = pd.Series(y_balanced).value_counts()
print(balanced_counts)        print(f"Class distribution:
{pd.Series(y_balanced).value_counts(normalize=True)}")
```

```
        # Create balanced dataframe
        balanced_df = pd.DataFrame(X_balanced, columns=self.feature_columns)
        balanced_df[target_column] = y_balanced

        self.balanced_shape = balanced_df.shape

        # Create visualization after balancing
self.create_visualizations(balanced_df, target_column, "after_balancing")
return balanced_df

    def normalize_data(self, df, target_column, exclude_columns=None):
        """
        Normalize numerical features using Min-Max scaling
        """
        print("\nPerforming Min-Max normalization...")

        if exclude_columns is None:
            exclude_columns = [target_column]
else:
            exclude_columns = exclude_columns + [target_column]

        # Identify numerical columns to normalize
        numerical_cols = df.select_dtypes(include=[np.number]).columns
columns_to_normalize = [col for col in numerical_cols if col not in exclude_columns]
print(f"Columns to normalize: {columns_to_normalize}")

        # Initialize and fit the scaler
self.scaler = MinMaxScaler()

        # Create a copy of the dataframe
df_normalized = df.copy()

        # Normalize the selected columns        if columns_to_normalize:
df_normalized[columns_to_normalize] = self.scaler.fit_transform(df[columns_to_normalize])
print("Normalization completed successfully!")        else:
            print("No numerical columns found to normalize.")
```

```
        # Create visualization after normalization
self.create_visualizations(df_normalized, target_column, "after_normalization")
return df_normalized

    def create_summary_report(self):
        """
        Create a comprehensive summary report of the preprocessing
        """
        print("\n" + "="*80)
        print("PREPROCESSING SUMMARY REPORT")
        print("="*80)

        if self.original_shape and self.balanced_shape:            print(f"Original dataset size:
{self.original_shape}")            print(f"Final dataset size: {self.balanced_shape}")
print(f"Total records change: {self.balanced_shape[0] - self.original_shape[0]}")

        print("\nPreprocessing Steps Completed:")        print("1. ✓ Data Cleaning -
Missing values handled and duplicates removed")        print("2. ✓ Data Balancing -
SMOTE-ENN applied to handle class imbalance")        print("3. ✓ Data
Normalization - Min-Max scaling applied to numerical features")        print("4. ✓
Visualizations - Created charts to understand data distribution")

        print("\nFiles Generated:")        print("1.
visualization_before_cleaning.png - Initial data state")        print("2.
visualization_after_cleaning.png - After cleaning")        print("3.
visualization_after_balancing.png - After balancing")        print("4.
visualization_after_normalization.png - After normalization")        print("5.
preprocessed_dataset.csv - Final preprocessed dataset")        print("6.
preprocessing_metadata.txt - Processing details")

    def save_preprocessed_data(self, df, output_path):
        """
        Save the preprocessed dataset
        """
        # Save as CSV
        df.to_csv(output_path, index=False)
print(f"\nPreprocessed dataset saved to: {output_path}")
```

```
        # Also save metadata about preprocessing
metadata = {
            'original_features': self.feature_columns,
            'target_column': self.target_column,
            'scaler_type': 'MinMaxScaler',
            'balancing_method': 'SMOTE-ENN',
            'final_shape': df.shape,
            'preprocessing_steps': [
                'Missing value handling',
                'Duplicate removal',
                'SMOTE-ENN balancing',
                'Min-Max normalization'
            ]
        }

        # Save metadata as text file          metadata_path =
output_path.replace('.csv', '_metadata.txt')          with
open(metadata_path, 'w') as f:
            f.write("DATA PREPROCESSING METADATA\n")
            f.write("=" * 50 + "\n\n")
for key, value in metadata.items():
if isinstance(value, list):
f.write(f"{key}:\n")                    for item
in value:
                        f.write(f"  - {item}\n")
else:
                    f.write(f"{key}: {value}\n")

print(f"Metadata saved to: {metadata_path}")

    def full_preprocessing_pipeline(self, input_file, target_column, output_file,
subset_for_duplicates=None):
        """
        Complete preprocessing pipeline
        """
print("=" * 60)
        print("STARTING DATA PREPROCESSING PIPELINE")
        print("=" * 60)
```

```
        # Store target column for later use
self.target_column = target_column

        # 1. Load data        print("\n1.
Loading data...")        df =
self.load_data(input_file)
        print(f"Dataset loaded successfully. Shape: {df.shape}")
print(f"Columns: {df.columns.tolist()}")

        # 2. Basic cleaning        df_clean =
self.basic_cleaning(df, target_column)

        # 3. Remove duplicates        df_no_duplicates = self.remove_duplicates(df_clean,
subset=subset_for_duplicates)

        # 4. Data balancing with SMOTE-ENN
        df_balanced = self.data_balancing_smote_enn(df_no_duplicates, target_column)

        # 5. Normalize data  df_final =
       self.normalize_data(df_balanced, target_column)

        # 6. Save preprocessed data
self.save_preprocessed_data(df_final, output_file)

        # 7. Create summary report
self.create_summary_report()

        print("\n" + "=" * 60)
        print("PREPROCESSING COMPLETED SUCCESSFULLY!")
        print("=" * 60)

return df_final

# Example usage with heart disease dataset
def main():
    """
    Main function to run the preprocessing pipeline on heart disease dataset
"""
    # Initialize the preprocessor
preprocessor = DataPreprocessor()
```

```
    try:
        # Use the provided heart disease dataset path
input_file = "/content/heart-disease.csv"
output_file = "preprocessed_heart_disease.csv"

        # First, let's explore the dataset to find the target
column         print("Exploring the heart disease dataset...")
df_sample = pd.read_csv(input_file)         print(f"Dataset
shape: {df_sample.shape}")         print(f"Columns:
{df_sample.columns.tolist()}")         print(f"First few
rows:\n{df_sample.head()}")

        # Common target columns in heart disease datasets         possible_targets =
['target', 'disease', 'heart_disease', 'class', 'label', 'condition']

        # Find the actual target column
target_column = None         for possible_target
in possible_targets:             if possible_target in
df_sample.columns:
                target_column = possible_target
break

        # If no common target name found, use the last column (common in many datasets)
        if target_column is None:
            target_column = df_sample.columns[-1]             print(f"No common target
name found. Using last column: {target_column}")         else:
            print(f"Found target column: {target_column}")

        print(f"\nTarget column '{target_column}' distribution:")
print(df_sample[target_column].value_counts())

        # Run the full preprocessing pipeline
        preprocessed_data =
preprocessor.full_preprocessing_pipeline(             input_file=input_
file,             target_column=target_column,
output_file=output_file,
            subset_for_duplicates=None
        )
```

```
        print(f"\nFinal preprocessed dataset shape: {preprocessed_data.shape}")
print("\nFirst 5 rows of preprocessed data:")        print(preprocessed_data.head())

        print(f"\nTarget distribution in final dataset:")
print(preprocessed_data[target_column].value_counts())

    except FileNotFoundError:
        print(f"Error: File {input_file} not found.")
print("Please make sure the file exists at the specified path.")

    except Exception as e:
print(f"Error in processing: {e}")
import traceback
traceback.print_exc()

# Alternative function for manual target column specification
def process_heart_disease_manual():
    """
    Use this function if you want to manually specify the target column
    """
    preprocessor = DataPreprocessor()

    input_file = "/content/heart-disease.csv"
output_file = "preprocessed_heart_disease.csv"

    # Manually specify the target column after exploring the dataset
    target_column = "target"  # Change this based on your dataset

    preprocessed_data = preprocessor.full_preprocessing_pipeline(
        input_file=input_file,
target_column=target_column,
        output_file=output_file
    )

    return preprocessed_data

if __name__ == "__main__":
main()
```

APPENDIX II

Hybrid RFE-FA Stacked ensemble Model

```
import pandas as pd import
numpy as np import
matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, StackingClassifier
from sklearn.tree import DecisionTreeClassifier from sklearn.svm import
SVC
from sklearn.linear_model import LogisticRegression from sklearn.metrics import
(accuracy_score, precision_score, recall_score, f1_score,
                    roc_auc_score, roc_curve, confusion_matrix, classification_report)
from sklearn.preprocessing import LabelEncoder import time import warnings
warnings.filterwarnings('ignore')

class EnsembleModel:    def
__init__(self):        self.models = {}
self.stacking_model = None
self.performance_metrics = {}
self.training_times = {}
self.testing_times = {}
self.feature_importance = None

  def split_data(self, df, target_column, test_size=0.25, random_state=42):
      """
      Split data into train and test sets (75:25 ratio)
      """
      print(f"\n{' DATA SPLITTING ':=^60}")

      X = df.drop(columns=[target_column])
y = df[target_column]

      # Split into 75% train, 25% test
      X_train, X_test, y_train, y_test = train_test_split(
          X, y, test_size=test_size, random_state=random_state, stratify=y
      )
```

```
    print(f"Dataset split completed:")    print(f"  - Training set: {X_train.shape[0]} samples
  ({X_train.shape[0]/len(X)*100:.1f}%)") print(f"  - Testing set: {X_test.shape[0]} samples
  ({X_test.shape[0]/len(X)*100:.1f}%)") print(f"  - Total features: {X_train.shape[1]}")

    # Show class distribution in each set
self._show_class_distribution(y_train, y_test)
return X_train, X_test, y_train, y_test

  def _show_class_distribution(self, y_train, y_test):
"""Show class distribution across splits"""
print(f"\nClass Distribution:")
    print(f'{'Set':<12} {'Class 0':<10} {'Class 1':<10} {'Total':<10}")       print(f'{'-
'*45}")

    for y_set, set_name in zip([y_train, y_test], ['Train', 'Test']):
class_0 = np.sum(y_set == 0)          class_1 = np.sum(y_set ==
1)
        total = len(y_set)          print(f"{set_name:<12} {class_0:<10}
{class_1:<10} {total:<10}")

  def build_base_models(self):
    """
    Build individual base models with optimized parameters
    """
    print(f"\n{' BUILDING BASE MODELS ':=^60}")

    self.models = {
      'Decision Tree':
DecisionTreeClassifier(            max_depth=
8,          min_samples_split=10,
min_samples_leaf=4,
          random_state=42
        ),
        'SVM':
SVC(            kernel='rbf',
C=0.8,
gamma='scale',
probability=True,
          random_state=42
```

```
        ),
        'Logistic Regression': LogisticRegression(
            C=0.8,
max_iter=1000,
random_state=42,
            solver='liblinear'
        ),

        'Random Forest':
RandomForestClassifier(
n_estimators=150,              max_depth=12,
                min_samples_split=8,
min_samples_leaf=2,
            random_state=42
        )
    }

    print("Base    models    initialized:")
for  name,  model  in  self.models.items():
print(f"  ✓  {name}")

  def build_stacking_ensemble(self):
    """
    Build stacking ensemble with optimized meta-classifier
    """
    print(f"\n{' BUILDING STACKING ENSEMBLE ':=^60}")

