Chapter 1 INTRODUCTION

1.1 Background

The increasing prevalence of stress-related illnesses and a growing emphasis on preventative healthcare have fueled a demand for accessible and convenient health monitoring solutions. Traditional methods of assessing physiological health metrics, such as heart rate, often require specialized equipment like chest straps or wristbands. These methods can be cumbersome, uncomfortable, and may not provide a holistic view of an individual's well-being, particularly regarding stress response and heart rate variability (HRV). This has created a need for innovative approaches that offer non-intrusive, real-time monitoring capabilities, empowering individuals to proactively manage their health and well-being.

Advancements in computer vision and signal processing have paved the way for novel techniques in remote photoplethysmography (rPPG), enabling the extraction of physiological signals from video footage. Eulerian Video Magnification (EVM) is a powerful technique within rPPG that amplifies subtle color variations in video frames, revealing microscopic changes in skin perfusion related to blood flow. This technology offers the potential to accurately detect heart rate and derive valuable insights into HRV and stress levels using readily available webcams, eliminating the need for physical contact or wearable sensors. This non-contact approach offers a significant advantage in terms of user comfort, convenience, and accessibility, opening up new possibilities for continuous health monitoring in everyday settings.

This project addresses the limitations of existing health monitoring methods by developing a comprehensive platform leveraging EVM technology for webcam-based heart rate detection, HRV analysis, and stress assessment. By processing video data captured from a standard webcam, the platform aims to provide users with insightful visualizations and a computed score representing their overall health and stress response. This approach not only offers a convenient and non-intrusive monitoring experience but also provides a more holistic understanding of an individual’s physiological state, potentially aiding in early detection of health issues and promoting proactive stress management.

1.2 Existing Systems and Their Limitations

Existing Systems:

- **Wearable Sensors:** Devices like smartwatches, fitness trackers, and chest straps are widely used for heart rate monitoring. These devices typically employ photoplethysmography (PPG) or electrocardiography (ECG) to measure heart rate. Some advanced wearables also offer HRV analysis and estimations of stress levels based on physiological data.
- **Medical-Grade Equipment:** ECG machines and Holter monitors are used in clinical settings for accurate heart rate and HRV assessment. These devices provide detailed diagnostic information but are not suitable for continuous, everyday monitoring due to their complexity and cost.
- **Smartphone Apps:** Several smartphone applications utilize the device's camera and flash to measure heart rate through PPG. While convenient, the accuracy of these apps can be affected by factors like finger pressure, ambient light, and skin tone.

Limitations of Existing Systems:

- **Inconvenience and Intrusion:** Wearable sensors can be uncomfortable to wear for extended periods and may interfere with daily activities. Medical-grade equipment is not designed for continuous, at-home use.
- **Limited Accessibility:** Wearable devices can be expensive, limiting their accessibility to a broader population.
- **Accuracy Concerns:** The accuracy of PPG-based heart rate measurements can be affected by motion artifacts, skin tone, and ambient light. Smartphone apps using PPG may have even lower accuracy due to variations in camera quality and user handling.
- **Lack of Comprehensive Insights:** While some wearables offer HRV analysis, the accuracy and reliability of these measurements can vary. Stress assessment based solely on heart rate data may not provide a complete picture of an individual's stress response.
- **Data Privacy Concerns:** Sharing health data with third-party app developers or cloud services raises concerns about data privacy and security.

1.3 Proposed System and Its Advantages

Proposed System:

The proposed system is a web-based health and stress monitoring platform that leverages Eulerian Video Magnification (EVM) technology to detect users' heart rates accurately using a standard webcam. The platform goes beyond basic heart rate monitoring by analyzing heart rate variability (HRV) and extracting physiological data related to stress from the webcam footage. This data is then processed to generate insightful graphs, charts, and a computed score representing the user's overall health and stress response.

The key components of the proposed system are:

- **EVM-Based Heart Rate Detection:** Employs EVM to magnify subtle color changes in facial regions related to blood flow, enabling real-time heart rate monitoring without external sensors.
- **HRV and Stress Analysis:** Utilizes advanced algorithms to process heart rate data, calculate HRV, and extract stress-related markers from webcam footage.
- **Visual Representation and Data Insights:** Generates graphs and charts to visualize heart rate, HRV, and stress levels over time, allowing users to identify patterns and trends.
- **Computed Score and Health Assessment:** Calculates a score based on heart rate, HRV, and stress data, providing a quantifiable measure of well-being.
- **User-Friendly Web Interface:** Offers an intuitive web interface for easy access and navigation.
- **Data Privacy and Security:** Prioritizes user privacy and data security through robust encryption and adherence to relevant regulations.

Advantages of the Proposed System:

- **Non-Intrusive and Convenient:** Eliminates the need for external sensors or devices, offering a comfortable and hassle-free monitoring experience. Users only need a webcam and internet access.

