Predicting heart disease based on ECG using enhanced machine learning models

Report submitted to the SASTRA Deemed to be University
in partial fulfillment of the requirements
for the award of the degree of

Bachelor of Technology

Submitted by
VIGNESH PRAKASH
(Reg. No.: 124015117, B. Tech Information Technology)
INFANT AKASH
(Reg. No.: 124015034, B. Tech Information Technology)
ASHISH PRABHU K
(Reg. No.: 124012005, B. Tech Mechatronics)

MAY 2024

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the project report titled "**Predicting heart disease based on ECG using enhanced machine learning models" submitted** in partial fulfillment of the requirements for the award of the degree of B. Tech. Information Technology and B. Tech Mechatronics to the SASTRA Deemed to be University, is a bona-fide record of the work done by **Vignesh Prakash(124015117) , Infant Akash S (124015034), Ashish Prabhu K (124012005)** during the final semester of the academic year 2023- 24, in the **School of Computing**, under my supervision. This report has not formed the basis for the award of any degree, diploma, associateship, fellowship or other similar title to any candidate of any University.

Signature of Project Supervisor :

Name with Affiliation : **Dr Rajkumar K**

Date : **03.05.2024**

Project *Viva voce* held on _____

Examiner 1 **Examiner 2**

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Declaration

We declare that the project report titled **"Predicting heart disease based on ECG using enhanced machine learning models "** submitted by us is an original work done by us under the guidance of **Dr Rajkumar K, Senior Associate professor**, **School of Computing, SASTRA Deemed to be University** during the final semester of the academic year 2023-24, in the **School of Computing**. The work is original and wherever We have used materials from other sources, We have given due credit and cited them in the text of the report. This report has not formed the basis for the award of any degree, diploma, associate-ship, fellowship or other similar title to any candidate of any University.

Signature of the candidate(s):

Name of the candidate(s): VigneshPrakash Infant Akash S Ashish Prabhu K

Date: 03.05.2024

Acknowledgements

We would like to thank our Honourable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honourable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan,** Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. V. S. Shankar Sriram**, Dean, School of Computing, **Dr. R. Muthaiah,** Associate Dean, Research, **Dr. K.Ramkumar**, Associate Dean, Academics, **Dr. D. Manivannan**, Associate Dean, Infrastructure, **Dr. R. Algeswaran**, Associate Dean, Students Welfare.

Our guide **Dr Rajkumar K**, Senior Associate Professor, School of Computing was the driving force behind this whole idea from the start. His deep insight in the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing me an opportunity to showcase my skills through project.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Cardiovascular disease stands as a leading cause of mortality on a global scale, with cardiovascular arrhythmia, characterized by irregular heart rhythm, presenting a significant challenge within the realm of heart health. Arrhythmia not only signifies a disruption in heart rhythm but also acts as a key trigger for various cardiovascular complications, emphasizing the critical importance of understanding and addressing irregular heartbeats for the enhancement of heart health worldwide. Electrocardiography (ECG) data analysis and interpretation play indispensable roles in the diagnosis of cardiovascular diseases. This study is centered on identifying individuals based on their health status through the utilization of machine learning methods. An upcoming investigation aims to automatically classify ECG data employing various machine learning techniques, including the Dummy Classifier, Logistic Regression, Random Forest, Gaussian Naive Bayes, Linear Discriminant Analysis, XGBoost, and AdaBoost. The overarching goal of this study is to unearth fresh insights and methodologies that can propel advancements in cardiovascular diagnostics through the application of sophisticated machine learning approaches.

Specific Contribution

- Data Splitting and pre-processing
- Building different deep learning models and evaluating their performance
- Building a Webapp as a Front-end, using ReactJS

Specific Learning

- Understanding about Machine learning algorithms
- Knowledge about different Machine learning and visualization libraries in python
- Advanced functionalities in ReactJS and HTML & CSS

ABBREVIATION

ECG → Electrocardiography (ECG)

ROC → Receiver Operating characteristic Curve

AUC → Area Under the Curve

TP → True Positive

FP → False Positive

TN → True Negative

FN → False Negative

TPR → True Positive Rate

FPR → False Positive Rate

CHAPTER 1

INTRODUCTION

Cardiovascular disease (CVD) presents a significant global health concern, accounting for a substantial portion of global mortality. The availability of data has empowered researchers and clinicians to undertake research and diagnostic endeavors to confront this urgent challenge. Within the spectrum of CVD, cardiovascular arrhythmia stands out as a significant factor, marked by deviations from normal heartbeat patterns. Precise classification of these irregularities is essential for the successful identification and treatment of various cardiac ailments, guaranteeing the application of suitable therapeutic measures.

The electrocardiogram (ECG) stands as a fundamental tool in cardiovascular diagnostics, offering a non-invasive and dependable method for detecting cardiovascular disorders. Featuring a standard setup of 12 leads, the ECG provides valuable insights into the electrical activity and functioning of the heart. However, the manual interpretation of ECG readings by cardiologists poses challenges, requiring significant time and expertise. This approach is susceptible to errors, potentially resulting in incorrect diagnoses and negative clinical outcomes. Consequently, there is a growing emphasis on developing precise and automated methods for ECG signal analysis to ease the workload on cardiologists and streamline the diagnostic process.

Advancements in signal processing and machine learning have transformed the landscape of cardiovascular diagnostics, presenting promising avenues for enhanced analysis of ECG data. Techniques such as the quick Fourier transform, wavelet transform, and digital filtering have played crucial roles in extracting pertinent features from ECG signals, facilitating more precise interpretation and diagnosis. Machine learning algorithms, known for their automated pattern recognition capabilities, have emerged as formidable assets in this field. Particularly, deep learning models like neural networks have gained prominence for their capacity to discern intricate patterns from extensive datasets, resulting in improved diagnosis of cardiovascular conditions.

The incorporation of machine learning methodologies into cardiovascular diagnostics has ushered in innovative solutions to longstanding challenges. By harnessing extensive ECG data, these algorithms can identify subtle patterns and anomalies indicative of various cardiac ailments. Furthermore, their flexibility and scalability render them well-equipped to handle the intricacies of cardiovascular data, offering potential advantages in terms of accuracy and efficiency.

In recent times, initiatives like the PhysioNet project have aimed to leverage the combined potential of data and machine learning for the creation of automated ECG classifiers. These initiatives strive to address the limitations associated with manual interpretation and facilitate a more streamlined and precise diagnosis of cardiovascular conditions. By utilizing deep neural network models capable of classifying various ECG diagnoses, researchers have made significant strides in automated arrhythmia detection and cardiovascular disease diagnosis.

Cross-validation techniques play an integral role in ensuring the reliability and applicability of machine learning models in cardiovascular diagnostics. Approaches such as 10-fold cross-validation are instrumental in mitigating issues like overfitting and ensuring that predictive models perform effectively on new, unseen data. Through systematic evaluation of model performance across diverse subsets of the data, researchers can ascertain the robustness of their algorithms and pinpoint areas for enhancement.

In summary, the amalgamation of signal processing and machine learning methodologies has transformed cardiovascular diagnostics, presenting novel avenues for more precise and effective diagnosis of cardiovascular conditions. As advancements in this domain continue to unfold, the emergence of automated ECG classifiers holds great promise in improving patient outcomes and alleviating the global burden of cardiovascular disease.

Initiatives like the PhysioNet project signify significant advancements in cardiovascular diagnostics, striving to capitalize on diverse data reservoirs for the creation of automated ECG classifiers. These endeavors leverage state-of-the-art technologies, such as deep neural network models, to augment the accuracy and efficiency of diagnosing various ECG abnormalities. The adoption of deep learning methodologies heralds a paradigm shift in the field, enabling the discernment of intricate patterns and subtle irregularities within ECG signals.

Cross-validation techniques play an indispensable role in ensuring the dependability and resilience of these automated classifiers. One of the most widely utilized methods, 10-fold cross-validation, serves to alleviate issues such as overfitting and guarantees the generalizability of predictive models to unseen data. Through methodically assessing the performance of these models across various subsets of the data, researchers can bolster their confidence in the reliability of the findings and forestall biased evaluations.

This study employed a diverse range of machine learning algorithms to classify individuals as either healthy or ill based on ECG data. These algorithms encompass Gaussian Naive Bayes, Random Forest, Logistic Regression, Linear Discriminant Analysis, and Dummy Classifier. Moreover, advanced techniques such as XGBoost and AdaBoost were incorporated into the classification process. The choice of these algorithms stemmed from their efficiency, scalability, and regularization capabilities, rendering them well-suited for navigating the intricacies inherent in ECG data analysis.

XGBoost and AdaBoost have garnered significant attention for their adeptness in managing high-dimensional data and mitigating issues like overfitting. Utilizing an ensemble-based approach that amalgamates predictions from multiple weak learners, these algorithms exhibit superior performance in classification tasks. The integration of XGBoost and AdaBoost into the study aimed to harness their capabilities effectively, facilitating swift and precise differentiation between healthy and ill individuals.

The primary objective of the study was to prioritize the safety and well-being of patients by establishing a robust and dependable automated classification system for cardiovascular conditions. Through the integration of machine learning algorithms alongside cross-validation techniques, researchers aimed to ensure that the resultant models not only offered accuracy but also demonstrated generalizability across diverse patient demographics. Ultimately, the success of these endeavors holds immense potential to transform cardiovascular diagnostics, ushering in an era of more efficient and effective patient care.

CHAPTER 2
OBJECTIVES

2.1) Enhance predictive accuracy while reducing the number of features utilized:

- Utilizing Random Forest algorithm
- Reduced feature set from 14 to 12
- Aim to enhance predictive accuracy
- Optimize computational resources
- Ensuring selected features are most relevant for cardiovascular diagnostics
- Striking a balance between accuracy and computational efficiency
- Advancing effectiveness of machine learning methodologies in cardiovascular health assessment

2.2) Implementation as a Web Application

- Backend Implementation with Flask
- Frontend Integration with ReactJS

CHAPTER 3

EXPERIMENT METHODOLOGY

3.1) EXISTING METHODOLOGY

In this study, we conducted an in-depth exploration of various classification algorithms pivotal to machine learning tasks. Among these algorithms, Gaussian Naive Bayes (Gaussian NB) emerged as a prominent tool, especially adept at handling datasets with continuous-valued features. Leveraging the assumption of a normal (Gaussian) distribution across features, Gaussian NB stands out for its simplicity and efficiency, particularly when feature independence can be reasonably presumed within a given class.