    # Define base estimators
base_estimators = [
        ('dt', self.models['Decision Tree']),
        ('svm', self.models['SVM']),
        ('lr', self.models['Logistic Regression']),
        ('rf', self.models['Random Forest'])
    ]

    # Meta-classifier - optimized for better performance
meta_classifier = RandomForestClassifier(
        n_estimators=200,
max_depth=10,
```

```
min_samples_split=5,
min_samples_leaf=1,
        random_state=42
    )

    # Create stacking classifier
self.stacking_model =
StackingClassifier(          estimators=base_estima
tors,          final_estimator=meta_classifier,
        cv=5,
        passthrough=True,  # Include original features
        n_jobs=-1
    )

    print("Stacking ensemble configured:")
    print("  - Base learners: Decision Tree, SVM, Logistic Regression, Random Forest")
    print("  - Meta-learner: Optimized Random Forest")
    print("  - Cross-validation: 5-fold")  print("  - Passthrough:
    Enabled (includes original features)")

  def train_models(self, X_train, y_train):
    """
    Train all models and measure training time
    """
    print(f"\n{' TRAINING MODELS ':=^60}")

    # Train individual models      for name, model in
self.models.items():          print(f"Training {name}...")
start_time = time.time()          model.fit(X_train, y_train)
training_time = time.time() - start_time
self.training_times[name] = training_time          print(f"  ✓
{name} trained in {training_time:.4f} seconds")

    # Train stacking ensemble
print("Training Stacking Ensemble...")
start_time = time.time()
self.stacking_model.fit(X_train, y_train)
training_time = time.time() - start_time
    self.training_times['Stacking Ensemble'] = training_time      print(f"
✓ Stacking Ensemble trained in {training_time:.4f} seconds")
```

```
    def evaluate_models(self, X_test, y_test):
        """
        Evaluate all models on test set
        """
        print(f"\n{' MODEL EVALUATION ':=^60}")

        # Evaluate on test set        test_results =
self._evaluate_on_set(X_test, y_test, "Test")

        # Store results
self.performance_metrics = test_results
return self.performance_metrics

    def _evaluate_on_set(self, X, y, set_name):
        """
        Evaluate models on a specific dataset
        """
        print(f"\nEvaluating on {set_name} set...")

        results = {}  all_models = {**self.models, 'Stacking Ensemble':
        self.stacking_model}

        for name, model in all_models.items():
start_time = time.time()

            # Predictions
            y_pred = model.predict(X)            y_pred_proba = model.predict_proba(X)[:, 1] if
hasattr(model, 'predict_proba') else None

            testing_time = time.time() - start_time
self.testing_times[name] = testing_time

            # Calculate metrics
            accuracy = accuracy_score(y, y_pred)
            precision = precision_score(y, y_pred, zero_division=0)
recall = recall_score(y, y_pred, zero_division=0)            f1 =
f1_score(y, y_pred, zero_division=0)
```

```
            # Specificity (True Negative Rate)            tn, fp,
fn, tp = confusion_matrix(y, y_pred).ravel()
specificity = tn / (tn + fp) if (tn + fp) > 0 else 0

            # AUC-ROC            auc_roc = roc_auc_score(y, y_pred_proba) if
y_pred_proba is not None else 0

            results[name] =
{                   'accuracy': accuracy,
                'precision': precision,
                'recall': recall,
                'sensitivity': recall,  # Sensitivity is same as recall
                'specificity': specificity,
                'f1_score': f1,
                'auc_roc': auc_roc,
                'testing_time': testing_time,
                'confusion_matrix': confusion_matrix(y, y_pred),
                'y_true': y,
                'y_pred': y_pred,
                'y_pred_proba': y_pred_proba
            }

            print(f"  ✓ {name:<20}: Accuracy = {accuracy:.4f}, F1 = {f1:.4f}, AUC = {auc_roc:.4f}") return

        results


    def create_comprehensive_visualizations(self):
        """
        Create comprehensive visualizations for model performance
        """
        print(f"\n{' CREATING VISUALIZATIONS ':=^60}")

        # Create multiple visualization figures
self._create_performance_comparison_chart()
        self._create_roc_curves()
self._create_confusion_matrices()
self._create_training_time_comparison()
self._create_feature_importance_plot()
```

```
self._create_detailed_metrics_chart()
self._create_comprehensive_results_table()

   def _create_performance_comparison_chart(self):
"""Create performance comparison bar chart"""
       fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(20, 12))
       fig.suptitle('MODEL PERFORMANCE COMPARISON (Test Set)', fontsize=16,
fontweight='bold')

       models = list(self.performance_metrics.keys())        metrics = ['accuracy',
'precision', 'recall', 'f1_score']        metric_names = ['Accuracy', 'Precision',
'Recall (Sensitivity)', 'F1-Score']        colors = ['#FF6B6B', '#4ECDC4',
'#45B7D1', '#96CEB4', '#FECA57']

       for idx, (metric, metric_name) in enumerate(zip(metrics, metric_names)):
ax = [ax1, ax2, ax3, ax4][idx]            values =
[self.performance_metrics[model][metric] for model in models]

           bars = ax.bar(models, values, color=colors[:len(models)], alpha=0.8)
ax.set_ylabel(metric_name)
           ax.set_title(f'{metric_name} Comparison')
           ax.set_ylim(0, 1)
           ax.tick_params(axis='x', rotation=45)
ax.grid(axis='y', alpha=0.3)

           # Add value labels on bars
for bar, value in zip(bars, values):
               height = bar.get_height()
               ax.text(bar.get_x() + bar.get_width()/2., height + 0.01,
   f'{value:.4f}', ha='center', va='bottom', fontweight='bold', fontsize=9)

       plt.tight_layout()
       plt.savefig('model_performance_comparison.png', dpi=300, bbox_inches='tight')     plt.show()

   def _create_roc_curves(self):
       """Create ROC curves for all models"""
   plt.figure(figsize=(12, 8))

       models = list(self.performance_metrics.keys())        colors =
['#FF6B6B', '#4ECDC4', '#45B7D1', '#96CEB4', '#FECA57']
```

```
        for idx, model_name in enumerate(models):            if
self.performance_metrics[model_name]['y_pred_proba'] is not None:
                fpr, tpr, _ =
roc_curve(                    self.performance_metrics[model_name]['y_true'],
self.performance_metrics[model_name]['y_pred_proba']
                )
                auc_score = self.performance_metrics[model_name]['auc_roc']
plt.plot(fpr, tpr, label=f'{model_name} (AUC = {auc_score:.4f})',
color=colors[idx], linewidth=2.5)

        plt.plot([0, 1], [0, 1], 'k--', alpha=0.5, label='Random Classifier')
plt.xlabel('False Positive Rate (1 - Specificity)')         plt.ylabel('True
Positive Rate (Sensitivity)')
        plt.title('ROC Curves - Model Comparison', fontsize=14, fontweight='bold')
plt.legend(fontsize=10)         plt.grid(alpha=0.3)         plt.xlim([0, 1])
plt.ylim([0, 1])
        plt.savefig('roc_curves_comparison.png', dpi=300, bbox_inches='tight')
plt.show()

    def _create_confusion_matrices(self):
        """Create confusion matrices for all
models"""          n_models =
len(self.performance_metrics)         fig, axes =
plt.subplots(2, 3, figsize=(18, 10))
        fig.suptitle('CONFUSION MATRICES - Test Set', fontsize=16, fontweight='bold')
axes = axes.flatten()         models = list(self.performance_metrics.keys())

        for idx, model_name in enumerate(models):
if idx < len(axes):
                cm = self.performance_metrics[model_name]['confusion_matrix']
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
                        xticklabels=['Predicted 0', 'Predicted 1'],
        yticklabels=['Actual 0', 'Actual 1'],                    ax=axes[idx])

                accuracy = self.performance_metrics[model_name]['accuracy']
         axes[idx].set_title(f'{model_name}\nAccuracy: {accuracy:.4f}', fontweight='bold')

        # Remove empty subplots     for idx
    in range(n_models, len(axes)):
    fig.delaxes(axes[idx])
```

```
        plt.tight_layout()
        plt.savefig('confusion_matrices.png', dpi=300, bbox_inches='tight')
plt.show()

    def _create_training_time_comparison(self):
"""Create training and testing time comparison"""
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))
        fig.suptitle('MODEL TRAINING AND TESTING TIME COMPARISON', fontsize=16,
fontweight='bold')        models = list(self.training_times.keys())

        # Training times        train_times = [self.training_times[model]
for model in models]        bars1 = ax1.bar(models, train_times,
color='#FF6B6B', alpha=0.8)        ax1.set_ylabel('Time (seconds)')
ax1.set_title('Training Time Comparison')
ax1.tick_params(axis='x', rotation=45)        ax1.grid(axis='y',
alpha=0.3)

        # Testing times        test_times = [self.testing_times[model] for
model in models]        bars2 = ax2.bar(models, test_times,
color='#4ECDC4', alpha=0.8)        ax2.set_ylabel('Time (seconds)')
ax2.set_title('Testing Time Comparison')
ax2.tick_params(axis='x', rotation=45)        ax2.grid(axis='y',
alpha=0.3)

        # Add value labels        for bars, ax in zip([bars1, bars2],
[ax1, ax2]):            for bar in bars:                height =
bar.get_height()                ax.text(bar.get_x() + bar.get_width()/2.,
height + 0.001,                    f'{height:.3f}s', ha='center',
va='bottom', fontsize=8)

        plt.tight_layout()
        plt.savefig('training_testing_times.png', dpi=300, bbox_inches='tight')
    plt.show()

    def _create_feature_importance_plot(self):
        """Create feature importance plot from Random Forest"""
if 'Random Forest' in self.models:        rf_model =
self.models['Random Forest']        if hasattr(rf_model,
'feature_importances_'):                feature_importance =
rf_model.feature_importances_
```

