

HEART DISEASE PREDICTION USING MULTI-LAYER PERCEPTRON ALGORITHM

A PROJECT REPORT

Submitted by

SADHAM HUSSAIN K

SANTHOSHKUMAR P

SARAVANAKUMAR K

VIGNESH P

In partial fulfillment for the award of the degree

Of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING



CHRISTIAN COLLEGE OF ENGINEERING AND TECHNOLOGY

ODDANCHATRAM-624619

ANNA UNIVERSITY : CHENNAI 600 025

MAY 2024

ANNA UNIVERSITY : CHENNAI 600025

BONAFIDE CERTIFICATE

Certified that this project report titled “**HEART DISEASE PREDICTION USING MULTI-LAYER PERCEPTRON ALGORITHM**” is the bonafide work of "**SADHAM HUSSAIN.K (920320104033), SANTHOSHKUMAR.P (920320104035), SARAVANAKUMAR.K (920320104036), VIGNESH.P (920320104701)** ” who carried out the project work under my supervision.

SIGNATURE

Dr.C.SUNDAR,M.E.,Ph.D.,

HEAD OF THE DEPARTMENT

Assistant Professor

Department of CSE,

Christian College of Engg & Tech,

Oddanchatram-624619.

SIGNATURE

Ms.P.RAJALAKSHMI, M.E.,

SUPERVISOR

Assistant Professor

Department of CSE,

Christian College of Engg & Tech,

Oddanchatram-624619.

Submitted for the university viva-voice examination held on.....

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We thank the most graceful creator of the universe, our almighty **GOD** who ideally supported us throughout this project.

At this moment of having successfully completed our project, we wish to convey our sincere thanks to the chairman **Prof. Dr. TOM CHERIAN ,MBBS, FRCS, (G SURG), FRCS(I-Collegiate), CCST (UK), Transplant Fellowship, (KINGS, LONDON)** and the management for providing all the facilities to us.

We would like to express our sincere thanks to our principal **Dr.K.PARIMALA GEETHA,M.E, Ph.D.**, for letting us to do our project and offering adequate duration in completing our project.

We would like to express our gratitude to our Head of Department of Computer Science And Engineering, **Dr. C. SUNDAR, M.E., Ph.D.**, for his constructive suggestions during our project.

With deep sense of gratitude, we extend our sincere thanks to our project guide and project coordinator **Ms. P. RAJALAKSHMI, M.E.**, Assistant Professor, Department of Computer Science And Engineering, who is our light house in the vast ocean of learning with her inspiring guidance & encouragement to complete the project.

Finally we also offer our thanks to all teaching and non –teaching staff members of Information Technology department and our friends for their kind help extended to us.

ABSTRACT

The Heart is one of the most vital structures in the human body. It is the centre of the circulatory system. Heart disease is a main life intimidating disease that can origin either death or a severe long-term disability. However, there is lack of effective tools to discover hidden relationships and trends in e-health data. Medical diagnosis is a complex task and plays a dynamic role in saving human lives so it needs to be executed accurately and efficiently. A suitable and precise computer based automated decision support system is required to reduce cost for achieving clinical tests. Health analytics have been proposed using ML to predict accurate patient data analysis. The data produced from health care industry is not mined. Data mining techniques can be used to build an intelligent model in medical field using data sets which involves risk factor of patients. The knowledge discovery in database (KDD) is startled with development of approaches and techniques for making use of data. This thesis provides an insight into deep learning and machine learning techniques used in diagnosing diseases. Numerous data mining classifiers have been conversed which has emerged in recent years for efficient and effective disease diagnosis. This thesis proposes a heart attack prediction system using Deep learning techniques, specifically Multi-Layer Perceptron (MLP) to predict the likely possibilities of heart related diseases of the patient. MLP is a very powerful classification algorithm that makes use of Deep Learning approach in Artificial Neural Network. The proposed model incorporates deep learning and data mining to provide the accurate results with minimum errors.

TABLE OF CONTENTS

CHAPTER NO	TOPIC	PAGE NO
	ABSTRACT	i
	LIST OF FIGURES	v
	LIST OF ABBREVIATIONS	vi
1	INTRODUCTION	1
	1.1 Heart Disease Prediction	1
	1.2 Data Science and Engineering	1
	1.3 Data Mining Types	2
	1.4 Data Mining Algorithms	3
	1.5 Applications of Data Mining	4
	1.6 Learning Algorithms in Data Mining	5
	1.7 Problems in Deep Learning Approaches	6
2	LITERATURE SURVEY	7
3	EXISTING SYSTEM	10
	3.1 Machine Learning Algorithms	10
	3.1.1 Classification and Regression Tree	10
	3.1.2 Support Vector Machine	11
	3.1.3 Random Forest Algorithm	11
	3.1.4 K-Nearest Neighbor Algorithm	11
	3.2 Challenges	12
	3.3 Disadvantages	12
4.	PROPOSED SYSTEM	13
	4.1 Algorithm	13
	4.2 Advantages	15
	4.3 Applications	16
5.	SYSTEM DESIGN	17
	5.1 System Architecture	17
	5.2 Modules	18

	5.2.1 Datasets Acquisition	18
	5.2.2 Preprocessing	18
	5.2.3 Feature Selection	19
	5.2.4 Classification	19
	5.2.5 Disease Diagnosis	20
6.	SYSTEM SPECIFICATIONS	21
	6.1 Hardware Specification	21
	6.2 Software Requirements	21
	6.3 Software Description	22
	6.3.1 Python	22
	6.3.2 Tensorflow Libraries in Python	26
	6.3.3 PyCharm	27
	6.3.4 MySQL	29
7	SYSTEM TESTING	31
	7.1 Testing	31
	7.1.1 Overview Of Testing	31
	7.2 Testing Methods	32
	7.2.1 Static Vs. Dynamic Testing	32
	7.2.2 White-Box Testing	32
	7.2.3 Static Testing Methods	33
	7.2.4 Black-Box Testing	33
	7.2.5 Grey-Box Testing	34
	7.3 Testing Levels	35
	7.3.1 Unit Testing	35
	7.3.2 Integration Testing	36
	7.3.3 System Testing	36
	7.3.4 Acceptance Testing	36
	7.4 Testing Approaches	36
	7.4.1 Bottom-Up	36
	7.4.2 Top-Down	36

7.5 Objectives Of Testing	37
7.5.1 Installation Testing	37
7.5.2 Compatibility Testing	37
7.5.3 Smoke And Sanity Testing	37
7.5.4 Regression Testing	37
7.5.5 Alpha Testing	38
7.5.6 Beta Testing	38
7.5.7 Functional Vs Non-Functional Testing	38
7.5.8 Destructive Testing	38
7.5.9 Software Performance Testing	38
7.5.10 Usability Testing	39
7.5.11 Accessibility Testing	39
7.5.12 Security Testing	39
7.5.13 Development Testing	39
8 CONCLUSION AND FUTURE	40
ENHANCEMENT	
8.1 Conclusion	40
8.2 Future Enhancement	40
APPENDIX	41
A1 CODING	41
A2 SCREENSHOT	66
REFERENCES	80

LIST OF FIGURES

FIGURE NO	FIGURE NAME	PAGE NO
Fig 4.1	MLP Algorithm	14
Fig5.1	System Architecture	17
Fig A2.1	Web Application Link	66
Fig A2.2	Web Application-Home Page	66
Fig A2.3	Admin Login Page	67
Fig A2.4	Admin Dashboard-User Information	68
Fig A2.5	Admin Dashboard-Doctor Information	68
Fig A2.6	Admin Dashboard-Drugs Information	69
Fig A2.7	New User Registration Page	69
Fig A2.8	User Login Page	70
Fig A2.9	User Dashboard-Personal Data	71
Fig A2.10	Heart Disease Prediction -1 & Prediction Result -1	72
Fig A2.11	Heart Disease Prediction -2 & Prediction Result -2	73
Fig A2.12	New Doctor Registration Page	74
Fig A2.13	Doctor Login Page	75
Fig A2.14	Appointment Scheduling Phase	76
Fig A2.15	Confirmation of appointment	77
Fig A2.16	Drug Assigning Phase	78
Fig A2.17	Report Generation Phase	79

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
CVD	Cardiovascular Diseases
SRS	Software Requirements Specification
XML	eXtensible Markup Language
JSON	Java Script Object Notation
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
IBM	International Business Machines
ML	Machine Learning
AI	Artificial Intelligence
SVM	Support Vector Machines
PNN	Probabilistic neural network
CNN	Convolutional Neural Network
ANN	Artificial Neural Network
KNN	K-Nearest Neighbor algorithm
ECG	Electrocardiogram
DT	Decision Trees
LR	Logistic Regression
RF	Random Forest
CART	Classification and Regression trees
EHR	Electronic Health Records
MLP	Multi-Layer Perceptron
CPU	Central Processing Unit
GPU	Graphics Processing Unit
API	Application Programming Interface
IDE	Integrated Development Environment
SQL	Structured Query Language

CHAPTER 1

1. INTRODUCTION

1.1 HEART DISEASE PREDICTION

Cardiovascular diseases (CVDs) continue to be a formidable global health challenge, particularly prevalent in industrialized nations, where their impact is felt through a significant toll on lives and quality of life. This project aims to address the pressing need for accurate and efficient prediction of cardiovascular diseases through the utilization of advanced technologies, specifically focusing on a deep learning algorithm known as the Multilayer Perceptron. The Software Requirements Specification (SRS) presented herein outlines a comprehensive framework designed not only to predict CVDs with improved accuracy but also to provide essential prescription details and recommend suitable healthcare professionals. In the face of preventable yet rising instances of CVDs, there exists a critical gap in preventive measures. The complexity of diagnosing heart diseases necessitates a sophisticated approach, and recent strides in machine learning provide a promising avenue. This project not only leverages these advancements but also explores the implementation of intelligent automated systems to support medical practitioners in predictive diagnosis and decision-making. The focus on the Multilayer Perceptron Neural Network with Back-propagation as the training algorithm underscores the commitment to enhancing the accuracy of the diagnostic system. The iterative application of the back-propagation algorithm, as demonstrated in the study, aims to minimize error rates and thereby improve the overall diagnostic accuracy of the system. Beyond prediction, the project envisions a holistic approach by extending its capabilities to provide detailed prescription information related to the predicted diseases and recommending healthcare professionals. By addressing these facets, the project seeks to contribute to the global effort in combating cardiovascular diseases, fostering timely medical care, and ultimately saving more effective, efficient and lifesaving treatments for cardiovascular diseases and problems.

1.2 DATA SCIENCE AND ENGINEERING

Data mining is the process of discovering patterns, trends, and insights from large datasets using various statistical and machine learning techniques. The goal of data mining is to extract valuable information from data and use it to make informed business decisions. Data mining involves several steps, including data collection, data cleaning, data integration, data transformation, data reduction, pattern identification, and evaluation. Data mining techniques include clustering, classification, regression, association rule mining, and anomaly detection.

Applications of data mining are widespread and include customer relationship management, fraud detection, market basket analysis, predictive maintenance, and healthcare. Data mining is an essential tool for businesses to make informed decisions and gain a competitive advantage.

Data mining can be performed on various types of data, including structured, semi-structured, and unstructured data. Structured data is organized and stored in a predefined format, such as databases, spreadsheets, or tables, and can be easily processed by machines. Semi-structured data, on the other hand, has some organizational structure but is not strictly defined, such as XML files, JSON, or HTML documents. Unstructured data, like text data or multimedia content, has no predefined structure, making it challenging to process and analyze. Data mining can be used in various fields, including marketing, healthcare, finance, and government. In marketing, data mining can be used to analyze customer behavior, predict buying patterns, and improve customer targeting. In healthcare, data mining can be used to predict the likelihood of a patient developing a particular disease, identify potential risks, and suggest treatment plans. In finance, data mining can be used to detect fraudulent transactions, identify investment opportunities, and predict market trends. In government, data mining can be used to detect potential threats to national security, monitor social media, and analyze demographic trends. To perform data mining, various tools and techniques are available, including programming languages like R and Python, data visualization software, and machine learning platforms like IBM Watson, Azure ML, and Google Cloud AI. It is essential to have domain knowledge and a clear understanding of the business problem and the data to use the appropriate tools and techniques effectively. In summary, data mining is a crucial technique for extracting valuable insights from large datasets and can be used in various fields to make informed decisions, improve business processes, and gain a competitive advantage.

1.3 DATA MINING TYPES

There are several types of data mining techniques, each with its unique strengths and applications. Here are some of the most commonly used data mining types:

Classification: This type of data mining technique is used to categorize data into different classes or groups based on predefined criteria. For example, classification can be used to classify customers into high, medium, and low-risk categories based on their purchase history or credit score.

Clustering: Clustering is used to group data based on similarities and differences, without any predefined criteria. This technique is useful for finding patterns and relationships in data and is often used in market segmentation or customer segmentation.

Regression: Regression analysis is used to predict numerical values based on other variables or factors. For example, regression can be used to predict a customer's future spending based on their past purchase history.

Association Rule Mining: This technique is used to find correlations between different variables or factors. For example, association rule mining can be used to identify the relationship between product sales and customer demographics.

Anomaly Detection: Anomaly detection is used to identify unusual patterns or outliers in data. This technique is useful for detecting fraud, security breaches, or defects in manufacturing processes.

Text Mining: Text mining is used to extract insights from unstructured data such as social media feeds, customer feedback, or product reviews. This technique can be used to identify customer sentiment, product trends, or emerging topics in the market.

Time Series Analysis: Time series analysis is used to identify trends and patterns in data over time. This technique is often used in forecasting sales or predicting market trends.

Each of these data mining types has its strengths and limitations, and the choice of technique depends on the business problem and the type of data available.

1.4 DATA MINING ALGORITHMS

There are various data mining algorithms available to extract useful information from large datasets. Here are some of the most commonly used data mining algorithms:

Decision Trees: Decision trees are a type of classification algorithm that creates a tree-like model of decisions and their possible consequences. It is a popular technique for predictive modeling, and it can be used in a wide range of applications.

K-Means Clustering: K-means clustering is a clustering algorithm used to group data points into clusters based on their similarities. It is a popular technique in unsupervised learning, and it can be used for customer segmentation, image processing, and natural language processing.

Apriori Algorithm: Apriori algorithm is used for association rule mining, which identifies the relationship between different variables or factors. It is used to discover hidden patterns in transactional datasets, such as retail sales, online shopping carts, or web log data.

Random Forest: Random Forest is an ensemble learning method that combines multiple decision trees to improve the accuracy of predictions. It is used for classification and regression tasks and can handle large datasets with high dimensionality.

Naive Bayes: Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem, which assumes that the features are independent of each other. It is used in text classification, sentiment analysis, and spam filtering.

Support Vector Machines: Support Vector Machines (SVMs) are a type of supervised learning algorithm that separates data into different classes by finding the best decision boundary between them. It is used for classification and regression tasks and can handle both linear and nonlinear data.

Gradient Boosting: Gradient boosting is another ensemble learning method that combines multiple weak models into a single strong model. It is used in regression and classification tasks and is particularly effective in handling complex datasets.

Each of these algorithms has its strengths and limitations, and the choice of algorithm depends on the type of data, the problem being solved, and the accuracy and speed required.

1.5 APPLICATIONS OF DATA MINING

Data science and engineering have a wide range of applications in various fields, including:

Healthcare: Data science and engineering can be used to analyze large medical datasets, identify patterns, and predict outcomes. It can help in disease diagnosis, personalized treatment planning, and drug discovery.

Finance: Data science and engineering can be used in finance to detect fraudulent activities, predict market trends, and identify investment opportunities. It can also help in risk management and credit scoring.

Marketing: Data science and engineering can be used to analyze customer behavior, predict buying patterns, and improve customer targeting. It can help in identifying new market segments, customer retention, and optimizing marketing campaigns.

Manufacturing: Data science and engineering can be used in manufacturing to optimize processes, reduce defects, and increase efficiency. It can also help in predictive maintenance, supply chain management, and quality control.

Transportation: Data science and engineering can be used in transportation to optimize routes, predict demand, and improve logistics. It can help in reducing transportation costs, improving delivery times, and minimizing environmental impact.

Education: Data science and engineering can be used in education to personalize learning, predict student outcomes, and improve teaching methods. It can help in identifying at-risk students, optimizing curriculums, and improving student engagement.

Energy: Data science and engineering can be used in the energy sector to optimize energy consumption, reduce waste, and increase efficiency. It can help in energy production, distribution, and storage.

These are just a few examples of the wide range of applications of data science and engineering. With the increasing amount of data being generated every day, the potential for data-driven insights and solutions is vast, and new applications are emerging all the time.

1.6 LEARNING ALGORITHMS IN DATA MINING

Machine learning and deep learning are two subfields of artificial intelligence that are commonly used in data mining to extract useful information from large datasets. Here's a brief overview of each technique:

Machine Learning: Machine learning is a subset of artificial intelligence that involves the development of algorithms that can learn from data and make predictions or decisions based on that data. In data mining, machine learning algorithms can be used for tasks such as classification, clustering, and regression.

Deep Learning: Deep learning is a subset of machine learning that involves the use of neural networks, which are designed to mimic the structure of the human brain. Deep learning algorithms can be used for tasks such as image recognition, speech recognition, and natural language processing.

In data mining, both machine learning and deep learning can be used to extract patterns and insights from large datasets. For example, a machine learning algorithm could be used to predict which customers are most likely to buy a certain product, based on their past purchase

history. A deep learning algorithm could be used to identify objects in images, such as identifying faces in a crowd. One of the key advantages of machine learning and deep learning is their ability to learn from large datasets and make accurate predictions or decisions based on that data.

1.7 PROBLEMS IN DEEP LEARNING APPROACHES

In data mining, machine learning and deep learning can be used to solve a wide range of problems, including:

- **Predictive Analytics:** Predictive analytics involves using historical data to make predictions about future events or behaviors. Machine learning algorithms can be used for tasks such as predictive modeling and time series forecasting. For example, machine learning algorithms can be used to predict customer churn or forecast sales revenue.
- **Anomaly Detection:** Anomaly detection involves identifying unusual patterns or events in a dataset that deviate from the norm. Machine learning algorithms can be used for tasks such as fraud detection and intrusion detection. For example, machine learning algorithms can be used to detect credit card fraud or identify network intrusions.
- **Recommender Systems:** Recommender systems involve suggesting items or products to users based on their preferences or past behavior. Machine learning algorithms can be used for tasks such as collaborative filtering and content-based filtering. For example, machine learning algorithms can be used to recommend movies or products to users based on their viewing or purchasing history.
- **Natural Language Processing:** Natural language processing involves processing and analyzing human language data. Deep learning algorithms can be used for tasks such as sentiment analysis and text classification. For example, deep learning algorithms can be used to classify customer reviews as positive or negative or identify the topic of a news article.
- **Computer Vision:** Computer vision involves analyzing and understanding visual data, such as images and videos. Deep learning algorithms can be used for tasks such as object recognition and image segmentation. For example, deep learning algorithms can be used to identify objects in images or recognize faces in videos.