- **Accessibility:** By utilizing readily available webcams, the platform is more accessible than systems requiring specialized hardware.
- **Potential for Improved Accuracy (Compared to Smartphone PPG):** EVM analyzes subtle changes in facial skin perfusion, potentially offering more accurate heart rate measurements compared to smartphone-based PPG, which can be affected by user movement and lighting conditions.
- **Comprehensive Insights:** Provides more comprehensive insights into health and stress levels by analyzing HRV and extracting stress-related markers from webcam footage, going beyond basic heart rate monitoring.
- **Visualizations and Data Insights:** Offers clear visualizations of health data, allowing users to easily understand trends and patterns in their physiological responses.
- **Quantifiable Health Assessment:** The computed score provides a simple and quantifiable measure of well-being, enabling users to track their progress and set goals.
- **Remote Monitoring Capabilities:** Enables remote health monitoring without the need for in-person visits or specialized equipment.
- **Cost-Effective:** Reduces the need for expensive wearable devices, making health monitoring more affordable.
- **Privacy-Focused:** Emphasizes data privacy and security through robust encryption and adherence to regulations.

Chapter 2

LITERATURE SURVEY

1. Traditional Heart Rate Monitoring

Traditional methods for measuring heart rate primarily involve electrocardiography (ECG) and auscultation. ECG is the gold standard, providing a precise electrical recording of the heart's activity using electrodes placed on the skin. Auscultation, using a stethoscope, involves listening to heart sounds. While accurate, these methods are not suitable for continuous, long-term monitoring in everyday settings. ECG requires specialized equipment and trained personnel, while auscultation is subjective and prone to human error.

2. Photoplethysmography (PPG) and Remote PPG (rPPG)

PPG is an optical technique that measures changes in blood volume in peripheral circulation. It typically uses a light source and a photodetector to measure the light reflected or transmitted through the tissue. PPG is widely used in wearable devices like smartwatches and fitness trackers. However, PPG signals can be affected by motion artifacts, ambient light, and skin tone.

Remote PPG (rPPG) extends the PPG technique by extracting physiological signals from video footage. It analyzes subtle changes in skin color caused by blood flow using a camera as the sensor. rPPG offers a non-contact and convenient approach to heart rate monitoring. However, it faces challenges related to motion, illumination changes, and variations in skin tone. Several algorithms have been developed to improve the robustness of rPPG, including:

- **Independent Component Analysis (ICA):** Used to separate the desired physiological signal from noise and artifacts.
- **Principal Component Analysis (PCA):** Similar to ICA, PCA helps to extract the dominant signal components.
- **Chrominance-based methods:** Exploit the different absorption spectra of hemoglobin at different wavelengths to enhance the signal quality.

3. Eulerian Video Magnification (EVM)

EVM is a video processing technique that amplifies subtle changes in video frames that are otherwise imperceptible to the human eye. It was initially developed for visualizing small motions and color variations in videos. EVM has found applications in various fields, including medical imaging, material science, and structural analysis. In the context of rPPG, EVM can amplify subtle color changes in the face caused by blood flow, enhancing the signal-to-noise ratio and improving the accuracy of heart rate detection.

Several studies have demonstrated the effectiveness of EVM for heart rate monitoring using webcams. These studies have shown promising results in controlled environments, but further research is needed to address challenges related to real-world scenarios with varying lighting conditions, user movement, and skin tones.

4. Heart Rate Variability (HRV) Analysis

HRV refers to the beat-to-beat variations in heart rate. It reflects the activity of the autonomic nervous system (ANS), which regulates various bodily functions, including heart rate, blood pressure, and respiration. HRV analysis has been used to assess stress, fatigue, and overall health status. Various time-domain and frequency-domain measures are used to quantify HRV:

- **Time-domain measures:** Include SDNN (standard deviation of normal-to-normal intervals), RMSSD (root mean square of successive differences), and pNN50 (percentage of successive[1] NN intervals differing by more than 50 ms).
- **Frequency-domain measures:** Analyze the power spectral density of HRV signals in different frequency bands, such as high frequency (HF), low frequency (LF), and very low frequency (VLF).

Research has shown that reduced HRV is associated with increased risk of cardiovascular disease, depression, and other health problems.

5. Stress Assessment Techniques

Stress can be assessed using various methods, including:

- **Physiological measures:** Heart rate, HRV, skin conductance, and cortisol levels.
- **Psychological questionnaires:** Self-report measures like the Perceived Stress Scale (PSS) and the State-Trait Anxiety Inventory (STAI).

While physiological measures provide objective data, they may not capture the full context of an individual's stress experience. Psychological questionnaires provide subjective insights but can be influenced by response bias. Combining physiological and psychological measures can provide a more comprehensive assessment of stress.

Chapter 3 ANALYSIS

3.1 Problem Statement

The increasing prevalence of stress-related health issues and the growing demand for accessible and convenient health monitoring solutions highlight a critical need for innovative approaches to assess physiological well-being. Existing methods for monitoring vital signs, such as heart rate, often rely on contact-based sensors or specialized medical equipment, which can be cumbersome, uncomfortable, and inaccessible for continuous, everyday use. While some consumer-grade wearables offer convenient monitoring, their accuracy can be compromised by factors like motion artifacts and skin tone variations. Furthermore, these devices often provide limited insights into more nuanced physiological indicators like heart rate variability (HRV), which is crucial for understanding stress response and overall autonomic nervous system function. This project addresses these limitations by developing a non-intrusive, webcam-based platform that leverages Eulerian Video Magnification (EVM) technology to accurately detect heart rate, analyze HRV, and assess stress levels, offering a more convenient, accessible, and comprehensive approach to health and well-being monitoring.

