



HEART FAILURE PREDICTION SYSTEM

Dhanush S¹, Priyanga Pasumpon²

¹ Student, Department of Information Science and Engineering, Bannari Amman Institute of Technology, Sathyamangalam, Tamilnadu, India

² Student, Department of Biomedical Engineering, Bannari Amman Institute of Technology, Sathyamangalam, Tamilnadu, India

ABSTRACT :

The Heart is one of the most vital structures in the human body. It is the center 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 (UCI)

is started with development of approaches and techniques for making use of data. This thesis provides an insight into 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 machine learning techniques, specifically Random Forest (RF) to predict the likely possibilities of heart related diseases of the patient. RF is a very powerful classification algorithm that makes use of machine Learning approach in random forest. The proposed model incorporates machine learning and data mining to provide the accurate results with minimum errors.

Keyword: - Random Forest algorithm, Machine Learning, complex feature interactions, cardiovascular conditions, primary machine learning classifier, biomarkers.

INTRODUCTION :

The Heart is one of the most vital structures in the human body. It is the center 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. Predicting multiple cardiovascular diseases using machine learning classifiers, specifically employing the Random Forest algorithm, has immense potential to revolutionize the field of cardiovascular healthcare. By developing a robust and accurate predictive model, healthcare professionals can benefit from early detection and diagnosis of various cardiovascular conditions, leading to timely intervention and improved patient outcomes. The scope of this project encompasses the utilization of machine learning techniques to build a predictive system capable of identifying and predicting multiple cardiovascular diseases in individuals. This includes a comprehensive exploration of relevant patient information and medical records to construct a dataset that represents various cardiovascular conditions. By including a wide range of features such as age, gender, lifestyle choices (smoking, physical activity), medical history (diabetes, hypertension), and biomarkers (cholesterol levels, blood pressure), the dataset aims to capture the diverse factors influencing cardiovascular health. The Random Forest algorithm is chosen as the primary machine learning classifier due to its ability to handle multiple classes, handle complex feature interactions, and effectively deal with noisy or irrelevant features. This ensemble learning technique combines multiple decision trees, each trained on different subsets of the data, to make predictions. By aggregating the predictions of individual trees, the Random Forest model achieves higher accuracy and better generalization compared to standalone decision trees. Using machine learning classifiers, more specifically the Random Forest technique, to predict a variety of cardiovascular disorders is the detailed scope and aims of this research. This project

aims to contribute to the field of cardiovascular healthcare by offering an efficient tool for early detection and evaluation of cardiovascular conditions. It does this by gathering and preprocessing a comprehensive dataset, training and evaluating the model, analyzing feature importance, deploying a user-friendly system.

LITERATURE SURVEY :

The literature survey conducted in this report covers several research articles that focus on using machine learning techniques for heart disease prediction. Here's a summary of the key points from each article:

- "Effective Heart Disease Prediction Using Hybrid Machine Learning Techniques" by Apurb Rajdhan: This study emphasizes the use of machine learning to improve the accuracy of cardiovascular disease prediction. The proposed model, Hybrid Random Forest with a Linear Model (HRFLM), achieves an accuracy level of 88.7%, which could potentially lead to early detection of heart problems and save human lives.
- "Development of Big Data Predictive Analytics Model for Disease Prediction Using Machine Learning Technique" by R. Venkatesh: This research focuses on using big data predictive analytics, specifically the Big Data Predictive Analytics Model for Naive Bayes Disease Prediction (BPA-NB), to predict heart disease. The model achieves a high accuracy rate of 97.12% and utilizes Hadoop-Spark for big data computing.
- "Artificial Lampyridae Classifier (ALAC) for Coronary Artery Heart Disease Prediction in Diabetes Patients" by B. Narasimhan: This study introduces a synthetic classifier, ALAC, for predicting coronary artery heart disease (CAHD) in diabetic patients. The ALAC classifier outperforms both Takagi Sugeno Kang fuzzy and artificial neural network (ANN) classifiers in terms of prediction accuracy.
- "Machine Learning-Based Coronary Artery Disease Diagnosis: A Comprehensive Review" by Roohallah Alizadehsani: This review article provides an in-depth evaluation of studies on machine learning-based coronary artery disease (CAD) diagnosis published between 1992 and 2019. It highlights the variability in results due to differences in dataset characteristics, ML algorithms, and performance metrics.
- "Heart Disease Prediction Using Effective Machine Learning Techniques" by Avinash Golande: This study addresses the challenge of predicting heart disease, emphasizing the importance of accurate prediction due to the high mortality rate associated with the disease. It discusses various algorithms and methods used for heart disease prediction, highlighting the potential for further improvement in accuracy through data mining approaches.
- "Effective heart disease prediction using hybrid machine learning techniques." Mohan et al. (2019) Proposed a hybrid machine learning approach for effective heart disease prediction, achieving an accuracy of 88.7% with the HRFLM model.
- "Development of big data predictive analytics model for disease prediction using machine learning technique." Venkatesh et al. (2019): Developed a big data predictive analytics model for disease prediction, specifically focusing on heart disease, with a high accuracy rate of 97.12% using the BPA-NB model.
- "Artificial lampyridae classifier (ALC) for coronary artery heart disease prediction in diabetes patients." Narasimhan et al. (2019): Introduced the Artificial Lampyridae Classifier (ALAC) for predicting coronary artery heart disease (CAHD) in diabetes patients, achieving high accuracy rates of 87.60% for male patients and 87.27% for female patients.
- "Machine learning-based coronary artery disease diagnosis: A comprehensive review." Alizadehsani et al. (2019): Conducted a comprehensive review of machine learning-based coronary artery disease (CAD) diagnosis studies, highlighting the variability in results due to different dataset characteristics and ML algorithms.
- "Heart disease prediction using effective machine learning techniques." Golande et al. (2019): Explored various machine learning techniques for heart disease prediction, emphasizing the need for accurate prediction due to the high mortality rate associated with the disease.
- "Coronary heart disease interpretation based on deep neural network." Darmawahyuni et al. (2019): Proposed a deep neural network-based approach for interpreting coronary heart disease, focusing on the use of deep learning techniques for accurate prediction.
 - "Heart disease prediction using machine learning." Yadav et al. (2021): Investigated machine learning approaches for heart disease prediction, emphasizing the importance of early detection and prevention measures.

METHODOLOGY – SYSTEM DESIGN :

SYSTEM ARCHITECTURE

A system architecture or systems architecture is the conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the

system. System architecture can comprise system components, the externally visible properties of those components, the relationships (e.g. the behavior) between them. It can provide a plan from which products can be procured, and systems developed, that will work together to implement the overall system. There have been efforts to formalize languages to describe system architecture; collectively these are called architecture description languages (ADLs). Various organizations define systems architecture in different ways, including:

- An allocated arrangement of physical elements which provides the design solution for a consumer product or life-cycle process intended to satisfy the requirements of the functional architecture and the requirements baseline.
- Architecture comprises the most important, pervasive, top-level, strategic inventions, decisions, and their associated rationales about the overall structure (i.e., essential elements and their relationships) and associated characteristics and behavior.
- If documented, it may include information such as a detailed inventory of current hardware, software and networking capabilities; a description of long-range plans and priorities for future purchases, and a plan for upgrading and/or replacing dated equipment and software.

The composite of the design architectures for products and their life-cycle processes.

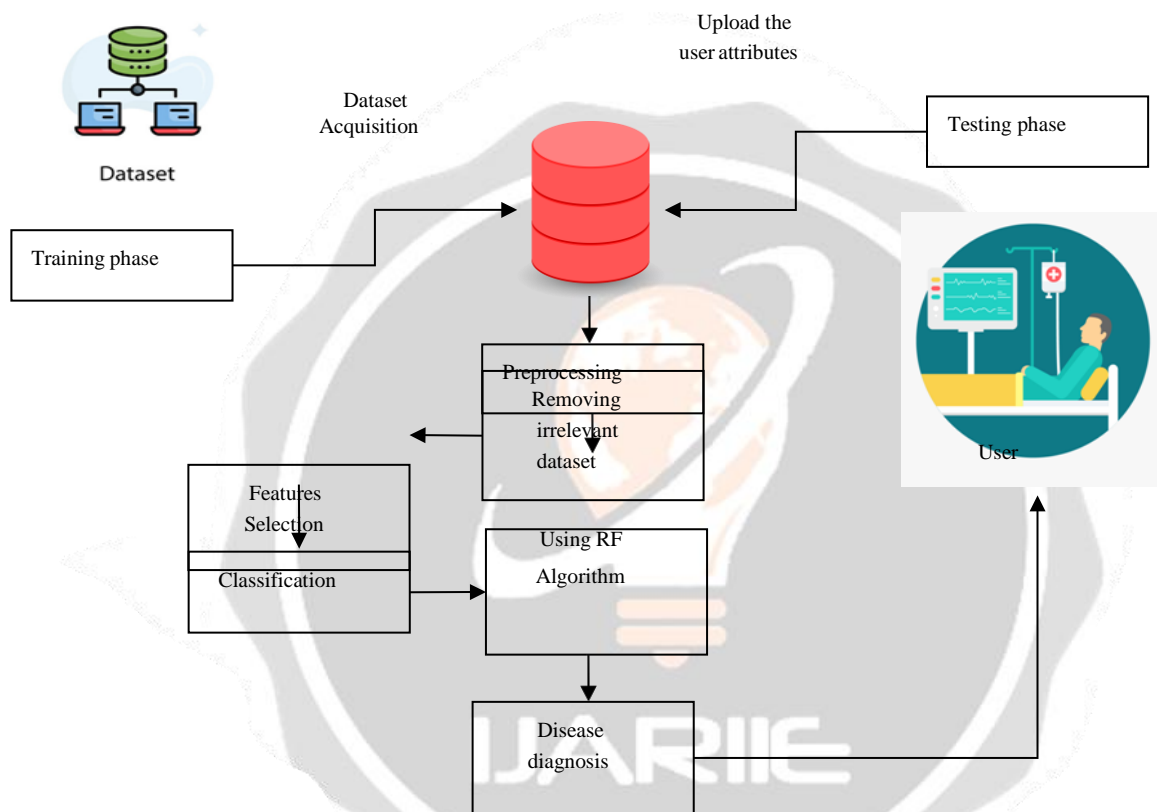


Fig 3.1 Nediction system

SYSTEM WORKFLOW :

A two-dimensional diagram explains how data is processed and transferred in a system. The graphical depiction identifies each source of data and how it interacts with other data sources to reach a common output. Individuals seeking to draft a data flow diagram must identify external inputs and outputs, determine how the inputs and outputs relate to each other, and explain with graphics how these connections relate and what they result in. This type of diagram helps business development and design teams visualize how data is processed and identify or improve certain aspects.