CHAPTER 2

2. LITERATURE SURVEY

Aleeza Nouman, et.al,...[1] implemented the framework for predict heart disease in the earlier stage by using emerging technologies in a better and secure manner that assists while saving lives. Emerging technologies like Machine Learning (ML) and blockchain are revolutionizing the existing healthcare infrastructure, which is a difficult task to securely and accurately forecast heart disease. Blockchain and ML are providing the best solutions to gather information while predicting heart disease. This study provides comprehensive reviews on different ML techniques (Support Vector Machine (SVM), Probabilistic neural network (PNN), Bayesian networks, Convolutional Neural Network (CNN), J48 and Artificial Neural Network (ANN)) in order to predict heart disease. Correct diagnosis and forecasting are essential concerns for practitioners as well as hospitals. These important issues should be taken into deliberation when predicting heart disease. Because of developments in computing technology, many services for real-time knowledge gathering and storage capacity are now feasible. As a result, a lot of health-related data is acquired, which is excellent for clinical research.

Abhay Agrahara, et.al,...[2] summarize the recent research with comparative results that has been done on heart disease prediction and also make analytical conclusions. From the study, it is observed Naive Bayes with Genetic algorithm; Decision Trees and Artificial Neural Networks techniques improve the accuracy of the heart disease prediction system in different scenarios. In this paper commonly used data mining and machine learning techniques and their complexities are summarized. And present a heart disease prediction use case showing how synthetic data can be used to address privacy concerns and overcome constraints inherent in small medical research data sets. While advanced machine learning algorithms, such as neural networks models, can be implemented to improve prediction accuracy, these require very large data sets which are often not available in medical or clinical research. We examine the use of surrogate data sets comprised of synthetic observations for modeling heart disease prediction. We generate surrogate data, based on the characteristics of original observations, and compare prediction accuracy results achieved from traditional machine learning models using both the original observations and the synthetic data.

Ibrahim M. El-Hasnony, et.al,...[3] evaluated the heart disease dataset with active learning methods. Many machine-learning-based studies have used this dataset for heart disease prediction and classification. In this paper, the proposed model tries to solve the problem of memorizing learning models. Active learning is a perfect method if the model does

not overfit the data instances. However, it is still good to train only on samples that significantly impact its performance. Hence, the goal was to achieve a model that generalizes the existing data, i.e., not memorization, but a generalization. In many cases, especially in high-dimensional settings, learning a model that works well on training data but fails on new data is common. In many cases of regular machine learning algorithms, the model has “overfit” or “underfit” the training data (i.e., it has simply memorized/unmemorized the data). So, five selection strategies for multi-label active learning are applied. The five methods are MMC, Random, Adaptive, QUIRE, and AUDI. The grid search with the label ranking classifier is implemented as the predictive modelling in each strategy. There have been several different conditions under which the system can be stopped. As a rule of thumb, the AL procedure is repeated several times (number of iterations). The base classifier’s performance is evaluated using a test set and an evaluation metric. The entire model was applied to the dataset of heart disease. As a result of the research, it was concluded that the learning model could be extrapolated to include new data

Md. Mahbubur Rahman, et.al,...[4] produced a manual and web-based automatic prediction system that can confer a conceptual report of clear warning of patient's heart condition. The proposed prediction system predicts heart disease using some health parameters. The system uses thirteen health parameters like age, sex, chest pain type, blood pressure, ECG, etc. Eight algorithms are used separately to diagnose heart disease accurately, namely KNN, XgBoost, Logistic Regression (LR), Support Vector Machine (SVM), Ada Boost, Decision tree (DT), Naïve Bayes, and Random Forest (RF). Decision Tree and Random Forest provide better performance than others among all methods. This research also established a website to easily check their heart condition from home instantly. In the proposed method, heart disease can be detected more efficiently and less costly within a short time. This paper worked with pre-processed data to train and test using machine learning algorithms. In the first stage, pre-processed data are divided into two parts. Most of those are used in the training phase (80%), and the rest (20%) are used in the testing phase. In the training and testing phase, the proposed system has trained our dataset using machine learning algorithms like Decision tree, XgBoost, KNN, Support vector machine, Naïve Bayes, Logistic Regression, AdaBoost, and Random Forest.

Raniya R. Sarra, et.al,...[5] implemented an enhanced model was implemented to increase the heart disease diagnosis and prediction accuracy, as well as to reduce computational load. The ML (SVM) algorithm was used as a classification model for enhanced heart disease

diagnoses. This model was performed on two famous heart disease datasets. The results showed increasing accuracy from 84.21% to 89.47 and from 85.29% to 89.7% in the Cleveland and Statlog datasets, respectively. Furthermore, the features used in the system were decreased from 14 to 6 features, which means that the computational load was reduced from 100% to approximately 42%. We anticipate that this work will contribute to the future development and implementation of heart disease prediction and diagnosis systems. Automatic heart disease prediction is a major global health concern. Effective cardiac treatment requires an accurate heart disease prognosis. Therefore, this paper proposes a new heart disease classification model based on the support vector machine (SVM) algorithm for improved heart disease detection. To increase prediction accuracy, the χ^2 statistical optimum feature selection technique was used. The suggested model's performance was then validated by comparing it to traditional models using several performance measures. Moreover, this paper critically evaluates the previous methods and presents the limitations in these methods. Finally, the article provides some future research directions in the domain of automated heart disease detection based on machine learning and multiple of data modalities. Based on different data modalities, the previously proposed studies were critically analysed and systematically organized. Moreover, in this study, we also pointed out the limitations and loop holes in the previously proposed methods for automated heart disease detection. Finally, to mitigate the problems present in previously developed methods and to provide better heart disease detection, some future directions were discussed for onward research in the domain of automated heart disease detection based on ML. We hope that this review will be helpful to those who intend to work in the domain of automated heart disease detection.

CHAPTER 3

3. EXISTING SYSTEM

Present days one of the major application areas of machine learning algorithms is medical diagnosis of diseases and treatment. Machine learning algorithms also used to find correlations and associations between different diseases. Nowadays many people are dying because of sudden heart attack. Prediction and diagnosing of heart disease becomes a challenging factor faced by doctors and hospitals both in India and abroad. In order to reduce number of deaths because of heart diseases, we have to predict whether person is at the risk of heart disease or not in advance. Data mining techniques and machine learning algorithms play a very important role in this area. Many researchers are carrying out their research in this area to develop software that can help doctors to take decision regarding both prediction and diagnosing of heart disease. In this project we focused on how data mining techniques can be used to predict heart disease in advance such that patient is well treated. An important task of any diagnostic system is the process of attempting to determine and/or identify a possible disease or disorder and the decision reached by this process. For this purpose, machine learning algorithms are widely employed. For these machine learning techniques to be useful in medical diagnostic problems, they must be characterized by high performance, the ability to deal with missing data and with noisy data, the transparency of diagnostic knowledge, and the ability to explain decisions. As people are generating more data everyday so there is a need for such a classifier which can classify those newly generated data accurately and efficiently. This System mainly focuses on the supervised learning technique called the Random forests for classification of data by changing the values of different hyper parameters in Random Forests Classifier. There have been numerous attempts to apply machine learning for classification-related tasks, such as aiding in the early diagnosis of cervical cancer. Support vector machines, k-nearest neighbours, Random Forest trees (RFT), Classification and regression trees (CART), and others are some of the most used machine learning techniques.

3.1 MACHINE LEARNING ALGORITHMS

3.1.1 CLASSIFICATION AND REGRESSION TREE: The CART algorithm, which is used in machine learning, shows how the values of the target variable can be predicted based on other factors. Each fork of the decision tree is divided into a predictor variable, and at the conclusion of each node is a prediction for the target variable. Depending on the threshold value of an attribute, nodes in the decision tree are divided into sub-nodes. The training set is the root node, which is divided into two by taking the best attribute and threshold value

into account. Additionally, the subsets are divided according to the same rationale. This continues until the tree has either produced all of its potential leaves or found its last clean sub-set.

3.1.2 SUPPORT VECTOR MACHINE: Regression, classification into one or more classes, and outliers' detection are the main problem statements for which support vector machines have been used. The two main functions of support vector machines are predicting the location of the hyper plane and solving a quadratic equation. Initially, this process was carried out using known, already-existing quadratic optimisation techniques.

3.1.3 RANDOM FOREST: A type of data analysis called Random Forest (RF) Classification generates models out of data classes. A categorization model foretells the existence of classes. Continuous-valued functions are modelled via numerical prediction. The algorithm used to correct the accuracy numbers uses classification and numerical prediction. For classification, regression, and other tasks, random forests are an ensemble learning technique that builds a large number of decision trees during the training phase and outputs the class that represents the mean of the classes (classification) or the mean/average prediction (regression). Numerous classification trees and a bootstrap sample are produced. Each tree is trained using methods using the training set of data. On the training set, train the model using random forest classification. predicting the outcomes of the test set, using a confusion matrix to assess accuracy.

3.1.4 K-NEAREST NEIGHBOR ALGORITHM: A non-parametric classifier called K-Nearest Neighbour is employed in both classification and regression. We cannot make assumptions about the distribution of the data because it is parametric. An explicit training phase is not required for KNN classifier. Data is separated into test set and training set for KNN. In order to classify a row of the test set, the nearest neighbour k points, depending on the Euclidean distance from the training set point, are observed.

3.2 CHALLENGES

Predicting cardiovascular disease using deep learning algorithms presents several challenges:

Data Quality and Quantity: Deep learning models require large amounts of high-quality data for effective training. Obtaining diverse and accurately labeled datasets for cardiovascular disease prediction can be challenging due to privacy concerns, data heterogeneity, and limited availability of labeled data.

Feature Selection and Extraction: Identifying relevant features from complex healthcare datasets is crucial for model performance. However, selecting appropriate features from heterogeneous data sources such as electronic health records (EHRs), medical images, genetic information, and lifestyle factors can be challenging. Deep learning models often rely on automatic feature learning, but ensuring the models learn clinically meaningful representations remains a challenge.

Imbalanced Data: Class imbalance, where one class (e.g., presence of cardiovascular disease) is significantly more prevalent than others, can lead to biased models that prioritize the majority class. Addressing class imbalance through techniques such as oversampling, undersampling, or cost-sensitive learning is essential to prevent biased predictions.

Interpretability and Explainability: Deep learning models are often viewed as "black boxes" due to their complex architectures and high-dimensional representations. Interpreting and explaining the predictions of deep learning models in the context of cardiovascular disease diagnosis is crucial for gaining trust from healthcare professionals and patients.

Generalization to Diverse Populations: Models trained on data from one demographic or population may not generalize well to other populations with different characteristics, leading to biased predictions. Ensuring the robustness and generalizability of deep learning models across diverse patient populations and healthcare settings is a significant challenge.

3.3 DISADVANTAGES

- Labelled data-based disease classification
- Provide high number of false positive
- Binary classification can be occurred
- Computational complexity

CHAPTER 4

4. PROPOSED SYSTEM

Cardiovascular disease continues to claim an alarming number of lives across the globe. CVD disease is the greatest scourge affecting the industrialized nations. CVD not only strikes down a significant fraction of the population without warning but also causes prolonged suffering and disability in an even larger number. Although large proportion of CVDs is preventable, they continue to rise mainly because preventive measures are inadequate. Heart disease diagnosis has become a difficult task in the field of medicine. This diagnosis depends on a thorough and accurate study of the patient's clinical tests data on the health history of an individual. The tremendous improvement in the field of machine learning aim at developing intelligent automated systems which helps the medical practitioners in predicting as well as making decisions about the disease. Such an automated system for medical diagnosis would enhance timely medical care followed by proper subsequent treatment thereby resulting in significant lifesaving. Incorporating the techniques of classification in these intelligent systems achieve at accurate diagnosis. Neural Networks has emerged as an important method of classification. Multi-layer Perceptron Neural Network with Back-propagation has been employed as the training algorithm in this work. This project proposes a diagnostic system for predicting heart disease with improved accuracy. The propagation algorithm has been repeated until minimum error rate was observed. And it is quite evident from the results presented in the previous section that the accuracy rate is maximized.

4.1 ALGORITHM

A multilayer ANN consists of an input layer, an output layer, and one or more hidden layers. The hidden layer can be used to carry out intermediate calculations before converting the input to the output layer. When a model is created for a particular application, it is trained using inputs and targets until it masters the ability to link a certain input with a particular output. A network is trained until the minimal value of the weight change during a training cycle is reached. After being trained, a model is verified by checking to see if it produces reliable results. Multilayered networks are capable of remembering data because of the enormous number of synaptic weights that are available in the network. An output layer neuron will occasionally qualify its output using a threshold function. We will nevertheless refer to them as neurons even though our issue involves artificial neurons. Synapses between neurons are

represented by connections, which are edges of a directed graph with nodes corresponding to synthetic neurons. There are five consecutive steps in the model-building process:

1. Deciding which data will be used as input and output for the supervised learning process.
2. Normalisation of the data, both at input and output.
3. The normalised data is trained using neural network learning.
4. Verifying the goodness of fit of the model.
5. evaluating the differences between anticipated and desired results.

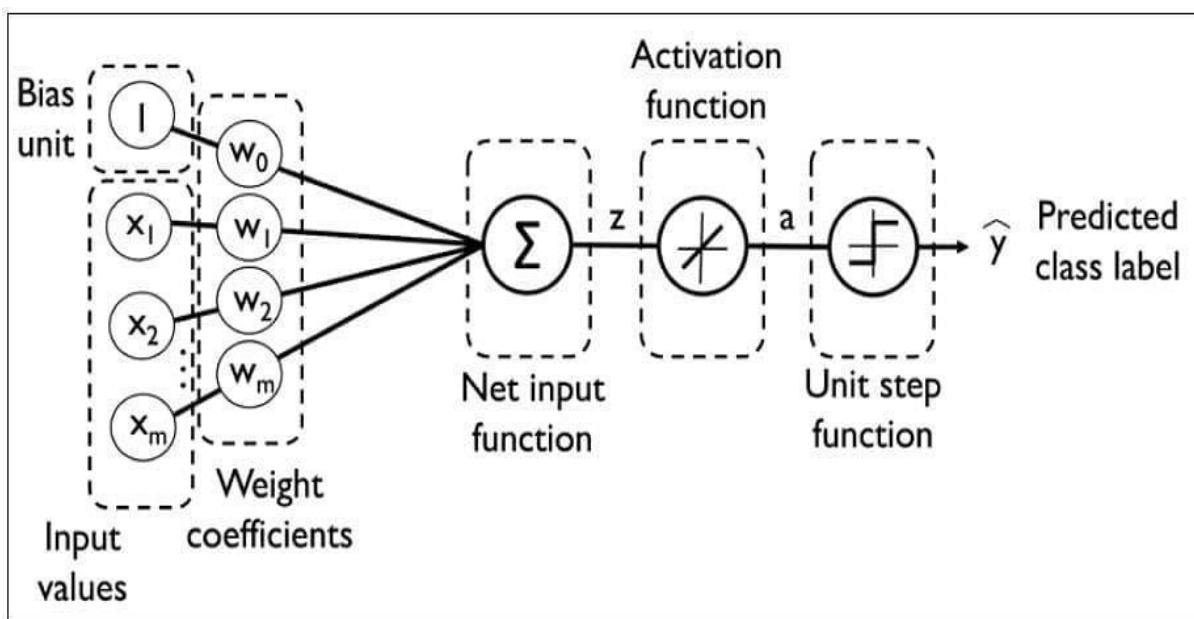


Figure 4.1 MLP Algorithm

A layered feed forward neural network is made up of layers, or subgroups of processing modules. Data is processed by a layer of processing components, which then transfers the results to a higher layer. The subsequent layer may then carry out its own calculations and transmit the outcomes to a subsequent layer. Last but not least, a subgroup of one or more processing elements determines the output of the network. Based on a weighted average of its inputs, each processing component performs its calculations. The first layer is the input layer, while the last layer is the output layer. The layers between these two are the ones that are concealed. The processing components are referred to as cells, neuromas, or artificial neurons since they are thought to function similarly to brain neurons.

The steps in the neural network algorithm are as follows:

Step 1: Randomly initialise the weights and biases.

Step 2 is to feed the machine with the training sample.

Step 3: Continue using the inputs, and then compute each unit's net input and output in the hidden and output layers.

Reverse the defect to the concealed layer in step four.

Update the weights and biases to account for the spread of errors.

Step 5: The network's weights and biases are automatically adjusted using training and learning functions, which are mathematical operations.

Terminating condition in step six

4.2 ADVANTAGES

- Accuracy is high
- Parallel processing
- Multiple heart diseases are predicted
- Reduce number of false positive rate

4.3 APPLICATIONS

Deep learning algorithms have numerous applications in the prediction and management of cardiovascular diseases:

- **Risk Stratification:** Deep learning models can analyze patient data, including medical history, laboratory results, imaging studies, and lifestyle factors, to stratify individuals into different risk categories for cardiovascular diseases. This enables healthcare providers to identify high-risk patients who may benefit from early intervention and preventive measures.
- **Early Detection and Diagnosis:** Deep learning algorithms can analyze medical images (e.g., echocardiograms, angiograms) and signals (e.g., electrocardiograms, blood pressure recordings) to detect early signs of cardiovascular abnormalities, such as heart failure, arrhythmias, or atherosclerosis. Early detection allows for timely intervention and improved patient outcomes.
- **Personalized Treatment Planning:** By analyzing large-scale patient data, deep learning models can identify patterns and correlations between patient characteristics, treatment strategies, and outcomes. This enables personalized treatment planning and optimization of therapeutic interventions based on individual patient profiles, leading to more effective and tailored healthcare delivery.
- **Drug Discovery and Development:** Deep learning techniques, such as deep generative models and deep reinforcement learning, can accelerate the drug discovery process by predicting the efficacy and safety of potential therapeutic compounds for cardiovascular diseases. These models can analyze molecular structures, biological pathways, and clinical trial data to identify promising drug candidates and optimize treatment regimens.
- **Population Health Management:** Deep learning models can analyze population-level health data, including electronic health records, claims data, and social determinants of health, to identify trends, risk factors, and disparities in cardiovascular health outcomes. This information can inform public health policies, resource allocation, and preventive interventions aimed at improving population health and reducing the burden of cardiovascular diseases.

CHAPTER 5

5. SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE

The system architecture for cardiovascular disease prediction leveraging deep learning algorithms encompasses several interconnected components. Initially, data acquisition and preprocessing modules collect diverse datasets, including heart disease like attributes such as age, gender, smoke level and so on. Subsequently, the model development and training phase involve the exploration of various deep learning architectures, such as Multi layer perceptron algorithm.