3.2 Objectives

- **Develop an EVM-based Heart Rate Detection System:** Implement and optimize an algorithm using EVM to accurately detect heart rate from video captured by a standard webcam, minimizing the impact of motion artifacts and varying lighting conditions.
- **Implement HRV Analysis:** Develop algorithms to process the extracted heart rate data and calculate relevant HRV metrics, providing insights into autonomic nervous system activity and stress response.
- **Develop a Stress Assessment Module:** Design and implement a module that analyzes physiological data extracted from webcam footage to assess stress levels, potentially incorporating additional features for a more holistic stress evaluation.
- **Design and Develop a User-Friendly Web Interface:** Create an intuitive and accessible web interface for users to easily capture video, view their health data

(heart rate, HRV, stress levels), and track their progress over time.

- **Evaluate the Platform's Performance:** Conduct rigorous testing to evaluate the accuracy and reliability of the platform's heart rate detection, HRV analysis, and stress assessment capabilities under various conditions (e.g., different lighting, user movement, skin tones).
- **Ensure Data Privacy and Security:** Implement robust security measures to protect user data and ensure compliance with relevant data privacy regulations.
- **Provide Visualizations and Reporting:** Develop clear and informative visualizations (graphs, charts) and reports to present the collected data in a user-friendly manner, enabling users to understand trends and patterns in their health and stress responses.

3.3 Aims of The Project

The aims of this project are to:

- **Provide a Non-Invasive and Convenient Health Monitoring Solution:** To develop a system that allows users to monitor their heart rate, HRV, and stress levels without the need for physical contact with sensors or wearable devices, utilizing readily available webcams.
- **Enhance Accessibility to Health Monitoring:** To create a cost-effective and easily accessible health monitoring solution that overcomes the limitations of expensive wearable devices, making it available to a broader population.
- **Offer Comprehensive Insights into Physiological Well-being:** To go beyond basic heart rate monitoring by providing analysis of HRV and assessing stress levels, offering a more holistic understanding of an individual's physiological state and stress response.
- **Promote Proactive Health Management:** To empower individuals to take control of their health by providing them with real-time data and insights, enabling them to identify trends, make informed lifestyle choices, and potentially detect early signs of health issues or stress-related problems.
- **Advance the Application of EVM Technology in Healthcare:** To contribute to the field of non-contact physiological measurement by demonstrating the effectiveness and potential of EVM technology for health and stress monitoring in real-world settings.
- **Develop a User-Friendly and Secure Platform:** To create a platform that is easy to use, accessible to users with varying technical skills, and ensures the privacy and security of sensitive health data.

3.4 Methodology

The methodology for this project will follow a structured approach, encompassing the following phases:

1. Data Acquisition:

- **Video Capture:** Participants will be recorded using standard webcams under controlled and semi-controlled environmental conditions. These conditions will vary in lighting (natural, artificial, low light), background complexity (plain, cluttered), and participant movement (still, slight head movements). This variety aims to test the robustness of the system.
- **Dataset Creation:** The recorded videos will be compiled into a dataset, with appropriate metadata (e.g., participant demographics, lighting conditions, movement levels). A smaller, controlled dataset will be used for initial algorithm development and testing, while a larger, more diverse dataset will be used for final evaluation.
- **Ground Truth Measurement (Optional):** To validate the accuracy of the EVM-based heart rate detection, simultaneous measurements using a contact-based heart rate monitor (e.g., pulse oximeter or ECG) could be collected for a subset of participants. This will provide a "ground truth" for comparison.

2. Preprocessing and Signal Extraction:

- **Face Detection and Tracking:** Algorithms for face detection (e.g., Haar cascades, deep learning-based detectors) will be used to automatically locate and track the participant's face in each video frame.
- **Region of Interest (ROI) Selection:** A specific region of interest (e.g., forehead, cheeks) will be selected within the detected face for analysis. This ROI should be relatively stable and contain sufficient perfusion information.
- **EVM Application:** The EVM algorithm will be applied to the selected ROI to amplify subtle color changes related to blood flow. Different EVM parameters (e.g., temporal filtering, spatial filtering) will be explored and optimized for heart rate signal extraction.

- **Signal Processing:** The extracted signal will be processed using techniques such as band-pass filtering, noise reduction, and artifact removal to improve signal quality and isolate the heart rate component.

3. Heart Rate and HRV Estimation:

- **Heart Rate Calculation:** The processed signal will be analyzed to detect peaks corresponding to heartbeats. The time intervals between these peaks will be used to calculate the instantaneous heart rate and average heart rate.
- **HRV Analysis:** Time-domain and frequency-domain HRV metrics will be calculated from the inter-beat intervals (IBIs). Time-domain measures like SDNN and RMSSD will be computed. Frequency-domain analysis using Fast Fourier Transform (FFT) or Wavelet Transform will be performed to analyze the power in different frequency bands (HF, LF).

4. Stress Assessment (If applicable):

- **Feature Extraction:** If stress assessment is included, additional features related to stress will be extracted from the video data (e.g., subtle changes in facial expressions, head movement).
- **Stress Level Classification/Regression:** Machine learning models (e.g., Support Vector Machines, Random Forests, Neural Networks) may be trained to classify or predict stress levels based on the extracted features. Ground truth for stress levels could be obtained through self-report questionnaires (e.g., PSS) or physiological measurements like skin conductance.