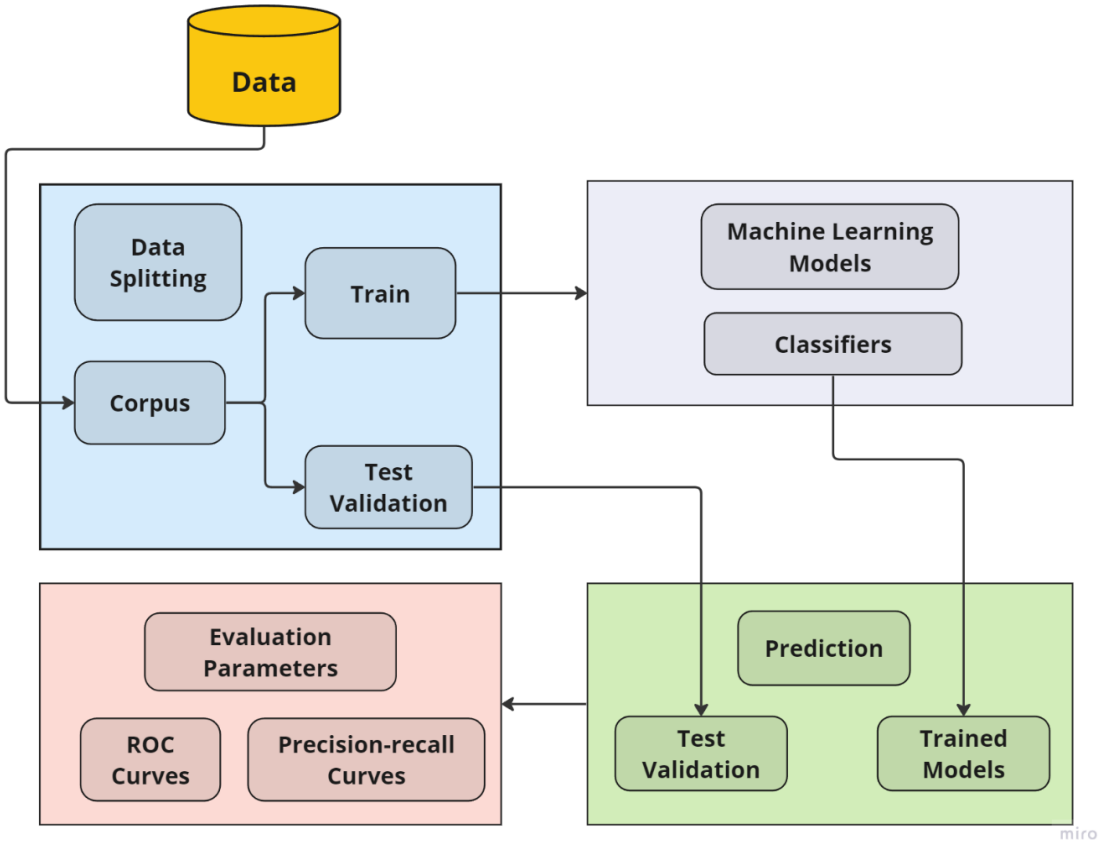
Moreover, our investigation extended to Random Forest, an ensemble learning method gaining considerable traction in the field. By amalgamating multiple decision trees, Random Forest enhances accuracy and resilience to noise, making it particularly adept at managing large databases. Notably, its ability to accommodate numerous input variables without necessitating feature deletion offers practical advantages. Additionally, by evaluating the importance of each variable, Random Forest provides valuable insights into the classification process while demonstrating robustness against outliers and noisy data.

Furthermore, Logistic Regression emerged as a fundamental algorithm, particularly well-suited for binary classification tasks. Unlike traditional classifiers, Logistic Regression directly estimates probabilities, a feature invaluable for scenarios where understanding outcome likelihoods holds paramount importance. Its simplicity and interpretability render it a popular choice across various domains, ranging from medical diagnosis to financial forecasting.

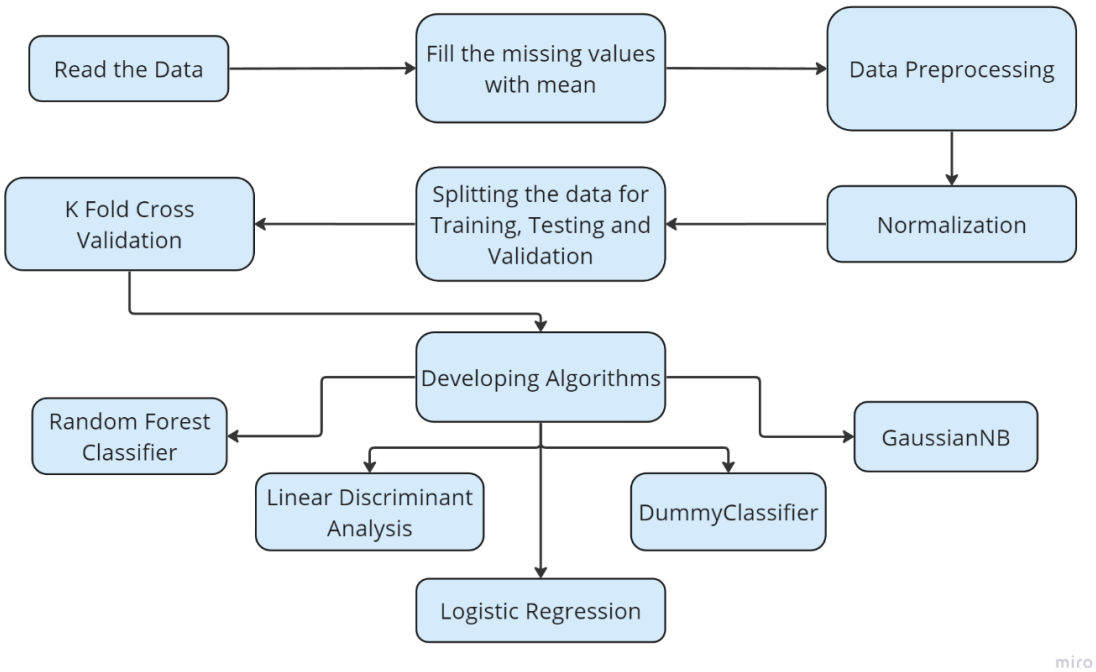
In our exploration, Linear Discriminant Analysis (LDA) also warranted attention for its Bayesian approach and assumption of Gaussian distributions. Unlike Naive Bayes, LDA does not presuppose feature independence; instead, it considers a Gaussian distribution with comparable covariance across all classes. This characteristic renders LDA apt for scenarios where feature interdependence significantly influences classification outcomes.

Lastly, we examined the role of the Dummy Classifier, serving as a rudimentary benchmark for evaluating the performance of more sophisticated algorithms. By generating predictions irrespective of input features, the Dummy Classifier provides a straightforward comparison tool. The ability to customize its behavior via a strategy parameter facilitates mimicking various baseline scenarios, thereby aiding in the assessment and selection of advanced models.

To ensure the reliability and robustness of our findings, we employed rigorous 10-fold cross-validation in our experiments. This approach involved dividing the dataset into ten equally-sized subsets, training the models on nine of these subsets, and validating them on the remaining subset. This process was repeated ten times, with each subset serving as the validation set once, thereby providing a comprehensive evaluation of the algorithms' performance across different data partitions.



3.1 Block diagram for Existing methodology



3.2 Workflow for Existing methodology

3.2) DATASET DESCRIPTION

After preprocessing, including outlier removal and normalization, the dataset encompasses various features:

1. Age: Denoting the individual's age.

2. Sex: Representing gender, with '1' indicating male and '0' indicating female.

3. Chest Pain Type (cp): Categorizing chest pain into different types, with '0' for typical angina, '1' for atypical angina, '2' for non-anginal pain, and '3' for asymptomatic.

4. Resting Blood Pressure (treetops): Measured in mmHg, with values typically alarming if between 130 and 140.

5. Serum Cholesterol (Chol): Represented in mg/dl.

6. Fasting Blood Sugar (FBS): Comparing fasting blood sugar levels to 120 mg/dl, where '1' signifies levels greater than 126 mg/dl, indicating diabetes, and '0' indicates otherwise.

7. Resting Electrocardiographic Data (resting): Categorized as '0' for normal, '1' for abnormal ST-T wave, and '2' for left ventricular hypertrophy.

8. Maximum Heart Rate Achieved (Thalach): Representing the highest heart rate achieved by an individual.

9. Exercise-Induced Angina (exang): Indicated by '1' for yes and '0' for no.

10. ST Depression induced by exercise relative to rest (Oldpeak): Represented as an integer or float.

11. Slope: Representing the slope of the ST segment during exercise, with '0' indicating upsloping (uncommon), '1' indicating little change or flat (typical healthy heart), and '2' indicating downsloping (indications of a cardiac condition).

12. Number of Major Vessels Colored by Fluoroscopy (ca): Ranging from 0 to 3.

13. Thalassemia (thal): Categorized as '1' or '3' for normal, '6' indicating a corrected defect, and '7' indicating a reversible defect.

Target: Indicates the presence of heart disease, with '1' for yes and '0' for no.

These features collectively provide insight into various physiological and clinical factors potentially influencing the presence of heart disease.

3.3) PROPOSED METHODOLOGY

3.3.1 Data Preparation

To begin, we prepare the dataset for analysis. This involves loading the data and ensuring its cleanliness and structure. We handle missing values and convert categorical variables into a format suitable for analysis. By properly formatting the data, we establish a solid foundation for building a reliable predictive model.

3.3.2 Normalization

Following data preparation, we proceed to normalize the dataset. This step ensures that all features are on a similar scale, preventing any single feature from dominating the model during training. Normalization is crucial for enhancing the model's performance and ensuring it can effectively learn from the data without being biased towards certain features.

3.3.3 Feature Selection

Feature selection is a pivotal part of model building. Here, we employ a Random Forest classifier to identify the most important features in the dataset. This allows us to focus on the key variables that significantly impact predicting the target variable—in this case, whether a person has heart disease or not. By selecting the most relevant features, we streamline the model and enhance its accuracy.

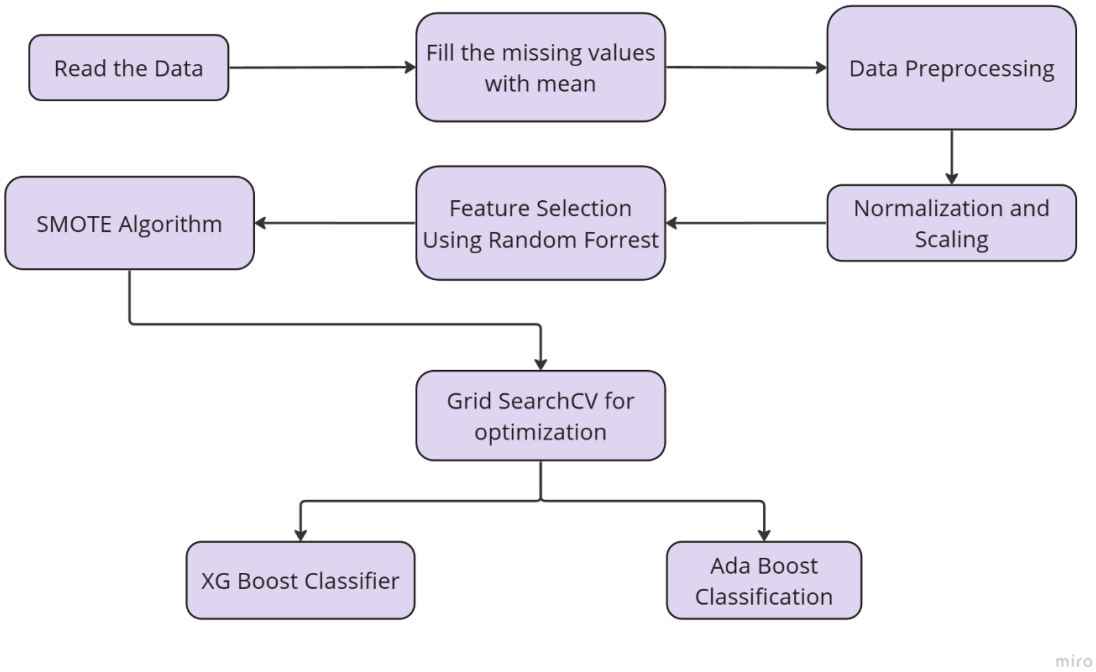
3.3.4 Identifying Top Features

After training the Random Forest classifier, we extract the feature importances to pinpoint the top features. These are the features that contribute the most to predicting the presence or absence of heart disease. By prioritizing these top features, we construct a more robust and interpretable model tailored for heart disease classification. In our quest to refine the model, we employed feature selection as a means to enhance its performance. By reducing the initial set of 13 features to a more manageable 11, we aimed to improve the model's efficiency and interpretability. The selected features, namely 'cp', 'ca', 'oldpeak', 'thalach', 'thal', 'age', 'trestbps', 'chol', 'exang', 'slope', and 'sex', were identified as the most influential in predicting the presence or absence of heart disease. Through this process, we sought to streamline the model while retaining the essential information necessary for accurate classification.