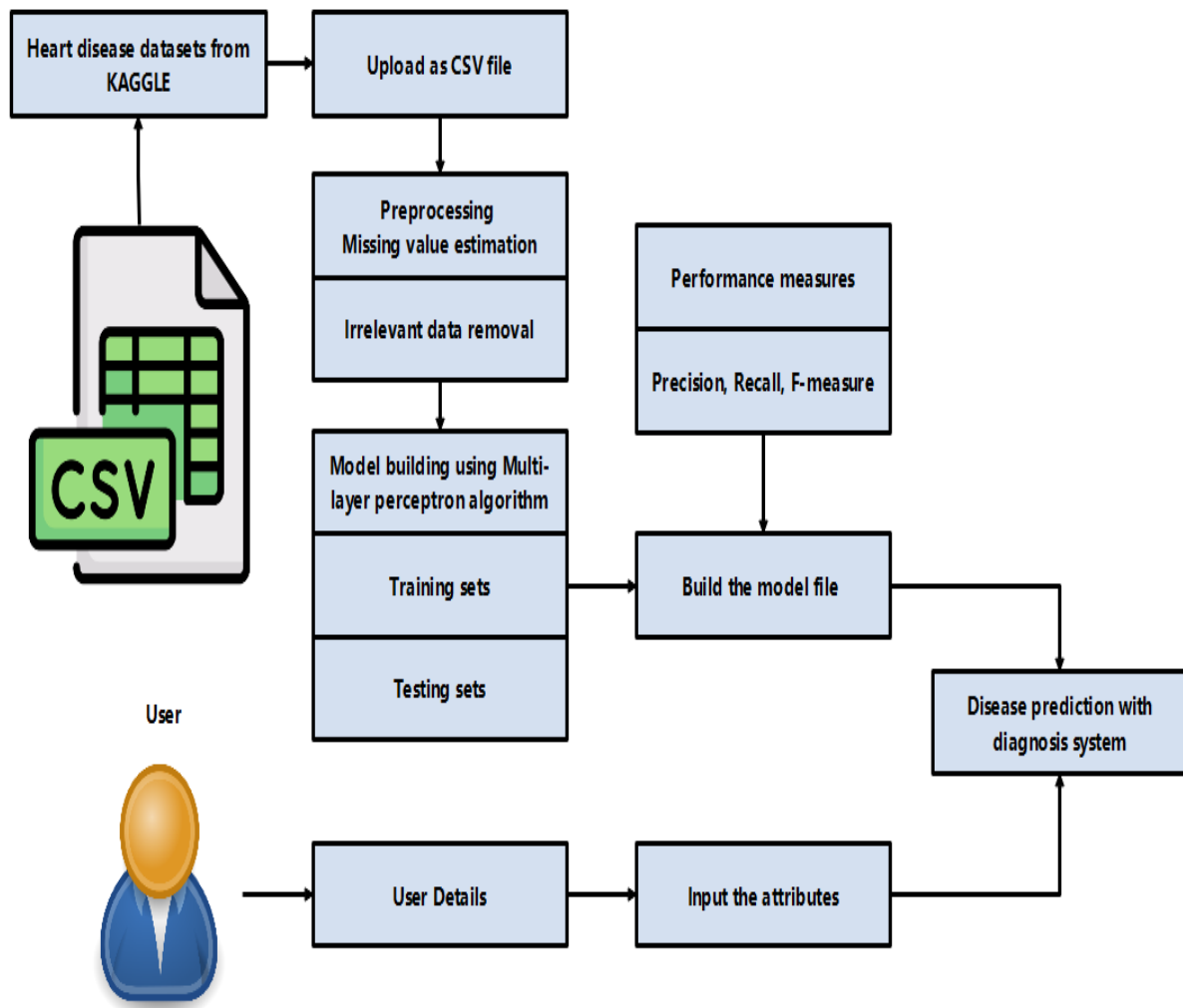


Figure 5.1 System Architecture

5.2 MODULES

- **MODULE 1: DATASETS ACQUISITION**
- **MODULE 2: PRE-PROCESSING**
- **MODULE 3: FEATURES SELECTION**
- **MODULE 4: CLASSIFICATION**
- **MODULE 5: DISEASE DIAGNOSIS**

5.2.1 DATASETS ACQUISITION

A data set (or dataset, although this spelling is not present in many contemporary dictionaries like Merriam-Webster) is a collection of data. Most commonly a data set corresponds to the contents of a single database table, or a single statistical data matrix, where every column of the table represents a particular variable, and each row corresponds to a given member of the data set in question. The data set lists values for each of the variables, such as height and weight of an object, for each member of the data set. Each value is known as a datum. The data set may comprise data for one or more members, corresponding to the number of rows. The term data set may also be used more loosely, to refer to the data in a collection of closely related tables, corresponding to a particular experiment or event. In this module, we can upload the cardiovascular datasets related to heart diseases which includes the attributes such as age, gender, height, weight, systolic blood pressure, diastolic blood pressure, cholesterol, glucose, smoke, alcohol, active status, cardio labels.

5.2.2 PREPROCESSING

Data pre-processing is an important step in the [data mining] process. The phrase "garbage in, garbage out" is particularly applicable to data mining and machine learning projects. Data-gathering methods are often loosely controlled, resulting in out-of-range values, impossible data combinations, missing values, etc. Analysing data that has not been carefully screened for such problems can produce misleading results. Thus, the representation and quality of data is first and foremost before running an analysis. If there is much irrelevant and redundant information present or noisy and unreliable data, then knowledge discovery during the training phase is more difficult. Data preparation and filtering steps can take considerable amount of processing time. In this module, we can

eliminate the irrelevant values and also estimate the missing values of data. Finally provide structured datasets.

5.2.3 FEATURES SELECTION

Feature selection refers to the process of reducing the inputs for processing and analysis, or of finding the most meaningful inputs. A related term, feature engineering (or feature extraction), refers to the process of extracting useful information or features from existing data. Filter feature selection methods apply a statistical measure to assign a scoring to each feature. The features are ranked by the score and either selected to be kept or removed from the dataset. The methods are often uni-variate and consider the feature independently, or with regard to the dependent variable. It can be used to construct the multiple heart diseases. In this module, select the multiple features from uploaded datasets. And train the datasets with various disease labels such as Coronary heart diseases, Cardiac arrest, High blood pressure, Arrhythmia and normal.

5.2.4 CLASSIFICATION

In this module implement classification algorithm to predict the heart diseases. And using deep learning algorithm such as multi-layer perceptron algorithm to predict the diseases. A multilayer perceptron (MLP) is a feed forward artificial neural network model that maps sets of input data onto a set of appropriate outputs. It (MLP) consists of multiple layers of nodes in a directed graph, and each layer is fully connected to the next one. Each node is a neuron with a nonlinear activation function except for the input nodes. MLP utilizes a supervised learning technique called back propagation for training the network. MLP is a modified form of the standard linear perceptron and can distinguish data that are not linearly separable. If a multilayer perceptron (MLP) has a simple on-off mechanism i.e. linear activation function in all neurons, to determine whether or not a neuron fires, then it is easily proved with linear algebra that any number of layers can be reduced to the standard two-layer input-output model. The gradient techniques are then applied to the optimization methods to adjust the weights to minimize the loss function in the network. Hence, the algorithm requires a known and a desired output for all inputs in order to compute the gradient of loss function. Usually, the generalization of Multilayer Feed Forward Networks is done using delta rule which possibly makes a chain of iterative rules to compute gradients for each layer. Back Propagation Algorithm necessitates the activation function to be different between the neurons. The ongoing researches on parallel, distributed computing and computational neuroscience are currently implemented with the concepts of MultiLayer Perceptron using a Back Propagation Algorithm.

The multilayer perceptron is the most known and most frequently used type of neural network. User can provide the features and automatically predict the diseases. A layered feed forward neural network is made up of layers, or subgroups of processing modules. Data is processed by a layer of processing components, which then transfers the results to a higher layer. The subsequent layer may then carry out its own calculations and transmit the outcomes to a subsequent layer. Last but not least, a subgroup of one or more processing elements determines the output of the network. Based on a weighted average of its inputs, each processing component performs its calculations. The first layer is the input layer, while the last layer is the output layer. The layers between these two are the ones that are concealed.

5.2.5 DISEASE DIAGNOSIS

Medical decision support system is a decision-support program which is designed to assist physicians and other health professionals with decision making tasks, such as determining diagnosis of patients' data. In this module, provide the diagnosis information based on predicted heart diseases. Proposed system provides improved accuracy in heart disease prediction. Risk factors are conditions or habits that make a person more likely to develop a disease.

CHAPTER 6

6. SYSTEM SPECIFICATION

6.1 HARDWARE SPECIFICATION

- Processor : Intel core processor 2.6.0 GHZ
- RAM : 4 GB
- Hard disk : 160 GB
- Compact Disk : 650 Mb

6.2 SOFTWARE REQUIREMENTS

- Server Side : Python 3.7.4(64-bit) or (32-bit)
- Client Side : HTML, CSS, Bootstrap
- Front End : Python
- Back end : MySQL
- IDE : PyCharm
- Server : WampServer 2i
- OS : Windows 10 64 –bit

6.3 SOFTWARE DESCRIPTION

6.3.1 PYTHON

Python is a high-level, interpreted programming language that is widely used in various domains such as web development, scientific computing, data analysis, artificial intelligence, machine learning, and more. It was first released in 1991 by Guido van Rossum and has since become one of the most popular programming languages due to its simplicity, readability, and versatility. One of the key features of Python is its easy-to-learn syntax, which makes it accessible to both novice and experienced programmers. It has a large standard library that provides a wide range of modules for tasks such as file I/O, networking, regular expressions, and more. Python also has a large and active community of developers who contribute to open-source libraries and packages that extend its capabilities. Python is an interpreted language, which means that it is executed line-by-line by an interpreter rather than compiled into machine code like C or C++. This allows for rapid development and testing, as well as easier debugging and maintenance of code. Python is used for a variety of applications, including web development frameworks such as Django and Flask, scientific computing libraries such as NumPy and Pandas, and machine learning libraries such as TensorFlow and PyTorch. It is also commonly used for scripting and automation tasks due to its ease of use and readability. Overall, Python is a powerful and versatile programming language that is widely used in a variety of domains due to its simplicity, ease of use, and active community.

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. In July 2018, Van Rossum stepped down as the leader in the language community. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library. Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open-source software and has a community-based development model, as do nearly all of Python's other implementations. Python and CPython are managed by the non-profit Python Software Foundation. Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large

standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach. While offering choice in coding methodology, the Python philosophy rejects exuberant syntax (such as that of Perl) in favor of a simpler, less-cluttered grammar. As Alex Martelli put it: "To describe something as 'clever' is not considered a compliment in the Python culture." Python's philosophy rejects the Perl "there is more than one way to do it" approach to language design in favour of "there should be one—and preferably only one—obvious way to do it".

Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of CPython that would offer marginal increases in speed at the cost of clarity. [When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. CPython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter. An important goal of Python's developers is keeping it fun to use. This is reflected in the language's name a tribute to the British comedy group Monty Python and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (from a famous Monty Python sketch) instead of the standard for and bar. A common neologism in the Python community is *pythonic*, which can have a wide range of meanings related to program style. To say that code is *pythonic* is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called *unpythonic*. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the

interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace.

Python also has a large and active community of developers who contribute to a wide range of open-source libraries and tools, making it easy to find and use pre-built code to solve complex problems.

Python has a wide range of applications, including:

Data Science: Python is one of the most popular languages for data science, thanks to libraries like NumPy, Pandas, and Matplotlib that make it easy to manipulate and visualize data.

Machine Learning: Python is also widely used in machine learning and artificial intelligence, with libraries like TensorFlow, Keras, and Scikit-learn that provide powerful tools for building and training machine learning models.

Web Development: Python is commonly used in web development, with frameworks like Django and Flask that make it easy to build web applications and APIs.

Scientific Computing: Python is used extensively in scientific computing, with libraries like SciPy and SymPy that provide powerful tools for numerical analysis and symbolic mathematics.

In addition to its versatility and ease of use, Python is also known for its portability and compatibility. Python code can be run on a wide range of platforms, including Windows, macOS, and Linux, and it can be integrated with other languages like C and Java.

Overall, Python is a powerful and versatile programming language that is well-suited for a wide range of applications, from data science and machine learning to web development and scientific computing. Its simplicity, readability, and large community of developers make it an ideal choice for beginners and experts alike.

There are two attributes that make development time in Python faster than in other programming languages:

1. Python is an interpreted language, which precludes the need to compile code before executing a program because Python does the compilation in the background. Because Python is a high-level programming language, it abstracts many sophisticated details from the programming code. Python focuses so much on this abstraction that its code can be understood by most novice programmers.

2. Python code tends to be shorter than comparable codes. Although Python offers fast development times, it lags slightly in terms of execution time. Compared to fully compiling languages like C and C++, Python programs execute slower. Of course, with the processing speeds of computers these days, the speed differences are usually only observed in benchmarking tests, not in real-world operations. In most cases, Python is already included in Linux distributions and Mac OS X machines.

One of the strengths of Python is its rich ecosystem of third-party libraries and tools. These libraries provide a wide range of functionality, from scientific computing and data analysis to web development and machine learning. Some popular Python libraries and frameworks include:

NumPy: a library for numerical computing in Python, providing support for large, multi-dimensional arrays and matrices, along with a large collection of mathematical functions to operate on these arrays.

Pandas: a library for data manipulation and analysis in Python, providing support for reading and writing data in a variety of formats, as well as powerful tools for manipulating and analyzing data.

Matplotlib: a plotting library for Python that provides a variety of visualization tools, including line plots, scatter plots, bar plots, and more.

TensorFlow: an open-source machine learning library for Python that provides a variety of tools and algorithms for building and training machine learning models.

Django: a popular web framework for Python that provides a full-stack framework for building web applications, with support for everything from URL routing to user authentication and database integration.

Python's popularity has also led to a large and active community of developers who contribute to open-source projects and share code and resources online. This community provides a wealth of resources for learning Python, including tutorials, online courses, and forums for asking and answering questions.

Overall, Python is a versatile and powerful programming language that is well-suited for a wide range of applications. Its simplicity, flexibility, and wide range of libra

6.3.2 TENSORFLOW LIBRARIES IN PYTHON

TensorFlow is an open-source machine learning framework developed by Google Brain Team. It is one of the most popular libraries for building and training machine learning models, especially deep neural networks. TensorFlow allows developers to build complex models with ease, including image and speech recognition, natural language processing, and more. One of the key features of TensorFlow is its ability to handle large-scale datasets and complex computations, making it suitable for training deep neural networks. It allows for parallelization of computations across multiple CPUs or GPUs, allowing for faster training times. TensorFlow also provides a high-level API called Keras that simplifies the process of building and training models. TensorFlow offers a wide range of tools and libraries that make it easy to integrate with other Python libraries and frameworks. It has built-in support for data preprocessing and visualization, making it easy to prepare data for training and analyze model performance. One of the major advantages of TensorFlow is its ability to deploy models to a variety of platforms, including mobile devices and the web.

Graph-based computation: TensorFlow uses a graph-based computation model, which allows for efficient execution of computations across multiple devices and CPUs/GPUs.

Automatic differentiation: TensorFlow provides automatic differentiation, which allows for efficient computation of gradients for use in backpropagation algorithms.

High-level APIs: TensorFlow provides high-level APIs, such as Keras, that allow developers to quickly build and train complex models with minimal code.

Preprocessing and data augmentation: TensorFlow provides a range of tools for preprocessing and data augmentation, including image and text preprocessing, data normalization, and more.

Distributed training: TensorFlow supports distributed training across multiple devices, CPUs, and GPUs, allowing for faster training times and more efficient use of resources.

Model deployment: TensorFlow allows for easy deployment of models to a variety of platforms, including mobile devices and the web.

Visualization tools: TensorFlow provides a range of visualization tools for analyzing model performance, including TensorBoard, which allows for real-time visualization of model training and performance.

6.3.3 PYCHARM

PyCharm is an integrated development environment (IDE) for Python programming language, developed by JetBrains. PyCharm provides features such as code completion, debugging, code analysis, refactoring, version control integration, and more to help developers write, test, and debug their Python code efficiently. PyCharm is available in two editions: Community Edition (CE) and Professional Edition (PE). The Community Edition is a free, open-source version of the IDE that provides basic functionality for Python development. The Professional Edition is a paid version of the IDE that provides advanced features such as remote development, web development, scientific tools, database tools, and more. PyCharm is available for Windows, macOS, and Linux operating systems. It supports Python versions 2.7, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, and 3.10.

Features:

- Intelligent code completion
- Syntax highlighting
- Code inspection
- Code navigation and search
- Debugging
- Testing
- Version control integration
- Web development support
- Scientific tools support
- Database tools support

Integration with other JetBrains tools

PyCharm's code completion feature can help speed up development by automatically suggesting code based on context and previously written code. It also includes a debugger that allows developers to step through code, set breakpoints, and inspect variables. PyCharm has integration with version control systems like Git, Mercurial, and Subversion. It also supports virtual environments, which allow developers to manage different Python installations and packages in isolated environments. The IDE also has features specifically geared towards web development, such as support for popular web frameworks like Django, Flask, and Pyramid. It includes tools for debugging, testing, and profiling web applications. PyCharm also provides

scientific tools for data analysis, visualization, and scientific computing, such as support for NumPy, SciPy, and matplotlib. It also includes tools for working with databases, such as PostgreSQL, MySQL, and Oracle. Overall, PyCharm is a powerful and feature-rich IDE that can greatly increase productivity for Python developers.

Customization:

PyCharm allows developers to customize the IDE to their liking. Users can change the color scheme, fonts, and other settings to make the IDE more comfortable to use. PyCharm also supports plugins, which allow developers to extend the IDE with additional features.

Collaboration:

PyCharm makes it easy for developers to collaborate on projects. It supports integration with popular collaboration tools such as GitHub, Bitbucket, and GitLab. It also includes features for code reviews, task management, and team communication.

Education:

PyCharm provides a learning environment for Python programming language. PyCharm Edu is a free, open-source edition of PyCharm that includes interactive courses and tutorials for learning Python. It provides an easy-to-use interface for beginners and includes features such as code highlighting, autocompletion, and error highlighting.

Support:

PyCharm has an active community of users who provide support through forums and social media. JetBrains also provides comprehensive documentation, tutorials, and training courses for PyCharm. For users who need more personalized support, JetBrains offers a paid support plan that includes email and phone support.

Pricing:

PyCharm Community Edition is free and open-source. PyCharm Professional Edition requires a paid license, but offers a 30-day free trial. JetBrains also offers a subscription-based pricing model that includes access to all JetBrains IDEs and tools.

Integrations:

PyCharm integrates with a wide range of tools and technologies commonly used in Python development. It supports popular Python web frameworks like Flask, Django, Pyramid,

and web2py. It also integrates with tools for scientific computing like NumPy, SciPy, and pandas. PyCharm also supports popular front-end technologies such as HTML, CSS, and JavaScript.

Performance:

PyCharm is known for its fast and reliable performance. It uses a combination of static analysis, incremental compilation, and intelligent caching to provide fast code completion and navigation. PyCharm also has a memory profiler that helps identify and optimize memory usage in Python applications.