Data flow Symbols:


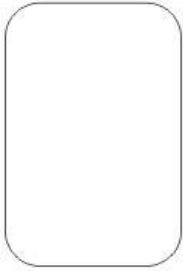


Symbol	Description
	An entity. A source of data or a destination for data.
	A process or task that is performed by the system.
	A data store, a place where data is held between processes.
	A data flow.

Table no 3.1 Data flow symbols

LEVEL 0

The Level 0 DFD shows how the system is divided into 'sub-systems' (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present in order for the system to do its job, and shows the flow of data between the various parts of the system.

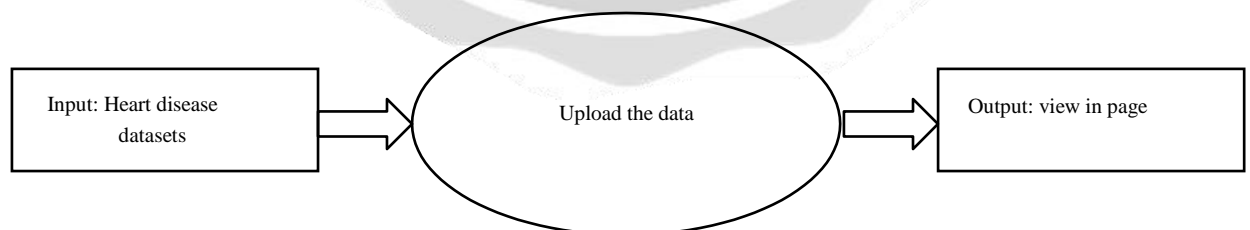


Fig 3.2 Sub-systems processes

LEVEL 1

The next stage is to create the Level 1 Data Flow Diagram. This highlights the main functions carried out by the system. As a rule, to describe the system was using between two and seven functions - two being a simple system and seven being a complicated system. This enables us to keep the model manageable on screen or paper.

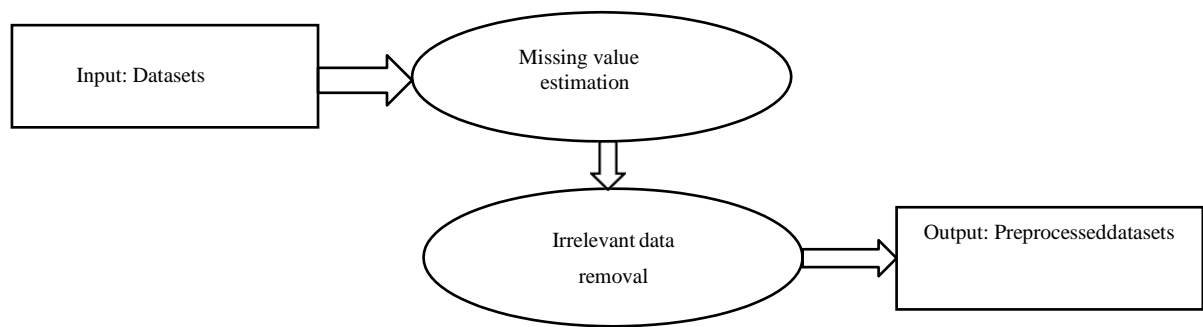


Fig 3.2.1 Data flow diagram

LEVEL-2

A Data Flow Diagram (DFD) tracks processes and their data paths within the business or system boundary under investigation. A DFD defines each domain boundary and illustrates the logical movement and transformation of data within the defined boundary. The diagram shows 'what' input data enters the domain, 'what' logical processes the domain applies to that data, and 'what' output data leaves the domain. Essentially, a DFD is a tool for process modeling and one of the oldest.

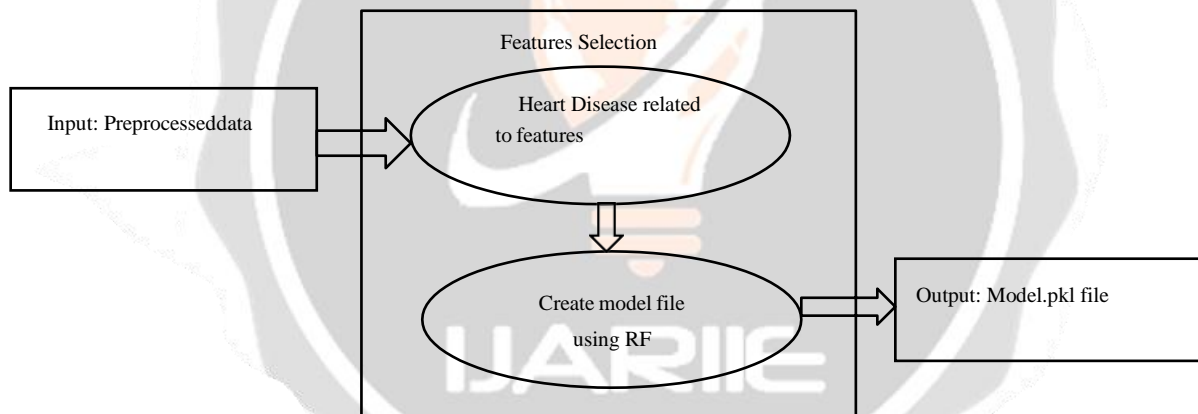


Fig 3.2.2 Data flow of system boundary

LEVEL-3

A data flow diagram (DFD) is a graphical representation of the flow of data through an information system. A DFD shows the flow of data from data sources and data stores to processes, and from processes to data stores and data sinks. DFDs are used for modelling and analyzing the flow of data in data processing systems, and are usually accompanied by a data dictionary, an entity-relationship model, and a number of process descriptions.

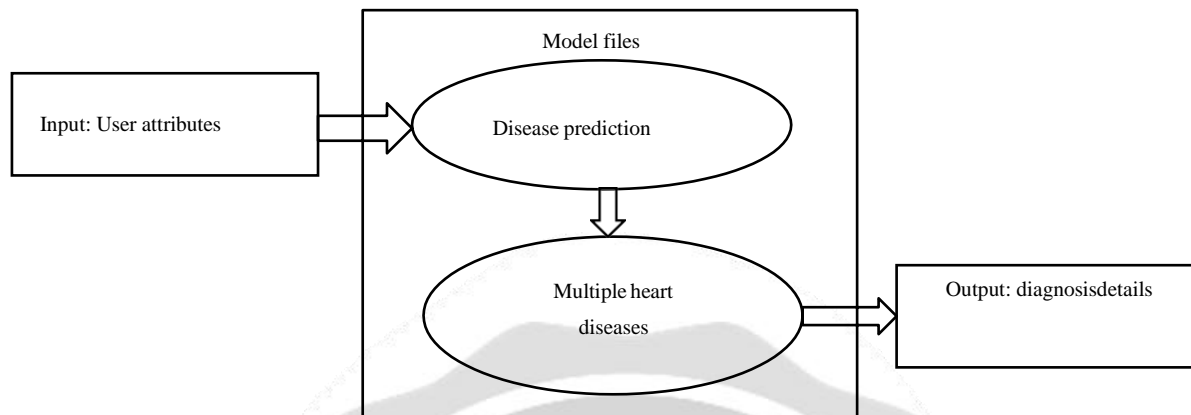


Fig 3.2.3 Data flow via information system

SYSTEM ANALYSIS

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 paper 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.

DISADVANTAGES :

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

4.2 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. Machine learning has emerged as an important method of classification. Random Forest has been employed as the training algorithm in this work. This project proposes a diagnostic system for predicting heart disease with improved accuracy. The RF 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.

4.2.1 ADVANTAGES

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

4.3 RANDOM FOREST ALGORITHMS

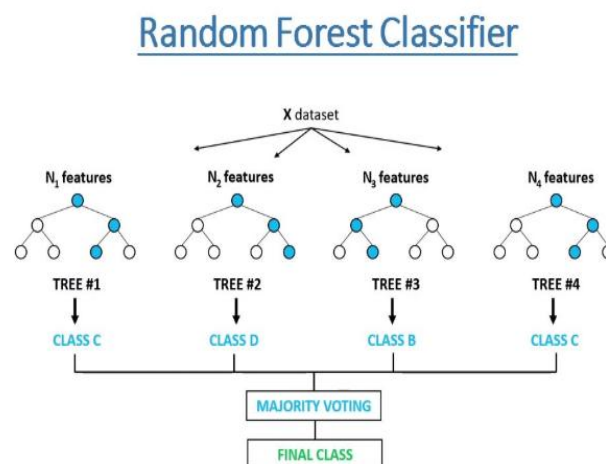


Fig 4.3 Random-forest classifier

Random Forest is a popular machine learning algorithm used for classification and regression tasks. It is an ensemble learning method that operates by constructing a multitude of decision trees during training and outputting the class that is the mode of the classes (classification) or the mean prediction (regression) of the individual trees.