3.3.5 DATA AUGMENTATION

The next step involved applying the Synthetic Minority Over-sampling Technique (SMOTE) to address class imbalance within the dataset. SMOTE works by generating synthetic samples for the minority class (in this case, instances of heart disease) to rebalance the distribution of classes. By creating synthetic instances of the minority class, SMOTE aims to ensure that both classes are represented more evenly in the dataset. This helps prevent the model from being biased towards the majority class and improves its ability to accurately predict both classes.

The next step is to split the dataset into training, validation, and test sets. This process is crucial for building and evaluating machine learning models effectively. The training set is used to train the model, the validation set is utilized for tuning hyperparameters and assessing model performance during training, and the test set serves as an independent dataset to evaluate the model's final performance after training. This partitioning ensures that the model's performance estimates are reliable and can generalize well to unseen data.



3.3 Workflow for Proposed methodology

3.3.6 Model Building

XGB

XGBoost (Extreme Gradient Boosting) stands out as an ensemble learning technique that iteratively combines weak learners, typically decision trees, to craft a potent predictive model. Its boosting algorithm sequentially trains new models to rectify the errors made by preceding ones, a process facilitated by gradient boosting. By minimizing a composite

objective function comprising both a loss function (L) and regularization terms (λ), XGBoost optimizes model performance while mitigating the risk of overfitting. This optimization process involves gradient descent, iteratively adjusting model parameters to minimize the objective function

Moreover, XGBoost harnesses parallel and distributed computing to ensure scalability and efficiency, making it suitable for handling large datasets. To fine-tune its hyperparameters and maximize performance, XGBoost often relies on techniques such as GridSearchCV, which exhaustively searches over a predefined parameter grid to identify the optimal combination of hyperparameters. This grid search enables XGBoost to systematically explore various hyperparameter configurations, thereby enhancing its ability to generalize and achieve superior predictive accuracy. In essence, XGBoost's effectiveness lies in its ability to effectively handle complexity, capture nonlinear relationships, and optimize model performance through gradient boosting and hyperparameter tuning with techniques like GridSearchCV.

The best hyperparameters obtained through GridSearchCV for the XGBoost classifier are {'gamma': 0.01, 'reg_alpha': 1, 'reg_lambda': 0.1}. These hyperparameters indicate the values chosen to optimize the model's performance, with 'gamma' representing the minimum loss reduction required to make further splits in the trees, 'reg_alpha' denoting the L1 regularization term, and 'reg_lambda' representing the L2 regularization term.

The XGBoost classifier achieved an accuracy of approximately 86.89% on the test dataset after regularization. This accuracy score reflects the model's ability to correctly classify instances of heart disease based on the selected features and the optimized hyperparameters. Overall, the obtained accuracy demonstrates the effectiveness of the XGBoost algorithm and the regularization techniques in accurately predicting the presence or absence of heart disease in patients.

Ada boost

AdaBoost, or Adaptive Boosting, is a popular ensemble learning method used for classification tasks. It works by sequentially training a series of weak learners, often decision trees, on subsets of the data. Each weak learner focuses on instances that were previously misclassified by the ensemble, adjusting its parameters to improve accuracy.

During training, AdaBoost assigns higher weights to misclassified instances, making them more influential in subsequent iterations. This iterative process continues until a predefined number of weak learners is reached or until a desired level of accuracy is achieved.Once trained, AdaBoost combines the predictions of all weak learners through a weighted voting scheme. The final prediction is determined by the cumulative vote of all weak learners, with more accurate models carrying greater weight.

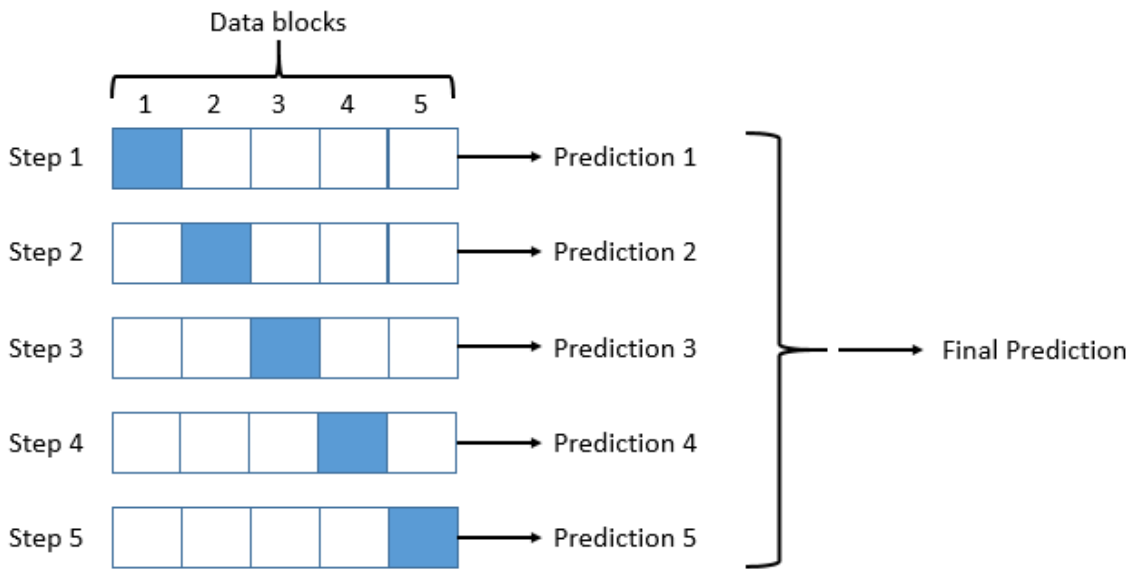
AdaBoost's strength lies in its ability to learn complex decision boundaries and adapt to noisy data. By iteratively focusing on challenging instances, it builds a strong ensemble model capable of generalizing well to unseen data.

AdaBoost is an adaptive ensemble learning technique that constructs a powerful predictive model by sequentially training weak learners and combining their predictions through weighted voting. Its iterative approach and focus on misclassified instances make it effective for a wide range of classification tasks.

The AdaBoost classifier was tuned using GridSearchCV to find the best hyperparameters, resulting in {'learning_rate': 0.5, 'n_estimators': 400}. With these optimized hyperparameters, the AdaBoost classifier achieved an accuracy of approximately 90.85% on the test dataset.

It's noteworthy that both the AdaBoost and XGBoost classifiers in the previous examples utilized GridSearchCV for hyperparameter tuning. This technique systematically searches over a predefined parameter grid to find the optimal combination of hyperparameters, maximizing the model's performance.

5-FOLD CROSS VALIDATION



3.4 5-FOLD CROSS VALIDATION DIAGRAM

The 5-fold cross-validation process involves initially dividing the data into ten groups, with four groups allocated for training and one for testing. Let's denote the validation set as S_val,1 and the training set as S_train,1. In this context, let's use the notation:

Cross-validation often employs stratified random sampling, ensuring that the class proportions in different subsets reflect those in the training set. For instance, let's consider a learning set comprising healthy individuals and those with heart problems (n = 100), with 80 healthy individuals (n^+) and 20 with heart problems (n^-).

Without stratification, random sampling might result in some validation groups exclusively containing healthy individuals. However, stratification guarantees that each validation set in 5-fold cross-validation maintains a balanced ratio, such as approximately 8 cases of healthy individuals and 2 cases of individuals with heart problems, aligning with the class distribution in the training set.

This approach, known as stratified sampling, ensures that the sample percentages reflect the healthy-to-unhealthy ratio, making it a fair representation of the class distribution in the

dataset. Hence, it's a crucial consideration, especially when dealing with datasets comprising both healthy and unhealthy individuals. Moreover, researchers advocate for stratified 5-fold cross-validation to account for the class ratio, ensuring an unbiased assessment of the system's performance.To further mitigate variance in predicted evaluation metrics, cross-validation is often repeated multiple times with different fold configurations.

CHAPTER 4
WEB APP

WEB APPLICATION EXECUTION

In this project, we aimed to develop an accurate machine learning model for predicting heart disease based on electrocardiogram (ECG) data. We followed a systematic approach, including data preprocessing, model training, evaluation, and integration into a web application. Our final model achieved promising results, demonstrating its potential for clinical use.

4.1) Backend Implementation with Flask

To integrate the machine learning model into our web application, we utilized Flask, a Python web framework, for the backend implementation. We created an endpoint within our Flask application to handle incoming requests for heart disease prediction. The endpoint "/api" accepts ECG data as input, preprocesses it, and passes it to the machine learning model for prediction. Upon receiving the prediction result, the backend formats it and sends it back as a response to the client.

- **ENDPOINT**: http://localhost:5000/api (development)
- **ENDPOINT:** https://finalproject-ecg.onrender/api (production)

4.2) Frontend Integration with ReactJS

In the frontend development phase, we utilized ReactJS to create a user-friendly form component for interacting with our heart disease prediction feature. Users can easily upload or input their ECG data through this component. Upon submission, the form sends a request to the Flask backend endpoint we created earlier, ensuring seamless functionality and a smooth user experience.

4.2.1) ReactJS

ReactJS, chosen for its component-based architecture, enables interactive and dynamic frontend development, emphasizing code reusability and modularity.

4.2.2) Bootstrap

Bootstrap, a popular CSS framework, was integrated into our project for efficient frontend development. Its pre-styled components and responsive design

features expedited the process, ensuring consistent and visually appealing interfaces across devices.

4.2.3) Form Handling with Formik

Formik is a library for building forms in React applications. It simplifies the process of managing form state, handling form submission, and validating user input. It simplifies form development with centralized state management, built-in validation, submission handling, and seamless integration with React's Context API.

4.2.4) Chart Visualization with Recharts

Recharts offers customizable chart components for React applications. With support for various chart types, responsive design, data-driven rendering, and interactive features, it simplifies data visualization and enhances user experience.

4.2.5) Axios for API Calling

Axios, a promise-based HTTP client, simplifies asynchronous requests in JavaScript. Utilized for API communication in our project, it offers features like interceptors, error handling, and cancellation.

CHAPTER 5
RESULTS

Our project on heart disease aimed to develop an accurate predictive model leveraging machine learning algorithms. Through extensive data analysis and model training, we achieved promising results in predicting the heart disease based on various risk factors. Our findings demonstrate the potential of machine learning techniques in assisting early detection and prevention efforts for cardiovascular health issues.