```
            plt.figure(figsize=(12, 8))                indices =
np.argsort(feature_importance)[::-1]

            plt.bar(range(len(feature_importance)), feature_importance[indices], color='#45B7D1')
plt.xlabel('Feature Index')                plt.ylabel('Feature Importance')
            plt.title('Random Forest Feature Importance', fontweight='bold')
plt.grid(axis='y', alpha=0.3)
            plt.savefig('feature_importance.png', dpi=300, bbox_inches='tight')
plt.show()

   def _create_detailed_metrics_chart(self):
"""Create detailed metrics comparison chart"""
models = list(self.performance_metrics.keys())
      metrics = ['accuracy', 'precision', 'recall', 'specificity', 'f1_score', 'auc_roc']
      metric_names = ['Accuracy', 'Precision', 'Sensitivity\n(Recall)', 'Specificity', 'F1-Score', 'AUC-
ROC']

      fig, ax = plt.subplots(figsize=(16, 8))

      x = np.arange(len(metrics))        width = 0.15        colors =
['#FF6B6B', '#4ECDC4', '#45B7D1', '#96CEB4', '#FECA57']

      for i, model_name in enumerate(models):            values =
[self.performance_metrics[model_name][metric] for metric in metrics]            ax.bar(x
+ i*width, values, width, label=model_name, color=colors[i], alpha=0.8)

      ax.set_xlabel('Metrics', fontweight='bold')
ax.set_ylabel('Score', fontweight='bold')
      ax.set_title('Detailed Metrics Comparison - Test Set', fontsize=14, fontweight='bold')
ax.set_xticks(x + width*len(models)/2)        ax.set_xticklabels(metric_names)
      ax.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
ax.grid(axis='y', alpha=0.3)     ax.set_ylim(0, 1)

      # Add value labels on bars for i,
      model_name in enumerate(models):
      values =
      [self.performance_metrics[model_name]
      [metric] for metric in metrics]    for j,
      value in enumerate(values):
      ax.text(j + i*width, value + 0.01,
      f'{value:.4f}',              ha='center',
```

```
        va='bottom', fontsize=8,
        fontweight='bold')

        plt.tight_layout()
        plt.savefig('detailed_metrics_comparison.png', dpi=300, bbox_inches='tight')
    plt.show()

    def _create_comprehensive_results_table(self):
        """Create a comprehensive results table visualization"""
models = list(self.performance_metrics.keys())
        metrics = ['accuracy', 'precision', 'recall', 'specificity', 'f1_score', 'auc_roc']
metric_names = ['Accuracy', 'Precision', 'Sensitivity', 'Specificity', 'F1-Score', 'AUC-ROC']

        # Create data for
table         table_data = []
for model in models:
            row = [model]
for metric in metrics:
                row.append(f"{self.performance_metrics[model][metric]:.4f}")
table_data.append(row)

        # Create figure and axis
        fig, ax = plt.subplots(figsize=(14, 6))
        ax.axis('tight')
ax.axis('off')

        # Create table
        table = ax.table(cellText=table_data,
colLabels=['Model'] + metric_names,
                         cellLoc='center',
loc='center',
bbox=[0, 0, 1, 1])

        # Style the table
        table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1, 1.5)

        # Color the header         for i in
range(len(metric_names) + 1):
```

```
            table[(0, i)].set_facecolor('#4ECDC4')
            table[(0, i)].set_text_props(weight='bold', color='white')

        # Highlight stacking ensemble row
        for i, row in enumerate(table_data):
            if row[0] == 'Stacking Ensemble':            for
    j in range(len(row)):            table[(i+1,
    j)].set_facecolor('#FECA57')
                    table[(i+1, j)].set_text_props(weight='bold')

        plt.title('COMPREHENSIVE MODEL PERFORMANCE COMPARISON',
fontsize=16, fontweight='bold', pad=20)
        plt.savefig('comprehensive_results_table.png', dpi=300, bbox_inches='tight')
plt.show()

    def generate_comprehensive_report(self):
        """
        Generate comprehensive performance report
        """
        print(f"\n{' COMPREHENSIVE PERFORMANCE REPORT ':=^80}")

        # Test set performance
        print(f"\n📊 TEST SET PERFORMANCE:")
        print(f"{'Model':<25} {'Accuracy':<10} {'Precision':<10} {'Sensitivity':<10} {'Specificity':<10}
{'F1-Score':<10} {'AUC-ROC':<10}")
        print(f"{'-'*95}")

        for model_name, metrics in self.performance_metrics.items():
print(f"{model_name:<25} {metrics['accuracy']:.4f}      {metrics['precision']:.4f}      "
                f"{metrics['sensitivity']:.4f}      {metrics['specificity']:.4f}      "
f"{metrics['f1_score']:.4f}      {metrics['auc_roc']:.4f}")

        # Training and testing times
        print(f"\n⏱️ COMPUTATIONAL PERFORMANCE:")
        print(f"{'Model':<25} {'Training Time (s)':<15} {'Testing Time (s)':<15}")
print(f"{'-'*60}")

        for model_name in self.training_times.keys():            train_time =
self.training_times[model_name]            test_time =
```

```
self.testing_times.get(model_name, 0)          print(f"{model_name:<25}
{train_time:.4f}         {test_time:.4f}")

      # Best performing model
      best_model = max(self.performance_metrics.items(),
key=lambda x: x[1]['accuracy'])

      print(f"\n🏆 BEST PERFORMING MODEL: {best_model[0]}")
print(f"    Accuracy:  {best_model[1]['accuracy']:.4f}") print(f"    F1-
Score:   {best_model[1]['f1_score']:.4f}")   print(f"         AUC-ROC:
{best_model[1]['auc_roc']:.4f}")
      print(f"   Sensitivity: {best_model[1]['sensitivity']:.4f}")
      print(f"   Specificity: {best_model[1]['specificity']:.4f}")

      # Verify stacking model performance      stacking_metrics =
  self.performance_metrics['Stacking Ensemble']
      print(f"\n✅ STACKING MODEL PERFORMANCE VERIFICATION:")
      print(f"   Accuracy: {stacking_metrics['accuracy']:.4f} {'✓' if stacking_metrics['accuracy'] >=
0.9182 else '✗'}")       print(f"
Required: ≥ 0.9182")

      # Model comparison
      print(f"\n📈 MODEL COMPARISON INSIGHTS:")
      base_models = [m for m in self.performance_metrics.keys() if m != 'Stacking Ensemble']
stacking_better = 0       total_metrics = 0

      for metric in ['accuracy', 'precision', 'sensitivity', 'specificity', 'f1_score', 'auc_roc']:
stacking_value = stacking_metrics[metric]          base_avg =
np.mean([self.performance_metrics[m][metric] for m in base_models])          if
stacking_value > base_avg:              stacking_better += 1          total_metrics += 1

      print(f"   Stacking ensemble outperforms base models in {stacking_better}/{total_metrics} metrics")
      print(f"   Average improvement over base models: {((stacking_metrics['accuracy'] - base_avg) /
base_avg * 100):.2f}%")

      # Model interpretation
      print(f"\n📈 MODEL INTERPRETATION:")
      print(f"• Stacking Ensemble combines predictions from all base models for superior performance")
print(f"• Random Forest provides robust feature importance and handles non-linear relationships")
print(f"• Logistic Regression offers good interpretability and calibration")       print(f"• SVM captures
```

```
complex patterns but may be computationally intensive")        print(f"• Decision Tree provides fast predictions and good interpretability")

# Main execution function def run_ensemble_modeling(preprocessed_file, target_column):
    """
    Run complete ensemble modeling pipeline
    """
    print(f"{' ENSEMBLE MODELING PIPELINE ':=^60}")

    # Load preprocessed data
    print(f"\n📥 Loading preprocessed data from {preprocessed_file}")
df = pd.read_csv(preprocessed_file)

    print(f"Dataset shape: {df.shape}")
    print(f"Target column: {target_column}")

    # Initialize ensemble model
   ensemble = EnsembleModel()

    # Split data (75:25 ratio)
    X_train, X_test, y_train, y_test =
ensemble.split_data(        df, target_column,
test_size=0.25
    )

    # Build models
ensemble.build_base_models()
ensemble.build_stacking_ensemble()

    # Train models
ensemble.train_models(X_train, y_train)

    # Evaluate models
ensemble.evaluate_models(X_test, y_test)

    # Create visualizations
ensemble.create_comprehensive_visualizations()
```

```
    # Generate report
ensemble.generate_comprehensive_report()
return ensemble

# Run the ensemble modeling
if __name__ == "__main__":
    # Configuration     preprocessed_file =
"preprocessed_heart_disease.csv"     target_column =
"target"  # Adjust based on your dataset

    try:
        # Run ensemble modeling         ensemble_model =
run_ensemble_modeling(preprocessed_file, target_column)

        print(f"\n{'🎉 ENSEMBLE MODELING COMPLETED SUCCESSFULLY! '=^60}")
print(f"Generated files:")
        print(f"  📊 model_performance_comparison.png")
print(f"  📈 roc_curves_comparison.png")        print(f"
🎯 confusion_matrices.png") print(f"  ⏱️
training_testing_times.png") print(f"  🔍
feature_importance.png")
        print(f"  📋 detailed_metrics_comparison.png")
        print(f"  📋 comprehensive_results_table.png")