Ease of Use:

PyCharm provides an intuitive and easy-to-use interface for developers. It has a well-organized menu structure, clear icons, and easy-to-navigate tabs. PyCharm also provides a variety of keyboard shortcuts and customizable keymaps that allow users to work efficiently without constantly switching between the mouse and keyboard.

Community:

PyCharm has a large and active community of developers who contribute to the development of the IDE. The PyCharm Community Edition is open-source, which means that anyone can contribute to its development. The PyCharm user community is also active in providing support, tips, and tutorials through forums, blogs, and social media.

6.3.4 MY SQL

MySQL is the world's most used open source relational database management system (RDBMS) as of 2008 that run as a server providing multi-user access to a number of databases. The MySQL development project has made its source code available under the terms of the GNU General Public License, as well as under a variety of proprietary agreements. MySQL was owned and sponsored by a single for-profit firm, the Swedish company MySQL AB, now owned by Oracle Corporation.

MySQL is a popular choice of database for use in web applications, and is a central component of the widely used LAMP open source web application software stack—LAMP is an acronym for "Linux, Apache, MySQL, Perl/PHP/Python." Free-software-open source projects that require a full-featured database management system often use MySQL. For

commercial use, several paid editions are available, and offer additional functionality. Applications which use MySQL databases include: TYPO3, Joomla, Word Press, phpBB, MyBB, Drupal and other software built on the LAMP software stack. MySQL is also used in many high-profile, large-scale World Wide Web products, including Wikipedia, Google(though not for searches), Imagebook Twitter, Flickr, Nokia.com, and YouTube.

Inter images

MySQL is primarily an RDBMS and ships with no GUI tools to administer MySQL databases or manage data contained within the databases. Users may use the included command line tools, or use MySQL "front-ends", desktop software and web applications that create and manage MySQL databases, build database structures, back up data, inspect status, and work with data records. The official set of MySQL front-end tools, MySQL Workbench is actively developed by Oracle, and is freely available for use.

Graphical

The official MySQL Workbench is a free integrated environment developed by MySQL AB that enables users to graphically administer MySQL databases and visually design database structures. MySQL Workbench replaces the previous package of software, MySQL GUI Tools. Similar to other third-party packages, but still considered the authoritative MySQL frontend, MySQL Workbench lets users manage database design & modeling, SQL development (replacing MySQL Query Browser) and Database administration (replacing MySQL Administrator).MySQL Workbench is available in two editions, the regular free and open source Community Edition which may be downloaded from the MySQL website, and the proprietary Standard Edition which extends and improves the feature set of the Community Edition.

CHAPTER 7

7.SYSTEM TESTING

7.1 TESTING

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs (errors or other defects). Software testing can be stated as the process of validating and verifying that a computer program/application/product:

1. meets the requirements that guided its design and development,
2. works as expected,
3. It can be implemented with the same characteristics, and satisfies the needs of stakeholders.

Software testing, depending on the testing method employed, can be implemented at any time in the development process. Traditionally most of the test effort occurs after the requirements have been defined and the coding process has been completed, but in the agile approaches most of the test effort is on-going. As such, the methodology of the test is governed by the chosen software development methodology. Different software development models will focus the test effort at different points in the development process. Newer development models, such as Agile, often employ test-driven development and place an increased portion of the testing in the hands of the developer, before it reaches a formal team of testers. In a more traditional model, most of the test execution occurs after the requirements have been defined and the coding process has been completed.

7.1.1 OVERVIEW OF TESTING

Testing can never completely identify all the defects within software. Instead, it furnishes a criticism or comparison that compares the state and behaviour of the product against oracles principles or mechanisms by which someone might recognize a problem. These oracles may include (but are not limited to) specifications, contracts, comparable products, past versions of the same product, inferences about intended or expected purpose, user or customer expectations, relevant standards, applicable laws, or other criteria. A primary purpose of testing

is to detect software failures so that defects may be discovered and corrected. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions. The scope of software testing often includes examination of code as well as execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do. In the current culture of software development, a testing organization may be separate from the development team. There are various roles for testing team members. Information derived from software testing may be used to correct the process by which software is developed. Every software product has a target audience. For example, the audience for video game software is completely different from banking software. Therefore, when an organization develops or otherwise invests in a software product, it can assess whether the software product will be acceptable to its end users, its target audience, its purchasers, and other stakeholders. Software testing is the process of attempting to make this assessment

7.2 TESTING METHODS

7.2.1 Static vs. Dynamic Testing

There are many approaches to software testing. Reviews, walkthroughs, or inspections are referred to as static testing, whereas actually executing programmed code with a given set of test cases is referred to as dynamic testing. Static testing can be omitted, and unfortunately in practice often is. Dynamic testing takes place when the program itself is used. Dynamic testing may begin before the program is 100% complete in order to test particular sections of code and are applied to discrete functions or modules. Typical techniques for this are either using stubs/drivers or execution from a debugger environment. The box approach Software testing methods are traditionally divided into white- and black-box testing. These two approaches are used to describe the point of view that a test engineer takes when designing test cases.

7.2.2 White-Box Testing

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) tests internal structures or workings of a program, as opposed to the functionality exposed to the end users. In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g., in-circuit testing (ICT). While white-box testing can be applied

at the unit, integration and system levels of the software testing process, it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during a system-level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements. Techniques used in white-box testing include: 1. API testing (application programming interface) - testing of the application using public and private APIs 2. Code coverage - creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once) 3. Fault injection methods - intentionally introducing faults to gauge the efficacy of testing strategies

7.2.3 Static Testing Methods

Code coverage tools can evaluate the completeness of a test suite that was created with any method, including black-box testing. This allows the software team to examine parts of a system that are rarely tested and ensures that the most important function points have been tested. Code coverage as a software metric can be reported as a percentage for: 1. Function coverage, which reports on functions executed 2. Statement coverage, which reports on the number of lines executed to complete the test 100% statement coverage ensures that all code paths, or branches (in terms of control flow) are executed at least once. This is helpful in ensuring correct functionality, but not sufficient since the same code may process different inputs correctly or incorrectly.

7.2.4 Black-Box Testing

Black-box testing treats the software as a "black box", examining functionality without any knowledge of internal implementation. The tester is only aware of what the software is supposed to do, not how it does it. Black-box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, state transition tables, decision table testing, fuzz testing, model-based testing, use case testing, exploratory testing and specification-based testing. Specification-based testing aims to test the functionality of software according to the applicable requirements. This level of testing usually requires thorough test cases to be provided to the tester, who then can simply verify that for a given input, the output value (or behavior), either "is" or "is not" the same as the expected value specified in the test case. Test cases are built around specifications and requirements, i.e., what the application is supposed to do. It uses external descriptions of the software, including specifications, requirements, and designs to derive test cases. These tests can be functional or non-functional, though usually

functional. Specification based testing may be necessary to assure correct functionality, but it is insufficient to guard against complex or high-risk situations. One advantage of the black box technique is that no programming knowledge is required. Whatever biases the programmers may have had, the tester likely has a different set and may emphasize different areas of functionality. On the other hand, black-box testing has been said to be "like a walk in a dark labyrinth without a flashlight." Because they do not examine the source code, there are situations when a tester writes many test cases to check something that could have been tested by only one test case, or leaves some parts of the program untested. This method of test can be applied to all levels of software testing: unit, integration, system and acceptance. It typically comprises most if not all testing at higher levels, but can also dominate unit testing as well.

7.2.5 Grey-Box Testing

Grey-box testing involves having knowledge of internal data structures and algorithms for purposes of designing tests, while executing those tests at the user, or black-box level. The tester is not required to have full access to the software's source code. Manipulating input data and formatting output do not qualify as grey-box, because the input and output are clearly outside of the "black box" that we are calling the system under test. This distinction is particularly important when conducting integration testing between two modules of code written by two different developers, where only the interfaces are exposed for test. However, modifying a data repository does qualify as grey-box, as the user would not normally be able to change the data outside of the system under test. Grey-box testing may also include reverse engineering to determine, for instance, boundary values or error messages. By knowing the underlying concepts of how the software works, the tester makes better-informed testing choices while testing the software from outside. Typically, a grey-box tester will be permitted to set up his testing environment; for instance, seeding a database; and the tester can observe the state of the product being tested after performing certain actions. For instance, in testing a database product he/she may fire an SQL query on the database and then observe the database, to ensure that the expected changes have been reflected. Grey-box testing implements intelligent test scenarios, based on limited information. This will particularly apply to data type handling, exception handling, and so on.

Visual testing

The aim of visual testing is to provide developers with the ability to examine what was happening at the point of software failure by presenting the data in such a way that the

developer can easily find the information he requires, and the information is expressed clearly. At the core of visual testing is the idea that showing someone a problem (or a test failure), rather than just describing it, greatly increases clarity and understanding. Visual testing therefore requires the recording of the entire test process – capturing everything that occurs on the test system in video format. Output videos are supplemented by real-time tester input via picture-in-a-picture webcam and audio commentary from microphones. Visual testing provides a number of advantages. The quality of communication is increased dramatically because testers can show the problem (and the events leading up to it) to the developer as opposed to just describing it and the need to replicate test failures will cease to exist in many cases. The developer will have all the evidence he requires of a test failure and can instead focus on the cause of the fault and how it should be fixed. Visual testing is particularly well-suited for environments that deploy agile methods in their development of software, since agile methods require greater communication between testers and developers and collaboration within small teams. Visual testing is gathering recognition in customer acceptance and usability testing, because the test can be used by many individuals involved in the development process. For the customer, it becomes easy to provide detailed bug reports and feedback, and for program users, visual testing can record user actions on screen, as well as their voice and image, to provide a complete picture at the time of software failure for the developer.

7.3 TESTING LEVELS

Tests are frequently grouped by where they are added in the software development process, or by the level of specificity of the test. The main levels during the development process as defined by the SWEBOK guide are unit-, integration-, and system testing that are distinguished by the test target without implying a specific process model. Other test levels are classified by the testing objective.

7.3.1 Unit Testing

Unit testing, also known as component testing, refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors. These types of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone

cannot verify the functionality of a piece of software, but rather is used to assure that the building blocks the software uses work independently of each other.

7.3.2 Integration Testing

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be localised more quickly and fixed. Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.

7.3.3 System Testing

System testing tests a completely integrated system to verify that it meets its requirements.

7.3.4 Acceptance Testing

At last the system is delivered to the user for Acceptance testing

7.4 TESTING APPROACHES

7.4.1 Bottom-Up

Bottom-Up Testing is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher-level components. The process is repeated until the component at the top of the hierarchy is tested. All the bottom or low-level modules, procedures or functions are integrated and then tested. After the integration testing of lower-level integrated modules, the next level of modules will be formed and can be used for integration testing. This approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage.

7.4.2 Top-Down

Top-Down Testing is an approach to integrated testing where the top integrated modules are tested and the branch of the module is tested step by step until the end of the related module.

7.5 OBJECTIVES OF TESTING

7.5.1 Installation Testing

An installation test assures that the system is installed correctly and working at actual customer's hardware.

7.5.2 Compatibility Testing

A common cause of software failure (real or perceived) is a lack of its compatibility with other application software, operating systems (or operating system versions, old or new), or target environments that differ greatly from the original (such as a terminal or GUI application intended to be run on the desktop now being required to become a web application, which must render in a web browser). For example, in the case of a lack of backward compatibility, this can occur because the programmers develop and test software only on the latest version of the target environment, which not all users may be running. This results in the unintended consequence that the latest work may not function on earlier versions of the target environment, or on older hardware that earlier versions of the target environment was capable of using. Sometimes such issues can be fixed by proactively abstracting operating system functionality into a separate program module or library.

7.5.3 Smoke and Sanity Testing

Sanity testing determines whether it is reasonable to proceed with further testing. Smoke testing is used to determine whether there are serious problems with a piece of software, for example as a build verification test.

7.5.4 Regression Testing

Regression testing focuses on finding defects after a major code change has occurred. Specifically, it seeks to uncover software regressions, or old bugs that have come back. Such regressions occur whenever software functionality that was previously working correctly stops working as intended. Typically, regressions occur as an unintended consequence of program changes, when the newly developed part of the software collides with the previously existing code. Common methods of regression testing include re-running previously run tests and checking whether previously fixed faults have re-emerged. The depth of testing depends on the phase in the release process and the risk of the added features. They can either be complete, for changes added late in the release or deemed to be risky, to be very shallow, consisting of

positive tests on each feature, if the changes are early in the release or deemed to be of low risk.

7.5.5 Alpha Testing

Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing, before the software goes to beta testing.

7.5.6 Beta Testing

Beta testing comes after alpha testing and can be considered a form of external user acceptance testing. Versions of the software, known as beta versions, are released to a limited audience outside of the programming team. The software is released to groups of people so that further testing can ensure the product has few faults or bugs. Sometimes, beta versions are made available to the open public to increase the feedback field to a maximal number of future users.

7.5.7 Functional vs Non-functional Testing

Functional testing refers to activities that verify a specific action or function of the code. These are usually found in the code requirements documentation, although some development methodologies work from use cases or user stories. Functional tests tend to answer the question of "can the user do this" or "does this particular feature work." Non-functional testing refers to aspects of the software that may not be related to a specific function or user action, such as scalability or other performance, behaviour under certain constraints, or security.

7.5.8 Destructive Testing

Destructive testing attempts to cause the software or a sub-system to fail. It verifies that the software functions properly even when it receives invalid or unexpected inputs, thereby establishing the robustness of input validation and error-management routines. Software fault injection, in the form of fuzzing, is an example of failure testing. Various commercial non-functional testing tools are linked from the software fault injection page; there are also numerous open-source and free software tools available that perform destructive testing.

7.5.9 Software Performance Testing

Performance testing is in general executed to determine how a system or sub-system performs in terms of responsiveness and stability under a particular workload. It can also serve

to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage. Load testing is primarily concerned with testing that the system can continue to operate under a specific load, whether that be large quantities of data or a large number of users. This is generally referred to as software scalability. The related load testing activity of when performed as a non-functional activity is often referred to as endurance testing. Volume testing is a way to test software functions even when certain components (for example a file or database) increase radically in size. Stress testing is a way to test reliability under unexpected or rare workloads.

7.5.10 Usability Testing

Usability testing is needed to check if the user interface is easy to use and understand. It is concerned mainly with the use of the application. It is important to check interface is working properly or not as planned

7.5.11 Accessibility Testing

Accessibility testing may include compliance with standards such as:

1. Americans with Disabilities Act of 1990
2. Section 508 Amendment to the Rehabilitation Act of 1973
3. Web Accessibility Initiative (WAI) of the World Wide Web Consortium

7.5.12 Security Testing

Security testing is essential for software that processes confidential data to prevent system intrusion by hackers. So, it is important to perform security testing to find loop holes in the developed software.

7.5.13 Development Testing

Development Testing is a software development process that involves synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce software development risks, time, and costs. It is performed by the software developer or engineer during the construction phase of the software development lifecycle. Rather than replace traditional QA focuses, it augments it.

CHAPTER 8

8. CONCLUSION AND FUTURE ENHANCEMENT

8.1 CONCLUSION

In this project the problem of constraining and summarizing different algorithms of data mining used in the field of medical prediction are discussed. The focus is on using different algorithms and combinations of several target attributes for intelligent and effective heart disease prediction using data mining. Data mining technology provides an important means for extracting valuable medical rules hidden in medical data and acts as an important role in disease prediction and clinical diagnosis. There is an increasing interest in using classification to identify disease which is present or not. In the current study, have demonstrated, using a large sample of patients hospitalized with classification. Classification algorithm is very sensitive to noisy data. If any noisy data is present then it causes very serious problems regarding to the processing power of classification. It not only slows down the task of classification algorithm but also degrades its performance. Hence, before applying classification algorithm it must be necessary to remove all those attributes from datasets who later on acts as noisy attributes. In this research work, we can implement pre-processing steps and implemented the classification rule algorithms namely multi-layer perceptron is used for classifying datasets which are uploaded by user. By analysing the experimental results, it is observed that the multi-layer perceptron technique has yields better result than other techniques.

8.2 FUTURE ENHANCEMENT

In future we tend to improve efficiency of performance by applying other data mining techniques and algorithms.

APPENDIX 1. CODING

APP.PY

```
from flask import Flask, render_template, flash, request, session, send_file
import pickle
import numpy as np
import mysql.connector
import sys

app = Flask(__name__)
app.config['DEBUG']
app.config['SECRET_KEY'] = '7d441f27d441f27567d441f2b6176a'

@app.route("/")

def homepage():
    return render_template('index.html')

@app.route("/Home")

def Home():
    return render_template('index.html')

@app.route("/AdminLogin")

def AdminLogin():
    return render_template('AdminLogin.html')

@app.route("/NewDoctor")

def NewDoctor():
    return render_template('NewDoctor.html')
```

```
@app.route("/DoctorLogin")
```

```
def DoctorLogin():
```

```
    return render_template('DoctorLogin.html')
```

```
@app.route("/UserLogin")
```

```
def UserLogin():
```

```
    return render_template('UserLogin.html')
```

```
@app.route("/NewUser")
```

```
def NewUser():
```

```
    return render_template('NewUser.html')
```

```
@app.route("/AdminHome")
```

```
def AdminHome():
```

```
    conn = mysql.connector.connect(user='root', password="", host='localhost',  
database='5heartbd')
```

```
    cur = conn.cursor()
```

```
    cur.execute("SELECT * FROM regtb ")
```

```
    data = cur.fetchall()
```

```
    return render_template('AdminHome.html', data=data)
```

```
@app.route("/adminlogin", methods=['GET', 'POST'])
```

```
def adminlogin():
```

```
    if request.method == 'POST':
```

```
        if request.form['uname'] == 'admin' and request.form['password'] == 'admin':
```

```
conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')
```

```
cur = conn.cursor()
```

```
cur.execute("SELECT * FROM regtb ")
```

```
data = cur.fetchall()
```

```
flash("Login successfully")
```

```
return render_template('AdminHome.html', data=data)
```

```
else:
```

```
flash("UserName Or Password Incorrect!")
```

```
return render_template('AdminLogin.html')
```

```
@app.route("/DoctorInfo")
```

```
def DoctorInfo():
```

```
conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')
```

```
cur = conn.cursor()
```

```
cur.execute("SELECT * FROM doctortb ")
```

```
data = cur.fetchall()
```

```
return render_template('DoctorInfo.html', data=data)
```

```
@app.route("/ADRemove")
```

```
def ADRemove():
```

```
id = request.args.get('id')
```

```
conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')
```

```
cursor = conn.cursor()
```

```
cursor.execute(
```

```

        "delete from doctortb where id='" + id + "'"")
conn.commit()
conn.close()

```

```

flash('Doctor info Remove Successfully!')