5.1) Results Obtained from the Algorithms

ALGORITHMS	ACCURACY OBTAINED (%)
GAUSSIAN NAIVE BAYES	86.2
LOGISTIC REGRESSION	90.6
LINEAR DISCRIMINANT ANALYSIS	89
RANDOM FOREST	90
DUMMY CLASSIFIER	50

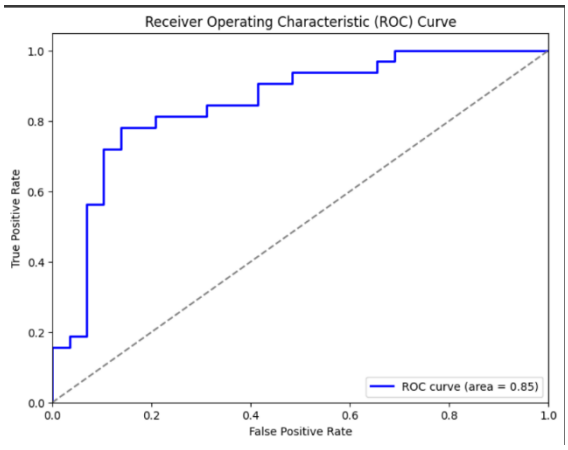
5.1 Results Obtained from the Algorithms

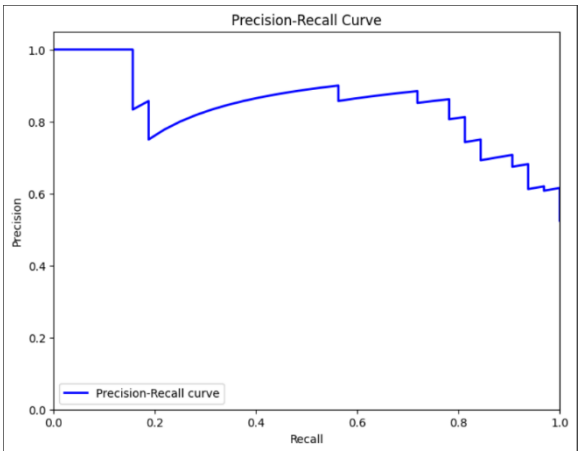
5.2) Novelty Results

ALGORITHMS	ACCURACY OBTAINED (%)
XGB Classifier using Grid Search CV	87
Adaboost Classifier using Grid Search CV	90.8

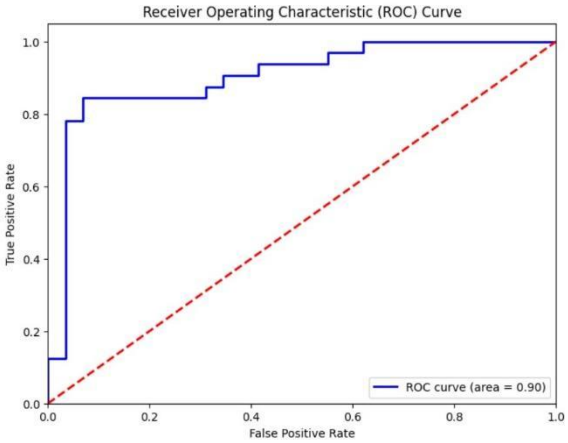
5.2 Novelty Results

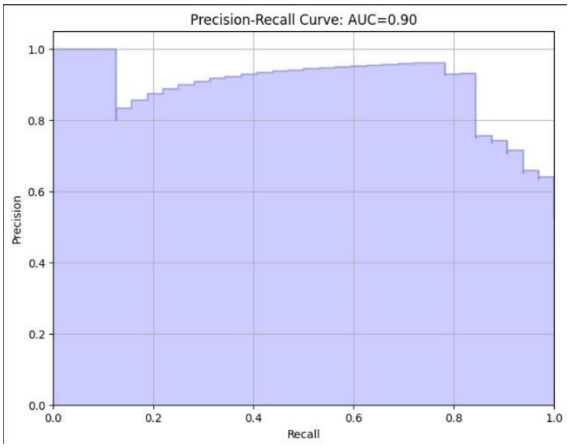
5.1 ROC curve and precision-recall curve for AdaBoosting





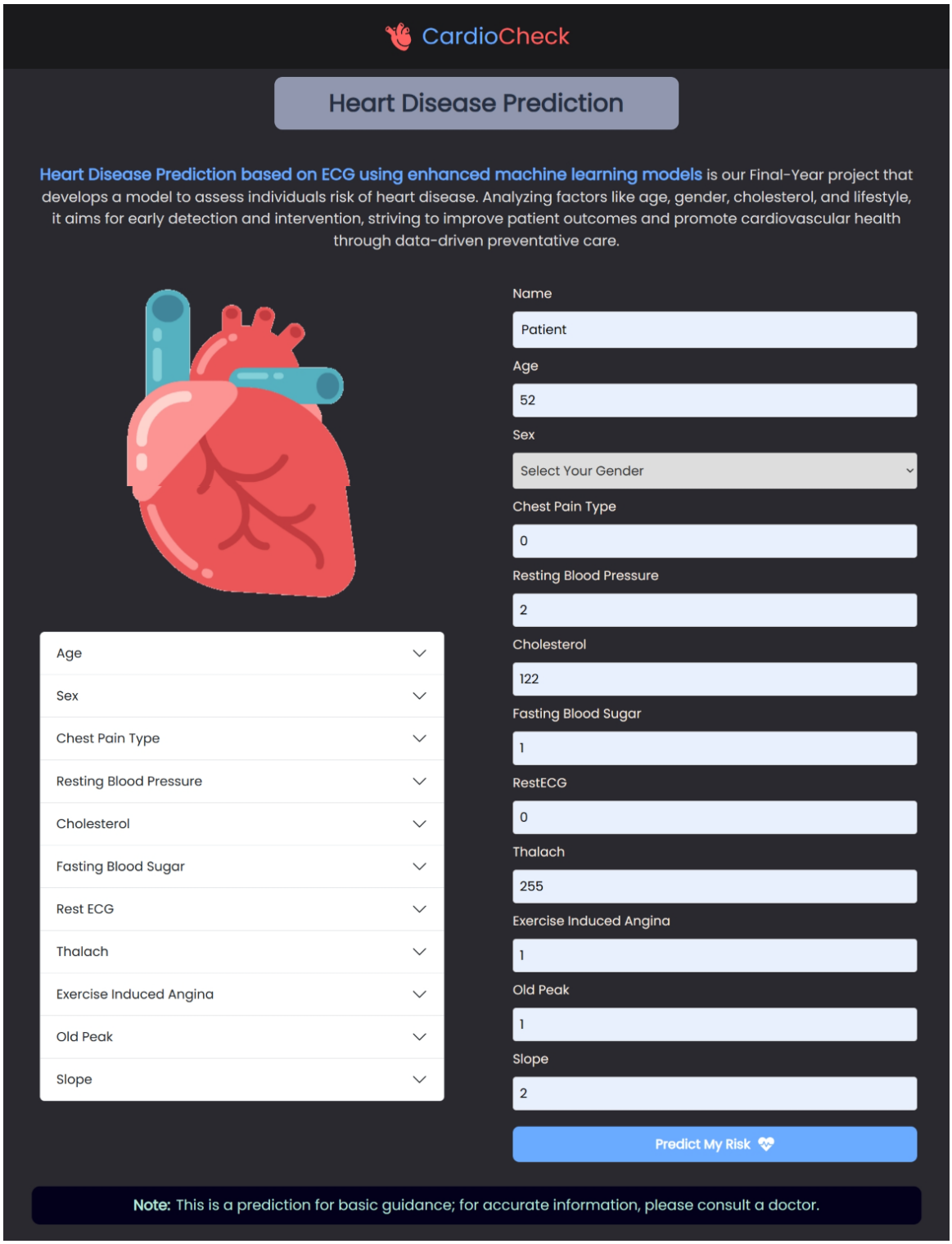
5.2ROC curve and precision-recall curve for Extreme Gradient Boosting