    except Exception as e:      print(f"\n❌ Error in
    ensemble modeling: {e}")
        import traceback
    traceback.print_exc()
```

APPENDIX III

RFE-Stack ensemble Model import pandas as pd import numpy as np
import matplotlib.pyplot as plt import seaborn as sns from
sklearn.model_selection import train_test_split, cross_val_score from
sklearn.preprocessing import StandardScaler, LabelEncoder from
sklearn.ensemble import RandomForestClassifier, StackingClassifier
from sklearn.tree import DecisionTreeClassifier from sklearn.svm import
SVC
from sklearn.linear_model import LogisticRegression from sklearn.metrics import
(accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score, roc_curve, confusion_matrix, classification_report) from xgboost
import XGBClassifier import warnings warnings.filterwarnings('ignore')

# Set style for better visualizations
plt.style.use('default') sns.set_palette("husl")

print("=" * 80)
print("STACKING ENSEMBLE MODEL FOR CARDIAC ARREST PREDICTION")
print("=" * 80)

Load the feature selected dataset
print("\n LOADING FEATURE SELECTED DATASET...") df
= pd.read_csv("/content/feature_selected_by_RFE_FA.csv")

print(f'Dataset Shape: {df.shape}") print(f"Features:
{df.shape[1] - 1}, Samples: {df.shape[0]}")

# Identify target column
target_columns = ['target', 'heart_disease', 'disease', 'class', 'output']
target_col = None for col in target_columns: if col in
df.columns: target_col = col break if target_col is None:
target_col = df.columns[-1] print(f"Target column: '{target_col}'")

Separate features and target

X = df.drop(columns=[target_col])
y = df[target_col]

print(f"\nSelected Features: {list(X.columns)}")
print(f"Target variable distribution:\n{y.value_counts()}")

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

print(f"\nTraining set: {X_train.shape}")
print(f"Testing set: {X_test.shape}")

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("\n🔄 STAGE 1: INDIVIDUAL BASE MODEL TRAINING AND EVALUATION")
print("=" * 60)

# Define base models
base_models = [
    ('decision_tree', DecisionTreeClassifier(random_state=42, max_depth=5)),
    ('svm', SVC(random_state=42, probability=True)),
    ('logistic_regression', LogisticRegression(random_state=42, max_iter=1000)),
    ('random_forest', RandomForestClassifier(random_state=42, n_estimators=100)),
('xgboost', XGBClassifier(random_state=42, eval_metric='logloss')) ]

# Meta learner meta_learner = RandomForestClassifier(random_state=42,
n_estimators=100)

# Store results
results = [] model_predictions = {} model_probabilities
= {} confusion_matrices = {} print("Training and
evaluating individual base models...")

# Train and evaluate each base model
for name, model in base_models:
    print(f"\n--- Training {name.upper()} ---")

    # Handle scaling requirements     if
name in ['svm', 'logistic_regression']:
        X_train_model = X_train_scaled
X_test_model = X_test_scaled     else:
```

```
        X_train_model = X_train
        X_test_model = X_test

    # Train model
model.fit(X_train_model, y_train)

    # Predictions     y_pred =
model.predict(X_test_model)

    # Get probabilities - FIXED: Handle all models properly
try:
        y_pred_proba = model.predict_proba(X_test_model)[:, 1]
except AttributeError:
        # For models without predict_proba, use decision function and normalize
y_pred_proba = model.decision_function(X_test_model)
                    y_pred_proba = (y_pred_proba - y_pred_proba.min()) / (y_pred_proba.max() -
            y_pred_proba.min())

    # Store predictions for stacking
model_predictions[name] = y_pred
model_probabilities[name] = y_pred_proba

    # Calculate confusion matrix      cm =
confusion_matrix(y_test, y_pred)
confusion_matrices[name] = cm

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)     precision =
precision_score(y_test, y_pred, zero_division=0)     sensitivity =
recall_score(y_test, y_pred)  # Sensitivity = Recall     specificity
= recall_score(1-y_test, 1-y_pred)  # Specificity     f1 =
f1_score(y_test, y_pred)     auc = roc_auc_score(y_test,
y_pred_proba)

    # Calculate additional metrics from confusion matrix
tn, fp, fn, tp = cm.ravel()     false_positive_rate = fp / (fp
+ tn) if (fp + tn) > 0 else 0     false_negative_rate = fn /
(fn + tp) if (fn + tp) > 0 else 0
```

```
    # Store results
results.append({        'Model':
name.upper(),
        'Accuracy': accuracy,
'Precision': precision,
        'Sensitivity': sensitivity,
        'Specificity': specificity,
        'F1-Score': f1,
        'AUC-ROC': auc,
        'True_Positive': tp,
        'True_Negative': tn,
        'False_Positive': fp,
        'False_Negative': fn,
        'False_Positive_Rate': false_positive_rate,
        'False_Negative_Rate': false_negative_rate
    })

    print(f"Accuracy: {accuracy:.4f}")
print(f"F1-Score: {f1:.4f}")     print(f"AUC-
ROC: {auc:.4f}")     print(f"Confusion
Matrix:\n{cm}")

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

print("\n📊 INDIVIDUAL MODEL PERFORMANCE COMPARISON")
print("=" * 50)
display_columns = ['Model', 'Accuracy', 'Precision', 'Sensitivity', 'Specificity', 'F1-Score', 'AUC-ROC']
print(results_df[display_columns].round(4))

#
        ==========================================================================
        =========
# INDIVIDUAL MODEL VISUALIZATIONS - SEPARATED FOR BETTER VISIBILITY
#
        ==========================================================================
        =========

print("\n📈 STAGE 2: INDIVIDUAL MODEL VISUALIZATIONS")
print("=" * 50)
```

```
# 1. Performance metrics comparison - SEPARATE PLOT plt.figure(figsize=(15,
8))
metrics = ['Accuracy', 'Precision', 'Sensitivity', 'Specificity', 'F1-Score']
x_pos = np.arange(len(metrics)) width = 0.15 for i, (idx, row) in
enumerate(results_df.iterrows()):

   plt.bar(x_pos + i*width,
         [row['Accuracy'], row['Precision'], row['Sensitivity'],
row['Specificity'], row['F1-Score']],            width,
label=row['Model'], alpha=0.8)

plt.xlabel('Metrics') plt.ylabel('Score')
plt.title('INDIVIDUAL MODEL PERFORMANCE COMPARISON', fontsize=14, fontweight='bold')
plt.xticks(x_pos + width*2, metrics)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True, alpha=0.3) plt.tight_layout() plt.show()

# 2. AUC-ROC comparison - SEPARATE PLOT
plt.figure(figsize=(12, 6))
plt.bar(results_df['Model'], results_df['AUC-ROC'], color='lightgreen', alpha=0.7)
plt.xlabel('Models') plt.ylabel('AUC-ROC Score')
plt.title('AUC-ROC SCORES BY MODEL', fontsize=14, fontweight='bold')
plt.xticks(rotation=45) plt.grid(True, alpha=0.3) plt.tight_layout() plt.show()

# 3. F1-Score comparison - SEPARATE PLOT plt.figure(figsize=(12,
6))
plt.bar(results_df['Model'], results_df['F1-Score'], color='lightcoral', alpha=0.7)
plt.xlabel('Models') plt.ylabel('F1-Score')
plt.title('F1-SCORES BY MODEL', fontsize=14, fontweight='bold')
plt.xticks(rotation=45) plt.grid(True, alpha=0.3) plt.tight_layout()
plt.show()

# 4. Confusion Matrices - SEPARATE PLOTS FOR EACH MODEL
print("\n CONFUSION MATRICES FOR INDIVIDUAL MODELS")
print("=" * 50)

model_names = ['decision_tree', 'svm', 'logistic_regression', 'random_forest', 'xgboost']
for i, name in enumerate(model_names):
```

```
   plt.figure(figsize=(8, 6))     cm =
confusion_matrices[name]     sns.heatmap(cm,
annot=True, fmt='d', cmap='Blues',
xticklabels=['Predicted 0', 'Predicted 1'],
yticklabels=['Actual 0', 'Actual 1'],
cbar_kws={'shrink': 0.8})
   plt.title(f'CONFUSION MATRIX - {name.upper()}\nTN:{cm[0,0]} FP:{cm[0,1]} | FN:{cm[1,0]}
          TP:{cm[1,1]}',
          fontsize=12, fontweight='bold')
plt.tight_layout()     plt.show()

# 5. ROC Curves for individual models - SEPARATE PLOT
plt.figure(figsize=(12, 8)) for name in model_names:     if
name in ['svm', 'logistic_regression']:         X_test_model =
X_test_scaled     else:
      X_test_model = X_test

   # Get the trained model     model_obj = next(m[1] for m in
base_models if m[0] == name)

   # Get probabilities safely
try:
      y_pred_proba = model_obj.predict_proba(X_test_model)[:, 1]
except AttributeError:
      # For models without predict_proba, use decision function
      y_pred_proba = model_obj.decision_function(X_test_model)
                  y_pred_proba = (y_pred_proba - y_pred_proba.min()) / (y_pred_proba.max() -
          y_pred_proba.min())

   fpr, tpr, _ = roc_curve(y_test, y_pred_proba)     auc_score = roc_auc_score(y_test,
y_pred_proba)     plt.plot(fpr, tpr, linewidth=2.5, label=f'{name.upper()} (AUC =
{auc_score:.3f})')

plt.plot([0, 1], [0, 1], 'k--', linewidth=2, label='Random Classifier', alpha=0.7)
plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.title('ROC CURVES - INDIVIDUAL MODELS', fontsize=14, fontweight='bold')
plt.legend(fontsize=10) plt.grid(True, alpha=0.3) plt.tight_layout() plt.show()

print("\n🔄 STAGE 3: STACKING ENSEMBLE MODEL")
```

```
print("=" * 45) print("Building Stacking

Ensemble Model...") # Create stacking

classifier stacking_model =

StackingClassifier(     estimators=base_mode

ls,     final_estimator=meta_learner,

    cv=5,
    passthrough=False )

# Train stacking model stacking_model.fit(X_train,
y_train)

# Predictions from stacking model y_pred_stack =
stacking_model.predict(X_test) y_pred_proba_stack =
stacking_model.predict_proba(X_test)[:, 1]

# Calculate confusion matrix for stacking cm_stack
= confusion_matrix(y_test, y_pred_stack)
confusion_matrices['stacking'] = cm_stack

# Calculate metrics for stacking model
stack_accuracy = accuracy_score(y_test, y_pred_stack)
stack_precision = precision_score(y_test, y_pred_stack, zero_division=0)
stack_sensitivity = recall_score(y_test, y_pred_stack) stack_specificity =
recall_score(1-y_test, 1-y_pred_stack) stack_f1 = f1_score(y_test,
y_pred_stack) stack_auc = roc_auc_score(y_test, y_pred_proba_stack)

# Calculate additional metrics from confusion matrix tn_stack, fp_stack, fn_stack, tp_stack =
cm_stack.ravel() false_positive_rate_stack = fp_stack / (fp_stack + tn_stack) if (fp_stack +
tn_stack) > 0 else 0 false_negative_rate_stack = fn_stack / (fn_stack + tp_stack) if (fn_stack
+ tp_stack) > 0 else 0

# Add stacking results to dataframe
stacking_results = {
    'Model': 'STACKING ENSEMBLE',
    'Accuracy': stack_accuracy,
'Precision': stack_precision,
```

```
    'Sensitivity': stack_sensitivity,
    'Specificity': stack_specificity,
    'F1-Score': stack_f1,
    'AUC-ROC': stack_auc,
    'True_Positive': tp_stack,
    'True_Negative': tn_stack,
    'False_Positive': fp_stack,
    'False_Negative': fn_stack,
    'False_Positive_Rate': false_positive_rate_stack,

    'False_Negative_Rate': false_negative_rate_stack
} results_df = pd.concat([results_df, pd.DataFrame([stacking_results])],

ignore_index=True)

print("\n📊 STACKING ENSEMBLE PERFORMANCE")
print("=" * 40)
stacking_performance = results_df[results_df['Model'] == 'STACKING ENSEMBLE']