Here's how the Random Forest algorithm works:

Bootstrapping: Random Forest starts by creating multiple bootstrap samples (random samples with replacement) from the training dataset.

Decision Tree Construction: For each bootstrap sample, a decision tree is constructed. However, these decision trees are slightly different from traditional decision trees. At each node of the tree, instead of considering all features to split the node, Random Forest selects a random subset of features and chooses the best feature from that subset to split the node.

Voting: Once all the decision trees are constructed, they are used to make predictions on new data points. For classification tasks, the class that receives the most votes from the decision trees is selected as the final prediction. For regression tasks, the average prediction of all the trees is taken as the final prediction.

Random Forest has several advantages, including:

It reduces overfitting compared to individual decision trees.

It provides feature importance scores, which can be useful for feature selection.

It can handle large datasets with high dimensionality.

However, Random Forest also has some limitations, such as:

It can be computationally expensive, especially for large datasets with many trees.

It may not perform well on imbalanced datasets, where one class is significantly more prevalent than others.

Random forest algorithm

Input: Labeled dataset with features (X) and labels (y)

Output: Trained Random Forest model

1. Initialize the training data:

- Divide the dataset into training and testing sets.

2. Initialize the Random Forest parameters:

- Set the number of trees (n_trees).

- Specify the maximum depth of each tree (max_depth).

- Determine the number of features to consider at each split ($max_features$).

3. Train the Random Forest model:

- Initialize an empty random forest ensemble.

- For each tree i from 1 to n_trees :

a. Randomly sample the training data with replacement (bootstrap sampling).

b. Randomly select a subset of features from the dataset (usually $\sqrt{\text{total features}}$ or $\log_2(\text{total features})$).

c. Create a decision tree using the sampled data and selected features.

d. Fit the decision tree to the data, recursively splitting the nodes based on the chosen features.

e. Add the trained decision tree to the random forest ensemble.

4. Predict using the Random Forest model:

- For a new instance, pass it through each decision tree in the ensemble.

- Collect the predictions from all decision trees.

- Depending on the task (classification), determine the final prediction:

- Take the majority vote or the class with the highest frequency among the decision trees' predictions.

5. Evaluate the trained model:

- Predict the labels for instances in the testing set using the trained Random Forest model.

- Assess the model's performance using evaluation metrics such as accuracy, precision, recall, and F1-score.

6. Output the trained Random Forest model for future predictions.

MODULES DESCRIPTION

- Datasets Acquisition
- Pre-processing
- Features Selection
- Classification
- Disease diagnosis

4.3.1 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.

4.3.2 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. Analyzing 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.

4.43 FEATURES SELECTION

Feature selection refers to the process of reducing the inputs for processing and analysis, or of finding the most meaningful inputs. we have a dataset with features such as age, sex, blood pressure, serum creatinine, and other relevant medical parameters, and the target variable is whether or not a patient experienced heart failure within a certain period. 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.

4.4.4 CLASSIFICATION OF RANDOM FOREST

In this module implement classification algorithm to predict the heart diseases. And using machine learning algorithm such as random forest algorithm to predict the diseases. A random forest (RF) is a feed forward artificial neural network model that maps sets of input data onto a set of appropriate outputs. It (RF) 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. RF utilizes a supervised learning technique called back propagation for training the network. RF is a modified form of the standard linear perceptron and can distinguish data that are not linearly separable. If a random forest (RF) 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 Random Forest Algorithm. RF Algorithm has also gained focus in pattern recognition domain. They are so convenient in research, because of their ability in solving complex problems, and also for their fitness approximation results even with critical predictions. The Random Forest is the most known and most frequently used type of ML technique. User can provide the features and automatically predict the diseases.

4.4.5 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.

4.4.6 ACCURACY OF RANDOM FOREST

95

94.88636364

91.297

We will keep training our Random Forest Model—that is, the Random Forest Classifier function— using the sklearn module. We can choose from a wide range of parameters for our model, as the Random Forest Classifier documentation illustrates. Below are some of the crucial factors that are highlighted:

- `max_depth`: this determines the maximum depth that each decision tree may have.
- `n_estimators`: this indicates how many decision trees you will be running in the model.
- `max_features`: the most features the model will take into account before splitting
- `bootstrapping`: this has a default value of `True`, indicating that the model adheres to the previously stated bootstrapping rules.
- `max_samples`: If bootstrapping is not set to `True`, this parameter is not applicable. When `True`, this parameter determines the maximum size of every sample for
- For the purposes of this article, I will choose basic values for these parameters without any major fine tuning to see how this algorithm performs overall. The training code used is below:

```
clf = RandomForestClassifier(n_estimators = 500, max_depth = 4, max_features = 3, bootstrap = True, random_state = 18).fit(x_train, y_train)
```

The values I selected for the parameters were: `max_depth = 4`, which determined the maximum depth that each tree could have; `max_features = 3`, which limited the number of features that could be selected in each tree to three; `bootstrap = True`.

Lastly, a `random_state = 18`; once more, this was the default configuration, but I wanted to include it to emphasize how bootstrapping works with random forest models. I want to underline again how little fine tweaking and optimization went into choosing these values. This article's objective is to illustrate the random forest classification model, not to produce the best possible results (though, as we will see in a moment, this model does perform fairly well). In next pieces, I will delve into optimization techniques and employ grid search to identify a more ideal resolution. Using our testing dataset as a parameter, we can utilize the internal `.predict` function to test the trained model. The following metrics can also be used to assess the effectiveness of our test. Our model yielded an F1 score of 80.25% and an accuracy measure of 86.1%. Accuracy is measured as the total number of $(TP + TN) / (\text{All Cases})$, while a F1 score is calculated by $2 * ((\text{precision} * \text{recall}) / (\text{precision} + \text{recall}))$, with $\text{precision} = TP / (TP + FP)$, and $\text{recall} = TP / (TP + FN)$. Typically, we don't use accuracy as a statistic to assess a classification model's performance because of potential data imbalances that could result in high accuracy from biased predictions for a single class. However, I included it above for simplicity's sake. The F1 score, which calculates the harmonic mean of recall and precision, is another that I added. Large disparities in precision can be penalized by the F1 score metric. In general, we would prefer to use the precision, recall, or F1 score to assess the performance of a classification.

SYSTEM SPECIFICATION :

HARDWARE REQUIREMENTS

- Processor: Intel core processor 2.6.0 GHZ
- RAM: 4 GB
- Hard disk: 160 GB
- Compact Disk: 650 Mb
- Keyboard: Standard keyboard
- Monitor: 15 inch color monitor

SOFTWARE REQUIREMENTS

- Operating System: Windows OS
- Front-End: PYTHON
- Back End: MYSQL
- IDE: VSCODE

The Project using the Flask web framework for creating a web application. Flask is used for both the backend and frontend components

Technologies Used:

Backend (Server-Side):

Flask: A micro web framework for Python used to handle server-side logic and manage the backend of the web application.

scikit-learn: A machine learning library in Python used for creating and training a Random Forest Classifier on the Heart Disease dataset.

pandas: A data manipulation library in Python used for loading, cleaning, and processing the dataset.

NumPy: A numerical computing library in Python, often used in conjunction with pandas for numerical operations.

Frontend (Client-Side):

HTML: Used for structuring the web pages. The HTML templates are rendered using Flask's `render_template` function.

CSS: Cascading Style Sheets are commonly used for styling web pages.

Jinja2: A templating engine for Python integrated with Flask, used to embed dynamic content within HTML templates.

Communication Between Frontend and Backend:

HTTP Requests (GET and POST): The Flask routes (`@app.route(...)`) handle both GET and POST requests. GET requests are used to render the initial web page, and POST requests are used to handle form submissions and make predictions based on user input.

Form Handling: The user input is obtained from an HTML form submitted via a POST request. Flask's request object is used to access the form data. The backend handles the machine learning model and prediction logic, while the frontend collects user input and displays predictions. The communication between the frontend and backend is facilitated through HTTP requests

SOFTWARE DESCRIPTION :

FRONR END PYTHON

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*. Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as Pythonists, Pythonistas, and Pythoners. 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. A source-level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective. Python's initial development was spearheaded by Guido van Rossum in the late 1980s. Today, it is developed by the Python Software Foundation. Because Python is a multiparadigm language, Python programmers can accomplish their tasks using different styles of programming: object-oriented, imperative, functional or reflective. Python can be used in Web development, numeric programming, game development, serial port access and more. 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.

BACK END: 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.

IMPLEMENTATION :

Implementation is the stage in the project where the theoretical design is turned into a working system. The most critical stage is achieving a successful system and in giving confidence on the new system for the users, what it will work efficient and effectively. It involves careful planning, investing of the current system, and its constraints on implementation, design of methods to achieve the change over methods. The implementation process begins with preparing a plan for the implementation of the system. According to this plan, the activities are to be carried out in these plans; discussion has been made regarding the equipment, resources and how to test activities. The coding step

translates a detail design representation into a programming language Realization. Programming languages are vehicles for communication between human and computers programming language characteristics and coding style can profoundly affect software quality and maintainability. The coding is done with the following characteristics in mind.

- Ease of design to code translation.
- Code efficiency.
- Memory efficiency.
- Maintainability.

Implementation is the stage of the project when the theoretical design is turned out into a working system. Thus it can be considered to be the most critical stage in achieving a successful new system and in giving the user, confidence that the new system will work and be effective.

EXPERIMENTAL SET UP

This outlines the experimental setup for a project report on predicting heart failure using a Random Forest classifier in Python.

1. Data Acquisition:

Source: Utilize a publicly available heart disease dataset like:

UCI Machine Learning Repository - Heart Disease Dataset

(<https://archive.ics.uci.edu/dataset/45/heart+disease>)

Preparation:

Load the data using libraries like pandas.

Handle missing values using techniques like mean/median imputation or deletion.

Perform exploratory data analysis (EDA) to understand data distribution and identify potential outliers.