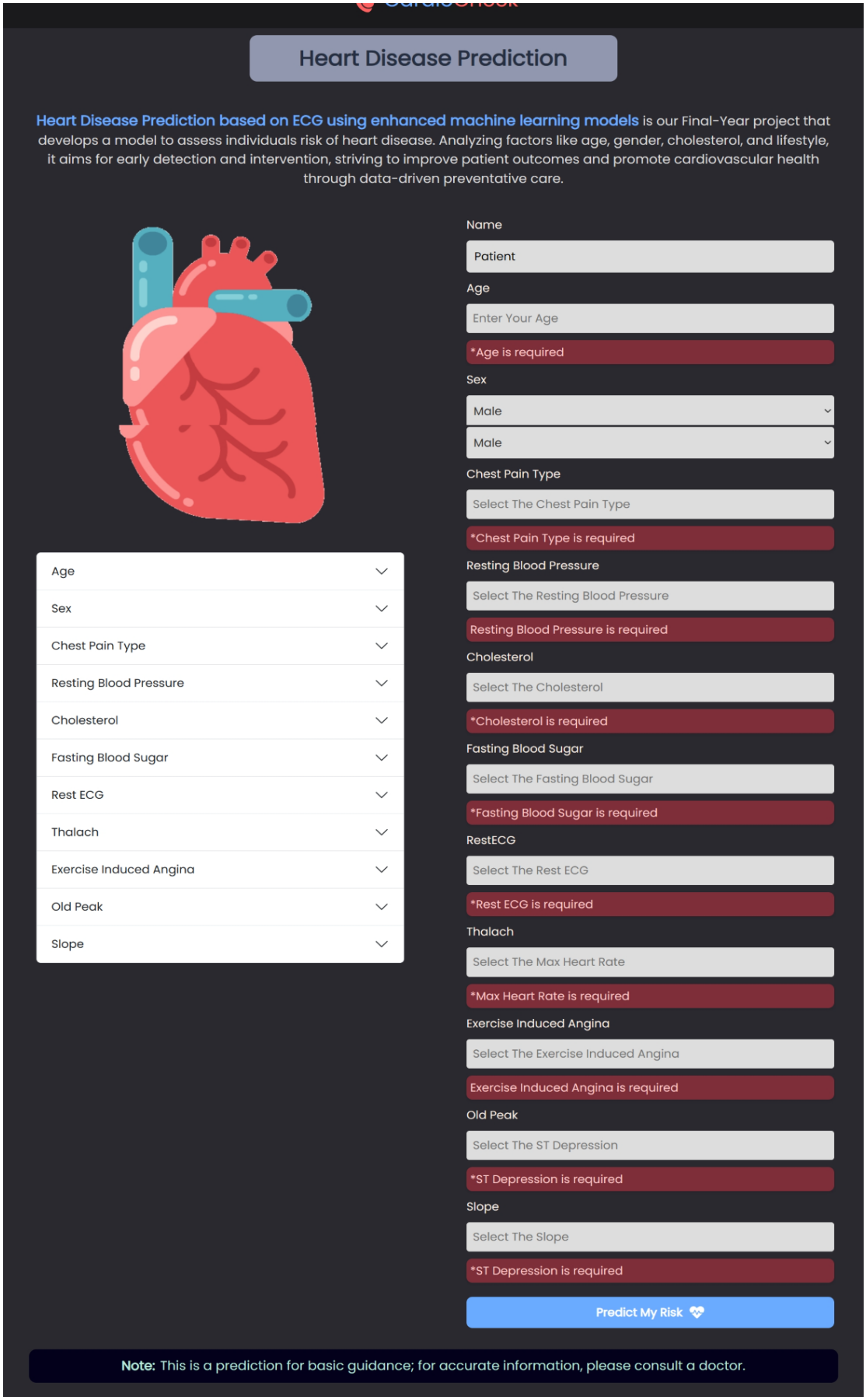




Web Application User Interface and Results

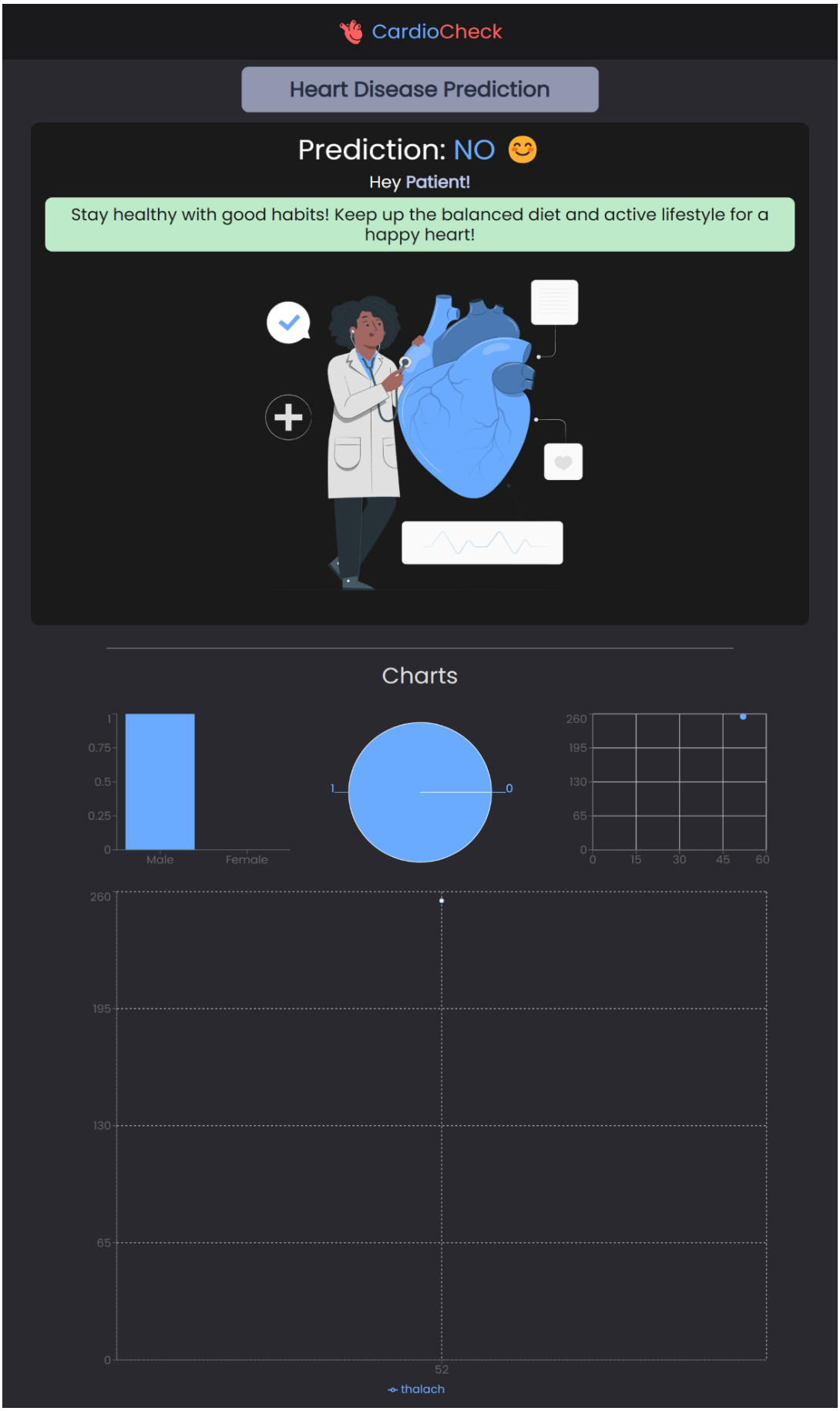
5.3 Home Page



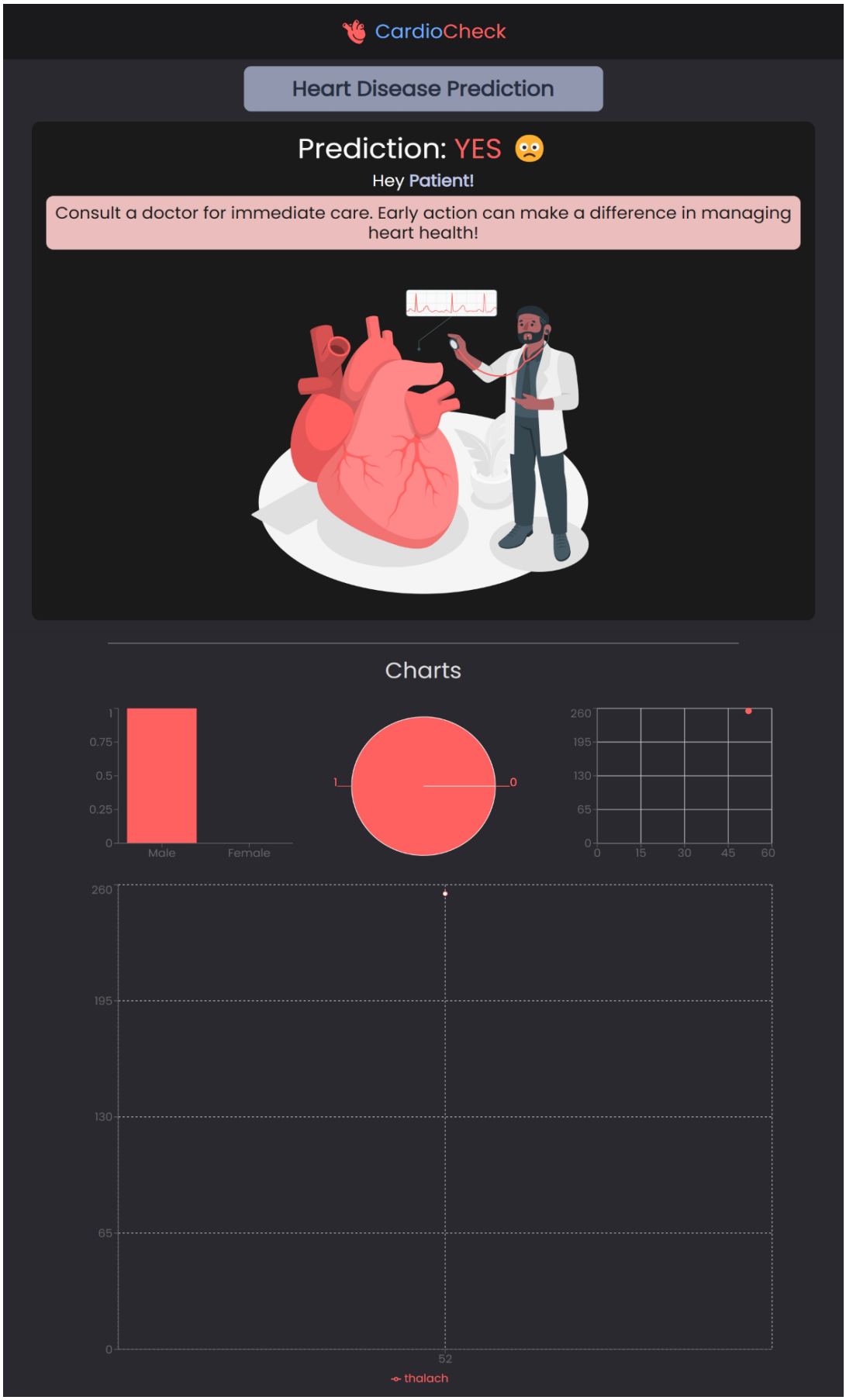


5.4_Home Page with Validation

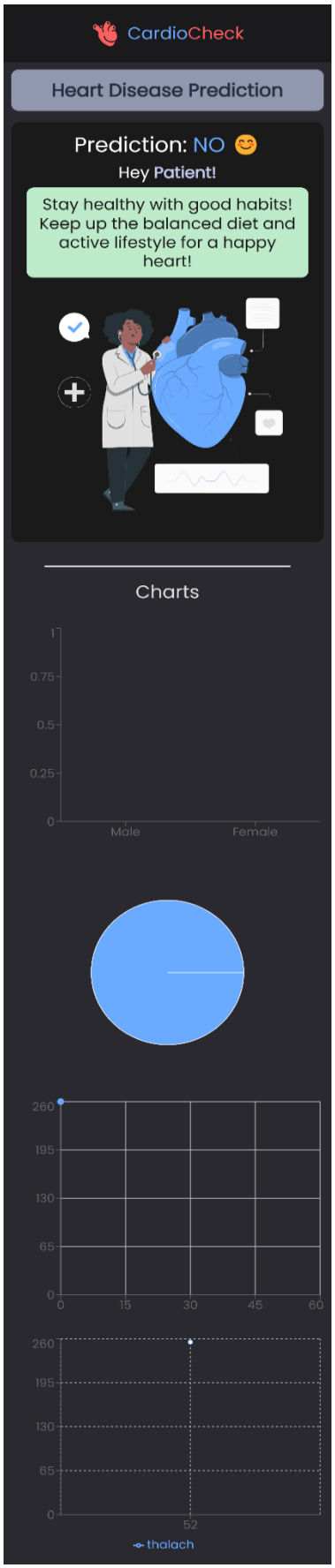
5.5 Prediction Page (NO)

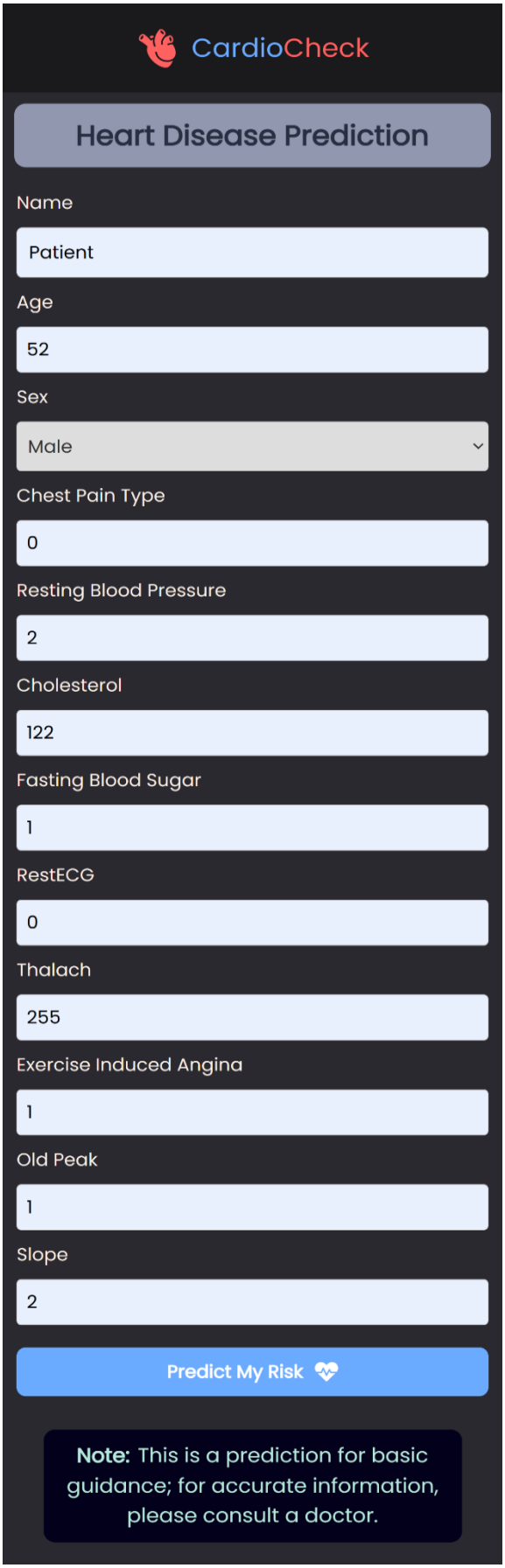


5.6 Prediction Page (YES)



5.7 Responsive Pages





CHAPTER 6

CONCLUSION

In this study, a dataset comprising ECGs from 302 individuals, including both healthy subjects and those with cardiovascular diseases, was analysed using machine learning algorithms. The algorithms employed for this analysis included Dummy Classifier, Logistic Regression, Random Forest, Gaussian Naïve Bayes, Linear Discriminant Analysis, XGBoost and AdaBoost. The objective was to accurately classify individuals as either healthy or suffering from heart disease based on their ECG data.