print(stacking_performance[display_columns].round(4)) print(f"\nConfusion Matrix
for Stacking Ensemble:") print(cm_stack)

#
        ================================================================================
        =========
# STACKING ENSEMBLE VISUALIZATIONS - SEPARATED
#
        ================================================================================
        =========

print("\n📈 STAGE 4: STACKING ENSEMBLE VISUALIZATIONS")
print("=" * 50)

# 1. Stacking Confusion Matrix - SEPARATE PLOT plt.figure(figsize=(8,
6))
sns.heatmap(cm_stack, annot=True, fmt='d', cmap='Greens',
xticklabels=['Predicted 0', 'Predicted 1'],
yticklabels=['Actual 0', 'Actual 1'],
cbar_kws={'shrink': 0.8})
```

```
plt.title('CONFUSION MATRIX - STACKING ENSEMBLE\nTN:{} FP:{} | FN:{}
TP:{}'.format(     cm_stack[0,0], cm_stack[0,1], cm_stack[1,0], cm_stack[1,1]),     fontsize=12,
fontweight='bold') plt.tight_layout() plt.show()

# 2. Stacking ROC Curve - SEPARATE PLOT
plt.figure(figsize=(10, 8))
fpr_stack, tpr_stack, _ = roc_curve(y_test, y_pred_proba_stack)
plt.plot(fpr_stack, tpr_stack, 'b-', linewidth=3, label=f'STACKING ENSEMBLE (AUC =
        {stack_auc:.3f})')
plt.plot([0, 1], [0, 1], 'k--', linewidth=2, label='Random Classifier', alpha=0.7)
plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.title('ROC CURVE - STACKING ENSEMBLE', fontsize=14, fontweight='bold')
plt.legend(fontsize=12) plt.grid(True, alpha=0.3)

plt.tight_layout() plt.show()

#
        ================================================================================
        =========
# COMPREHENSIVE COMPARISON VISUALIZATIONS - SEPARATED
#
        ================================================================================
        =========

print("\n📊 STAGE 5: COMPREHENSIVE MODEL COMPARISON")
print("=" * 50)

# 1. Radar Chart Comparison - SEPARATE PLOT plt.figure(figsize=(10, 10))
categories = ['Accuracy', 'Precision', 'Sensitivity', 'Specificity', 'F1-Score', 'AUC-ROC']

def normalize_radar_data(data):
return [d * 100 for d in data]

angles = np.linspace(0, 2*np.pi, len(categories), endpoint=False).tolist()
angles += angles[:1]  # Complete the circle

fig, ax = plt.subplots(figsize=(10, 10), subplot_kw=dict(projection='polar'))
colors = ['red', 'blue', 'green', 'orange', 'purple', 'brown', 'black']
```

```
for idx, row in results_df.iterrows():     values =
normalize_radar_data([         row['Accuracy'],
row['Precision'], row['Sensitivity'],         row['Specificity'],
row['F1-Score'], row['AUC-ROC']
   ])
   values += values[:1]  # Complete the circle
   ax.plot(angles, values, 'o-', linewidth=2, label=row['Model'], color=colors[idx])
ax.fill(angles, values, alpha=0.1, color=colors[idx])

ax.set_xticks(angles[:-1])
ax.set_xticklabels(categories)
ax.set_ylim(0, 100)
plt.title('COMPREHENSIVE MODEL COMPARISON\n(RADAR CHART)', size=16,
          fontweight='bold', pad=20)
plt.legend(bbox_to_anchor=(1.2, 1), loc='upper left', fontsize=10)
plt.tight_layout() plt.show()

# 2. Stacking vs Best Base Model - SEPARATE PLOT

plt.figure(figsize=(10, 6))
best_base_f1 = results_df[results_df['Model'] != 'STACKING ENSEMBLE']['F1-Score'].max()
stacking_f1 = stacking_results['F1-Score'] improvement = ((stacking_f1 - best_base_f1) /
best_base_f1) * 100

models_compare = ['Best Base Model', 'Stacking Ensemble']
f1_scores_compare = [best_base_f1, stacking_f1]

bars = plt.bar(models_compare, f1_scores_compare, color=['lightblue', 'lightgreen'], alpha=0.7)
plt.ylabel('F1-Score', fontsize=12)
plt.title(f'STACKING vs BEST BASE MODEL\nImprovement: {improvement:.2f}%', fontsize=14,
fontweight='bold') plt.grid(True, alpha=0.3)

# Add value labels on bars for bar, value in zip(bars,
f1_scores_compare):     plt.text(bar.get_x() + bar.get_width()/2,
bar.get_height() + 0.01,
          f'{value:.4f}', ha='center', va='bottom', fontsize=12, fontweight='bold')
plt.tight_layout() plt.show()

# 3. All Models ROC Curves - SEPARATE PLOT
plt.figure(figsize=(12, 10)) # Individual models
for name in model_names:     if name in ['svm',
```

```
'logistic_regression']:         X_test_model =
X_test_scaled     else:
        X_test_model = X_test     model_obj = next(m[1] for m in

base_models if m[0] == name)

   # Get probabilities safely
try:
        y_pred_proba = model_obj.predict_proba(X_test_model)[:, 1]
except AttributeError:
        # For models without predict_proba, use decision function
        y_pred_proba = model_obj.decision_function(X_test_model)
                    y_pred_proba = (y_pred_proba - y_pred_proba.min()) / (y_pred_proba.max() -
            y_pred_proba.min())

   fpr, tpr, _ = roc_curve(y_test, y_pred_proba)     auc_score = roc_auc_score(y_test,
y_pred_proba)     plt.plot(fpr, tpr, alpha=0.7, linewidth=2, label=f'{name.upper()} (AUC =
{auc_score:.3f})') # Stacking model

plt.plot(fpr_stack, tpr_stack, 'k-', linewidth=4,
label=f'STACKING ENSEMBLE (AUC = {stack_auc:.3f})')

plt.plot([0, 1], [0, 1], 'k--', alpha=0.5, linewidth=2, label='Random Classifier')
plt.xlabel('False Positive Rate', fontsize=12) plt.ylabel('True Positive Rate',
fontsize=12)
plt.title('ROC CURVES - ALL MODELS', fontsize=14, fontweight='bold')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', fontsize=10)
plt.grid(True, alpha=0.3) plt.tight_layout() plt.show()

# 4. Error Rates Comparison - SEPARATE PLOT plt.figure(figsize=(12,
8)) fpr_rates = [row['False_Positive_Rate'] for _, row in
results_df.iterrows()] fnr_rates = [row['False_Negative_Rate'] for _, row
in results_df.iterrows()] models = results_df['Model']

for i, (fpr, fnr, model) in enumerate(zip(fpr_rates, fnr_rates, models)):
plt.scatter(fpr, fnr, s=150, label=model, alpha=0.7)     plt.annotate(model, (fpr, fnr),
xytext=(8, 8), textcoords='offset points', fontsize=10)

plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('False Negative Rate', fontsize=12)
```

```
plt.title('ERROR RATES COMPARISON\n(Lower Left = Better Performance)', fontsize=14,
fontweight='bold') plt.legend(fontsize=10) plt.grid(True, alpha=0.3) plt.tight_layout() plt.show()

# 5. True Positive vs True Negative - SEPARATE PLOT
plt.figure(figsize=(12, 8)) tp_counts = [row['True_Positive'] for _,
row in results_df.iterrows()] tn_counts = [row['True_Negative'] for _,
row in results_df.iterrows()] models = results_df['Model']

x_pos = np.arange(len(models))
width = 0.35

plt.bar(x_pos - width/2, tp_counts, width, label='True Positives', alpha=0.7, color='green')
plt.bar(x_pos + width/2, tn_counts, width, label='True Negatives', alpha=0.7, color='blue')

plt.xlabel('Models', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('TRUE POSITIVE vs TRUE NEGATIVE COUNTS', fontsize=14, fontweight='bold')
plt.xticks(x_pos, models, rotation=45) plt.legend(fontsize=10) plt.grid(True, alpha=0.3) plt.tight_layout()
plt.show()

# 6. Model Ranking by F1-Score - SEPARATE PLOT plt.figure(figsize=(12,
8))
ranked_models = results_df.sort_values('F1-Score', ascending=True)
plt.barh(ranked_models['Model'], ranked_models['F1-Score'], color='lightcoral', alpha=0.7)
plt.xlabel('F1-Score', fontsize=12)
plt.title('MODEL RANKING BY F1-SCORE', fontsize=14, fontweight='bold') plt.grid(True,
alpha=0.3)

# Add value labels for i, (value, model) in enumerate(zip(ranked_models['F1-Score'],
ranked_models['Model'])):
    plt.text(value + 0.01, i, f'{value:.4f}', va='center', fontsize=10, fontweight='bold')
plt.tight_layout() plt.show()

# 7. Performance Metrics Table - SEPARATE DISPLAY
print("\n📋 PERFORMANCE METRICS SUMMARY TABLE")
print("=" * 50)
display_columns_extended = ['Model', 'Accuracy', 'Precision', 'Sensitivity', 'Specificity', 'F1-Score',
            'AUC-ROC', 'False_Negative']
print(results_df[display_columns_extended].round(4).to_string(index=False))
```

```
print("\n🎯 CONFUSION MATRIX INTERPRETATION FOR BEGINNERS")
print("=" * 55)

print("""
📊 UNDERSTANDING CONFUSION MATRICES:

For each model's confusion matrix:

   [ TN  FP ]   ← Actual Negative Class
[ FN  TP ]   ← Actual Positive Class

Where:
• TN (True Negative): Correctly predicted non-cardiac arrest cases
• FP (False Positive): Wrongly predicted cardiac arrest (False Alarm)
• FN (False Negative): Missed cardiac arrest cases (Dangerous!)
• TP (True Positive): Correctly predicted cardiac arrest cases

🔍 MEDICAL INTERPRETATION:

✅ IDEAL SCENARIO:
• High TP: Catch most actual cardiac arrest cases
• High TN: Correctly identify healthy patients
• Low FP: Avoid unnecessary treatments/scares • Low FN: Don't miss actual cardiac arrest cases

⚠️ CLINICAL IMPORTANCE:
• False Negatives are CRITICAL in cardiac arrest prediction
• Missing a cardiac arrest case can be life-threatening
• False Positives cause unnecessary stress but are less dangerous

📈 WHAT TO LOOK FOR:
1. High diagonal values (TN and TP)
2. Low off-diagonal values (FP and FN)
3. Especially watch for low FN values""")

# Print detailed confusion matrix analysis
print("\n📋 DETAILED CONFUSION MATRIX ANALYSIS:")
print("=" * 45) for _, row in results_df.iterrows():
model_name = row['Model']
```

```
                    tp, tn, fp, fn = row['True_Positive'], row['True_Negative'], row['False_Positive'],
        row['False_Negative']
    total = tp + tn + fp + fn

    print(f"\n{model_name}:")     print(f"  True Positives: {tp} ({tp/total*100:.1f}%) -
Correct cardiac arrest predictions")     print(f"  True Negatives: {tn} ({tn/total*100:.1f}%)
- Correct healthy predictions")     print(f"  False Positives: {fp} ({fp/total*100:.1f}%) -
False alarms")     print(f"  False Negatives: {fn} ({fn/total*100:.1f}%) - Missed cardiac
arrest cases")

print("\n🎯 FINAL RESULTS AND INTERPRETATION")
print("=" * 50)

# Find best performing models
best_base_model = results_df[results_df['Model'] != 'STACKING ENSEMBLE'].nlargest(1, 'F1-
        Score').iloc[0] best_overall_model =
results_df.nlargest(1, 'F1-Score').iloc[0]

print(f"🏆 BEST BASE MODEL: {best_base_model['Model']}") print(f"
F1-Score: {best_base_model['F1-Score']:.4f}") print(f"   Accuracy:
{best_base_model['Accuracy']:.4f}") print(f"   False Negatives:
{best_base_model['False_Negative']}") print(f"\n🏆 BEST OVERALL
MODEL: {best_overall_model['Model']}") print(f"   F1-Score:
{best_overall_model['F1-Score']:.4f}") print(f"   Accuracy:
{best_overall_model['Accuracy']:.4f}") print(f"   False Negatives:
{best_overall_model['False_Negative']}")

improvement_f1                =                ((best_overall_model['F1-Score']                -
            best_base_model['F1-Score'])          / best_base_model['F1-Score']) * 100
print(f"\n📈 IMPROVEMENT: {improvement_f1:.2f}% F1-Score improvement with Stacking")

print("""
📖 INTERPRETATION FOR BEGINNERS:

🔍 WHAT IS STACKING ENSEMBLE?
• Stacking combines multiple machine learning models to create a superior model
• Base Models: Decision Tree, SVM, Logistic Regression, Random Forest, XGBoost
• Meta-Learner: Random Forest that learns to combine base model predictions
```