5. Web Interface Development:

- **Frontend Development:** A user-friendly web interface will be developed using HTML, CSS, and JavaScript. The interface will allow users to upload videos, view their heart rate, HRV, and stress data, and track their progress over time.
- **Backend Development:** A backend system will be developed using a suitable framework (e.g., Python/Flask, Node.js) to handle video processing, data storage, and communication with the frontend.
- **Data Storage and Security:** A secure database will be used to store user data, and

appropriate security measures (e.g., encryption, access control) will be implemented to protect user privacy.

6. Evaluation and Validation:

- **Performance Metrics:** The performance of the heart rate detection and HRV analysis algorithms will be evaluated using metrics such as Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and correlation with ground truth measurements (if available).
- **Usability Testing:** Usability testing will be conducted with a group of target users to evaluate the ease of use and user experience of the web interface.
- **Stress Test:** The system will be subjected to stress testing to assess its performance under high load conditions.

7. Iteration and Refinement:

- Based on the evaluation results, the algorithms, web interface, and overall system will be iteratively refined to improve performance, usability, and robustness.

3.5 Software Requirements Specification

System requirement specifications gathered by extracting the appropriate information to implement the system. Moreover, the SRS delivers a complete knowledge of the system to understand what this project is going to achieve without any constraints on how to achieve this goal. This SRS not providing the information to outside characters but it hides the plan and gives little implementation details.

3.5.1 Hardware System Configuration

● Processor:	Intel i3+Processor
● RAM:	4GB (minimum)
● Hard Disk:	250GB

3.5.2 Software Requirements

● Operating System:	Windows 7/8/10/11
● Front End:	HTML, CSS
● Back End:	Flask.

● Scripts:	JavaScript.
● Technology:	Python, Machine Learning.
● IDE:	VS Code
● Browser:	Chrome, Firefox, Edge

3.6 Functional Requirements

Functional requirements define the core features and operations the system must perform to meet user needs.

User Management:

- **Sign Up:** The system shall allow new users to create an account by providing a unique username and password. The system shall securely store user credentials.
- **Login:** The system shall authenticate registered users based on their username and password, granting access to the dashboard.
- **Logout:** Users shall be able to log out of the system, terminating their session and securely removing session information.
- **Password Reset:** Users shall be able to reset their password via a secure email-based process.

Video Processing:

- **Video Upload:** The system shall allow users to upload video files captured by their webcams.
- **Automated Face Detection and Tracking:** The system shall automatically detect and track the user's face within the uploaded video.
- **EVM Processing**: The system shall apply the EVM algorithm to extract physiological signals from the detected facial region.
- **Signal Processing:** The system shall filter and process the extracted signal to remove noise and artifacts.

Physiological Data Analysis:

- **Heart Rate Calculation:** The system shall calculate the user's heart rate from the processed signal.

- **HRV Analysis**: The system shall calculate relevant HRV metrics (e.g., SDNN, RMSSD, LF/HF ratio).
- **Stress Assessment (Optional)**: If implemented, the system shall assess the user's stress level based on extracted physiological features.

Data Visualization and Reporting:

- **Data Visualization:** The system shall display heart rate, HRV, and stress level data in clear and informative graphs and charts (e.g., time series plots, bar charts).
- **Data History:** The system shall store and allow users to access their historical data.
- **Report Generation:** The system shall generate reports summarizing the user's health and stress data over specified time periods.
- **Data Export:** Users shall be able to export their health data in various formats (e.g., CSV, PDF).

User Interface:

- **Dashboard:** The system shall have a user-friendly dashboard where users can manage their profile, upload videos, and view their health data and reports.
- **Interactive Forms/Video Upload Interface:** The system shall provide a clear and intuitive interface for uploading video files.
- **Responsive Design:** The system shall be mobile-friendly and adapt to different screen sizes.

3.7 Non-Functional Requirements

Non-functional requirements define the quality attributes and constraints of the system, such as performance, security, and usability.

Performance:

- **Processing Time:** The system shall process a typical video (e.g., 30 seconds) and generate results within a reasonable timeframe (e.g., under 2 minutes).

- **Heart Rate Accuracy:** The system shall achieve a heart rate detection accuracy comparable to or exceeding that of smartphone-based PPG applications (e.g., within ±5 bpm compared to a reference measurement).
- **HRV Accuracy:** The calculated HRV metrics shall correlate strongly with measurements obtained from established methods (e.g., ECG).
- **Scalability:** The system shall be able to handle a reasonable number of concurrent users (e.g., up to 100 concurrent users) without significant performance degradation.
- **Response Time:** User interface interactions (e.g., button clicks, page loads) shall have a response time of less than 2 seconds.

Usability:

- **Ease of Use:** The system shall be easy to use and understand for users with varying levels of technical expertise. The interface shall be intuitive and require minimal training.
- **Learnability:** Users shall be able to learn how to use the system quickly and efficiently.
- **Accessibility**: The system shall adhere to accessibility guidelines (e.g., WCAG) to ensure usability for people with disabilities.
- **User Interface Consistency:** The user interface shall maintain a consistent look and feel throughout the application.