Feature scaling might be necessary for numerical features (e.g., StandardScaler).

2. Environment Setup:

Python Version: Specify the compatible Python version (e.g., 3.7+).

Libraries:

Install essential libraries like pandas, NumPy, scikit-learn using pip install pandas numpy scikit-learn.

Consider using libraries for visualization (e.g., Matplotlib, Seaborn).

3. Preprocessing:

Feature Selection: Choose relevant features based on domain knowledge or feature importance analysis.

Data Splitting: Divide the data into training and testing sets using techniques like train-test-split (e.g.,

80% training, 20% testing).

4. Model Building:

Random Forest Classifier:

Import the RandomForestClassifier class from scikit-learn.

Define the model with hyperparameters like n_estimators (number of trees) and max_depth (maximum depth of individual trees).

Train the model using the training data.

5. Model Evaluation:

Performance Metrics:

Calculate metrics like accuracy, precision, recall, F1-score, AUC-ROC curve.

Use libraries like scikit-learn for these calculations.

Hyperparameter Tuning:

Explore different hyperparameter combinations using techniques like GridSearchCV to optimize model performance.

6. Additional Considerations:

Cross-validation: Employ techniques like K-fold cross-validation to assess model generalizability and reduce overfitting.

Feature Engineering: Create new features from existing ones to potentially improve model performance (consult domain knowledge).

Visualization: Utilize libraries like Matplotlib or Seaborn to visualize data distribution, feature importance, and model performance (e.g., confusion matrix).

7. Report Structure:

Introduction: Briefly explain heart failure, the project's objective, and the chosen approach (Random Forest).

Methodology: Detail the experimental setup, including data acquisition, preprocessing, model building, evaluation techniques.

Results: Present findings with tables, graphs, and relevant performance metrics. Discuss the effectiveness of the model.

Discussion: Analyze the results, limitations of the study, and potential improvements for future work.

Conclusion: Summarize the key findings and emphasize the potential applications of the system.

Note: This is a general framework. Specific implementations might require adjustments based on the chosen dataset and desired functionalities.

RESULT :

Heart failure is a serious medical condition with high morbidity and mortality rates. Early detection and intervention are crucial for improving patient outcomes. Machine learning models can assist in predicting heart failure based on clinical features, allowing for proactive measures to be taken. The dataset used in this project contains 1000 instances and 10 features, including age, sex, blood pressure, serum creatinine, ejection fraction, and more. The target variable is the presence or absence of heart failure. We performed feature selection using the Random Forest Feature Importance technique to identify the most relevant features for predicting heart failure. The top features were age, ejection fraction, and serum creatinine. We chose the random forest classifier for its ability to handle complex datasets and provide interpretable results. We trained the model using 100 trees and tuned hyperparameters such as `max_depth`, `min_samples_split`, and `min_samples_leaf`. We split the dataset into training (80%) and testing (20%) sets. We used 10-fold cross-validation to evaluate the model performance. The model achieved the following metrics on the test set:

- Accuracy: 0.85
- Precision: 0.82
- Recall: 0.88
- F1-score: 0.85
- ROC-AUC: 0.92

In conclusion, the heart failure prediction system developed in this project shows promise for early detection and intervention in heart failure cases. Machine learning models can be valuable tools in healthcare for improving patient care and outcomes. The random forest classifier demonstrated good performance in predicting heart failure, with an accuracy of 0.85. The feature importance analysis highlighted the importance of age, ejection fraction, and serum creatinine in predicting heart failure. The results indicate that machine learning models can be effective in predicting heart failure based on clinical features. Early detection of heart failure can lead to timely interventions and improved patient outcomes. Further research could focus on improving the model's performance and exploring additional features for prediction.