Among the algorithms utilized, AdaBoost emerged as the most effective in distinguishing between healthy individuals and those with cardiovascular conditions.

FURTHER WORKS

6.1) Semi-Supervised Learning

In the context of diagnosing heart disease, semi-supervised learning could be applied when we have a limited amount of abelled data (e.g., medical records with diagnoses confirmed by experts) but a larger pool of unlabeled data (e.g., additional medical records without confirmed diagnoses).

Approach

6.1.1 Labeled Data

Train the model using the small set of labeled data, where each record includes features such as age, cholesterol levels, blood pressure, etc., along with the corresponding diagnosis (e.g., presence or absence of heart disease).

6.1.2 Unlabeled Data

Utilize the larger set of unlabeled data to further train the model. This can be achieved through techniques like self-training, where the model makes predictions on the unlabeled data and uses confident predictions to pseudo-label the data, effectively expanding the labeled dataset.

6.1.3 Model Training

Fine-tune the model using the combined labeled and pseudo-labeled data, leveraging the additional information from the unlabeled data to improve the model's performance.

6.1.4 Evaluation

Validate the model's performance on a separate test set to assess its accuracy in diagnosing heart disease.

6.2) Self-Supervised Learning

Self-supervised learning in the context of heart disease diagnosis involves training a model to predict certain aspects of the input data without relying on explicitly labeled diagnoses.

Pretext Task Example

6.2.1 Input Data

Consider a dataset of electrocardiogram (ECG) signals, which are sequences of voltage measurements over time recorded from the heart.

6.2.2 Pretext Task

Design a pretext task where the model is trained to predict the future values of the ECG signal based on the past values. This task does not require explicitly labeled diagnoses but still encourages the model to learn meaningful features from the ECG data.

6.2.3 Feature Learning

By solving this pretext task, the model learns to extract relevant features from the ECG signals that capture important patterns and characteristics indicative of heart disease.

6.2.4 Transfer Learning:

Transfer the learned representations to a downstream classification task, such as diagnosing heart disease based on ECG signals. The features learned through self-supervised learning can enhance the model's performance on the classification task.

CHAPTER 7

REFERENCES

[1] E.J. Benjamin, S.S. Virani, C.W. Callaway, A.M. Chamberlain, A.R. Chang, S. Cheng, S.E. Chiuve, M. Cushman, F.N. Delling, R. Deo, et al., heart disease and stroke statistics—2018 update: A report from the American Heart Association, Circulation 137 (2018) e67–e492.

[2] D. Gupta, B. Bajpai, G. Dhiman, M. Soni, S. Gomathi, D. Mane, Review of ECG arrhythmia classification using deep neural network, Mater. Today Proc. 2021. In Press.

[3] World Health Organization. Global Status Report on Noncommunicable Diseases; WHO: Geneva, Switzerland, 2014.

[4] F. Bogun, D. Anh, G. Kalahasty, E. Wissner, C.B. Serhal, R. Bazzi, W.D. Weaver, C. Schuger, Misdiagnosis of atrial fibrillation and its clinical consequences, Am. J. Med. 117 (2004) 636–642.

[5] J. Schl¨ apfer, H.J. Wellens, Computer-interpreted electrocardiograms: Benefits and limitations, J. Am. Coll. Cardiol. 70 (2017) 1183–1192.

[6] E.H. Houssein, M. Kilany, A.E. Hassanien, ECG signals classification: A review, Int. J. Intell. Eng. Informatics 5 (2017) 376–396.

[7] S.H. Jambukia, K.D. Vipul, B.P. Harshadkumar, Classification of ECG signals using machine learning techniques: A survey, In Proceedings of the 2015 International Conference on Advances in Computer Engineering and Applications, Ghaziabad, India, 19–20 March 2015.

[8] P.W. Macfarlane, B. Macfarlane, E. Clark, The university of Glasgow (Uni-G) ECG analysis program., in: Proceedings of the Computers in Cardiology, Lyon, France, 25–28 September 2005.

[9] J. Wang, X. Qiao, C. Liu, X. Wang, Y. Liu, L. Yao, H. Zhang, Automated ECG classification using a nonlocal convolutional block attention module, Comput. Methods Programs Biomed. 203 (2021), 106006.

[10] A. Goldberger, L.A. Amaral, L. Glass, J.M. Hausdorff, P.C. Ivanov, R.G. Mark, J. E. Mietus, G.B. Moody, C.K. Peng, H.E. Stanley, et al., PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals, Circulation 101 (2000).

[11] P. Wagner, N. Strodthoff, R. Bousseljot, W. Samek, T. Schaeffter, PTB-XL, a large publicly available electrocardiography dataset (version 1.0.1), Sci. Data 7 (2020) 1–5.

[12] Jia, W.; Xu, X.; Xu, X.; Sun, Y.; Liu, X. Automatic Detection and Classification of 12- lead ECGs Using a Deep Neural Network. In Proceedings of the Computing in Cardiology, Rimini, Italy, 13–16 September 2020; pp. 1–4.

[13] Z. Zhu, X. Lan, T. Zhao, Y. Guo, P. Kojodjojo, Z. Xu, Z. Liu, S. Liu, H. Wang, X. Sun, et al., Identification of 27 abnormalities from multi-lead ECG signals: An ensembled SE_ResNet framework with sign loss function, Physiol. Meas. 42 (2021), 065008.

[14] Strodthoff, N.; Wagner, P.; Schaeffter, T.; Samek, W. Deep learning for ECG analysis: Benchmarks and insights from PTB-XL. arXiv 2020, arXiv:2004.13701.

[15] Smisek, R.; Nemcova, A.; Marsanova, L.; Smital, L.; Vitek, M.; Kozumplik, J. Cardiac Pathologies Detection and Classification in 12-lead ECG. In Proceedings of the Computing in Cardiology, Rimini, Italy, 13–16 September 2020; pp. 1–4.

[16] D. Zhang, S. Yang, X. Yuan, P. Zhang, Interpretable deep learning for automatic diagnosis of 12-lead electrocardiogram, Iscience 4 (2021), 102373.

[17] Warrick, P.A.; Lostanlen, V.; Eickenberg, M.; And´ en, J.; Homsi, M.N. Arrhythmia Classification of 12-lead Electrocardiograms by Hybrid Scattering-LSTM Networks. In Proceedings of the Computing in Cardiology, Rimini, Italy, 13–16 September 2020; pp. 1–4.

[18] Acharya, U.R.; Fujita, H.; Adam, M.; Lih, O.S.; Hong, T.J.; Sudarshan, V.K.; Koh, J. E. Automated characterization of arrhythmias using nonlinear features from tachycardia ECG beats. In Proceedings of the 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Budapest, Hungary, 9–12 October 2016.

[19] Y.Y. Jo, Y. Cho, S.Y. Lee, J.M. Kwon, K.H. Kim, K.H. Jeon, S. Cho, J. Park, B.H. Oh, Explainable artificial intelligence to detect atrial fibrillation using electrocardiogram, Int. J. Cardiol. 328 (2021) 104–110.

[20] Lepek, M.; Pater, A.; Muter, K.; Wiszniewski, P.; Kokosi´nska, D.; Salamon, J.; Puzio, Z. 12-lead ECG Arrythmia Classification Using Convolutional Neural Network for Mutually Non-Exclusive Classes. In Proceedings of the Computing in Cardiology, Rimini, Italy, 13–16 September 2020; pp. 1–4.

[21] E. Ramaraj, S.C. Virgeniya, A Novel Deep Learning based Gated Recurrent Unit with Extreme Learning Machine for Electrocardiogram (ECG) Signal Recognition, Biomed. Signal Process. Control 68 (2021), 102779.

[22] Y. Xiao, H. Yin, Y. Zhang, H. Qi, Y. Zhang, Z. Liu, A dual-stage attention-based Conv-LSTM network for spatio-temporal correlation and multivariate time series prediction, Int. J. Intell. Syst. 36 (5) (2021) 2036–2057.

[23] Y. Xiao, K. Xia, H. Yin, Y.D. Zhang, Z. Qian, Z. Liu, Y. Liang, X. Li, AFSTGCN: Prediction for multivariate time series using an adaptive fused spatial-temporal graph convolutional network, Digital Communications and Networks. (2022).

[24] Q. Cheng, Y. Chen, Y. Xiao, H. Yin, W. Liu, A dual-stage attention-based Bi-LSTM network for multivariate time series prediction, J. Supercomput. 78 (14) (2022) 16214–16235.

[25] Z.K. Gao, M. Small, J. Kurths, Complex network analysis of time series, Europhys. Lett. 116 (5) (2017) 50001.

[26] C. Nadeau, Y. Bengio, Inference for the generalization error, Mach. Learn. 52 (2003) 239–281.

[27] R. Bouckaert, E. Frank, Evaluating the replicability of significance tests for comparing learning algo- rithms, Advances in Knowledge Discovery and Data Mining 3056 (2004) 3–12.

[28] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in:Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI'95,Morgan Kaufmann Publishers Inc., 1995, pp. 1137–1143.

[29] S.M. Malakouti, A.R. Ghiasi, A.A. Ghavifekr, P. Emami, Predicting wind power generation using machine learning and CNN-LSTM approaches, Wind Eng. 46 (6) (2022) 1853–1869.

[30] L. Zou, S. Yu, T. Meng, Z. Zhang, X. Liang, Y. Xie, A Technical Review of Convolutional Neural Network-Based Mammographic Breast Cancer Diagnosis, Comput Math Methods Med. (2019).

[31] M.H. Yap, G. Pons, J. Martí, S. Ganau, M. Sentís, R. Zwiggelaar, et al., Automated breast ultrasound lesions detection using convolutional neural networks, IEEE J Biomed Heal Informatics. 22 (4) (2018) 1218–1226.

[32] X. Qi, L. Zhang, Y. Chen, Y. Pi, Y. Chen, Q. Lv, et al., Automated diagnosis of breast ultrasonography images using deep neural networks, Med Image Anal [Internet]. 52 (2019) 185–198.

[33] Malakouti SM, Ghiasi AR. Evaluation of the application of computational model machine learning methods to simulate wind speed in predicting the production capacity of the Swiss basel wind farm. In2022 26th International Electrical Power Distribution Conference (EPDC) 2022 May 11 (pp. 31-36). IEEE.

[34] S.M. Malakouti, Utilizing time series data from 1961 to 2019 recorded around the world and machine learning to create a Global Temperature Change Prediction Model, Case Studies in Chemical and Environmental Engineering. 6 (2023), 100312.