Security:

- **Data Confidentiality:** User data, including videos and extracted physiological data, shall be protected from unauthorized access and disclosure. Data shall be encrypted both in transit (using HTTPS) and at rest (using appropriate encryption algorithms).
- **Data Integrity:** The system shall ensure that user data is not tampered with or modified without authorization.
- **Authentication:** The system shall use secure authentication mechanisms (e.g., strong password policies, secure password storage) to verify user identities.
- **Authorization:** The system shall implement authorization controls to restrict access to specific functionalities and data based on user roles.
- **Vulnerability Protection:** The system shall be protected against common web vulnerabilities (e.g., cross-site scripting, SQL injection).

Compatibility:

- **Web Browser Compatibility:** The system shall be compatible with the latest versions of major web browsers (e.g., Chrome, Firefox, Safari, Edge).
- **Operating System Compatibility:** The system shall be accessible on common desktop and mobile operating systems (e.g., Windows, macOS, Linux, Android, iOS).

Reliability:

- **Availability:** The system shall be available to users for at least 99% of the time, excluding scheduled maintenance.
- **Fault Tolerance:** The system shall be designed to minimize the impact of hardware or software failures.
- **Data Backup and Recovery:** The system shall implement regular data backups and have a recovery plan in place to restore data in case of data loss.

Maintainability:

- **Code Maintainability:** The codebase shall be well-structured, documented, and follow coding best practices to facilitate maintenance and future development.
- **Modularity**: The system shall be designed in a modular way to allow for easy modification and extension of functionalities.

Chapter 4

SYSTEM DESIGN

The system design for the AI-driven wind turbine site selection system involves outlining how the components of the software will work together to achieve the desired outcomes. It encompasses both the system architecture and detailed design. These elements ensure that the system is robust, scalable, efficient, and user-friendly.

4.1 System Architecture

The system architecture represents the high-level structure of the wind turbine site selection system, showing how different components interact with each other. The architecture is designed to handle user requests, process data, and deliver the site evaluation results using AI and machine learning algorithms.

4.1.1 Architectural Overview

The system follows a client-server architecture in which:

The client side (frontend) is responsible for interacting with users through the interface.

The server side (backend) performs computations, manages data storage, and interacts with the machine learning models.

The key components of the architecture include:

- **Frontend (Client):** The user interface, built using HTML, CSS, JavaScript, and potentially a framework like React.js, communicates with the backend to send data and display results.

- **Backend (Server):** The backend, developed using Python and Flask, handles the business logic, manages user authentication, interacts with machine learning models, and communicates with the database.
- **Database:** A relational database (e.g., SQLite or MySQL) stores user credentials, input data, evaluation results, and other project-specific information.
- **Machine Learning Model:** Pre-trained models, loaded into the backend, perform predictions and evaluations based on the input data.
- **API Layer:** A layer of API endpoints (e.g., RESTful API) exposes necessary backend services to the frontend, enabling communication and data exchange.

4.1.2 Detailed Components of the Architecture

- **User Interface (UI):** The UI is the first point of contact for the user. It collects input data like wind speed, environmental factors, terrain, and economic viability. It also displays the evaluation results (efficiency score and optimality status) along with any visualizations (e.g., bar charts, radar charts) to enhance the user experience. The UI communicates with the backend through API requests to send user data and receive results.
- **Web Server (Flask):** The Flask web framework handles HTTP requests, including POST requests for submitting site data and GET requests for fetching evaluation results. It is responsible for routing user requests, validating inputs, and invoking the machine learning models to perform the site selection process.
- **Machine Learning Component:** The backend will integrate machine learning models (e.g., Random Forest, Gradient Boosting, Neural Networks) that predict wind potential, assess terrain, evaluate economic feasibility, and calculate overall efficiency. Models will be pre-trained on historical data and will be loaded into the system to perform real-time evaluations when users submit new site data.
- **Database:** The system will use a database like SQLite or MySQL to store user information (credentials, session data) and site evaluation data. The database schema will have tables to store user details, input data, and results from the site evaluations.

- **API Layer:** RESTful APIs will be used to enable smooth communication between the frontend and backend. For example: A POST request to submit site data for evaluation. A GET request to fetch the results of an evaluation.
- **Authentication & Session Management:** The system should implement secure user authentication, possibly through a login system, which ensures that only authorized users can access the system. Sessions will be used to maintain user state between requests, ensuring a consistent experience.

4.2 Detailed Design

Detailed design refers to the lower-level components of the system, focusing on the implementation and how each component will be developed. It includes the specific interactions between the system's modules and the flow of data through the system.

4.2.1 User Interface (Frontend Design)

The frontend design will focus on providing an intuitive, responsive, and visually appealing interface for users. It will consist of:

Login/Signup Pages:

- **Functionality:** Users can register an account or log in using their credentials. If credentials are incorrect, an error message is shown.
- **UI Elements:** Form fields for username, password, and submit buttons.
- **Flow:** Once logged in, users will be directed to the dashboard for site evaluation.

Dashboard:

- **Functionality:** After logging in, users can access the main dashboard, where they can input data for site evaluation and view the results.
- **UI Elements:** Input fields for site data (e.g., wind speed, economic viability). Dropdown menus for selecting environmental impact, accessibility, etc. A button to submit the form and trigger the evaluation process.
- **Flow:** After submitting the form, the frontend sends the data to the backend for processing.