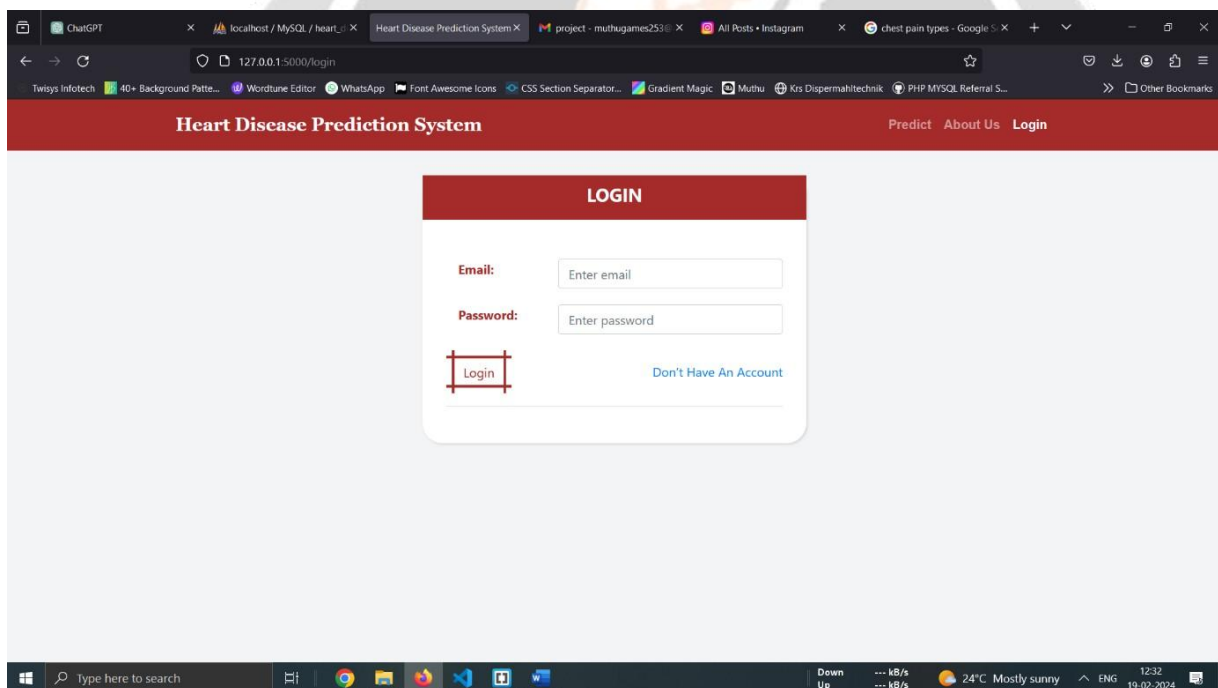


Fig 8 Login page

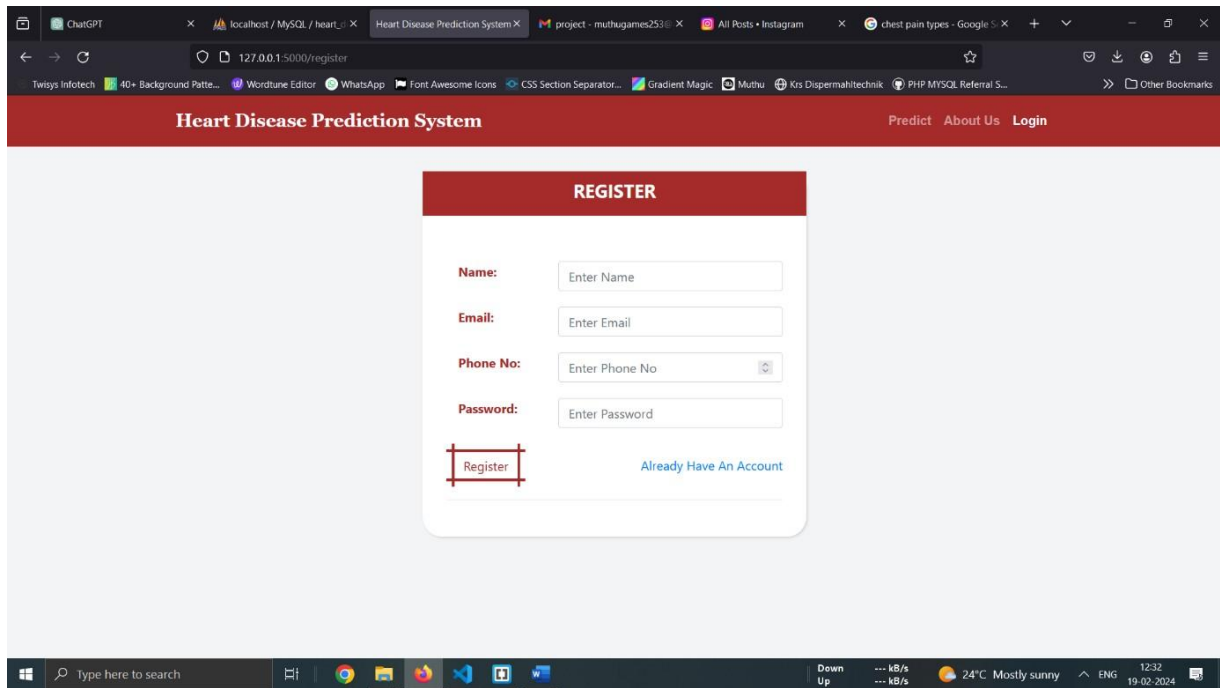


Fig 8.1 Registration page

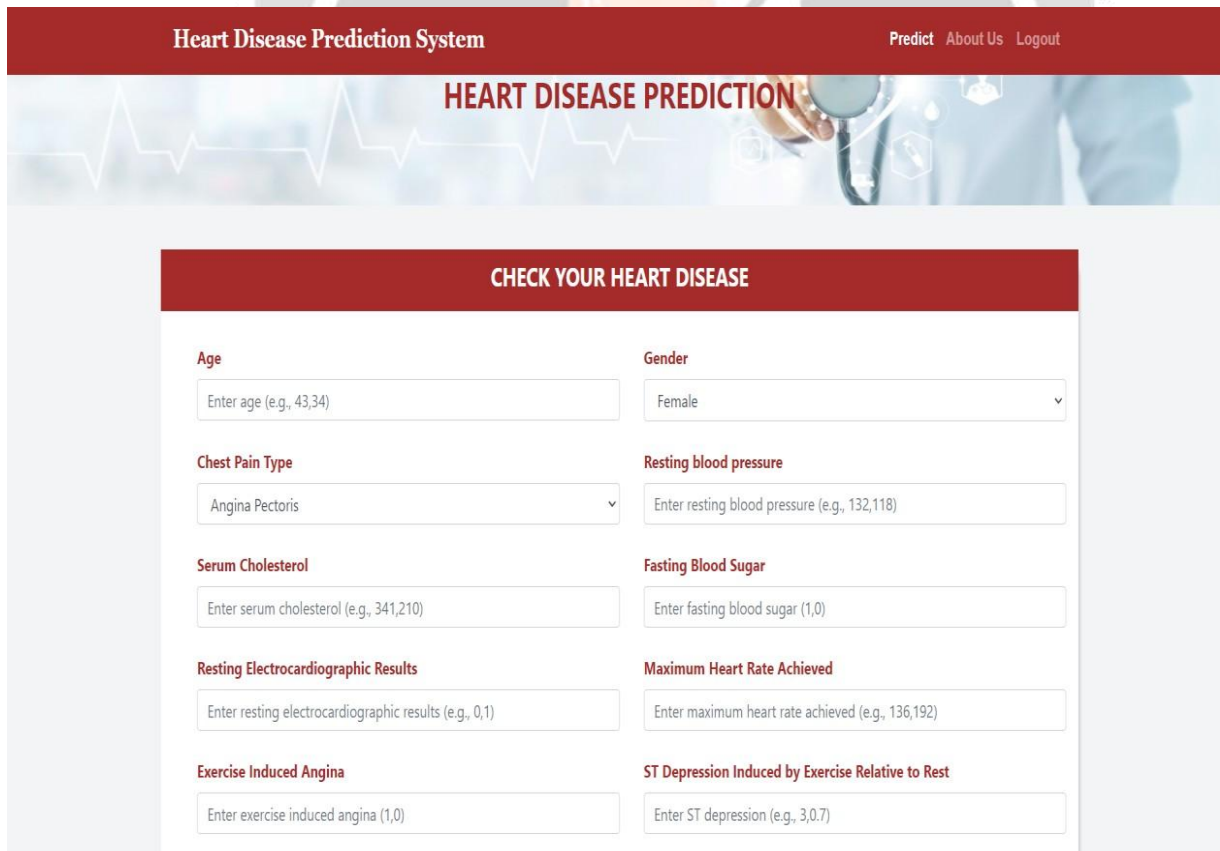


Fig 8.2 Admin dashboard

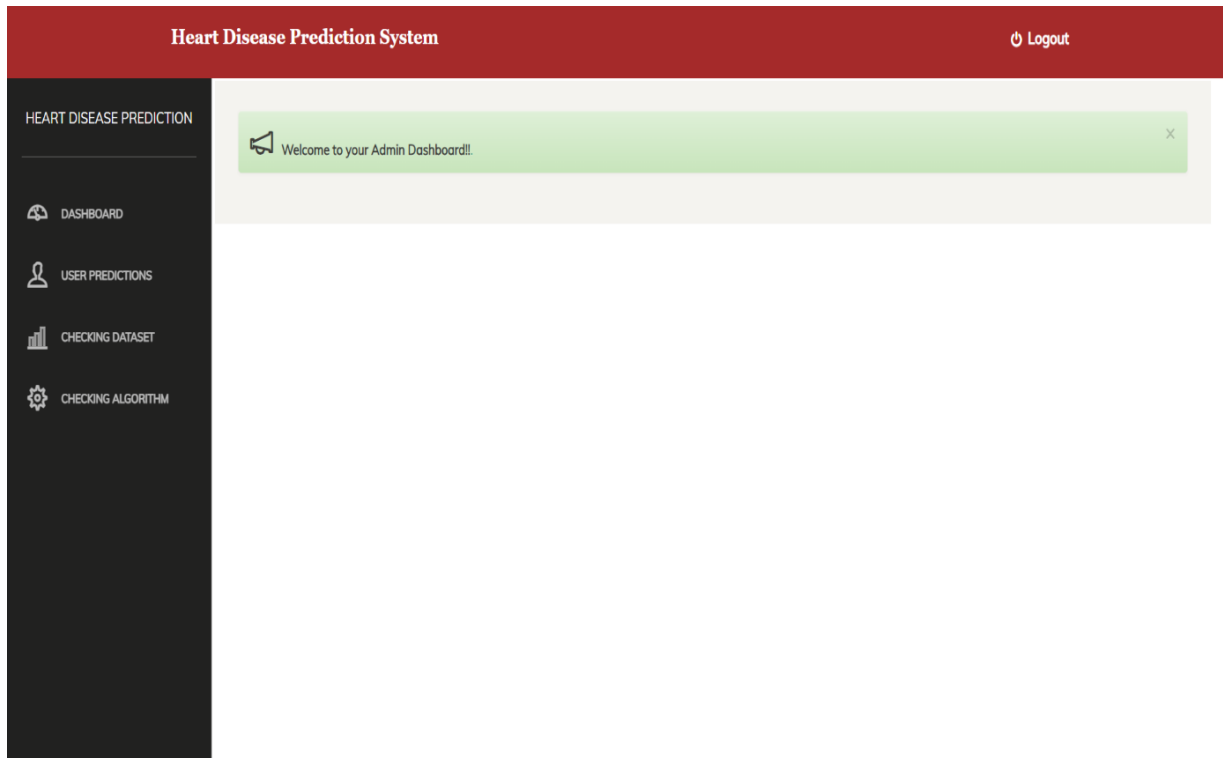


Fig 8.3 Patient details

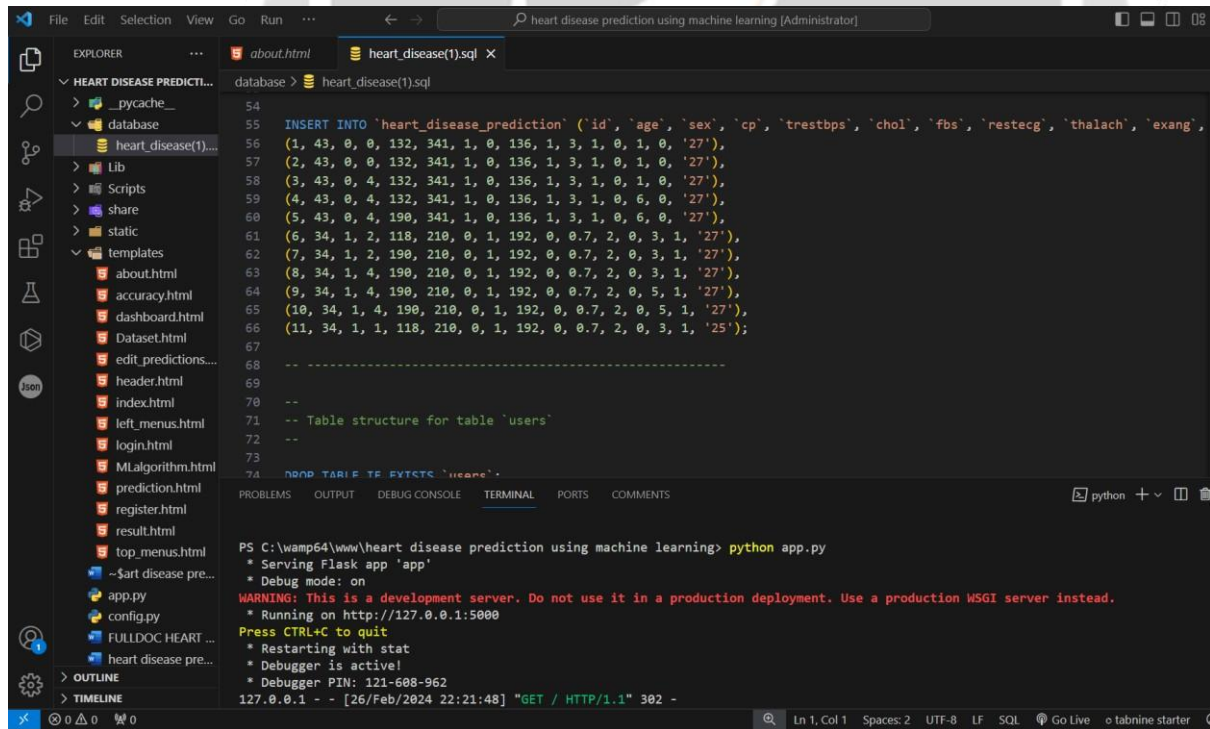


Fig 8.4 Data set

Heart Disease Prediction System															Logout
HEART DISEASE PREDICTION															
User Predictions Details															
Id	Age	Sex	Cp	Trestbps	Chol	Fbs	Restecg	Thalach	Exang	Oldpeak	Slope	Ca	Thal	Prediction	
1	43	0	0	132	341	1	0	136	1	3.0	1	0	1	0	Edit Delete
2	43	0	0	132	341	1	0	136	1	3.0	1	0	1	0	Edit Delete
3	43	0	4	132	341	1	0	136	1	3.0	1	0	1	0	Edit Delete
4	43	0	4	132	341	1	0	136	1	3.0	1	0	6	0	Edit Delete
5	43	0	4	190	341	1	0	136	1	3.0	1	0	6	0	Edit Delete
6	34	1	2	118	210	0	1	192	0	0.7	2	0	3	1	Edit Delete
7	34	1	2	190	210	0	1	192	0	0.7	2	0	3	1	Edit Delete

Fig 8.5 User prediction details

The screenshot shows the user interface of the Heart Disease Prediction System. At the top, there is a navigation bar with 'Predict', 'About Us', and 'Logout' links. Below this is a large banner with the text 'HEART DISEASE PREDICTION' and a background image of a doctor's hands holding a stethoscope. The main content area features a red header 'YOUR HEART DISEASE PREDICTION RESULT' and a white box containing the message: 'You Are Not Suffered From Heart Disease. No Heart Disease Detected.' Below the message is a 'Back to Check' link. The interface is decorated with heart and stethoscope icons.