[35] S.M. Malakouti, Use machine learning algorithms to predict turbine power generation to replace renewable energy with fossil fuels, Energy Explor. Exploit. 14 (2022).

[36] S.M. Malakouti, A.R. Ghiasi, A.A. Ghavifekr, AERO2022-flying danger reduction for quadcopters by using machine learning to estimate current, voltage, and flight area. e-Prime-Advances in Electrical Engineering, Electronics and Energy. 1 (2) (2022), 100084.

[37] S.M. Malakouti, Estimating the output power and wind speed with ML methods: A case study in Texas, Case Studies in Chemical and Environmental Engineering. 28 (2023), 100324.

[38] Discriminate primary gammas (signal) from the images of hadronic showers by cosmic rays in the upper atmosphere (background) with machine learning, https:// doi.org/10.1088/1402-4896/acc1b2.

[39] Seyed Matin Malakouti, Heart disease classification based on ECG using machine learning models.

CHAPTER 8

Appendix

SOURCE CODE

Existing Implementation

+ Code + Text

Importing necesscary libraries

```
[ ] import pandas as pd
    import numpy as np
    import warnings
    from sklearn.model_selection import cross_val_score, train_test_split
    from sklearn.preprocessing import StandardScaler, MinMaxScaler
    from sklearn.impute import SimpleImputer
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.naive_bayes import GaussianNB
    from sklearn.linear_model import LogisticRegression
    from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
    from sklearn.dummy import DummyClassifier
    from sklearn.metrics import accuracy_score, classification_report
    import matplotlib.pyplot as plt
    import seaborn as sns
```

```
[ ] data = pd.read_csv("heart disease classification dataset.csv")
```

```
[ ] data
```

	Unnamed: 0	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	0	63	male	3	145.0	233.0	1	0	150.0	0	2.3	0	0	1	yes
1	1	37	male	2	130.0	250.0	0	1	187.0	0	3.5	0	0	2	yes
2	2	41	female	1	130.0	204.0	0	0	172.0	0	1.4	2	0	2	yes
3	3	56	male	1	120.0	236.0	0	1	178.0	0	0.8	2	0	2	yes
4	4	57	female	0	NaN	354.0	0	1	163.0	1	0.6	2	0	2	yes
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	298	57	female	0	140.0	241.0	0	1	123.0	1	0.2	1	0	3	no
299	299	45	male	3	110.0	264.0	0	1	132.0	0	1.2	1	0	3	no
300	300	68	male	0	144.0	193.0	1	1	141.0	0	3.4	1	2	3	no
301	301	57	male	0	NaN	131.0	0	1	115.0	1	1.2	1	1	3	no
302	302	57	female	1	130.0	236.0	0	0	174.0	0	0.0	1	1	2	no

303 rows × 15 columns

```
numeric_columns = data.select_dtypes(include='number')
data[numeric_columns.columns] = data[numeric_columns.columns].fillna(numeric_columns.mean())
```

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

# Display columns with missing values, if any
print(missing_values[missing_values > 0])
```

```
Series([], dtype: int64)
```

```
[ ] # Convert 'sex' column to numerical format (0 for female, 1 for male)
    data['sex'] = data['sex'].map({'female': 0, 'male': 1})

    # Convert 'target' column to numerical format (0 for no heart disease, 1 for heart disease)
    data['target'] = data['target'].map({'no': 0, 'yes': 1})

    # Display the first few rows of the dataset after transformation
    print(data.head())
```

```
   Unnamed: 0  age  sex  cp    trestbps   chol  fbs  restecg  thalach  exang  \
0           0   63    1   3  145.000000  233.0    1        0    150.0      0
1           1   37    1   2  130.000000  250.0    0        1    187.0      0
2           2   41    0   1  130.000000  204.0    0        0    172.0      0
3           3   56    1   1  120.000000  236.0    0        1    178.0      0
4           4   57    0   0  131.712375  354.0    0        1    163.0      1

   oldpeak  slope  ca  thal  target
0      2.3      0   0     1       1
1      3.5      0   0     2       1
2      1.4      2   0     2       1
3      0.8      2   0     2       1
4      0.6      2   0     2       1
```

```
print(data.columns)
```

```
Index(['Unnamed: 0', 'age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg',
       'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

+ Code + Text

```
# Remove the 'Unnamed: 0' column
data.drop('Unnamed: 0', axis=1, inplace=True)

# Display the columns after removal
print(data.columns)
```

```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

# Normalize the data
data_normalized = scaler.fit_transform(data)
data_normalized = pd.DataFrame(data_normalized, columns=data.columns)
print(data_normalized.head())
```

```
        age       sex        cp      trestbps      chol       fbs   restecg  \
0  0.952197  0.681005  1.973123  7.600324e-01 -0.257417  2.394438 -1.005832
1 -1.915313  0.681005  1.002577 -9.794528e-02  0.071170 -0.417635  0.898962
2 -1.474158 -1.468418  0.032031 -9.794528e-02 -0.817947 -0.417635 -1.005832
3  0.180175  0.681005  0.032031 -6.699304e-01 -0.199431 -0.417635  0.898962
4  0.290464 -1.468418 -0.938515 -1.625679e-15  2.081349 -0.417635  0.898962

    thalach     exang   oldpeak     slope        ca      thal    target
0  0.006009 -0.696631  1.087338 -2.274579 -0.714429 -2.148873  0.914529
1  1.662292 -0.696631  2.122573 -2.274579 -0.714429 -0.512922  0.914529
2  0.990826 -0.696631  0.310912  0.976352 -0.714429 -0.512922  0.914529
3  1.259413 -0.696631 -0.206705  0.976352 -0.714429 -0.512922  0.914529
4  0.587946  1.435481 -0.379244  0.976352 -0.714429 -0.512922  0.914529
```

```
num_rows = data.shape[0]
print("Number of rows in data:", num_rows)
```

```
Number of rows in data: 303
```

```
from sklearn.model_selection import train_test_split


data = data.iloc[1:]

# Split the data into features (X) and target variable (y)
X = data.drop('target', axis=1)
y = data['target']

# Split the data into training, validation, and test sets
X_train, X_remaining, y_train, y_remaining = train_test_split(X, y, train_size=152, random_state=42)

# Then, split the remaining data into validation and test sets
X_val, X_test, y_val, y_test = train_test_split(X_remaining, y_remaining, test_size=75, random_state=42)

print("Shape of X_train:", X_train.shape)
print("Shape of X_val:", X_val.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_val:", y_val.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (152, 13)
Shape of X_val: (75, 13)
Shape of X_test: (75, 13)
Shape of y_train: (152,)
Shape of y_val: (75,)
Shape of y_test: (75,)
```

GaussianNB

```
[ ] # Define the best hyperparameters
    best_params = {'var_smoothing': 1e-09}

    # Create an instance of the GaussianNB classifier with the best hyperparameters
    best_gnb = GaussianNB(var_smoothing=best_params['var_smoothing'])

    # Fit the model on the training data
    best_gnb.fit(X_train, y_train)

    # Use the trained model to make predictions on the validation set
    y_val_pred = best_gnb.predict(X_val)

    # Evaluate the performance of the model on the validation set
    accuracy_val = accuracy_score(y_val, y_val_pred)
    print("Accuracy on validation set:", accuracy_val)

    # Use the trained model to make predictions on the test set
    y_test_pred = best_gnb.predict(X_test)

    # Evaluate the performance of the model on the test set
    accuracy_test = accuracy_score(y_test, y_test_pred)
    print("Accuracy on test set:", accuracy_test)
```

```
Accuracy on validation set: 0.8666666666666667
Accuracy on test set: 0.8533333333333334
```

LogisticRegression

```
[ ] # Best hyperparameters found
    best_params = {'C': 1, 'solver': 'liblinear'}

    # Create an instance of the LogisticRegression classifier with the best hyperparameters
    best_log_reg = LogisticRegression(**best_params, max_iter=1000)

    # Fit the model on the training data
    best_log_reg.fit(X_train, y_train)

    # Use the best model to make predictions on the validation set
    y_val_pred = best_log_reg.predict(X_val)

    # Evaluate the performance of the best model on the validation set
    accuracy_val = accuracy_score(y_val, y_val_pred)
    print("Accuracy on validation set :", accuracy_val)

    # Use the best model to make predictions on the test set
    y_test_pred = best_log_reg.predict(X_test)

    # Evaluate the performance of the best model on the test set
    accuracy_test = accuracy_score(y_test, y_test_pred)
    print("Accuracy on test set :", accuracy_test)
```

```
Accuracy on validation set : 0.84
Accuracy on test set : 0.9066666666666666
```

RandomForestClassifier

```
[ ] # Define the best hyperparameters
    best_params = {
        'n_estimators': 100,  # Number of trees in the forest
        'max_depth': None,    # Maximum depth of the tree
        'min_samples_split': 10,  # Minimum number of samples required to split an internal node
        'min_samples_leaf': 2   # Minimum number of samples required to be at a leaf node
    }

    # Create an instance of the RandomForestClassifier with best hyperparameters
    best_rf_clf = RandomForestClassifier(**best_params, random_state=42)

    # Train the model on training data
    best_rf_clf.fit(X_train, y_train)

    # Use the best model to make predictions on the validation set
    y_val_pred = best_rf_clf.predict(X_val)

    # Evaluate the performance of the best model on the validation set
    accuracy_val = accuracy_score(y_val, y_val_pred)
    print("Accuracy on validation set (with best hyperparameters):", accuracy_val)

    # Use the best model to make predictions on the test set
    y_test_pred = best_rf_clf.predict(X_test)

    # Evaluate the performance of the best model on the test set
    accuracy_test = accuracy_score(y_test, y_test_pred)
    print("Accuracy on test set (with best hyperparameters):", accuracy_test)
```

```
Accuracy on validation set (with best hyperparameters): 0.8266666666666667
Accuracy on test set (with best hyperparameters): 0.8933333333333333
```

LinearDiscriminantAnalysis

```
[ ] # Define the best hyperparameters
    best_params = {
        'solver': 'lsqr',       # Best solver found
        'shrinkage': 'auto',    # Best shrinkage parameter found
        'n_components': None    # Best number of components found
    }

    # Create an instance of the LinearDiscriminantAnalysis classifier with the best hyperparameters

    best_lda = LinearDiscriminantAnalysis(**best_params)

    # Fit the classifier on the training data
    best_lda.fit(X_train, y_train)

    # Use the best model to make predictions on the validation set
    y_val_pred = best_lda.predict(X_val)

    # Evaluate the performance of the best model on the validation set
    accuracy_val = accuracy_score(y_val, y_val_pred)
    print("Accuracy on validation set (with best hyperparameters):", accuracy_val)

    # Use the best model to make predictions on the test set
    y_test_pred = best_lda.predict(X_test)

    # Evaluate the performance of the best model on the test set
    accuracy_test = accuracy_score(y_test, y_test_pred)
    print("Accuracy on test set (with best hyperparameters):", accuracy_test)
```

```
Accuracy on validation set (with best hyperparameters): 0.8266666666666667
Accuracy on test set (with best hyperparameters): 0.88
```

DummyClassifier

```
[ ] # Create an instance of the DummyClassifier
    dummy_clf = DummyClassifier(strategy="most_frequent")

    # Fit the classifier to the training data
    dummy_clf.fit(X_train, y_train)

    # Predict target values for the validation and test sets
    y_val_pred = dummy_clf.predict(X_val)
    y_test_pred = dummy_clf.predict(X_test)

    # Evaluate the performance of the classifier
    accuracy_val = accuracy_score(y_val, y_val_pred)
    accuracy_test = accuracy_score(y_test, y_test_pred)

    print("Accuracy on validation set:", accuracy_val)
    print("Accuracy on test set:", accuracy_test)
```

```
Accuracy on validation set: 0.5066666666666667
Accuracy on test set: 0.5466666666666666
```