🎯 KEY INSIGHTS FROM CONFUSION MATRICES:

✅ WHY STACKING WORKS:
• Reduces both False Positives and False Negatives
• Combines strengths of different algorithms
• Provides more reliable predictions for critical medical decisions

💡 CLINICAL RECOMMENDATION:
The model with the lowest False Negative rate is most suitable for cardiac arrest prediction, as missing actual cases can have serious consequences. """)

```
# Save results
print(f"\n💾 SAVING RESULTS...")
results_df.to_csv("stacking_ensemble_results.csv", index=False)
print("✅ Results saved as 'stacking_ensemble_results.csv'")

# Save confusion matrices
confusion_matrices_df = pd.DataFrame({
    'Model': list(confusion_matrices.keys()),
    'Confusion_Matrix': [str(cm) for cm in confusion_matrices.values()]
}) confusion_matrices_df.to_csv("confusion_matrices.csv",
index=False) print("✅ Confusion matrices saved as
'confusion_matrices.csv'")

print(f"\n📁 FILES CREATED:")
print(f"   1. stacking_ensemble_results.csv - Complete performance metrics")
print(f"   2. confusion_matrices.csv - All confusion matrices")
print(f"   3. Multiple visualization charts - Model comparisons and insights")

print("\n" + "=" * 80)
print("STACKING ENSEMBLE MODELING COMPLETED SUCCESSFULLY! 🎉")
print("=" * 80)
```

APPENDIX IV

FA-Stacked ensemble Model

```
import pandas as pd import numpy as np import matplotlib.pyplot as plt
import seaborn as sns from sklearn.model_selection import
train_test_split, cross_val_score from sklearn.preprocessing import
StandardScaler, LabelEncoder from sklearn.ensemble import
RandomForestClassifier, StackingClassifier from sklearn.tree import
DecisionTreeClassifier from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression from sklearn.metrics import
(accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score, roc_curve, confusion_matrix, classification_report) from xgboost
import XGBClassifier import warnings import time warnings.filterwarnings('ignore')

# Set style for better visualizations
plt.style.use('default') sns.set_palette("husl")

print("=" * 80)
print("STACKING ENSEMBLE MODEL FOR CARDIAC ARREST PREDICTION")
print("=" * 80)

# Load the feature selected dataset
print("\n LOADING FEATURE SELECTED DATASET...") df
= pd.read_csv("/content/feature_selected_by_RFE_FA.csv")

print(f"Dataset Shape: {df.shape}") print(f"Features:
{df.shape[1] - 1}, Samples: {df.shape[0]}")

# Identify target column
target_columns = ['target', 'heart_disease', 'disease', 'class', 'output']
target_col = None for col in target_columns:     if col in
df.columns:         target_col = col         break if target_col is None:
target_col = df.columns[-1] print(f"Target column: '{target_col}'")

# Separate features and target X =
df.drop(columns=[target_col]) y =
df[target_col]

print(f"\nSelected Features: {list(X.columns)}")
print(f"Target variable distribution:\n{y.value_counts()}")
```

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

print(f"\nTraining set: {X_train.shape}")
print(f"Testing set: {X_test.shape}")

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("\n🔄 STAGE 1: INDIVIDUAL BASE MODEL TRAINING AND EVALUATION")
print("=" * 60)

# Define base models
base_models = [
    ('decision_tree', DecisionTreeClassifier(random_state=42, max_depth=5)),
    ('svm', SVC(random_state=42, probability=True)),
    ('logistic_regression', LogisticRegression(random_state=42, max_iter=1000)),
    ('random_forest', RandomForestClassifier(random_state=42, n_estimators=100)),
('xgboost', XGBClassifier(random_state=42, eval_metric='logloss')) ]

# Meta learner meta_learner = RandomForestClassifier(random_state=42,
n_estimators=100)

# Store results
results = [] model_predictions = {} model_probabilities =
{} confusion_matrices = {} training_times = {}
testing_times = {} print("Training and evaluating
individual base models...") # Train and evaluate each
base model for name, model in base_models:
print(f"\n--- Training {name.upper()} ---")

   # Handle scaling requirements     if
name in ['svm', 'logistic_regression']:
       X_train_model = X_train_scaled
X_test_model = X_test_scaled     else:
```

```
        X_train_model = X_train
X_test_model = X_test

    # Train model with timing
start_train = time.time()
model.fit(X_train_model, y_train)
end_train = time.time()     train_time
= end_train - start_train
training_times[name] = train_time

    # Predictions with timing     start_test
= time.time()     y_pred =
model.predict(X_test_model)
    end_test = time.time()
test_time = end_test - start_test
testing_times[name] = test_time

    # Get probabilities - FIXED: Handle all models properly
try:
        y_pred_proba = model.predict_proba(X_test_model)[:, 1]
except AttributeError:
        # For models without predict_proba, use decision function and normalize
y_pred_proba = model.decision_function(X_test_model)
        y_pred_proba = (y_pred_proba - y_pred_proba.min()) / (y_pred_proba.max() - y_pred_proba.min())

    # Store predictions for stacking
model_predictions[name] = y_pred
model_probabilities[name] = y_pred_proba

    # Calculate confusion matrix     cm =
confusion_matrix(y_test, y_pred)
confusion_matrices[name] = cm

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, zero_division=0)

    sensitivity = recall_score(y_test, y_pred)  # Sensitivity =
Recall     specificity = recall_score(1-y_test, 1-y_pred)  #
Specificity     f1 = f1_score(y_test, y_pred)     auc =
roc_auc_score(y_test, y_pred_proba)
```

```
    # Calculate additional metrics from confusion matrix
tn, fp, fn, tp = cm.ravel()     false_positive_rate = fp / (fp
+ tn) if (fp + tn) > 0 else 0     false_negative_rate = fn /
(fn + tp) if (fn + tp) > 0 else 0

    # Store results
results.append({
        'Model': name.upper(),
        'Accuracy': accuracy,
'Precision': precision,
        'Sensitivity': sensitivity,
        'Specificity': specificity,
        'F1-Score': f1,
        'AUC-ROC': auc,
        'True_Positive': tp,
        'True_Negative': tn,
        'False_Positive': fp,
        'False_Negative': fn,
        'False_Positive_Rate': false_positive_rate,
        'False_Negative_Rate': false_negative_rate,
        'Training_Time_Seconds': train_time,
        'Testing_Time_Seconds': test_time
    })

    print(f"Accuracy: {accuracy:.4f}")     print(f"F1-
Score: {f1:.4f}")     print(f"AUC-ROC: {auc:.4f}")
print(f"Training Time: {train_time:.4f} seconds")
print(f"Testing Time: {test_time:.4f} seconds")
print(f"Confusion Matrix:\n{cm}")

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

print("\n📊 INDIVIDUAL MODEL PERFORMANCE COMPARISON")
print("=" * 50)
display_columns = ['Model', 'Accuracy', 'Precision', 'Sensitivity', 'Specificity', 'F1-Score', 'AUC-ROC',
'Training_Time_Seconds', 'Testing_Time_Seconds']
print(results_df[display_columns].round(4))
#
```

```
# ============================================================================
# INDIVIDUAL MODEL VISUALIZATIONS - SEPARATED FOR BETTER VISIBILITY
# ============================================================================

print("\n📈 STAGE 2: INDIVIDUAL MODEL VISUALIZATIONS")
print("=" * 50)

# 1. Performance metrics comparison - SEPARATE PLOT
plt.figure(figsize=(15, 8))
metrics = ['Accuracy', 'Precision', 'Sensitivity', 'Specificity', 'F1-Score']
x_pos = np.arange(len(metrics))
width = 0.15

for i, (idx, row) in enumerate(results_df.iterrows()):
    plt.bar(x_pos + i*width,
            [row['Accuracy'], row['Precision'], row['Sensitivity'],
             row['Specificity'], row['F1-Score']],
            width, label=row['Model'], alpha=0.8)

plt.xlabel('Metrics')
plt.ylabel('Score')
plt.title('INDIVIDUAL MODEL PERFORMANCE COMPARISON', fontsize=14, fontweight='bold')
plt.xticks(x_pos + width*2, metrics)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

# 2. AUC-ROC comparison - SEPARATE PLOT
plt.figure(figsize=(12, 6))
plt.bar(results_df['Model'], results_df['AUC-ROC'], color='lightgreen', alpha=0.7)
plt.xlabel('Models')
plt.ylabel('AUC-ROC Score')
plt.title('AUC-ROC SCORES BY MODEL', fontsize=14, fontweight='bold')
plt.xticks(rotation=45)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

# 3. Training Time Comparison - NEW PLOT
plt.figure(figsize=(12, 6))
plt.bar(results_df['Model'], results_df['Training_Time_Seconds'], color='orange', alpha=0.7)
plt.xlabel('Models')
plt.ylabel('Training Time (seconds)')
```

```
plt.title('TRAINING TIME COMPARISON', fontsize=14, fontweight='bold')
plt.xticks(rotation=45) plt.grid(True, alpha=0.3) plt.tight_layout() plt.show()

# 4. Testing Time Comparison - NEW PLOT plt.figure(figsize=(12,
6))
plt.bar(results_df['Model'], results_df['Testing_Time_Seconds'], color='purple', alpha=0.7)
plt.xlabel('Models')
plt.ylabel('Testing Time (seconds)')
plt.title('TESTING TIME COMPARISON', fontsize=14, fontweight='bold')
plt.xticks(rotation=45) plt.grid(True, alpha=0.3) plt.tight_layout() plt.show()

# 5. F1-Score comparison - SEPARATE PLOT plt.figure(figsize=(12,
6))
plt.bar(results_df['Model'], results_df['F1-Score'], color='lightcoral', alpha=0.7)
plt.xlabel('Models') plt.ylabel('F1-Score')
plt.title('F1-SCORES BY MODEL', fontsize=14, fontweight='bold')
plt.xticks(rotation=45) plt.grid(True, alpha=0.3) plt.tight_layout()
plt.show()