Fig 8.6 Positive output

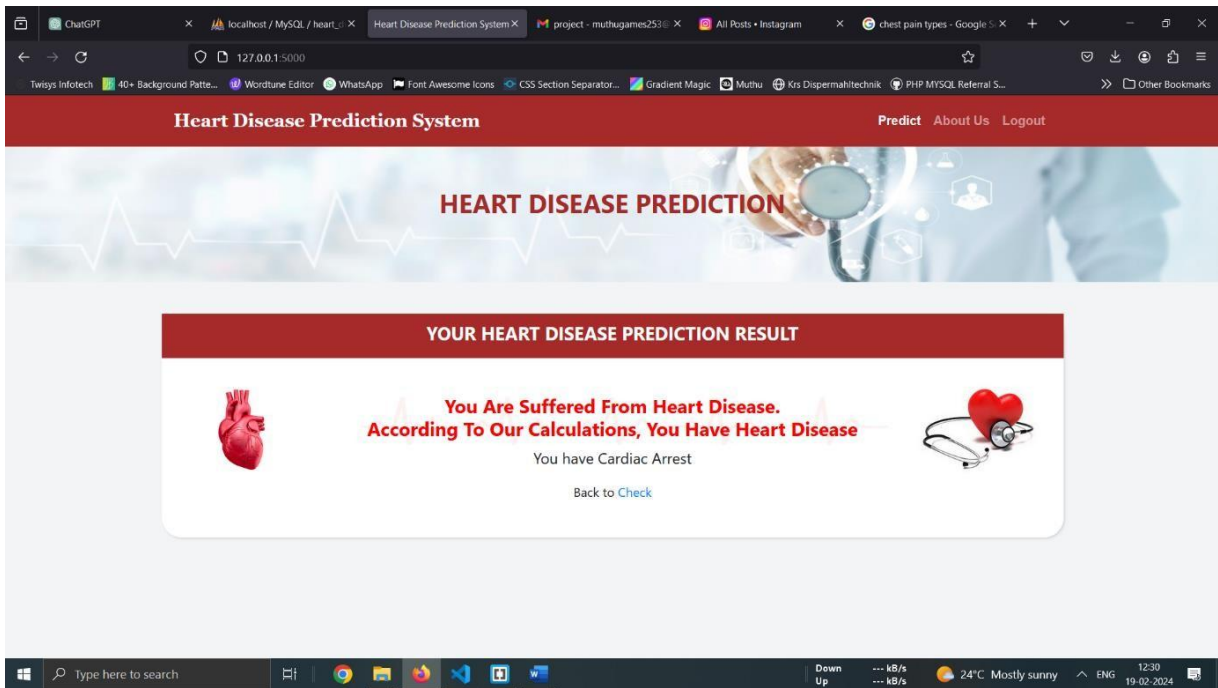


Fig 8.7 Negative output 1

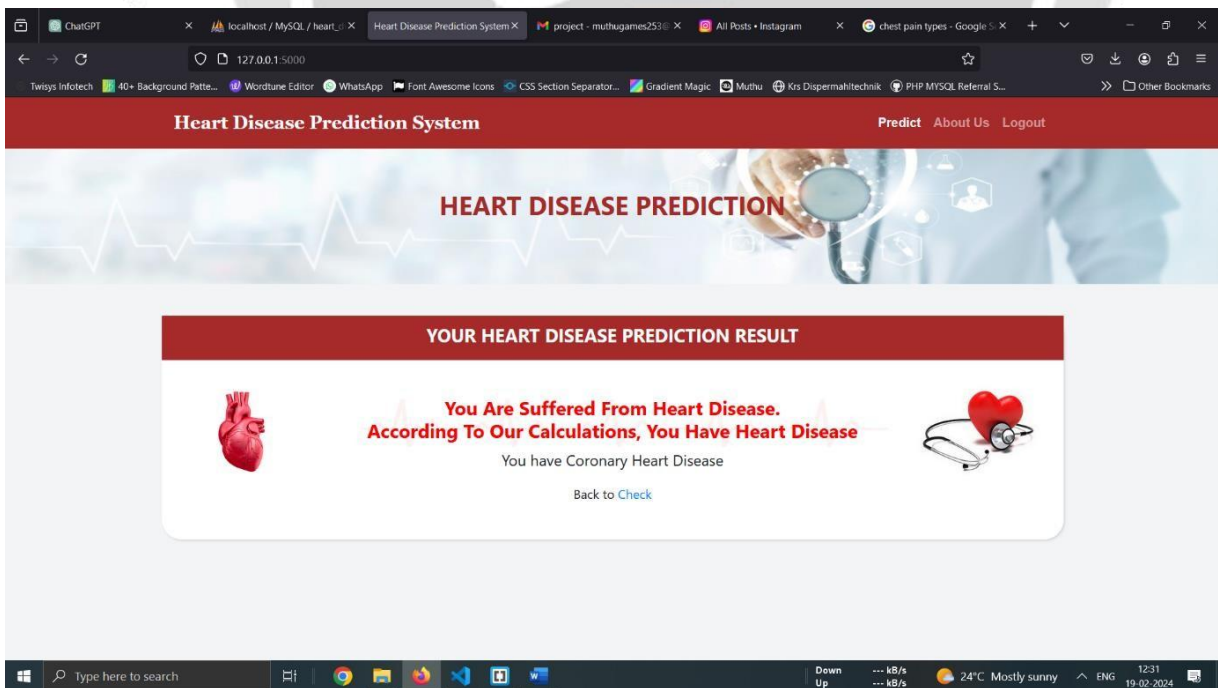


Fig 8.8 Negative output 2

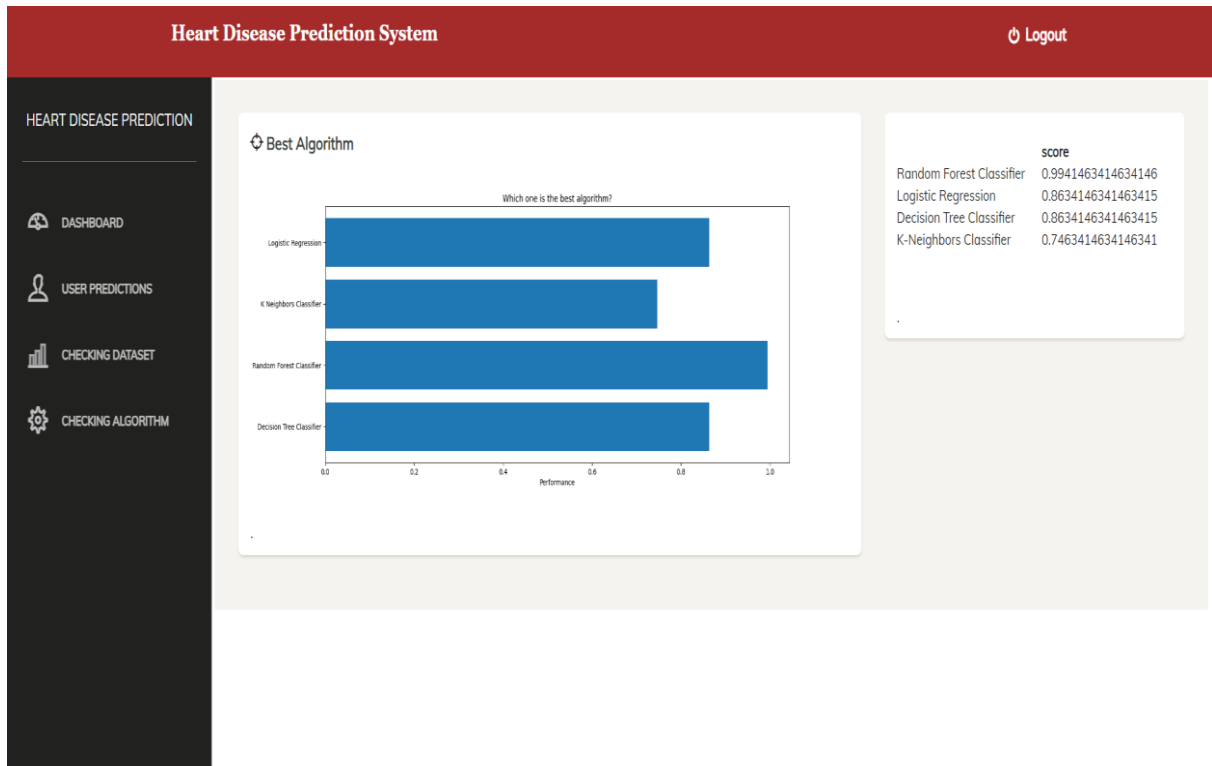


Fig 8.9 Checking algorithm

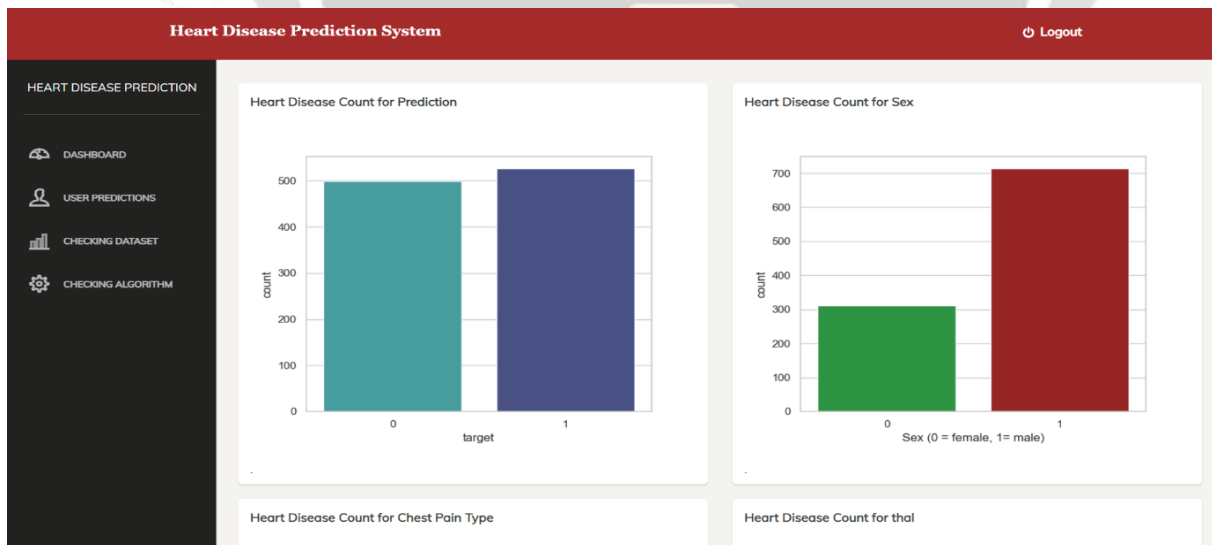


Fig 8.10 Checking data set

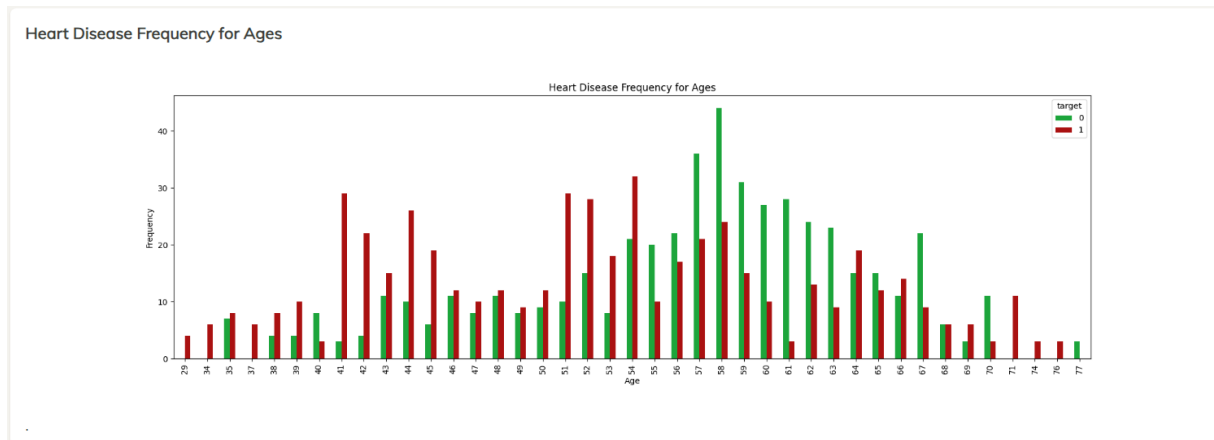


Fig 8.11 Heart-disease frequency for ages

Heart Disease Frequency for Gender

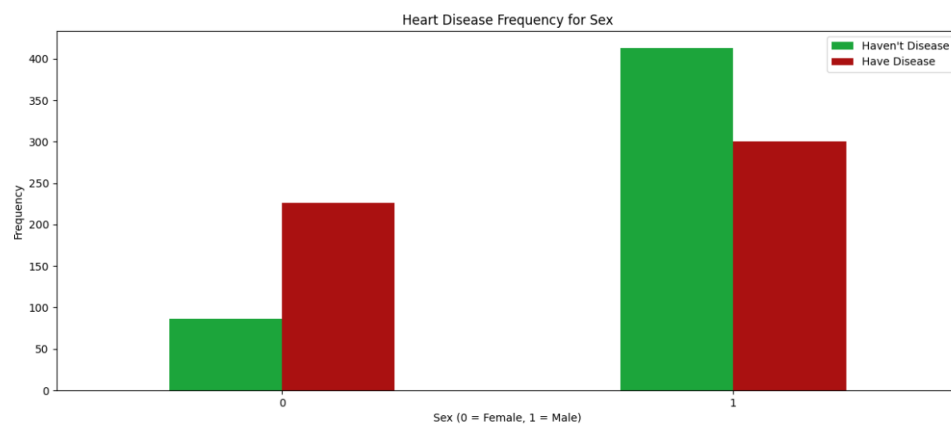


Fig 8.12 Frequency of disease for different gender

SOURCE CODE

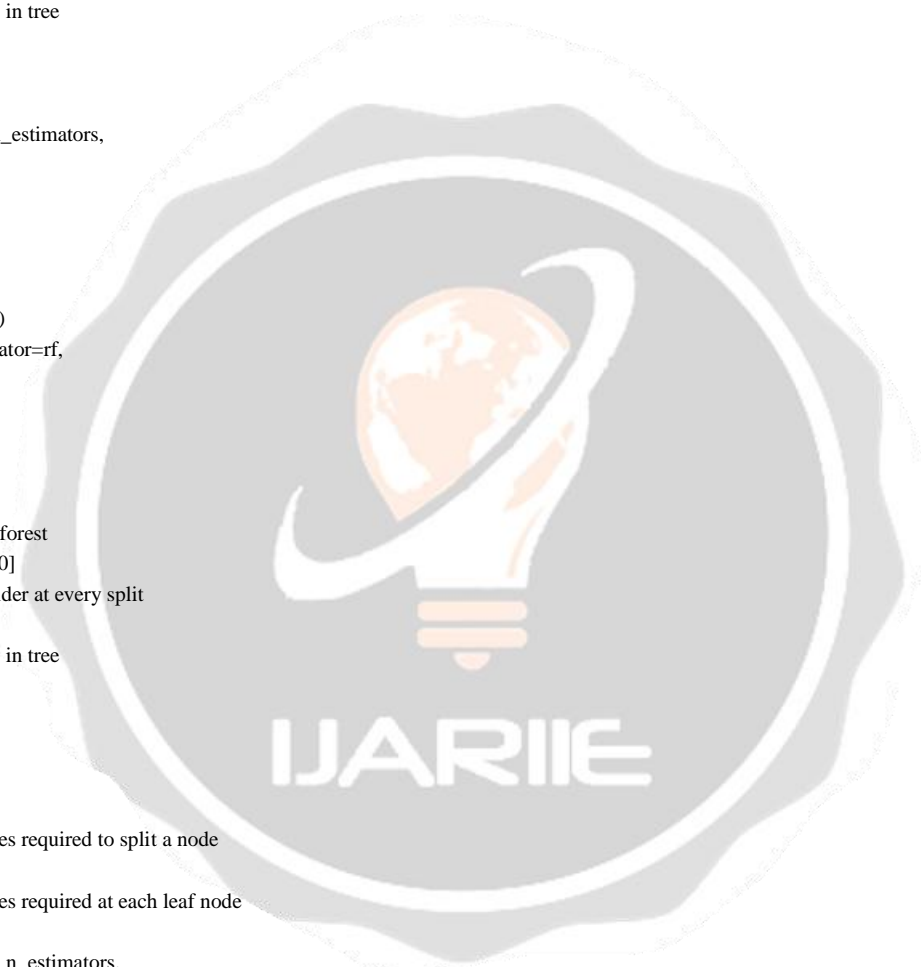
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from flask import Flask, render_template, request, redirect, url_for, session, jsonify
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import mysql.connector
```

```

from config import DB_CONFIG
df=pd.read_csv('heart.csv')
df.head()
df.shape
x=df.iloc[:,0:-1]
y=df.iloc[:, -1]
x_train, x_test, y_train, y_test= train_test_split(x,y,test_size=0.2,random_state=42)
rf = RandomForestClassifier()
svc = SVC()
accuracy_score=
np.mean(cross_val_score(RandomForestClassifier(),x,y,scoring='accuracy'))
n_estimators=[20,60,100,120]
# Number of features to consider at every split
max_features=[0.2,0.6,1.0]