```

```

conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')
cur = conn.cursor()
cur.execute("SELECT * FROM doctortb ")
data = cur.fetchall()
return render_template('DoctorInfo.html', data=data)

```

#In this phase, the code explains about how the app routes to the home page
#The code it will continues to the appointment page.

```

@app.route("/AURemove")
def AURemove():
    id = request.args.get('id')

    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

    cursor = conn.cursor()
    cursor.execute(
        "delete from regtb where id='" + id + "'"")
    conn.commit()
    conn.close()

    flash('User info Remove Successfully!')

    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

```

```

cur = conn.cursor()
cur.execute("SELECT * FROM regtb ")
data = cur.fetchall()
return render_template('AdminHome.html', data=data)

```

```

@app.route("/ADrugInfo")

```

```

def ADrugInfo():

```

```

    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

```

```

    cur = conn.cursor()
    cur.execute("SELECT * FROM apptb ")
    data = cur.fetchall()

```

```

    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

```

```

    cur = conn.cursor()
    cur.execute("SELECT * FROM drugtb ")
    data1 = cur.fetchall()
    return render_template('ADrugInfo.html', data=data, data1=data1)

```

```

@app.route("/newdoct", methods=['GET', 'POST'])

```

```

def newdoct():

```

```

    if request.method == 'POST':
        name = request.form['name']

        mobile = request.form['mobile']
        email = request.form['email']

        address = request.form['address']

```

```

specialist = request.form['Specialist']

uname = request.form['uname']
password = request.form['password']

conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

cursor = conn.cursor()

cursor.execute(
    "INSERT INTO doctortb VALUES ('" + name + "','" + email + "','" + mobile + "','" +
address + "','" + specialist + "','" + uname + "','" + password + "')"
)

conn.commit()

conn.close()

flash('Doctor Register Successfully')

return render_template('DoctorLogin.html')

@app.route("/doctlogin", methods=['GET', 'POST'])

def doctlogin():

    if request.method == 'POST':

        username = request.form['uname']

        password = request.form['password']

        session['ename'] = request.form['uname']

        conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

        cursor = conn.cursor()

        cursor.execute("SELECT * from doctortb where username='" + username + "' and
Password='" + password + "'")

        data = cursor.fetchone()

        if data is None:

```

```

        flash('Username or Password is wrong')

        return render_template('DoctorLogin.html', data=data)
    else:

        conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

        cur = conn.cursor()

        cur.execute("SELECT * FROM doctortb where username='" + username + "' and
Password='" + password + "'")

        data = cur.fetchall()

        flash("Login successfully")

        return render_template('DoctorHome.html', data=data)

```

```

@app.route("/DoctorHome")

```

```

def DoctorHome():

```

```

    username = session['ename']

    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

    # cursor = conn.cursor()

    cur = conn.cursor()

    cur.execute("SELECT * FROM doctortb where username='" + username + "' ")

    data = cur.fetchall()

    return render_template('DoctorHome.html', data=data)

```

```

@app.route("/DAppoitmentInfo")

```

```

def DAppoitmentInfo():

```

```

    username = session['ename']

    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

    cur = conn.cursor()

```



```
cur.execute("SELECT * FROM apptb where DoctorName='" + username + "' and
Status='waiting' ")
```

```
data = cur.fetchall()
```

```
conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')
```

```
cur = conn.cursor()
```

```
cur.execute("SELECT * FROM apptb where DoctorName='" + username + "' and
Status!='waiting' ")
```

```
data1 = cur.fetchall()
```

```
return render_template('DAppoitmentInfo.html', data=data, data1=data1)
```

```
@app.route("/Accept")
```

```
def Accept():
```

```
id = request.args.get('id')
```

```
conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')
```

```
cursor = conn.cursor()
```

```
cursor.execute(
```

```
    "update apptb set status='Accept' where id='" + id + "'"")
```

```
conn.commit()
```

```
conn.close()
```

```
conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')
```

```
cursor = conn.cursor()
```

```
cursor.execute("SELECT * FROM apptb where id='" + id + "'")
```

```
data1 = cursor.fetchone()
```

```

if data1:

    mobile = data1[2]

    sendmsg(mobile, "Appointment Request Accept..!")

    flash('Appointment Status Update Successfully!')

    username = session['ename']

    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

    cur = conn.cursor()

    cur.execute("SELECT * FROM apptb where DoctorName='" + username + "' and
Status='waiting' ")

    data = cur.fetchall()


    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

    cur = conn.cursor()

    cur.execute("SELECT * FROM apptb where DoctorName='" + username + "' and
Status!='waiting' ")

    data1 = cur.fetchall()

    return render_template('DAppoitmentInfo.html', data=data, data1=data1)

@app.route("/Reject")

def Reject():

    id = request.args.get('id')

    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

    cursor = conn.cursor()

```

```

cursor.execute(
    "update apptb set status='Reject' where id='" + id + "'"")
conn.commit()
conn.close()

conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')
cursor = conn.cursor()
cursor.execute("SELECT * FROM apptb where id='" + id + "'")
data1 = cursor.fetchone()

if data1:
    mobile = data1[2]

    sendmsg(mobile, "Appointment Request Reject..!")

flash('Appointment Status Update Successfully!')

username = session['ename']

conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')
cur = conn.cursor()

cur.execute("SELECT * FROM apptb where DoctorName='" + username + "' and
Status='waiting' ")
data = cur.fetchall()

conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')
cur = conn.cursor()

cur.execute("SELECT * FROM apptb where DoctorName='" + username + "' and
Status!='waiting' ")
data1 = cur.fetchall()

```

```
return render_template('DAppoitmentInfo.html', data=data, data1=data1)
```

```
@app.route("/AssignDrug")
```

```
def AssignDrug():
```

```
    id = request.args.get('id')
```

```
    st = request.args.get('st')
```

```
    session['apid'] = id
```

```
    if st == "Accept":
```

```
        return render_template('DAssignDrug.html')
```

```
    else:
```

```
        flash("Appointment Reject")
```

```
        username = session['ename']
```

```
        conn = mysql.connector.connect(user='root', password="", host='localhost',  
database='5heartbd')
```

```
        cur = conn.cursor()
```

```
        cur.execute("SELECT * FROM apptb where DoctorName='" + username + "' and  
Status='waiting' ")
```

```
        data = cur.fetchall()
```

```
        conn = mysql.connector.connect(user='root', password="", host='localhost',  
database='5heartbd')
```

```
        cur = conn.cursor()
```

```
        cur.execute("SELECT * FROM apptb where DoctorName='" + username + "' and  
Status!='waiting' ")
```

```
        data1 = cur.fetchall()
```

```

        return render_template('DAppoitmentInfo.html', data=data, data1=data1)

@app.route("/drugs", methods=['GET', 'POST'])

def drugs():
    if request.method == 'POST':

        date = request.form['date']
        minfo = request.form['minfo']
        oinfo = request.form['oinfo']
        file = request.files['file']
        file.save("static/upload/" + file.filename)

        conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

        cursor = conn.cursor()
        cursor.execute("SELECT * FROM apptb where id='" + session['apid'] + "'")
        data = cursor.fetchone()

        if data:
            uname = data[1]
            mobile = data[2]
            email = data[3]
            dname = data[4]

        else:

            return 'Incorrect username / password !'

        conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

```

```

        cursor = conn.cursor()

        cursor.execute(
            "INSERT INTO drugtb VALUES ('" + uname + "','" + mobile + "','" + email + "','" +
            dname + "','" +
                minfo + "','" + oinfo + "','" + file.filename + "','" + date + "')"
        )
        conn.commit()
        conn.close()

```

```

        conn = mysql.connector.connect(user='root', password="", host='localhost',
        database='5heartbd')

        cur = conn.cursor()

        cur.execute("SELECT * FROM drugtb where DoctorName='" + dname + "' ")

        data = cur.fetchall()

        return render_template('DrugsInfo.html', data=data)

```

```

@app.route("/DrugsInfo")

```

```

def DrugsInfo():

```

```

    username = session['ename']

    conn = mysql.connector.connect(user='root', password="", host='localhost',
    database='5heartbd')

    cur = conn.cursor()

    cur.execute("SELECT * FROM drugtb where DoctorName='" + username + "' ")

    data = cur.fetchall()

    return render_template('DrugsInfo.html', data=data)

```

```

@app.route('/download')

```

```

def download():

```

```

    id = request.args.get('id')

    conn = mysql.connector.connect(user='root', password="", host='localhost',
    database='5heartbd')

```

```

cursor = conn.cursor()
cursor.execute("SELECT * FROM drugtb where id = '" + str(id) + "'")
data = cursor.fetchone()
if data:
    filename = "static\\upload\\"+data[7]

    return send_file(filename, as_attachment=True)

else:
    return 'Incorrect username / password !'

@app.route("/newuser", methods=['GET', 'POST'])

def newuser():
    if request.method == 'POST':
        name = request.form['name']
        mobile = request.form['mobile']

        email = request.form['email']

        address = request.form['address']

        uname = request.form['uname']
        password = request.form['password']

        conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

        cursor = conn.cursor()

        cursor.execute(

            "INSERT INTO regtb VALUES ('" + name + "','" + email + "','" + mobile + "','" +
address + "','" + uname + "','" + password + "')"

```

```

conn.commit()

conn.close()

flash('User Register successfully')


return render_template('UserLogin.html')


@app.route("/userlogin", methods=['GET', 'POST'])

def userlogin():
    if request.method == 'POST':
        username = request.form['uname']
        password = request.form['password']
        session['uname'] = request.form['uname']

        conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

        cursor = conn.cursor()

        cursor.execute("SELECT * from regtb where username='" + username + "' and
Password='" + password + "'")

        data = cursor.fetchone()

        if data is None:

            flash('Username or Password is wrong')

            return render_template('UserLogin.html')

        else:

            conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

            cur = conn.cursor()

            cur.execute("SELECT * FROM regtb where username='" + username + "' and
Password='" + password + "'")

            data = cur.fetchall()

```



```

        flash("Login successfully")

        return render_template('UserHome.html', data=data)

@app.route("/UserHome")

def UserHome():
    uname = session['uname']

    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

    # cursor = conn.cursor()

    cur = conn.cursor()
    cur.execute("SELECT * FROM regtb where username='" + uname + "' ")
    data = cur.fetchall()
    return render_template('UserHome.html', data=data)

@app.route("/Heart")

def Heart():
    return render_template('Heart.html')

@app.route("/heart", methods=['GET', 'POST'])
def heart():

    if request.method == 'POST':

        Answer = "
        Prescription = "
        uname = session['uname']
        age1 = request.form['age']

```

```

gender = request.form['gender']
height = request.form['height']
weight = request.form['weight']
aphi = request.form['aphi']
aplo = request.form['aplo']
choles = request.form['choles']
glucose = request.form['glucose']
smoke = request.form['smoke']
alcohol = request.form['alcohol']
activity = request.form['activity']

```

```

age = int(age1)*(365)

```

#In this phase, the code explains about how the app routes to the prediction page

#The code it will continues to the appointment page.

```

filename2 = 'Heart/heart-model.pkl'
classifier2 = pickle.load(open(filename2, 'rb'))

```

```

data = np.array([[int(age), int(gender), int(height), int(weight), int(aphi), float(aplo),
float(choles), int(glucose), int(smoke), int(alcohol), int(activity)]])

```

```

print(data)

```

```

my_prediction = classifier2.predict(data)

```

```

print(my_prediction[0])

```

```

if my_prediction == 1:

```

```

    session['Ans'] = 'Yes'

```

```

    # Heart Cancer Diabetes

```

```

    session['dtype'] = 'heart'

```

```

    Answer = session['uname'] + ' :According to our Calculations, You have Heart disease'

```

```

print('Hello:According to our Calculations, You have Heart disease')

ans = 'Heart disease'

Prescription = "Angiotensin-converting enzyme (ACE) inhibitors, Food: fish or
seafood."

else:

    Answer = session['uname'] + " Congratulations!! You DON'T have Heart disease"
    ans = 'No Heart disease'
    print('Congratulations!! You DON T have Heart disease')
    Prescription = "Nill"

    session['Ans'] = 'No'
    # Heart Cancer Diabetes
    session['dtype'] = 'heart'

return render_template('Answer.html', data=Answer,pre =Prescription)

#In this phase, the code explains about how the app routes to the appointment page
#The code it will continues to the appointment scheduling page.

@app.route("/ViewDoctor", methods=['GET', 'POST'])

def ViewDoctor():
    if request.method == 'POST':

        ans = session['Ans']

        if ans == 'Yes':

            conn = mysql.connector.connect(user='root', password="", host='localhost',
            database='5heartbd')

```

```

# cursor = conn.cursor()

cur = conn.cursor()

cur.execute("SELECT * FROM doctortb ")

data = cur.fetchone()

if data is None:

    uname = session['uname']

    flash('No Doctor Found!')

    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

    cur = conn.cursor()

    cur.execute("SELECT * FROM regtb where username='" + uname + "' ")

    data = cur.fetchall()

    return render_template('UserHome.html', data=data)

else:

    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

    cur = conn.cursor()

    cur.execute("SELECT * FROM doctortb ")

    data = cur.fetchall()

    return render_template('ViewDoctor.html', data=data)

#The below code explains about the nature of login credentials

#It shows the efficiency of the secure login purpose

else:

    uname = session['uname']

```

```
conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')
```

```
cur = conn.cursor()
```

```
cur.execute("SELECT * FROM regtb where username='" + uname + "' ")
```

```
data = cur.fetchall()
```

```
return render_template('UserHome.html', data=data)
```

```
@app.route("/Appointment")
```

```
def Appointment():
```

```
    dname = request.args.get('id')
```

```
    session['dname'] = dname
```

```
    return render_template('Appointment.html')
```

```
@app.route("/appointment", methods=['GET', 'POST'])
```

```
def appointment():
```

```
    if request.method == 'POST':
```

```
        dname = session['dname']
```

```
        uname = session['uname']
```

```
        date = request.form['date']
```

```
        info = request.form['info']
```

```
        conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')
```

```
        cursor = conn.cursor()
```

```
        cursor.execute("SELECT * FROM regtb where UserNAme='" + uname + "'")
```

```
        data = cursor.fetchone()
```

```
        if data:
```

```
            mobile = data[3]
```

```
            email = data[2]
```

```

else:

    return 'Incorrect username / password !'


    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

    cursor = conn.cursor()

    cursor.execute("SELECT * FROM doctortb where UserName='" + dname + "'")

    data1 = cursor.fetchone()

    if data1:

        mobile = data1[3]


        sendmsg(mobile,"Appointment Request Received..!")


        conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

        cursor = conn.cursor()

        cursor.execute(

            "INSERT INTO apptb VALUES ('" + uname + "','" + mobile + "','" + email + "','" +
dname + "','" + date + "','" + info + "','waiting')")

        conn.commit()

        conn.close()


        uname = session['uname']

        conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

        cur = conn.cursor()

        cur.execute("SELECT * FROM apptb where username='" + uname + "' ")

        data = cur.fetchall()

        conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

        cur = conn.cursor()

        cur.execute("SELECT * FROM drugtb where username='" + uname + "' ")

```

```

        data1 = cur.fetchall()

        return render_template('UDrugsInfo.html', data=data, data1=data1)

@app.route("/UDrugsInfo")

def UDrugsInfo():

    uname = session['uname']

    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

    cur = conn.cursor()

    cur.execute("SELECT * FROM apptb where username='" + uname + "' ")

    data = cur.fetchall()

    conn = mysql.connector.connect(user='root', password="", host='localhost',
database='5heartbd')

    cur = conn.cursor()

    cur.execute("SELECT * FROM drugtb where username='" + uname + "' ")

    data1 = cur.fetchall()

    return render_template('UDrugsInfo.html', data=data, data1=data1)

def sendmsg(targetno,message):

    import requests

    requests.post(

"http://sms.creativepoint.in/api/push.json?apikey=6555c521622c1&route=transsms&sender=
FSSMSS&mobilenos=" + targetno + "&text=Dear customer your msg is " + message + " Sent
By FSMSG FSSMSS")

if __name__ == '__main__':

    app.run(debug=True, use_reloader=True)

```

HEARTDEPLOYMENT.PY

```
import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import seaborn as sns

from matplotlib import pyplot as plt

import os

import pickle

from sklearn.model_selection import train_test_split

#https://www.kaggle.com/datasets/sulianova/cardiovascular-disease-dataset/data

df = pd.read_csv("cardio_train.csv", sep=";")

print(df.head())

print(df.info())

from matplotlib import rcParams

rcParams['figure.figsize'] = 11, 8

df['years'] = (df['age'] / 365).round().astype('int')

sns.countplot(x='years', hue='cardio', data=df, palette="Set2");

plt.show()

df_categorical = df.loc[:, ['cholesterol', 'gluc', 'smoke', 'alco', 'active']]

sns.countplot(x="variable", hue="value", data=pd.melt(df_categorical));

plt.show()
```



```
df_long = pd.melt(df, id_vars=['cardio'], value_vars=['cholesterol', 'gluc', 'smoke', 'alco',  
'active'])
```

```
sns.catplot(x="variable", hue="value", col="cardio",  
            data=df_long, kind="count");
```

```
plt.show()
```

```
df.isnull().values.any()
```

```
X = df[['age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo', 'cholesterol', 'gluc', 'smoke', 'alco',  
'active']]
```

```
y = df['cardio']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=0)
```

```
from sklearn.neural_network import MLPClassifier
```

```
#from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import classification_report
```

```
# Train the model
```

```
#classifier = MLPClassifier(hidden_layer_sizes=(100, 50), activation='relu', solver='adam',  
max_iter=500)
```

```
classifier = MLPClassifier()
```

```
classifier.fit(X_train, y_train)
```

```

y_pred = classifier.predict(X_test)

print(classification_report(y_test, y_pred))

clreport = classification_report(y_test, y_pred)

print("Accuracy on training set: {:.2f}".format(classifier.score(X_train, y_train)))

print("Accuracy on test set: {:.3f}".format(classifier.score(X_test, y_test)))

Tacc = "Accuracy on training set: {:.2f}".format(classifier.score(X_train, y_train))

Testacc = "Accuracy on test set: {:.3f}".format(classifier.score(X_test, y_test))

# Creating a pickle file for the classifier

filename = 'heart-model.pkl'

pickle.dump(classifier, open(filename, 'wb'))

"data = np.array([[age, gender, height, weight, aphi, aplo, choles, glucose, smoke, alcohol]])

my_prediction = classifier2.predict(data)

print(my_prediction[0])"