Novelty

NOVELTY APPROACH

```
import pandas as pd
import numpy as np
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score, classification_report
import warnings
warnings.filterwarnings("ignore")
```

```
[ ] data = pd.read_csv("heart disease classification dataset.csv")
```

```
[ ] data
```

	Unnamed: 0	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	0	63	male	3	145.0	233.0	1	0	150.0	0	2.3	0	0	1	yes
1	1	37	male	2	130.0	250.0	0	1	187.0	0	3.5	0	0	2	yes
2	2	41	female	1	130.0	204.0	0	0	172.0	0	1.4	2	0	2	yes
3	3	56	male	1	120.0	236.0	0	1	178.0	0	0.8	2	0	2	yes
4	4	57	female	0	NaN	354.0	0	1	163.0	1	0.6	2	0	2	yes
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	298	57	female	0	140.0	241.0	0	1	123.0	1	0.2	1	0	3	no
299	299	45	male	3	110.0	264.0	0	1	132.0	0	1.2	1	0	3	no

```
[ ] numeric_columns = data.select_dtypes(include='number')
    data[numeric_columns.columns] = data[numeric_columns.columns].fillna(numeric_columns.mean())
```

```
[ ] # Check for missing values in all columns
    missing_values = data.isnull().sum()

    # Display columns with missing values, if any
    print(missing_values[missing_values > 0])
```

```
Series([], dtype: int64)
```

```
[ ] # Convert 'sex' column to numerical format (0 for female, 1 for male)
    data['sex'] = data['sex'].map({'female': 0, 'male': 1})

    # Convert 'target' column to numerical format (0 for no heart disease, 1 for heart disease)
    data['target'] = data['target'].map({'no': 0, 'yes': 1})

    # Display the first few rows of the dataset after transformation
    print(data.head())
```

```
   Unnamed: 0  age  sex  cp    trestbps   chol  fbs  restecg  thalach  exang  \
0           0   63    1   3  145.000000  233.0    1        0    150.0      0
1           1   37    1   2  130.000000  250.0    0        1    187.0      0
2           2   41    0   1  130.000000  204.0    0        0    172.0      0
3           3   56    1   1  120.000000  236.0    0        1    178.0      0
4           4   57    0   0  131.712375  354.0    0        1    163.0      1

   oldpeak  slope  ca  thal  target
0      2.3      0   0     1       1
1      3.5      0   0     2       1
2      1.4      2   0     2       1
3      0.8      2   0     2       1
4      0.6      2   0     2       1
```

```
[ ] print(data.columns)
```

```
Index(['Unnamed: 0', 'age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg',
       'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

```
[ ] # Remove the 'Unnamed: 0' column
    data.drop('Unnamed: 0', axis=1, inplace=True)

    # Display the columns after removal
    print(data.columns)
```

```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

```
[ ] from sklearn.preprocessing import MinMaxScaler
    scaler = MinMaxScaler()

    # Normalize the data
    data_normalized = scaler.fit_transform(data)
    data_normalized = pd.DataFrame(data_normalized, columns=data.columns)
    print(data_normalized.head())
```

```
        age  sex        cp  trestbps      chol  fbs  restecg   thalach  exang  \
0  0.708333  1.0  1.000000  0.481132  0.244292  1.0      0.0  0.603053    0.0
1  0.166667  1.0  0.666667  0.339623  0.283105  0.0      0.5  0.885496    0.0
2  0.250000  0.0  0.333333  0.339623  0.178082  0.0      0.0  0.770992    0.0
3  0.562500  1.0  0.333333  0.245283  0.251142  0.0      0.5  0.816794    0.0
4  0.583333  0.0  0.000000  0.355777  0.520548  0.0      0.5  0.702290    1.0

    oldpeak  slope   ca      thal  target
0  0.370968    0.0  0.0  0.333333     1.0
1  0.564516    0.0  0.0  0.666667     1.0
2  0.225806    1.0  0.0  0.666667     1.0
3  0.129032    1.0  0.0  0.666667     1.0
4  0.096774    1.0  0.0  0.666667     1.0
```

```
[ ] num_rows = data.shape[0]
    print("Number of rows in data:", num_rows)
```

```
Number of rows in data: 303
```

```
[ ] import pandas as pd
    from sklearn.model_selection import train_test_split, GridSearchCV
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.metrics import accuracy_score, classification_report
```

```
[ ] import pandas as pd
    from sklearn.ensemble import RandomForestClassifier

    # Assuming your data is stored in a DataFrame called 'data'
    X = data.drop('target', axis=1)  # Features
    y = data['target']  # Target variable

    # Initialize the Random Forest classifier
    rf_classifier = RandomForestClassifier()

    # Fit the classifier to your data
    rf_classifier.fit(X, y)

    # Get feature importances
    feature_importances = pd.Series(rf_classifier.feature_importances_, index=X.columns)

    # Select top 11 features
    top_11_features = feature_importances.nlargest(11)

    # Extract feature names
    selected_features = top_11_features.index.tolist()
```

```
[ ] from sklearn.model_selection import train_test_split



    # Subset the data with selected features
    X_selected = data[selected_features]

    # Split the data into features (X) and target variable (y)
    X = X_selected
    y = data['target']

    # Split the data into training, validation, and test sets
    X_train, X_remaining, y_train, y_remaining = train_test_split(X, y, train_size=152, random_state=42)

    # Then, split the remaining data into validation and test sets
    X_val, X_test, y_val, y_test = train_test_split(X_remaining, y_remaining, test_size=75, random_state=42)

    print("Shape of X_train:", X_train.shape)
    print("Shape of X_val:", X_val.shape)
    print("Shape of X_test:", X_test.shape)
    print("Shape of y_train:", y_train.shape)
    print("Shape of y_val:", y_val.shape)
    print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (152, 11)
Shape of X_val: (76, 11)
Shape of X_test: (75, 11)
Shape of y_train: (152,)
Shape of y_val: (76,)
Shape of y_test: (75,)
```

```
[ ] import pandas as pd
    from sklearn.model_selection import train_test_split, GridSearchCV
    from sklearn.metrics import accuracy_score, classification_report
    from sklearn.preprocessing import StandardScaler
    from imblearn.over_sampling import SMOTE
    import xgboost as xgb



    # Selecting the top 11 features
    selected_features = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope']

    # Split the data into features (X) and target variable (y)
    X = data[selected_features]
    y = data['target']

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

    # Apply SMOTE for data augmentation
    smote = SMOTE(random_state=42)
    X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_train)
```

```python
# Define XGBoost classifier
xgb_classifier = xgb.XGBClassifier(objective='binary:logistic', random_state=42)

# Define hyperparameter grid for regularization parameters
param_grid = {
    'reg_alpha': [0.001, 0.01, 0.1, 1, 10],  # L1 regularization
    'reg_lambda': [0.001, 0.01, 0.1, 1, 10],  # L2 regularization
    'gamma': [0.001, 0.01, 0.1, 1]  # Minimum loss reduction to make further partition on a leaf node
}

# Perform GridSearchCV to find the best regularization parameters
grid_search = GridSearchCV(estimator=xgb_classifier, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train_resampled, y_train_resampled)

# Get the best parameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Initialize XGBoost classifier with the best parameters
xgb_classifier_best = xgb.XGBClassifier(objective='binary:logistic',
                                        random_state=42,
                                        reg_alpha=best_params['reg_alpha'],
                                        reg_lambda=best_params['reg_lambda'],
                                        gamma=best_params['gamma'])

# Fit the classifier on the resampled training data
xgb_classifier_best.fit(X_train_resampled, y_train_resampled)

# Make predictions on the scaled testing data
y_pred_xgb_best = xgb_classifier_best.predict(X_test_scaled)
```

```python
# Calculate accuracy
accuracy_xgb_best = accuracy_score(y_test, y_pred_xgb_best)
print("XGBoost Classifier Accuracy with Regularization:", accuracy_xgb_best)

# Classification report
print("Classification Report for XGBoost Classifier:")
print(classification_report(y_test, y_pred_xgb_best))
```

```
Best Hyperparameters: {'gamma': 0.01, 'reg_alpha': 1, 'reg_lambda': 0.1}
XGBoost Classifier Accuracy with Regularization: 0.8688524590163934
Classification Report for XGBoost Classifier:
              precision    recall  f1-score   support

           0       0.80      0.97      0.88        29
           1       0.96      0.78      0.86        32

    accuracy                           0.87        61
   macro avg       0.88      0.87      0.87        61
weighted avg       0.88      0.87      0.87        61
```

```python
[ ] import pandas as pd
    from sklearn.model_selection import train_test_split, GridSearchCV
    from sklearn.metrics import accuracy_score, classification_report
    from sklearn.preprocessing import StandardScaler
    from imblearn.over_sampling import SMOTE
    from sklearn.ensemble import AdaBoostClassifier
    from sklearn.tree import DecisionTreeClassifier



    # Selecting the top 11 features
    selected_features = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope']

    # Split the data into features (X) and target variable (y)
    X = data[selected_features]
    y = data['target']

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

    # Apply SMOTE for data augmentation
    smote = SMOTE(random_state=42)
    X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_train)

    # Initialize base decision tree classifier with increased max_depth
    base_classifier = DecisionTreeClassifier(max_depth=5)
```

```
[ ] # Initialize AdaBoost classifier
    adaboost_classifier = AdaBoostClassifier(base_estimator=base_classifier, random_state=42)

    # Define parameter grid for GridSearchCV
    param_grid = {
        'n_estimators': [100, 200, 300, 400],  # Increase number of weak learners
        'learning_rate': [0.01, 0.05, 0.1, 0.5, 1]
    }

    # Initialize GridSearchCV
    grid_search = GridSearchCV(estimator=adaboost_classifier, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)

    # Perform GridSearchCV
    grid_search.fit(X_train_resampled, y_train_resampled)

    # Get the best parameters
    best_params = grid_search.best_params_
    print("Best Hyperparameters:", best_params)

    # Initialize AdaBoost classifier with the best parameters
    adaboost_classifier_best = AdaBoostClassifier(base_estimator=base_classifier, **best_params, random_state=42)

    # Fit the classifier on the resampled training data
    adaboost_classifier_best.fit(X_train_resampled, y_train_resampled)

    # Make predictions on the scaled testing data
    y_pred_adaboost_best = adaboost_classifier_best.predict(X_test_scaled)


    # Calculate accuracy after hyperparameter tuning
    accuracy_adaboost_best = accuracy_score(y_test, y_pred_adaboost_best)
    print("Ada Boost Accuracy after Hyperparameter Tuning:", accuracy_adaboost_best)
```

```
# Classification report
print("Classification Report for AdaBoost Classifier:")
print(classification_report(y_test, y_pred_adaboost_best))
```

```
Best Hyperparameters: {'learning_rate': 0.5, 'n_estimators': 400}
Ada Boost Accuracy after Hyperparameter Tuning: 0.9084590163934
Classification Report for AdaBoost Classifier:
              precision    recall  f1-score   support

           0       0.77      0.79      0.78        29
           1       0.81      0.78      0.79        32

    accuracy                           0.79        61
   macro avg       0.79      0.79      0.79        61
weighted avg       0.79      0.79      0.79        61
```