Result Display:

- **Functionality:** After the backend processes the data, the UI displays the efficiency score, optimality result (e.g., "Optimal" or "Not Optimal"), and visual representations (e.g., bar charts showing contributing factors).
- **UI Elements:** Efficiency score displayed in large text, visualizations for clarity, and recommendations for improving site suitability.
- **Responsiveness:** The UI will adapt to different screen sizes, ensuring usability on desktops, tablets, and mobile devices. The design will follow modern web design practices, including using frameworks like Bootstrap or Tailwind CSS for responsiveness.

4.2.2 Backend Design (Server-Side)

The backend is responsible for the core business logic, including data validation, calculations, and interfacing with the machine learning model. It consists of the following components:

Flask Application: Functionality: Handle incoming requests, interact with the database, and communicate with the machine learning models.

Routing: Define routes for different functionalities:

/login: Handle user authentication.

/signup: Handle user registration.

/dashboard: Display the site evaluation page.

/evaluate: Process the site data and return the results.

Machine Learning Integration:

- **Functionality:** Load pre-trained machine learning models (e.g., using joblib or pickle) and use them to evaluate site data.
- **Process:** Accept the input parameters (wind speed, environmental impact, economic viability). Apply the model to predict the site's efficiency score. Return the results to the frontend, including visual feedback on whether the site is optimal or not.
- **Data Validation:** The backend will validate user input to ensure that data is correctly formatted before processing (e.g., checking for valid numeric input

for wind speed). If any data is invalid, the system will return an error message to the user on the frontend.

- **Session Management:** The Flask app will manage user sessions, ensuring users stay logged in during their interaction with the site. Flask-Login or Flask- Security can be used for implementing secure user sessions.
- **Database Interaction:** The backend will interact with the database to store user data and evaluation results. It will use SQL queries to fetch user information, validate login credentials, and store new evaluation results.

4.2.3 Database Design

The database schema will consist of several tables:

- **Users Table:**

  Stores user credentials (username,

  password). Fields: id, username,

  password_hash.

- **Evaluations Table:**

  Stores the results of each site evaluation.

  Fields: id, user_id, wind_speed, environmental_impact, economic_viability, humidity, accessibility_score, efficiency_score, optimality_result.

- **Session Table:**

  Stores session-related data to maintain user authentication across requests.

  2.4 API Layer Design

The API layer will define various endpoints for handling frontend requests:

- **POST /evaluate:**

  Accepts data from the frontend (wind speed, environmental impact, etc.). Returns the site evaluation results, including efficiency scores and recommendations.

- **GET /results:**

Retrieves past evaluation results stored in the database and presents them on the frontend.

4.3 Data Flow Diagram (DFD)

A Data Flow Diagram (DFD) visually represents how data flows within the system and how different components of the system interact with one another. It helps identify the processes involved, the data they manipulate, and how they relate to other system components.

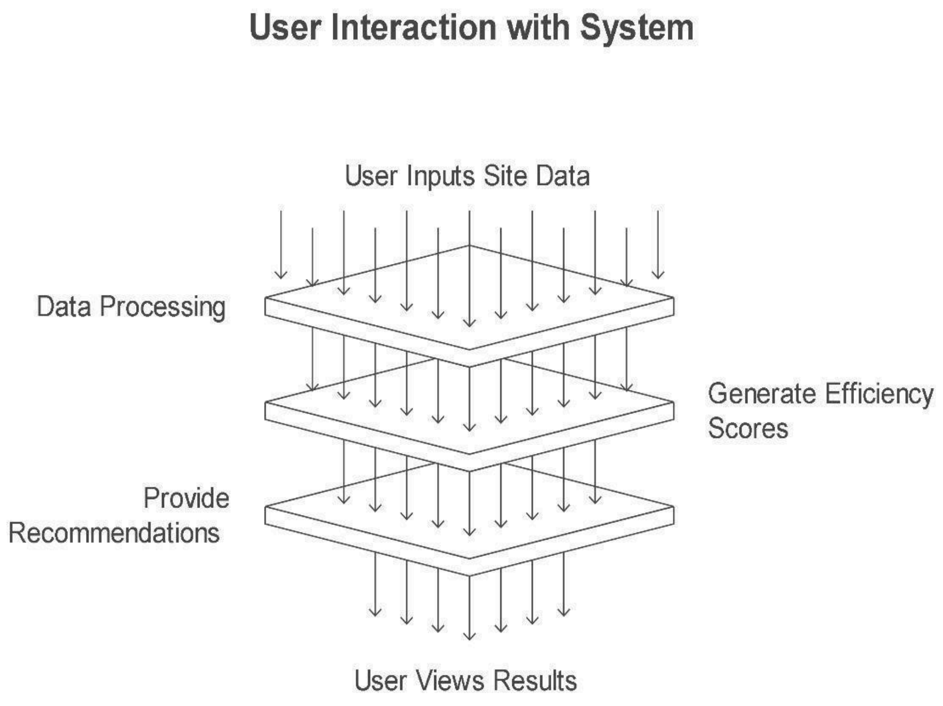


Figure 4.2: User Interaction with System.

Level 1 DFD for the AI-driven Wind Turbine Site Selection System:

1. Context Diagram (Level 0 DFD)

The Context Diagram (Level 0) shows the system as a single process and highlights its interactions with external entities. In this case, the system interacts with Users (who provide input data and receive results) and the Machine Learning Model (which processes the data to generate results).