```

APPENDIX 2: SCREENSHOTS

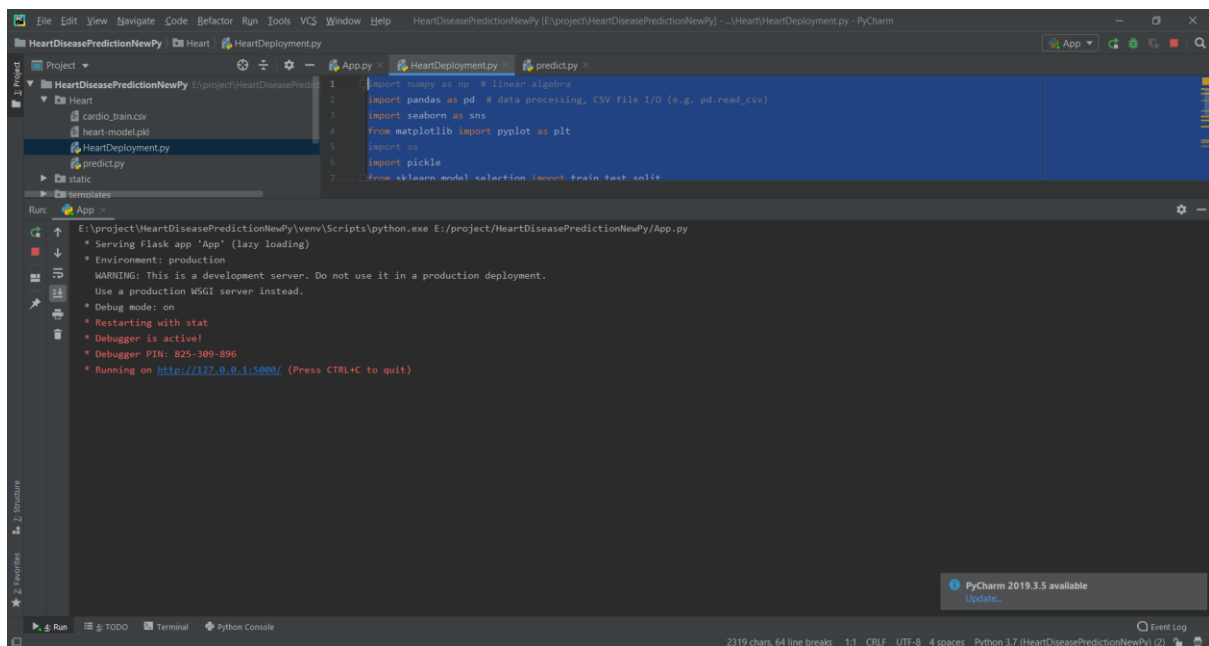


Figure A2.1 Web Application Link

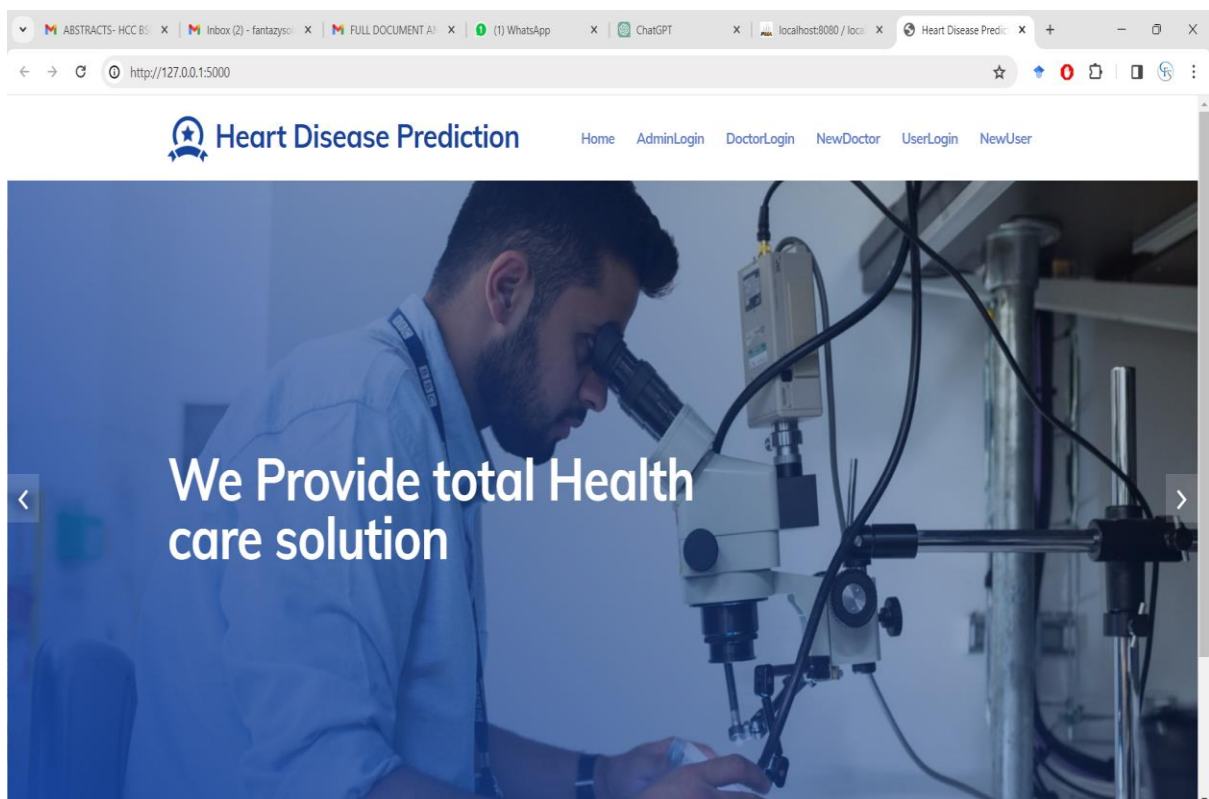


Figure A2.2 Web Application-Home Page

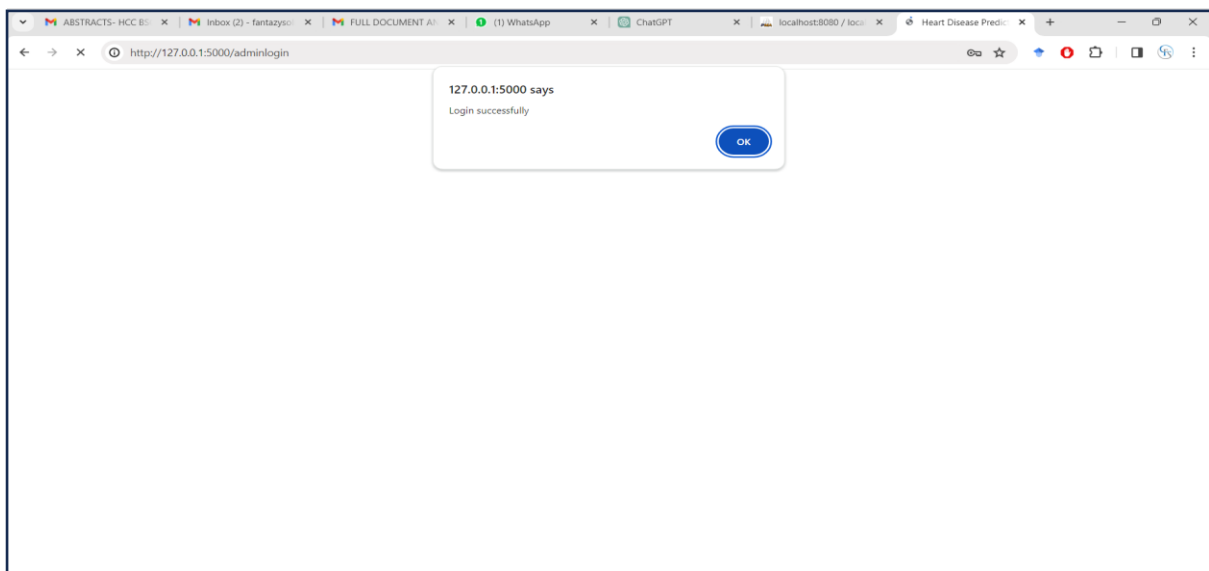
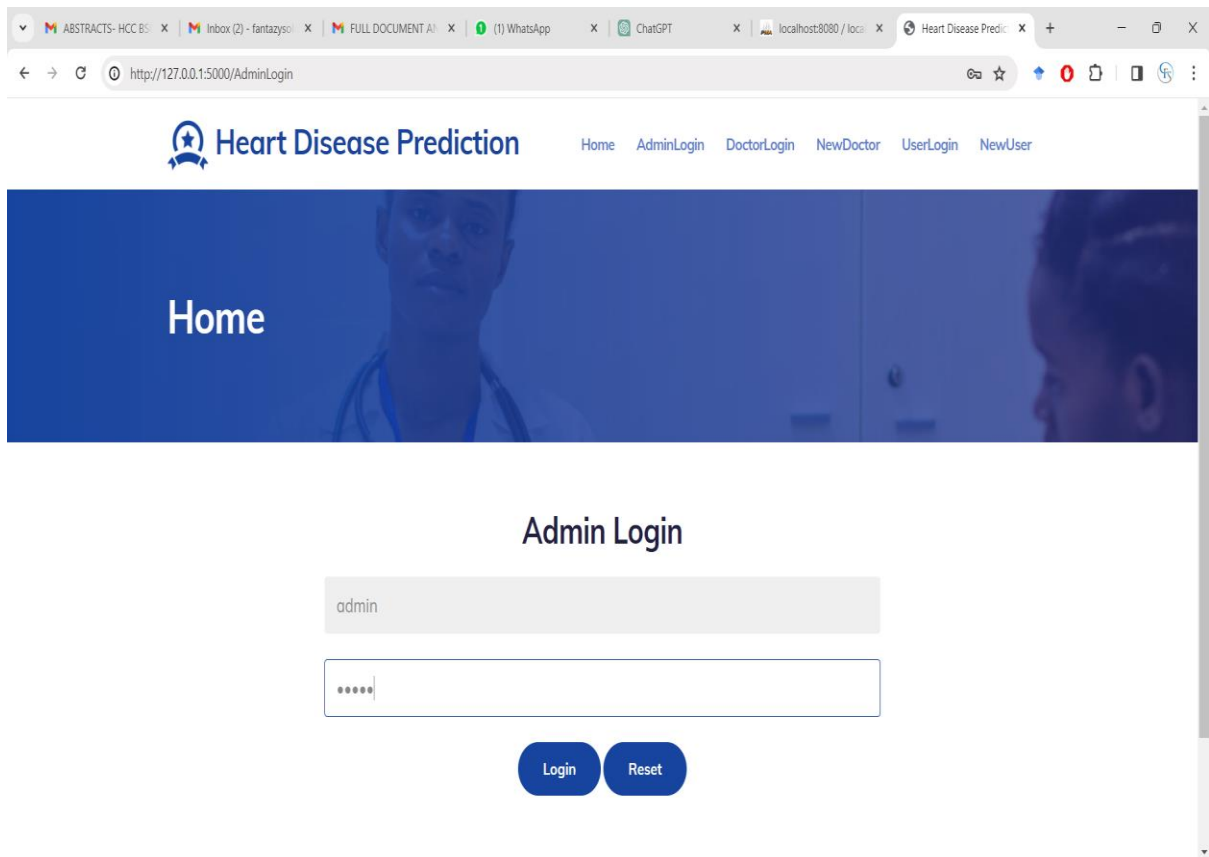


Figure A2.3 Admin Login Page

The screenshot shows the 'Admin' section of the 'Heart Disease Prediction' web application. The browser address bar indicates the URL is `http://127.0.0.1:5000/adminlogin`. The page features a blue header with the application logo and navigation links: Home, DoctorInfo, DrugInfo, and Logout. Below the header is a large blue banner with the word 'Admin' in white. The main content area is titled 'User Information' and contains a table with user details.

Name	EmailId	Phone	Address	UserName	Remove
sangeeth Kumar	sangeeth5535@gmail.com	9486365535	No 16, Samnath Plaza, Madurai Main Road, Melapudhur	san	Remove

At the bottom of the page, a footer states: ©All Rights Reserved. Design by Heart Disease Prediction.

Figure A2.4 Admin Dashboard – User Information

The screenshot shows the 'Admin' section of the 'Heart Disease Prediction' web application, specifically the 'Doctor Information' page. The browser address bar indicates the URL is `http://127.0.0.1:5000/DoctorInfo`. The page layout is consistent with the previous screenshot, featuring a blue header with the application logo and navigation links: Home, DoctorInfo, DrugInfo, and Logout. Below the header is a large blue banner with the word 'Admin' in white. The main content area is titled 'Doctor Information' and contains a table with doctor details.

Name	EmailId	Phone	Address	Specialist	UserName	Remove
sangeeth Kumar	sangeeth5535@gmail.com	9486365535	No 16, Samnath Plaza, Madurai Main Road, Melapudhur	heart	sangeeth	Remove

At the bottom of the page, a footer states: ©All Rights Reserved. Design by Heart Disease Prediction.

Figure A2.5 Admin Dashboard – Doctor Information

The screenshot shows the Admin Dashboard for the Heart Disease Prediction application. The page has a blue header with the application name and navigation links. Below the header is a large blue banner with the word 'Admin'. The main content area displays two tables: 'Appointment Information' and 'Drugs Information'.

Appointment Information

Username	Mobile	EmailId	DoctorName	Date	info	Status
san	No 16, Samnath Plaza, Madurai Main Road, Melapudhur	9486365535	sangeeth	2023-04-29	Cancer	Accept

Drugs Information

UserName	Phone	MailId	DoctorName	Medicine	Info	Date	Download
san	No 16, Samnath Plaza, Madurai Main Road, Melapudhur	9486365535	sangeeth	dola	nill	2023-04-30	Report
san	No 16, Samnath Plaza, Madurai Main Road, Melapudhur	9486365535	sangeeth	dola	nill	2023-04-30	Report
san	No 16, Samnath Plaza, Madurai Main Road, Melapudhur	9486365535	sangeeth	nill	gh	2023-04-30	Report

Figure A2.6 Admin Dashboard – Drugs Information

The screenshot shows the 'New User Register Here..!' page. It features a registration form with several input fields and two buttons at the bottom.

New User Register Here..!

Form fields (from top to bottom):

- First Name: rajiya
- Mobile: 9874563210
- Email: rajiya41@gmail.com
- Address: TRICHY
- Second Name: rajiya
- Password: (masked with dots)

Buttons: Login, Reset

Figure A2.7 New User Registration Page

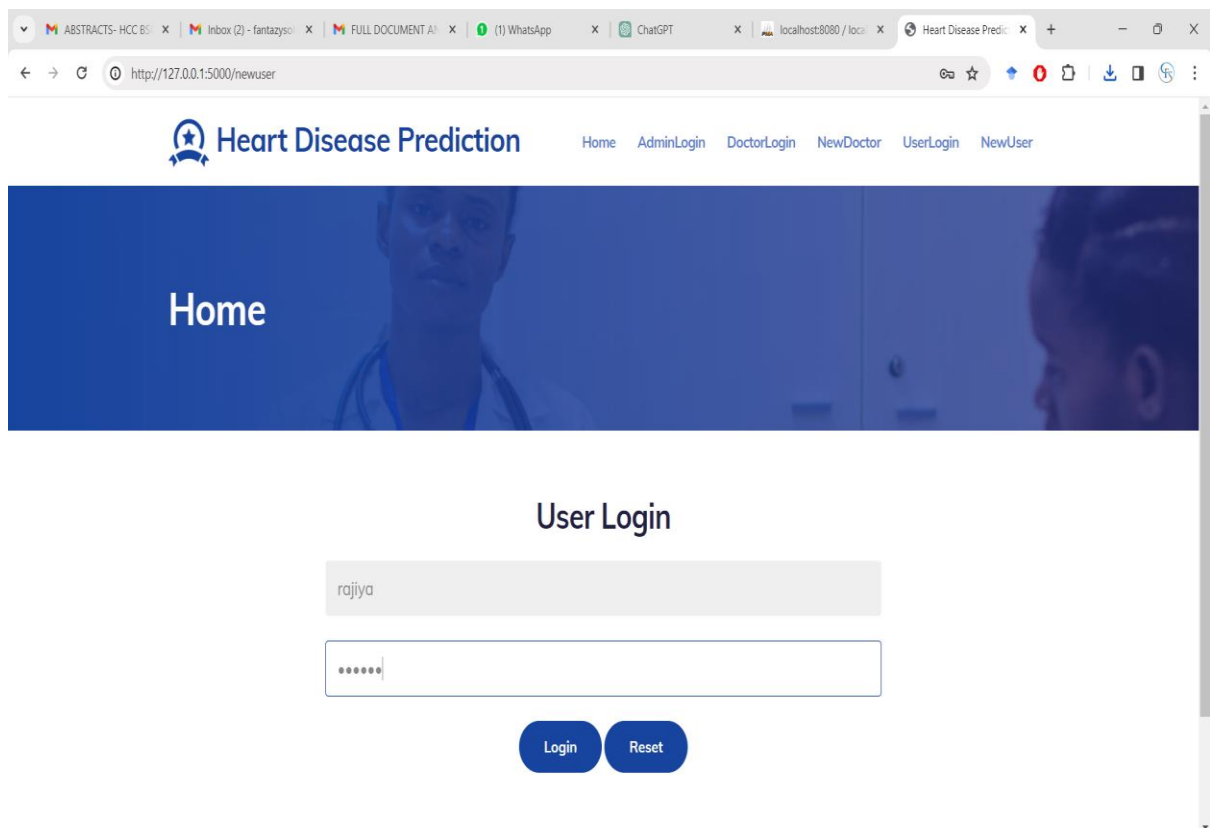
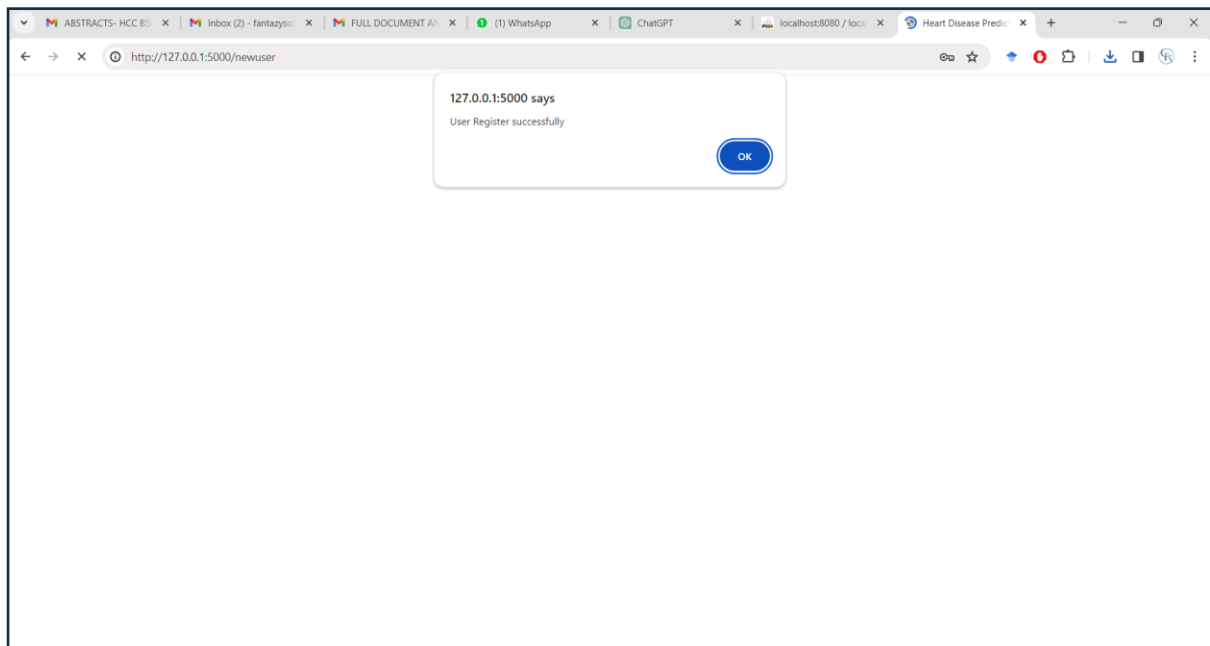