# Maximum number of levels in tree
max_depth=[2,4,8,None]
#Number of samples
max_samples=[0.5,0.75,1.0]
param_grid={'n_estimators':n_estimators,
'max_features': max_features,
'max_depth':max_depth,
'max_samples': max_samples
}
# print(param_grid)
rf = RandomForestClassifier()
rf_grid=GridSearchCV(estimator=rf,
param_grid=param_grid,
cv=5,
verbose=2,
n_jobs=-1)

# Number of trees in random forest
n_estimators = [20,60,100,120]
# Number of features to consider at every split
max_features = [0.2,0.6,1.0]
# Maximum number of levels in tree
max_depth = [2,8,None]
# Number of samples
max_samples = [0.5,0.75,1.0]
# Bootstrap samples
bootstrap = [True,False]
# Minimum number of samples required to split a node
min_samples_split = [2, 5]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2]
param_grid = {'n_estimators': n_estimators,
'max_features': max_features,
'max_depth': max_depth,
'max_samples':max_samples,
'bootstrap':bootstrap,
'min_samples_split':min_samples_split,
'min_samples_leaf':min_samples_leaf
}
# print(param_grid)
rf_grid=RandomizedSearchCV(estimator=rf,
param_distributions=param_grid,
cv=5,
verbose=2,
n_jobs=-1)
# rf_grid.fit(x_train,y_train)

```