# 6. Confusion Matrices - SEPARATE PLOTS FOR EACH MODEL
print("\n CONFUSION MATRICES FOR INDIVIDUAL MODELS")
print("=" * 50)

model_names = ['decision_tree', 'svm', 'logistic_regression', 'random_forest', 'xgboost']
for i, name in enumerate(model_names):
    plt.figure(figsize=(8, 6))     cm =
confusion_matrices[name]     sns.heatmap(cm,
annot=True, fmt='d', cmap='Blues',
xticklabels=['Predicted 0', 'Predicted 1'],
yticklabels=['Actual 0', 'Actual 1'],
cbar_kws={'shrink': 0.8})
    plt.title(f'CONFUSION MATRIX - {name.upper()}\nTN:{cm[0,0]} FP:{cm[0,1]} | FN:{cm[1,0]}
TP:{cm[1,1]}',           fontsize=12,
   fontweight='bold')  plt.tight_layout()

    plt.show()

# 7. ROC Curves for individual models - SEPARATE PLOT
plt.figure(figsize=(12, 8)) for name in model_names:     if
```

```
name in ['svm', 'logistic_regression']:        X_test_model =
X_test_scaled     else:
        X_test_model = X_test

    # Get the trained model     model_obj = next(m[1] for m in
base_models if m[0] == name)

    # Get probabilities safely
try:
        y_pred_proba = model_obj.predict_proba(X_test_model)[:, 1]
except AttributeError:
        # For models without predict_proba, use decision function
y_pred_proba = model_obj.decision_function(X_test_model)
        y_pred_proba = (y_pred_proba - y_pred_proba.min()) / (y_pred_proba.max() - y_pred_proba.min())

    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)     auc_score = roc_auc_score(y_test,
y_pred_proba)     plt.plot(fpr, tpr, linewidth=2.5, label=f'{name.upper()} (AUC =
{auc_score:.3f})')

plt.plot([0, 1], [0, 1], 'k--', linewidth=2, label='Random Classifier', alpha=0.7)
plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.title('ROC CURVES - INDIVIDUAL MODELS', fontsize=14, fontweight='bold')
plt.legend(fontsize=10) plt.grid(True, alpha=0.3) plt.tight_layout() plt.show()

print("\n🔄 STAGE 3: STACKING ENSEMBLE MODEL")
print("=" * 45) print("Building Stacking

Ensemble Model...")

# Create stacking classifier
stacking_model =
StackingClassifier(     estimators=base_m
odels,     final_estimator=meta_learner,
    cv=5,
   passthrough=False )

# Train stacking model with timing start_train_stack
= time.time() stacking_model.fit(X_train, y_train)
end_train_stack = time.time() train_time_stack =
end_train_stack - start_train_stack
```

```
# Predictions from stacking model with timing
start_test_stack = time.time()
y_pred_stack = stacking_model.predict(X_test) end_test_stack =
time.time() test_time_stack = end_test_stack - start_test_stack
y_pred_proba_stack = stacking_model.predict_proba(X_test)[:, 1]

# Calculate confusion matrix for stacking cm_stack
= confusion_matrix(y_test, y_pred_stack)
confusion_matrices['stacking'] = cm_stack

# Calculate metrics for stacking model
stack_accuracy = accuracy_score(y_test, y_pred_stack)
stack_precision = precision_score(y_test, y_pred_stack, zero_division=0)
stack_sensitivity = recall_score(y_test, y_pred_stack) stack_specificity =
recall_score(1-y_test, 1-y_pred_stack) stack_f1 = f1_score(y_test,
y_pred_stack) stack_auc = roc_auc_score(y_test, y_pred_proba_stack)

# Calculate additional metrics from confusion matrix tn_stack, fp_stack, fn_stack, tp_stack =
cm_stack.ravel() false_positive_rate_stack = fp_stack / (fp_stack + tn_stack) if (fp_stack +
tn_stack) > 0 else 0 false_negative_rate_stack = fn_stack / (fn_stack + tp_stack) if (fn_stack
+ tp_stack) > 0 else 0

# Add stacking results to dataframe
stacking_results = {
    'Model': 'STACKING ENSEMBLE',
    'Accuracy': stack_accuracy,
'Precision': stack_precision,
    'Sensitivity': stack_sensitivity,
    'Specificity': stack_specificity,
    'F1-Score': stack_f1,
    'AUC-ROC': stack_auc,
    'True_Positive': tp_stack,
    'True_Negative': tn_stack,
    'False_Positive': fp_stack,
    'False_Negative': fn_stack,
    'False_Positive_Rate': false_positive_rate_stack,
    'False_Negative_Rate': false_negative_rate_stack,
    'Training_Time_Seconds': train_time_stack,
    'Testing_Time_Seconds': test_time_stack
```

```
} results_df = pd.concat([results_df, pd.DataFrame([stacking_results])],

ignore_index=True)

print("\n📊 STACKING ENSEMBLE PERFORMANCE")
print("=" * 40)
stacking_performance = results_df[results_df['Model'] == 'STACKING ENSEMBLE']
print(stacking_performance[display_columns].round(4)) print(f"\nConfusion Matrix
for Stacking Ensemble:") print(cm_stack)

#
==========================================================================
===
# STACKING ENSEMBLE VISUALIZATIONS - SEPARATED
#
==========================================================================
===

print("\n📈 STAGE 4: STACKING ENSEMBLE VISUALIZATIONS")
print("=" * 50)

# 1. Stacking Confusion Matrix - SEPARATE PLOT plt.figure(figsize=(8,
6))
sns.heatmap(cm_stack, annot=True, fmt='d', cmap='Greens',
xticklabels=['Predicted 0', 'Predicted 1'],
yticklabels=['Actual 0', 'Actual 1'],
cbar_kws={'shrink': 0.8})
plt.title('CONFUSION MATRIX - STACKING ENSEMBLE\nTN:{} FP:{} | FN:{}
TP:{}'.format(     cm_stack[0,0], cm_stack[0,1], cm_stack[1,0], cm_stack[1,1]),     fontsize=12,
fontweight='bold') plt.tight_layout() plt.show()

# 2. Stacking ROC Curve - SEPARATE PLOT
plt.figure(figsize=(10, 8))
fpr_stack, tpr_stack, _ = roc_curve(y_test, y_pred_proba_stack)
plt.plot(fpr_stack, tpr_stack, 'b-', linewidth=3, label=f'STACKING ENSEMBLE (AUC =
{stack_auc:.3f})')
plt.plot([0, 1], [0, 1], 'k--', linewidth=2, label='Random Classifier', alpha=0.7)
plt.xlabel('False Positive Rate', fontsize=12) plt.ylabel('True Positive Rate',
fontsize=12)
```

```
plt.title('ROC CURVE - STACKING ENSEMBLE', fontsize=14, fontweight='bold')
plt.legend(fontsize=12) plt.grid(True, alpha=0.3) plt.tight_layout() plt.show()

#
==========================================================================
===
# COMPREHENSIVE COMPARISON VISUALIZATIONS - SEPARATED
#
==========================================================================
===

print("\n📊 STAGE 5: COMPREHENSIVE MODEL COMPARISON")
print("=" * 50)

# 1. Radar Chart Comparison - SEPARATE PLOT plt.figure(figsize=(10, 10))
categories = ['Accuracy', 'Precision', 'Sensitivity', 'Specificity', 'F1-Score', 'AUC-ROC']

def normalize_radar_data(data):
return [d * 100 for d in data]

angles = np.linspace(0, 2*np.pi, len(categories), endpoint=False).tolist()
angles += angles[:1]  # Complete the circle

fig, ax = plt.subplots(figsize=(10, 10), subplot_kw=dict(projection='polar'))
colors = ['red', 'blue', 'green', 'orange', 'purple', 'brown', 'black']

for idx, row in results_df.iterrows():     values =
normalize_radar_data([         row['Accuracy'],
row['Precision'], row['Sensitivity'],         row['Specificity'],
row['F1-Score'], row['AUC-ROC']
    ])
    values += values[:1]  # Complete the circle
    ax.plot(angles, values, 'o-', linewidth=2, label=row['Model'], color=colors[idx])
ax.fill(angles, values, alpha=0.1, color=colors[idx])

ax.set_xticks(angles[:-1])
ax.set_xticklabels(categories)
ax.set_ylim(0, 100)
plt.title('COMPREHENSIVE MODEL COMPARISON\n(RADAR CHART)', size=16,
fontweight='bold', pad=20)
```

```
plt.legend(bbox_to_anchor=(1.2, 1), loc='upper left', fontsize=10) plt.tight_layout()

plt.show()

# 2. Performance vs Training Time - NEW PLOT
plt.figure(figsize=(12, 8)) for idx, row in results_df.iterrows():
plt.scatter(row['Training_Time_Seconds'], row['F1-Score'], s=150, label=row['Model'], alpha=0.7)
plt.annotate(row['Model'], (row['Training_Time_Seconds'], row['F1-Score']),                xytext=(8,
8), textcoords='offset points', fontsize=10)

plt.xlabel('Training Time (seconds)', fontsize=12) plt.ylabel('F1-Score',
fontsize=12)
plt.title('PERFORMANCE vs TRAINING TIME TRADE-OFF', fontsize=14, fontweight='bold')
plt.legend(fontsize=10) plt.grid(True, alpha=0.3) plt.tight_layout() plt.show()

# 3. Stacking vs Best Base Model - SEPARATE PLOT plt.figure(figsize=(10,
6))
best_base_f1 = results_df[results_df['Model'] != 'STACKING ENSEMBLE']['F1-Score'].max()
stacking_f1 = stacking_results['F1-Score'] improvement = ((stacking_f1 - best_base_f1) /
best_base_f1) * 100

models_compare = ['Best Base Model', 'Stacking Ensemble']
f1_scores_compare = [best_base_f1, stacking_f1]

bars = plt.bar(models_compare, f1_scores_compare, color=['lightblue', 'lightgreen'], alpha=0.7)
plt.ylabel('F1-Score', fontsize=12)
plt.title(f'STACKING vs BEST BASE MODEL\nImprovement: {improvement:.2f}%', fontsize=14,
fontweight='bold') plt.grid(True, alpha=0.3)

# Add value labels on bars for bar, value in zip(bars,
f1_scores_compare):    plt.text(bar.get_x() + bar.get_width()/2,
bar.get_height() + 0.01,
         f'{value:.4f}', ha='center', va='bottom', fontsize=12, fontweight='bold')
plt.tight_layout() plt.show()

# 4. All Models ROC Curves - SEPARATE PLOT
plt.figure(figsize=(12, 10)) # Individual models
for name in model_names:    if name in ['svm',
'logistic_regression']:       X_test_model =
X_test_scaled    else:
```