Entities:

- **User:** Interacts with the system, providing site data and receiving results.
- **Machine Learning Model:** Processes site data to evaluate efficiency and determine optimality.

Data Flow:

- **User → System:** The user inputs site data (wind speed, environmental impact, etc.) into the system.
- **System → Machine Learning Model:** The system sends site data to the model for evaluation.
- **Machine Learning Model → System:** The model sends back the efficiency score and optimality results.
- **System → User:** The system sends the evaluation results (efficiency score and recommendations) to the user.

2. Level 1 DFD

In the Level 1 DFD, the system is broken down into subprocesses, and the flow of data between those subprocesses is depicted in greater detail.

Processes:

- **User Authentication:** Manages login and registration, including user credential validation.
- **Data Input and Validation:** Handles user-provided site data (wind speed, terrain, etc.), ensuring it is complete and valid.
- **Result Generation and Display:** Displays the evaluation results, including the efficiency score and recommendations, to the user.

External Entities:

- **User:** Provides input data for the site and views the evaluation results.
- **Machine Learning Model:** Evaluates site data to determine the efficiency score.

Level 1 DFD Breakdown:

Process 1: User Authentication

Inputs:

- **Username and Password:** Sent by the user during login or registration.

Outputs:

- **Authenticated** User: The system checks credentials, and if valid, grants access to the dashboard.
- **Error Message:** If authentication fails, the system returns an error message.

Process 2: Data Input and Validation

Inputs:

- **Site Data:** The user provides data such as wind speed, environmental impact, terrain, etc.

Outputs:

- **Validated Data:** If the data is correct, it is passed to the site evaluation process.
- **Error Message:** If the data is invalid, an error message prompts the user to correct the input.

Process 3: Site Evaluation

Inputs:

- **Validated Site Data:** Data provided by the user, such as wind speed, economic viability, etc.
- **Machine Learning Model:** The system sends the data to the trained machine learning model to assess site suitability.

Outputs:

- **Efficiency Score:** Calculated score indicating how suitable the site is for wind turbine installation.

- **Optimality Result**: "Optimal" or "Not Optimal" based on the efficiency score threshold.

Process 4: Result Generation and Display

Inputs:

- **Efficiency Score and Optimality Result:** Generated from the site evaluation process.

Outputs:

- **Evaluation Results:** Displayed to the user on the dashboard, showing efficiency score and optimality status.
- **Recommendations:** Suggestions for improving site suitability, if necessary.