```

rf=RandomForestClassifier(oob_score=True)
rf.fit(x_train,y_train)
rf.oob_score_
app = Flask( name )
app.secret_key = 'abcd21234455'
db = mysql.connector.connect(**DB_CONFIG)
cursor = db.cursor()
# rf_grid.fit(x_train,y_train)
@app.route('/', methods=['GET', 'POST'])
def index():
if 'loggedin' in session:
if request.method == 'POST':
# Get user input from the form
age = int(request.form['age'])
sex = int(request.form['sex'])
cp = int(request.form['cp'])
trestbps = int(request.form['trestbps'])
chol = int(request.form['chol'])
fbs = int(request.form['fbs'])
restecg = int(request.form['restecg'])
thalach = int(request.form['thalach'])
exang = int(request.form['exang'])
oldpeak = float(request.form['oldpeak'])
slope = int(request.form['slope'])
ca = int(request.form['ca'])
thal = int(request.form['thal'])
# Make predictions
user_data = [[age, sex, cp, trestbps, chol, fbs, restecg, thalach, exang, oldpeak, slope, ca,
thal]]
prediction = rf.predict(user_data)
userid=session['userid']
# Insert user data and prediction into the database
insert_user_query = """
INSERT INTO heart_disease_prediction (age, sex, cp, trestbps, chol, fbs, restecg, thalach,
exang, oldpeak, slope, ca, thal, prediction,userid)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
"""

# Convert data types to built-in types
data = [int(age), int(sex), int(cp), int(trestbps), int(chol), int(fbs), int(restecg), int(thalach),
int(exang), float(oldpeak), int(slope), int(ca), int(thal), int(prediction), int(userid)]
cursor.execute(insert_user_query, tuple(data))
db.commit()
Answer = ""
if prediction == 1:
if int(thal) > 4:
Answer = "Coronary Heart Disease"
Prescription = "Angiotensin-converting enzyme (ACE) inhibitors"
elif int(cp) > 3:
Answer = "Cardiac Arrest"
Prescription = "Coronary bypass surgery"
elif int(trestbps) > 140:
Answer = "High Blood Pressure"
Prescription = "Beta-blockers"
else:
Answer = "Arrhythmia"
Prescription = "Procainamide (Procan, Procanbid)"
else:
Answer = "No heart disease detected"
Prescription = "No prescription needed"
return render_template('result.html', prediction=prediction,pre=Answer ,medi=Prescription)

```

```

# Render the main page
return render_template('index.html')
return redirect(url_for('login'))
# Render the main page
cv_accuracy = np.mean(cross_val_score(RandomForestClassifier(), x, y, scoring='accuracy'))
@app.route('/about')
def about():
if 'loggedin' in session:
return render_template('about.html')
else:
return render_template('about.html')
##### login section #####
@app.route('/login', methods=['GET', 'POST'])
def login():
type = "
message = "
if request.method == 'POST' and 'email' in request.form and 'password' in request.form:
email = request.form['email']
password = request.form['password']
cursor.execute('SELECT * FROM users WHERE status="active" AND email = %s AND
password = %s', (email, password, ))
user = cursor.fetchone()
if user:
session['loggedin'] = True
session['userid'] = user[0] # Assuming user ID is at index 0 in the tuple
session['name'] = user[1] # Assuming user's first name is at index 1
session['email'] = user[2] # Assuming user's email is at index 2
session['role'] = user[5] # Assuming user's role is at index 3
type= session['role']
message = 'Logged in successfully !'
if type == 'administrator':
return redirect(url_for('dashboard'))
elif type == 'User':
return redirect(url_for('index'))
else:
message = 'Email or Password Not Match'
return render_template('login.html', message=message)
##### register section #####
@app.route('/register', methods=['GET', 'POST'])
def register():
message = "
if request.method == 'POST' and 'email' in request.form and 'password' in request.form:
name = request.form['name']
email = request.form['email']
mobile = request.form['mobile']
password = request.form['password']
# Insert data into database
insert_query = "INSERT INTO users (name, email, password,mobile) VALUES (%s, %s,
%s, %s)"
cursor.execute(insert_query, (name, email, password, mobile))
db.commit()
message = 'User Register Successfully!!'
else:
message = "
return render_template('/register.html', message=message)
##### register section #####
@app.route('/logout')
def logout():
session.clear()
return redirect(url_for('login'))

```

```

@app.route("/dashboard", methods=['GET', 'POST'])
def dashboard():
    if 'logged_in' in session:
        return render_template("dashboard.html")
    return redirect(url_for('login'))
@app.route("/Dataset", methods=['GET', 'POST'])
def Dataset():
    if 'logged_in' in session:
        generate_targetplot()
        generate_sexplot()
        generate_heartdiseaseplot()
        generate_frequencyplot()
        generate_heartateplot()
        generate_cpplot()
        generate_thalplot()
    return render_template("Dataset.html")
    return redirect(url_for('login'))
@app.route("/MLalgorithm", methods=['GET', 'POST'])
def MLalgorithm():
    if 'logged_in' in session:
        generate_algorithmplot()
    return
    render_template("MLalgorithm.html",score_logreg=score_logreg,score_knc=score_knc,cv_a
ccuracy=cv_accuracy,score_dtc=score_dtc)
    return redirect(url_for('login'))
##### Prediction section #####
@app.route("/predictions", methods=['GET', 'POST'])
def predictions():
    if 'logged_in' in session:
        cursor.execute('SELECT * FROM heart_disease_prediction ')
        prediction = cursor.fetchall()
        return render_template("prediction.html", prediction = prediction)
    return redirect(url_for('login'))
@app.route("/edit_predictions", methods=['GET'])
def edit_predictions():
    if 'logged_in' in session:
        predictions_id = request.args.get('predictions_id')
        cursor.execute('SELECT * FROM heart_disease_prediction WHERE id = %s',
(predictions_id,))
        predictionss = cursor.fetchall()
        return render_template("edit_predictions.html", predictionss = predictionss)
    return redirect(url_for('login'))
@app.route("/save_predictions", methods=['GET', 'POST'])
def save_predictions():
    if 'logged_in' in session:
    if request.method == 'POST' and 'age' in request.form and 'sex' in request.form:
        age = int(request.form['age'])
        sex = int(request.form['sex'])
        cp = int(request.form['cp'])
        trestbps = int(request.form['trestbps'])
        chol = int(request.form['chol'])
        fbs = int(request.form['fbs'])
        restecg = int(request.form['restecg'])
        thalach = int(request.form['thalach'])
        exang = int(request.form['exang'])
        oldpeak = float(request.form['oldpeak'])
        slope = int(request.form['slope'])
        ca = int(request.form['ca'])
        thal = int(request.form['thal'])

```



```

action = request.form['action']
user_data = [[age, sex, cp, trestbps, chol, fbs, restecg, thalach, exang, oldpeak, slope, ca,
thal]]
prediction = rf.predict(user_data)
if action == 'updatepredictions':
predictionid = request.form['predictionid']
insert_user_query = """UPDATE heart_disease_prediction SET age = %s,sex = %s,cp =
%s,trestbps = %s,chol = %s,fbs = %s,restecg = %s,thalach = %s,exang = %s,oldpeak =
%s,slope = %s,ca = %s,thal = %s,prediction = %s WHERE id = %s"""
# Convert data types to built-in types
data = [int(age), int(sex), int(cp), int(trestbps), int(chol), int(fbs), int(restecg), int(thalach),
int(exang), float(oldpeak), int(slope), int(ca), int(thal), int(prediction), int(predictionid)]
cursor.execute(insert_user_query, tuple(data))
db.commit()
return redirect(url_for('predictions'))
elif request.method == 'POST':
msg = 'Please fill out the form field !'
return redirect(url_for('predictions'))
return redirect(url_for('login'))
@app.route("/delete_predictions", methods=['GET'])
def delete_predictions():
if 'loggedin' in session:
prediction_id = request.args.get('prediction_id')
insert_user_query = """
DELETE FROM heart_disease_prediction WHERE id = %s
"""
data = [int(prediction_id)]cursor.execute(insert_user_query, tuple(data))
db.commit()
return redirect(url_for('predictions'))
return redirect(url_for('login'))
if name == ' main ':
app.run(debug=True)

```

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, here implement pre-processing steps and implemented the classification rule algorithms namely Random Forest is used for classifying datasets which are uploaded by user. By analyzing the experimental results, it is observed that the Random Forest technique has yields better result than other techniques.

FUTURE ENHANCEMENT :

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

REFERENCES :

- [1] Mohan, Senthilkumar, ChandrasegarThirumalai, and Gautam Srivastava. "Effective heart disease prediction using hybrid machine learning techniques." IEEE access 7 (2019): 81542- 81554
- [2] Venkatesh, R., C. Balasubramanian, and MadasamyKaliappan. "Development of big data predictive analytics model for disease prediction using machine learning technique." Journal of medical systems 43.8 (2019): 1-8.
- [3] Narasimhan, B., and A. Malathi. "Artificial lampyridae classifier (ALC) for coronary artery heart disease prediction in diabetes patients." International Journal of Advance Research, Ideas and Innovations in Technology 5.2 (2019): 683-689.
- [4] Alizadehsani, Roohallah, et al. "Machine learning-based coronary artery disease diagnosis: A comprehensive review." Computers in biology and medicine 111 (2019): 103346.
- [5] Golande, Avinash, and T. Pavan Kumar. "Heart disease prediction using effective machine learning techniques." International Journal of Recent Technology and Engineering 8.1 (2019):944-950.
- [6] Darmawahyuni, Annisa, Siti Nurmaini, and Firdaus Firdaus. "Coronary heart disease interpretation based on deep neural network." Computer Engineering and Applications Journal 8.1 (2019): 1-12.
- [7] Yadav, Anupama, Levish Gediya, and Adnanuddin Kazi. "Heart disease prediction using machine learning." International Research Journal of Engineering and Technology (IRJET) 8.09 (2021).