Figure A2.8 User Login Page

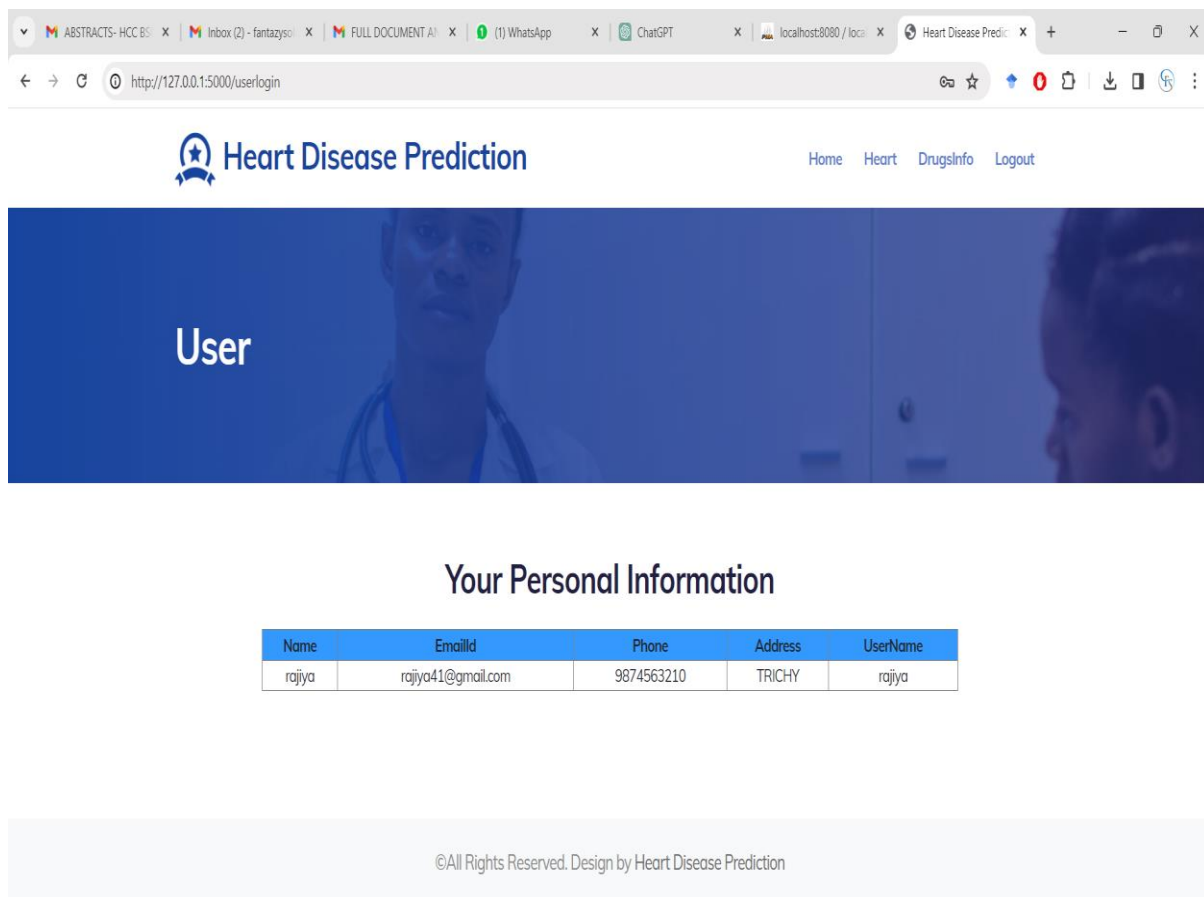
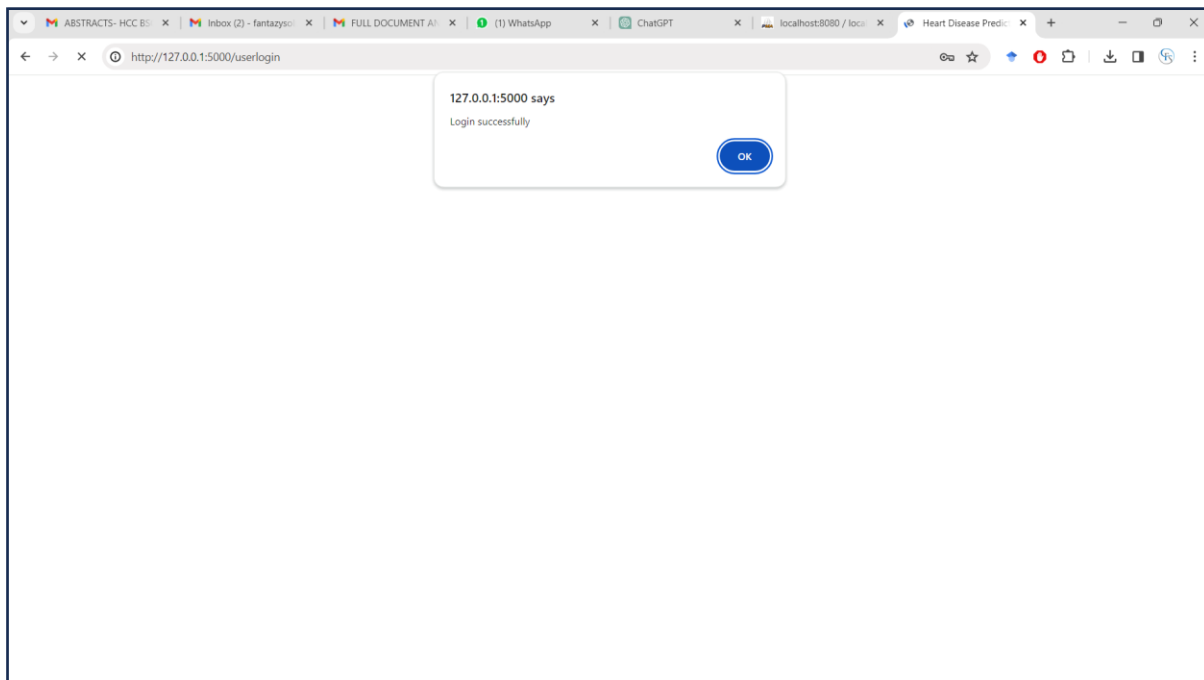


Figure A2.9 User Dashboard – Personal Data

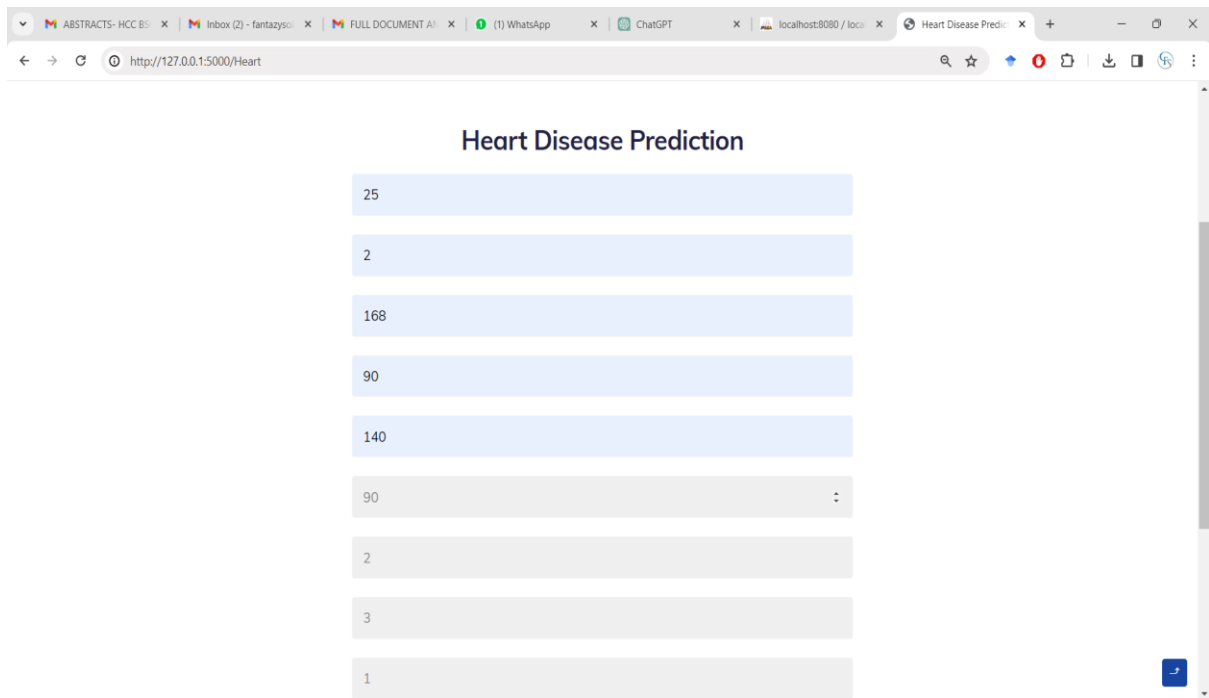


Figure A2.10 Heart Disease Prediction - 1

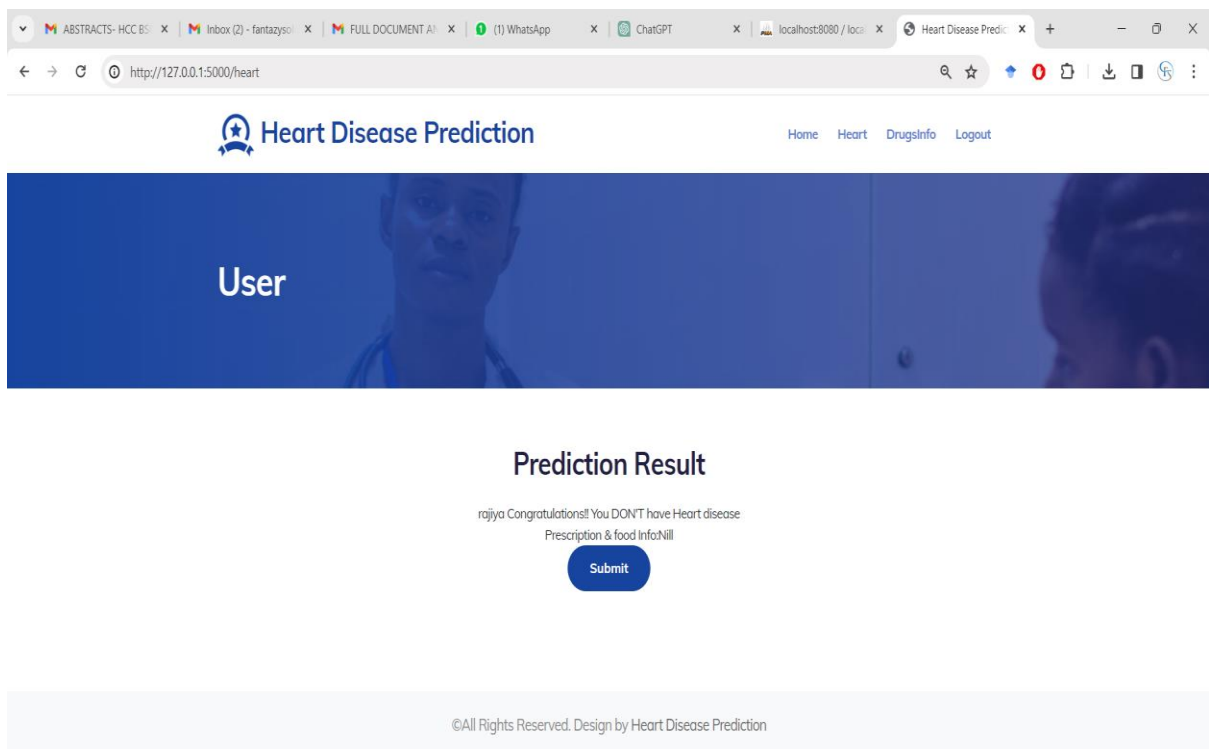


Figure A2.10 Prediction Result - 1

1

147

90

140

90

3

1

0

1

1

Predict Reset

©All Rights Reserved. Design by Heart Disease Prediction

Figure A2.11 Heart Disease Prediction - 2

Heart Disease Prediction

Home Heart DrugsInfo Logout

User

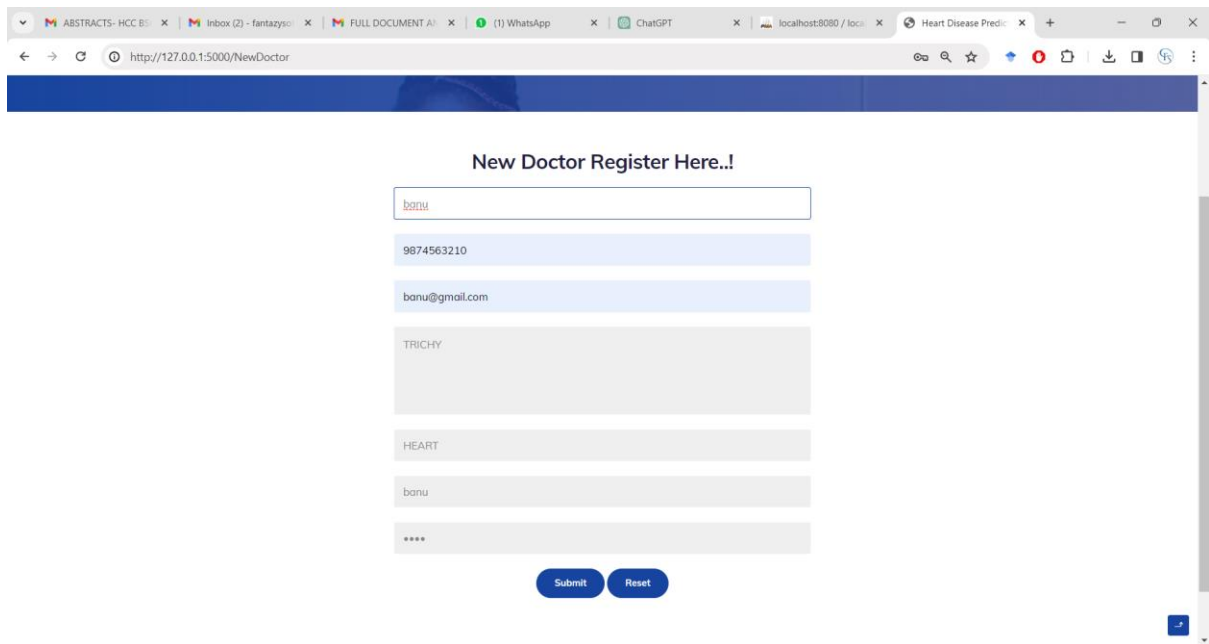
Prediction Result

rajya: According to our Calculations, You have Heart disease
Prescription & food Info: Angiotensin-converting enzyme (ACE) inhibitors, Food: fish or seafood.

Submit

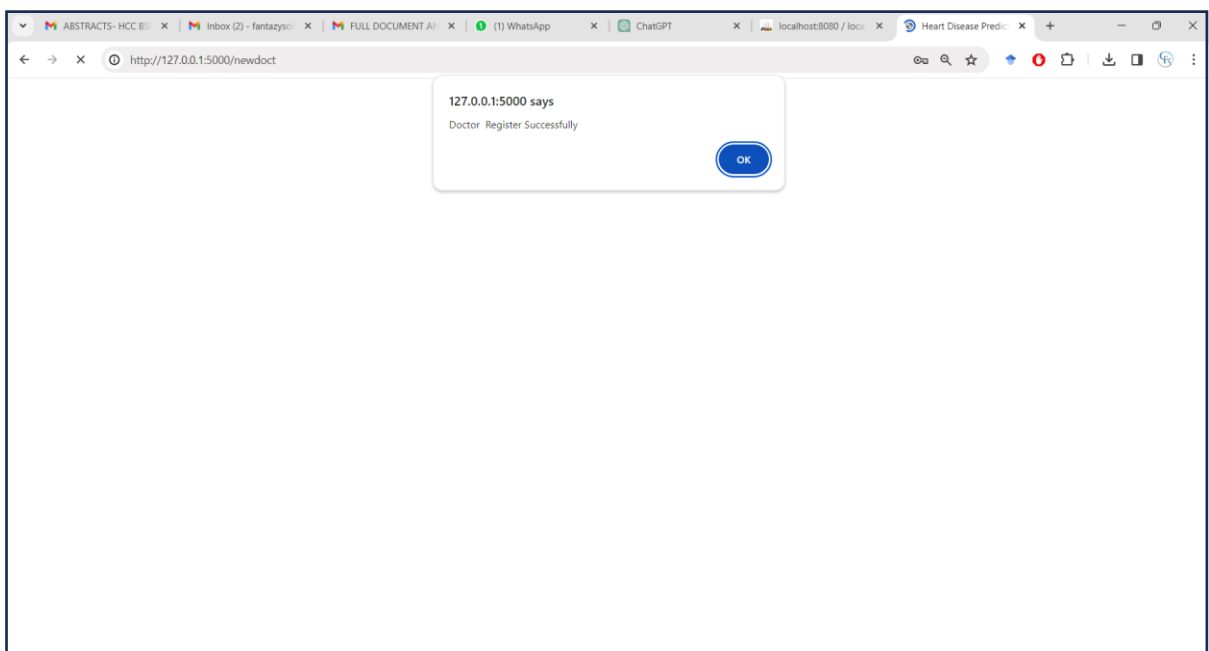
©All Rights Reserved. Design by Heart Disease Prediction

Figure A2.11 Prediction Result - 2



A screenshot of a web browser showing the 'New Doctor Register Here...!' page. The browser's address bar displays 'http://127.0.0.1:5000/NewDoctor'. The page features a registration form with the following fields: a text input for 'banu', a light blue input for '9874563210', a light blue input for 'banu@gmail.com', a light gray input for 'TRICHY', a light gray input for 'HEART', a light gray input for 'banu', and a light gray input for '****'. Below the form are two blue buttons labeled 'Submit' and 'Reset'.