```
        X_test_model = X_test     model_obj = next(m[1] for m in

base_models if m[0] == name)

    # Get probabilities safely
try:
        y_pred_proba = model_obj.predict_proba(X_test_model)[:, 1]
except AttributeError:
        # For models without predict_proba, use decision function
y_pred_proba = model_obj.decision_function(X_test_model)
        y_pred_proba = (y_pred_proba - y_pred_proba.min()) / (y_pred_proba.max() - y_pred_proba.min())

    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)     auc_score = roc_auc_score(y_test,
y_pred_proba)     plt.plot(fpr, tpr, alpha=0.7, linewidth=2, label=f'{name.upper()} (AUC =
{auc_score:.3f})')

# Stacking model
plt.plot(fpr_stack, tpr_stack, 'k-', linewidth=4,
label=f'STACKING ENSEMBLE (AUC = {stack_auc:.3f})')

plt.plot([0, 1], [0, 1], 'k--', alpha=0.5, linewidth=2, label='Random Classifier')
plt.xlabel('False Positive Rate', fontsize=12) plt.ylabel('True Positive Rate',
fontsize=12)
plt.title('ROC CURVES - ALL MODELS', fontsize=14, fontweight='bold')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', fontsize=10)
plt.grid(True, alpha=0.3) plt.tight_layout() plt.show()

# 5. Error Rates Comparison - SEPARATE PLOT plt.figure(figsize=(12,
8)) fpr_rates = [row['False_Positive_Rate'] for _, row in
results_df.iterrows()] fnr_rates = [row['False_Negative_Rate'] for _, row
in results_df.iterrows()] models = results_df['Model']

for i, (fpr, fnr, model) in enumerate(zip(fpr_rates, fnr_rates, models)):
plt.scatter(fpr, fnr, s=150, label=model, alpha=0.7)     plt.annotate(model, (fpr, fnr),
xytext=(8, 8), textcoords='offset points', fontsize=10)

plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('False Negative Rate', fontsize=12)
plt.title('ERROR RATES COMPARISON\n(Lower Left = Better Performance)', fontsize=14,
fontweight='bold') plt.legend(fontsize=10) plt.grid(True, alpha=0.3)
```

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

# 6. True Positive vs True Negative - SEPARATE PLOT
plt.figure(figsize=(12, 8)) tp_counts = [row['True_Positive'] for _,
row in results_df.iterrows()] tn_counts = [row['True_Negative'] for _,
row in results_df.iterrows()] models = results_df['Model']

x_pos = np.arange(len(models))
width = 0.35

plt.bar(x_pos - width/2, tp_counts, width, label='True Positives', alpha=0.7, color='green')
plt.bar(x_pos + width/2, tn_counts, width, label='True Negatives', alpha=0.7, color='blue')

plt.xlabel('Models', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('TRUE POSITIVE vs TRUE NEGATIVE COUNTS', fontsize=14, fontweight='bold')
plt.xticks(x_pos, models, rotation=45) plt.legend(fontsize=10) plt.grid(True, alpha=0.3)
plt.tight_layout() plt.show()

# 7. Model Ranking by F1-Score - SEPARATE PLOT plt.figure(figsize=(12,
8))
ranked_models = results_df.sort_values('F1-Score', ascending=True)
plt.barh(ranked_models['Model'], ranked_models['F1-Score'], color='lightcoral', alpha=0.7)
plt.xlabel('F1-Score', fontsize=12)
plt.title('MODEL RANKING BY F1-SCORE', fontsize=14, fontweight='bold') plt.grid(True,
alpha=0.3)

# Add value labels for i, (value, model) in enumerate(zip(ranked_models['F1-Score'],
ranked_models['Model'])):
    plt.text(value + 0.01, i, f'{value:.4f}', va='center', fontsize=10, fontweight='bold')
plt.tight_layout() plt.show()

# 8. Training Time Analysis - NEW PLOT
plt.figure(figsize=(12, 8))
ranked_by_time = results_df.sort_values('Training_Time_Seconds', ascending=True)
plt.barh(ranked_by_time['Model'], ranked_by_time['Training_Time_Seconds'], color='teal', alpha=0.7)
plt.xlabel('Training Time (seconds)', fontsize=12)
plt.title('MODEL TRAINING TIME COMPARISON', fontsize=14, fontweight='bold') plt.grid(True,
alpha=0.3)
```

```
# Add value labels for i, (value, model) in
enumerate(zip(ranked_by_time['Training_Time_Seconds'],
ranked_by_time['Model'])):
    plt.text(value + 0.01, i, f'{value:.4f}s', va='center', fontsize=10, fontweight='bold')
plt.tight_layout() plt.show()

# 9. Performance Metrics Table - SEPARATE DISPLAY
print("\n📋 PERFORMANCE METRICS SUMMARY TABLE") print("="
* 50)
display_columns_extended = ['Model', 'Accuracy', 'Precision', 'Sensitivity', 'Specificity', 'F1-Score',
'AUC-ROC', 'Training_Time_Seconds', 'Testing_Time_Seconds', 'False_Negative']
print(results_df[display_columns_extended].round(4).to_string(index=False))

print("\n🎯 CONFUSION MATRIX INTERPRETATION FOR BEGINNERS")
print("=" * 55)

print("""
📊 UNDERSTANDING CONFUSION MATRICES:

For each model's confusion matrix:

   [ TN  FP ]  ← Actual Negative Class
[ FN  TP ]  ← Actual Positive Class

Where:
• TN (True Negative): Correctly predicted non-cardiac arrest cases
• FP (False Positive): Wrongly predicted cardiac arrest (False Alarm)
• FN (False Negative): Missed cardiac arrest cases (Dangerous!)
• TP (True Positive): Correctly predicted cardiac arrest cases

🔍 MEDICAL INTERPRETATION:

✅ IDEAL SCENARIO:
• High TP: Catch most actual cardiac arrest cases
• High TN: Correctly identify healthy patients
• Low FP: Avoid unnecessary treatments/scares• Low FN: Don't miss actual cardiac arrest
  cases

⚠ CLINICAL IMPORTANCE:
```

```
• False Negatives are CRITICAL in cardiac arrest prediction
• Missing a cardiac arrest case can be life-threatening
• False Positives cause unnecessary stress but are less dangerous

📈 WHAT TO LOOK FOR:
1. High diagonal values (TN and TP)
2. Low off-diagonal values (FP and FN)
3. Especially watch for low FN values""")

# Print detailed confusion matrix analysis
print("\n📋 DETAILED CONFUSION MATRIX ANALYSIS:")
print("=" * 45) for _, row in results_df.iterrows():
model_name = row['Model']
    tp, tn, fp, fn = row['True_Positive'], row['True_Negative'], row['False_Positive'],
row['False_Negative']     total = tp + tn + fp + fn

    print(f"\n{model_name}:")     print(f"  True Positives: {tp} ({tp/total*100:.1f}%) -
Correct cardiac arrest predictions")     print(f"  True Negatives: {tn} ({tn/total*100:.1f}%)
- Correct healthy predictions")     print(f"  False Positives: {fp} ({fp/total*100:.1f}%) -
False alarms")     print(f"  False Negatives: {fn} ({fn/total*100:.1f}%) - Missed cardiac
arrest cases")

print("\n⏱️  TRAINING AND TESTING TIME ANALYSIS:")
print("=" * 45) for _, row in results_df.iterrows():
    print(f"{row['Model']}:")     print(f"  Training Time:
{row['Training_Time_Seconds']:.4f} seconds")     print(f"  Testing Time:
{row['Testing_Time_Seconds']:.4f} seconds")

print("\n🎯 FINAL RESULTS AND INTERPRETATION")
print("=" * 50)

# Find best performing models
best_base_model = results_df[results_df['Model'] != 'STACKING ENSEMBLE'].nlargest(1,
'F1Score').iloc[0] best_overall_model = results_df.nlargest(1, 'F1-Score').iloc[0]

print(f"🏆 BEST BASE MODEL: {best_base_model['Model']}")
print(f"    F1-Score: {best_base_model['F1-Score']:.4f}") print(f"
Accuracy: {best_base_model['Accuracy']:.4f}")
```

```
print(f"   Training Time: {best_base_model['Training_Time_Seconds']:.4f} seconds")
print(f"   False Negatives: {best_base_model['False_Negative']}")

print(f"\n🏆 BEST OVERALL MODEL: {best_overall_model['Model']}")
print(f"   F1-Score: {best_overall_model['F1-Score']:.4f}")
print(f"   Accuracy: {best_overall_model['Accuracy']:.4f}")
print(f"   Training Time: {best_overall_model['Training_Time_Seconds']:.4f} seconds")
print(f"   False Negatives: {best_overall_model['False_Negative']}")
improvement_f1 = ((best_overall_model['F1-Score'] - best_base_model['F1-Score']) /
best_base_model['F1-Score']) * 100 print(f"\n📈 IMPROVEMENT: {improvement_f1:.2f}%
F1-Score improvement with Stacking")

print("""
📖 INTERPRETATION FOR BEGINNERS:

🔍 WHAT IS STACKING ENSEMBLE?
• Stacking combines multiple machine learning models to create a superior model
• Base Models: Decision Tree, SVM, Logistic Regression, Random Forest, XGBoost
• Meta-Learner: Random Forest that learns to combine base model predictions

🎯 KEY INSIGHTS FROM CONFUSION MATRICES:

✅ WHY STACKING WORKS:
• Reduces both False Positives and False Negatives
• Combines strengths of different algorithms
• Provides more reliable predictions for critical medical decisions

💡 CLINICAL RECOMMENDATION:
The model with the lowest False Negative rate is most suitable for cardiac arrest
prediction, as missing actual cases can have serious consequences.

⏱️ TIME CONSIDERATIONS:
• Training time includes model fitting and parameter optimization
• Testing time measures prediction speed on new data
• Stacking typically takes longer to train but provides better performance""")

# Save results
print(f"\n💾 SAVING RESULTS...")
```

```
results_df.to_csv("stacking_ensemble_results.csv", index=False) print("✅
Results saved as 'stacking_ensemble_results.csv'")

# Save confusion matrices
confusion_matrices_df = pd.DataFrame({
    'Model': list(confusion_matrices.keys()),
    'Confusion_Matrix': [str(cm) for cm in confusion_matrices.values()]
}) confusion_matrices_df.to_csv("confusion_matrices.csv",
index=False) print("✅ Confusion matrices saved as
'confusion_matrices.csv'")

print(f"\n📁 FILES CREATED:")
print(f"   1. stacking_ensemble_results.csv - Complete performance metrics")
print(f"   2. confusion_matrices.csv - All confusion matrices")
print(f"   3. Multiple visualization charts - Model comparisons and insights")
print("\n" + "=" * 80)
print("STACKING ENSEMBLE MODELING COMPLETED SUCCESSFULLY! 🎉")
print("=" * 80)
```