Figure A2.12 New Doctor Registration Page



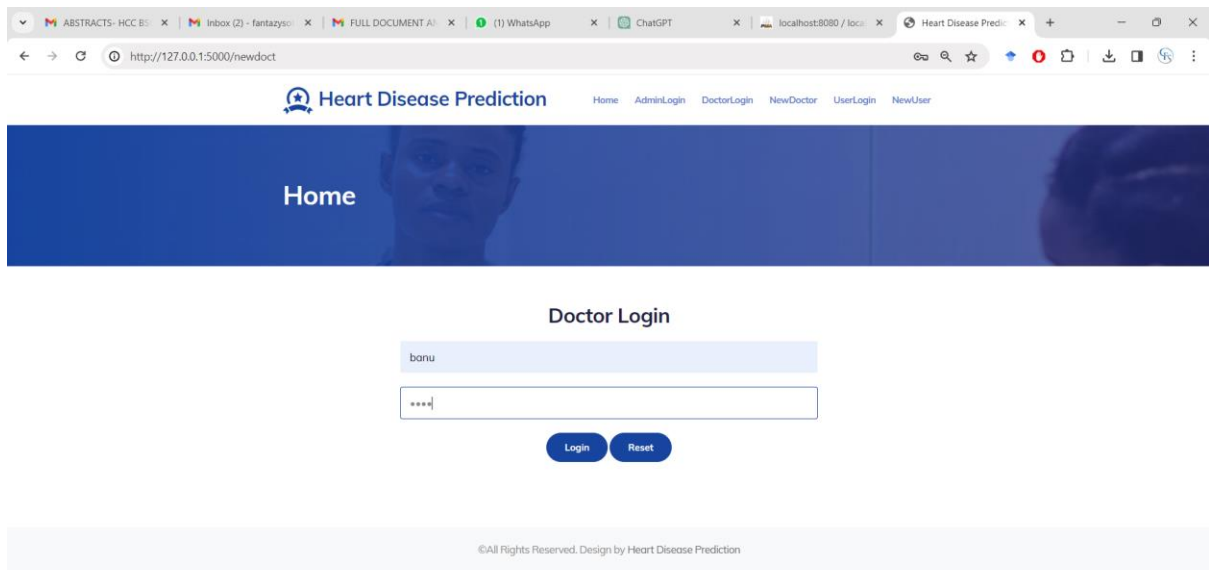
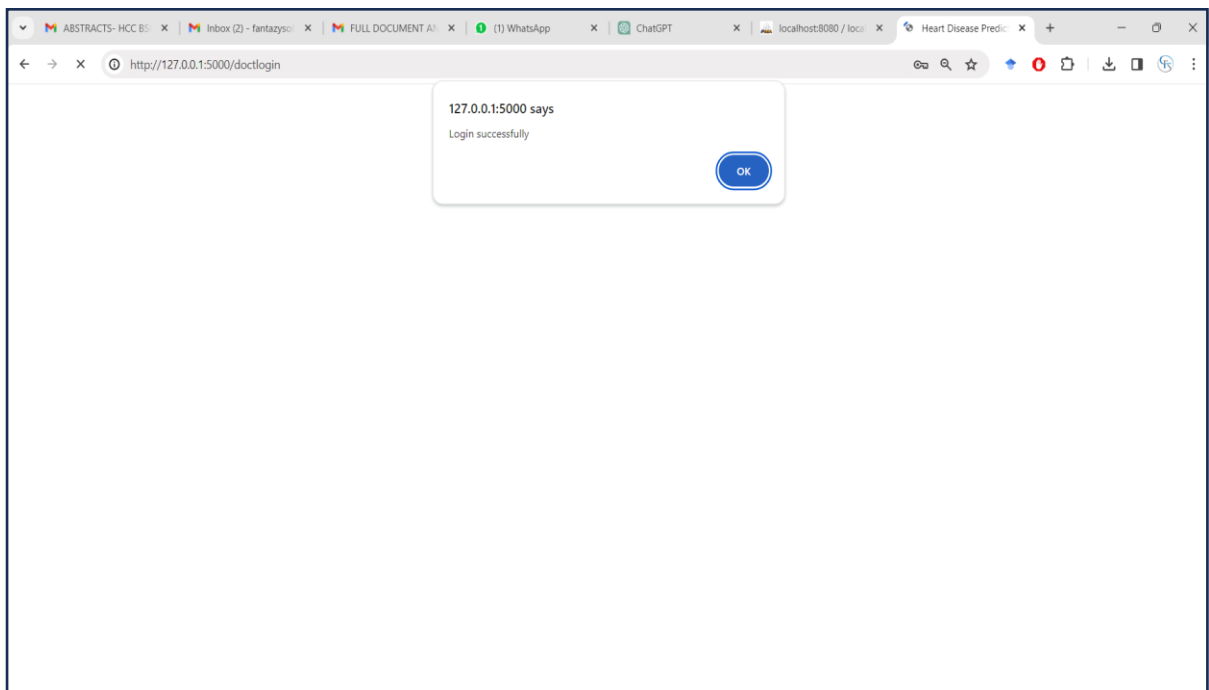


Figure A2.13 Doctor Login Page



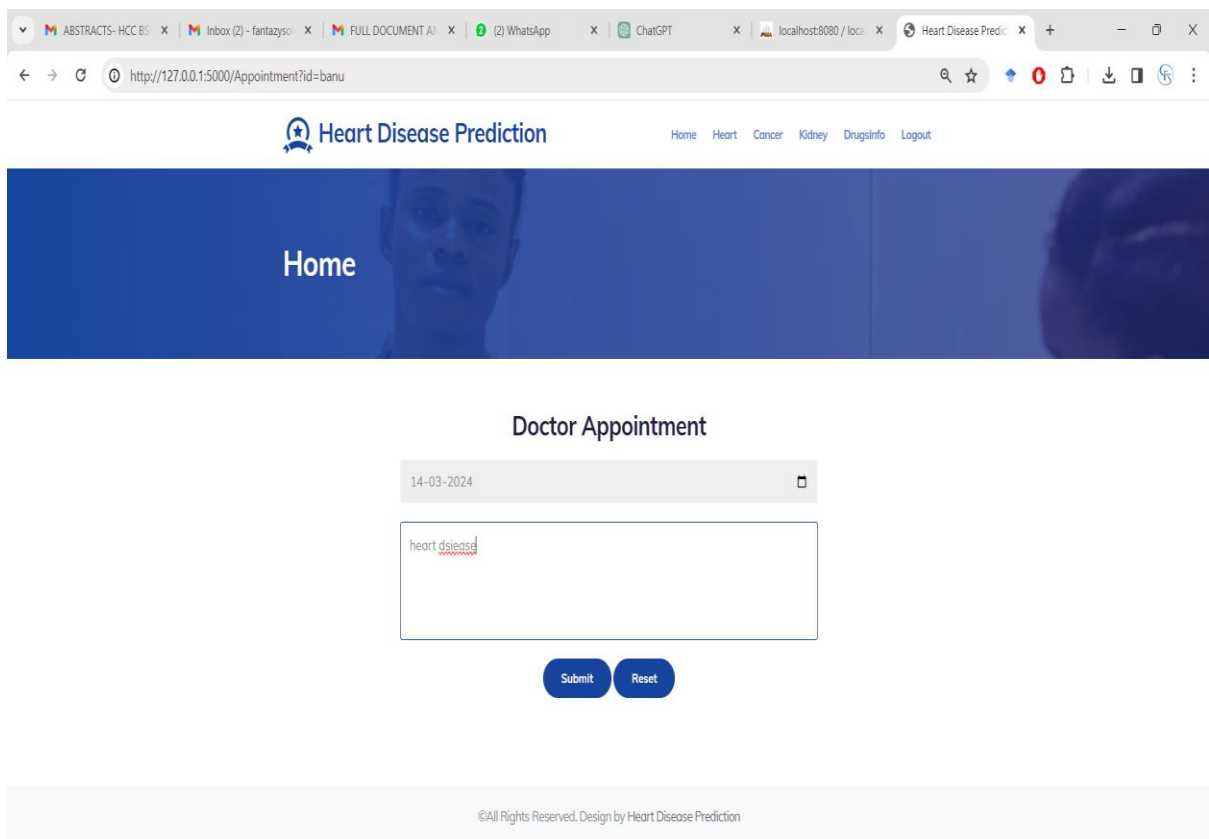
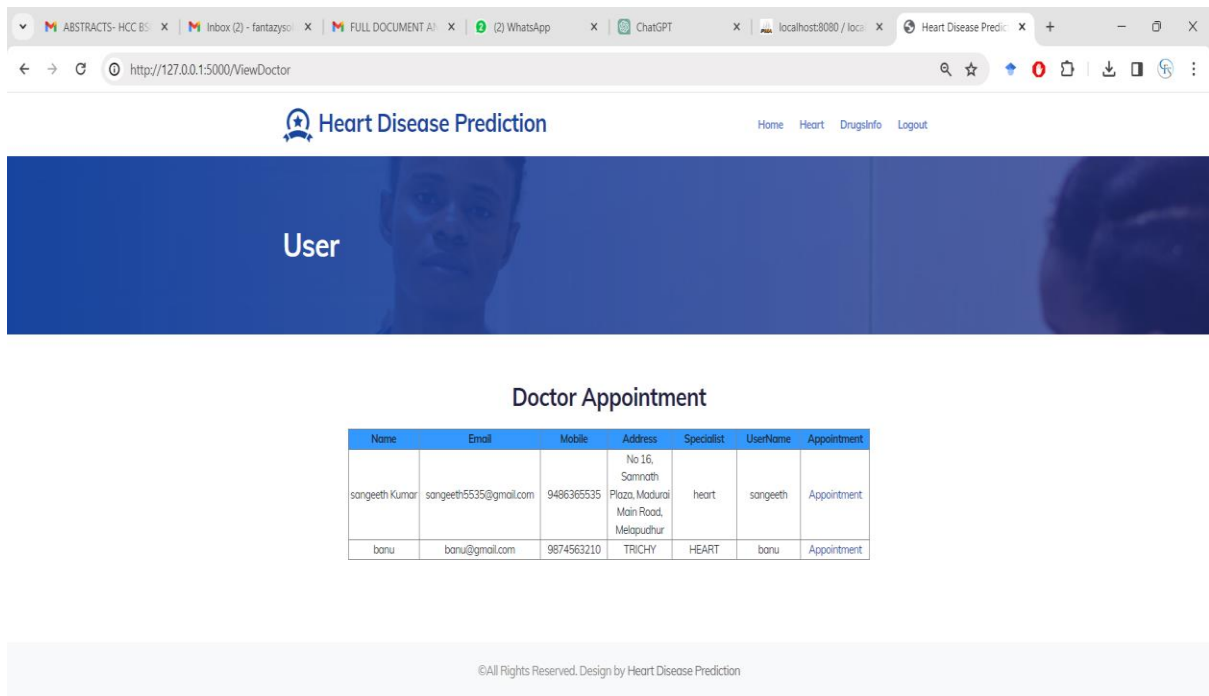


Figure A2.14 Appointment Scheduling Phase

Assign Drugs

14-03-2024

Nitrates. ...
 Angiotensin-converting enzyme (ACE) inhibitors. ...
 Angiotensin-2 receptor blockers (ARBs) ...
 Calcium channel blockers. ...
 Diuretics.

OtherInfo

Choose File download.jpg

Submit Reset

©All Rights Reserved. Design by Heart Disease Prediction

Heart Disease Prediction

Home AppointmentInfo DrugsInfo Logout

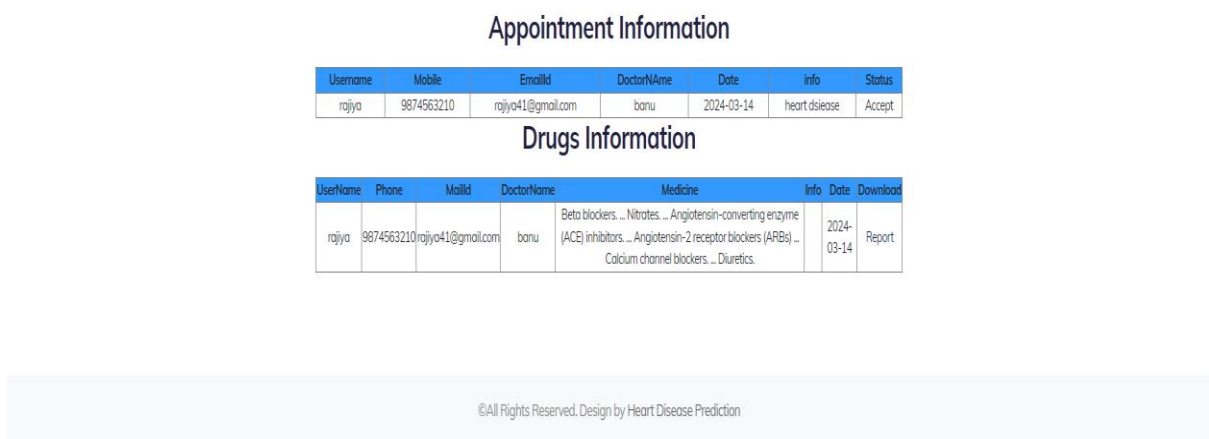
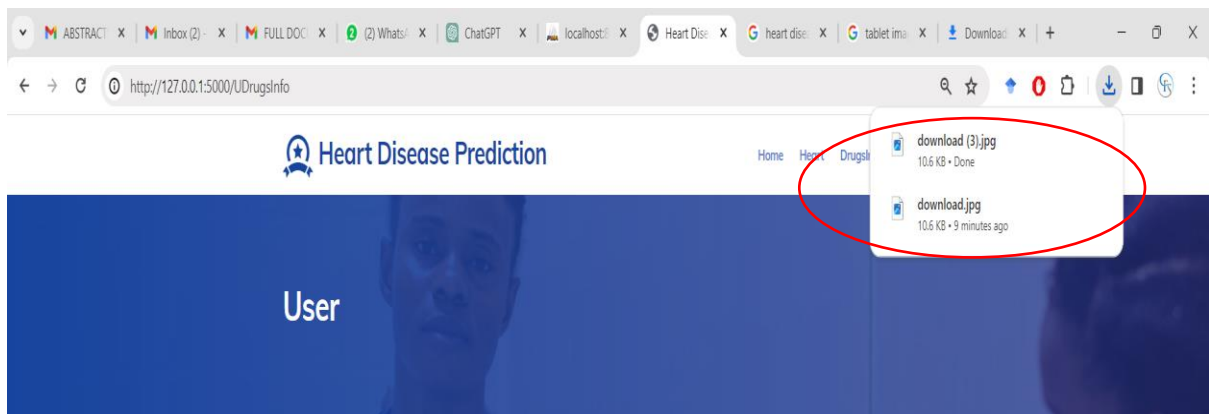
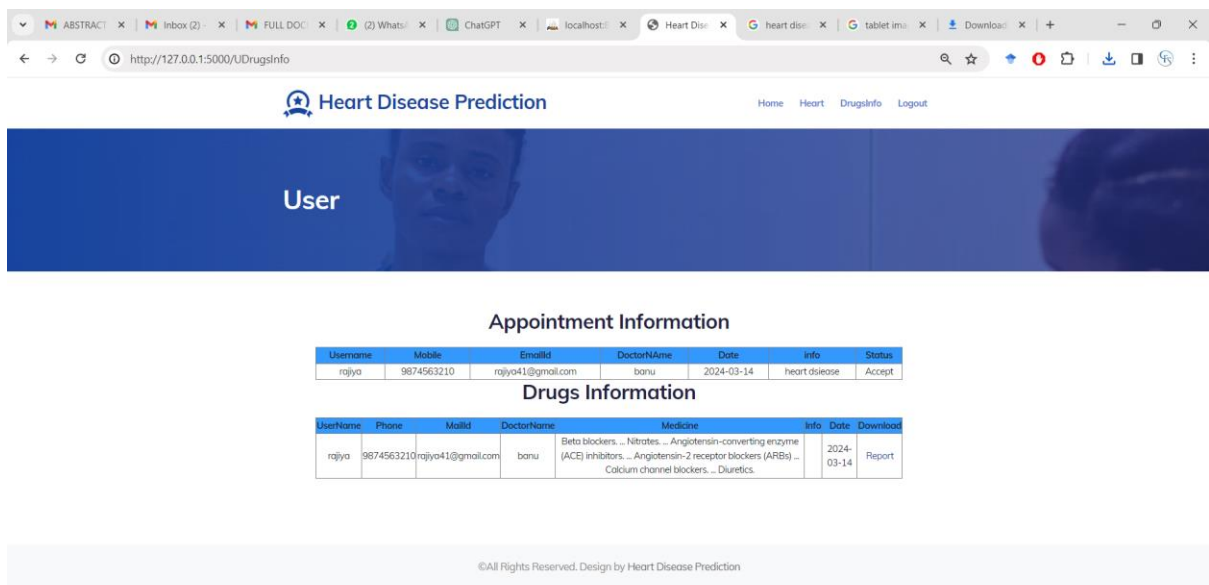
Doctor

Drugs Information

UserName	Phone	DoctorName	Medicine	Info	Date	Download
rajiya	9874563210	banu	Beta blockers ... Nitrates. ... Angiotensin-converting enzyme (ACE) inhibitors ... Angiotensin-2 receptor blockers (ARBs) ... Calcium channel blockers ... Diuretics.		2024-03-14	Report.

©All Rights Reserved. Design by Heart Disease Prediction

Figure A2.16 Drug Assigning Phase



127.0.0.1:5000/download?id=4

Figure A2.17 Report Generation Phase

REFERENCES

- [1] Ashir, Javeed, et al. "Machine learning-based automated diagnostic systems developed for heart failure prediction using different types of data modalities: A systematic review and future directions." *Computational and Mathematical Methods in Medicine* 2022 (2022).
- [2] Aleeza, Nouman and Salman Muneer. "A systematic literature review on heart disease prediction using blockchain and machine learning techniques." *International Journal of Computational and Innovative Sciences* 1.4 (2022): 1-6.
- [3] Abdul, Saboor, et al. "A method for improving prediction of human heart disease using machine learning algorithms." *Mobile Information Systems* 2022 (2022).
- [4] Chiradeep, Gupta, et al. "Cardiac Disease Prediction using Supervised Machine Learning Techniques." *Journal of Physics: Conference Series*. Vol. 2161. No. 1. IOP Publishing, 2022.
- [5] El-Hasnony, Ibrahim M., et al. "Multi-label active learning-based machine learning model for heart disease prediction." *Sensors* 22.3 (2022): 1184.
- [6] Md Mahbubur, Rahman. "A web-based heart disease prediction system using machine learning algorithms." *Network Biology* 12.2 (2022): 64.
- [7] Raniya R., Sarra, et al. "Enhanced heart disease prediction based on machine learning and χ^2 statistical optimal feature selection model." *Designs* 6.5 (2022): 87.
- [8] Ramesh, T. R., et al. "Predictive analysis of heart diseases with machine learning approaches." *Malaysian Journal of Computer Science* (2022): 132-148.
- [9] Suresh, Tamilarasi, et al. "A hybrid approach to medical decision-making: diagnosis of heart disease with machine-learning model." *Int J Elec Comp Eng* 12.2 (2022): 1831-1838.
- [10] Thilagavathi, G., et al. "Heart disease prediction using machine learning algorithms." 2022 *International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*. IEEE, 2022.