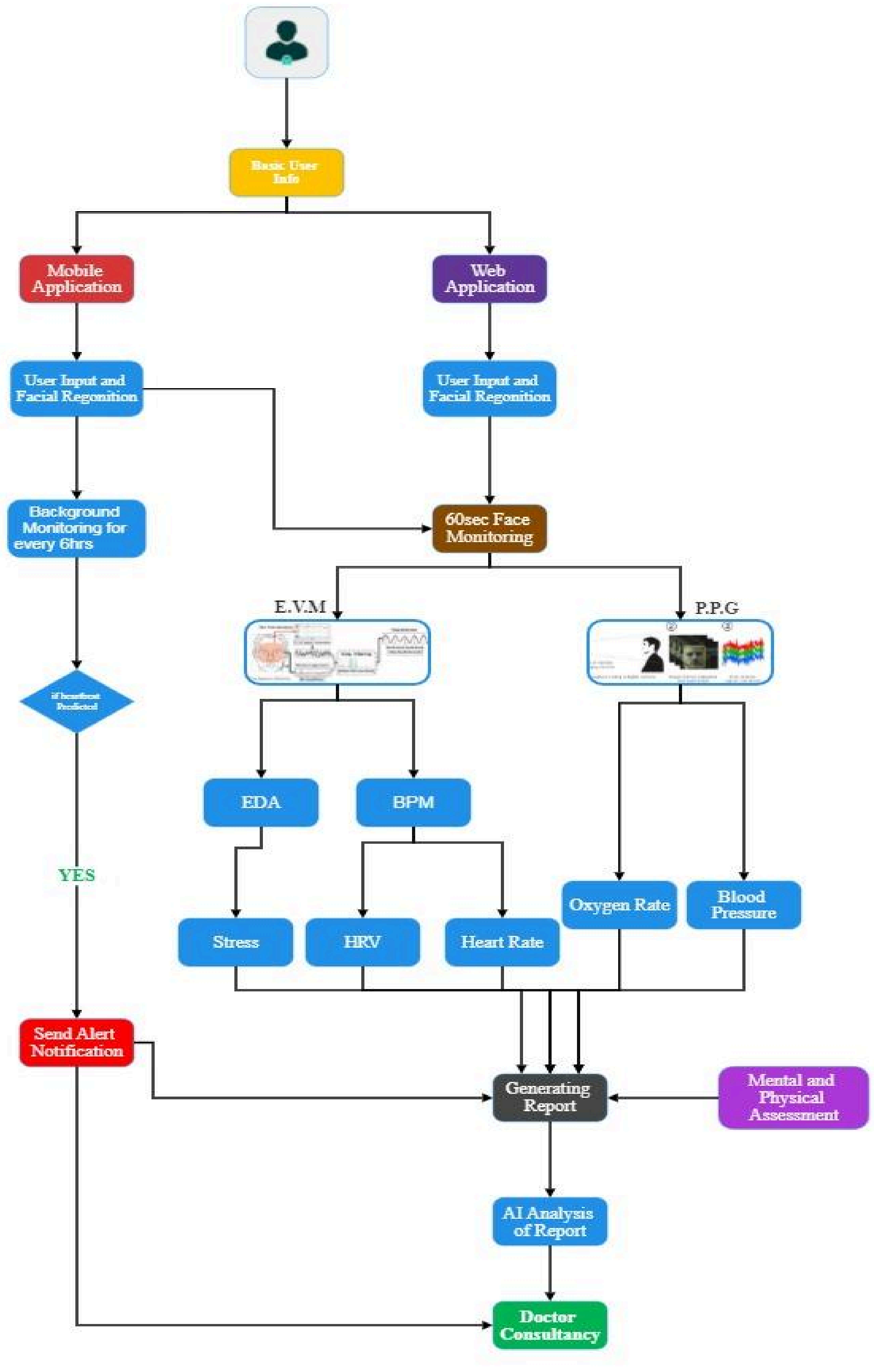


Figure 4.3: Data Flow Diagram.

Chapter 5

IMPLEMENTATION

The implementation of the AI-driven system for wind turbine site selection involves transforming the design into a functional system. This section outlines the step-by-step process of how the system is built and integrated, focusing on its core components: frontend development, backend implementation, machine learning integration, database setup, and deployment.

1.1 Frontend Implementation

The frontend provides the user interface, enabling users to interact with the system.

1.1.1 Tools and Technologies

HTML/CSS: For structuring and styling the web pages.

JavaScript: For adding interactivity to the web interface.

1.1.2 Steps

Create the Login/Signup Page:

Use HTML forms to capture user credentials.

Add JavaScript for client-side validation (e.g., ensure password fields are not empty).

Develop the Dashboard:

Create a data input form for site parameters (e.g., wind speed, economic viability).

Result Display:

Use dynamic elements (e.g., charts via Chart.js or D3.js) to display efficiency scores and site evaluations.

1.2 Backend Implementation

The backend handles the application logic, including user authentication, data validation, and communication with the machine learning model.

1.2.1 Tools and Technologies

Python: Primary programming language for backend development.

Flask: Lightweight web framework for handling HTTP requests and routing.

RESTful APIs: For communication between the frontend and backend.

1.2.2 Steps

Set Up Flask Application:

Initialize a Flask app with routes for user authentication (/login, /signup) and site evaluation (/evaluate).

Define templates for rendering HTML pages and handling HTTP requests.

Implement User Authentication:

Use Flask-Login to manage user sessions.

Hash passwords using bcrypt or Werkzeug.security for secure storage.

Data Validation:

Add input validation logic to ensure submitted site data is correct (e.g., wind speed > 0, categorical values match predefined options).

Integrate Machine Learning Models:

Load pre-trained models using libraries like joblib or pickle.

Write functions to send user input to the model and return predictions (e.g., efficiency scores and optimality results).

Connect to Database:

Use sqlite3 or an ORM like SQLAlchemy to store and retrieve user data, evaluation results, and session details.

Test Backend APIs:

Test API endpoints using tools like Postman to ensure they respond correctly to valid and invalid requests.

1.3 Machine Learning Integration

The machine learning model is the core of the site evaluation process.

1.3.1 Tools and Technologies

Scikit-learn: For training and deploying models like Random Forest or Gradient Boosting.

NumPy/Pandas: For handling and preprocessing data.

Joblib: For saving and loading pre-trained models.

1.3.2 Steps

Train the Model:

Use historical wind data and site parameters to train a regression model that predicts efficiency scores.

Split the dataset into training and test sets for model evaluation.

Perform hyperparameter tuning (e.g., adjusting tree depth for Random Forest) to optimize performance.

Save the Model:

Serialize the trained model using joblib.dump for integration into the backend.

Integrate with Backend:

Load the model in the Flask app using joblib.load.

Create a function to preprocess user input, run predictions, and return results.

Validate the Model:

Use validation datasets to ensure the model's predictions align with real-world outcomes.

Measure performance using metrics like Mean Squared Error (MSE) and R-squared (R^2).

1.4 Database Setup

The database stores user credentials, input data, and evaluation results.

1.4.1 Tools and Technologies

SQLite: Lightweight database for local development.

SQLAlchemy: For managing database operations in Python.

1.4.2 Steps

Design Database Schema:

Create tables for:

Users (id, username, password_hash).

Evaluations (id, user_id, wind_speed, efficiency_score, optimality_result).

Initialize the Database:

Write scripts to set up the database structure.

Use Flask's database extension or raw SQL queries to create tables.

Integrate with Flask:

Connect the Flask app to the database using SQLAlchemy.

Write functions for storing and retrieving user data and evaluation results.

1.5 Deployment

Deployment makes the system accessible to end users.

1.5.1 Tools and Technologies

Hosting Services: Heroku, AWS, or Google Cloud.

Version Control: Git for tracking changes and collaborating with team members.

1.5.2 Steps

Prepare the Application:

Ensure the Flask app is production-ready (disable debug mode, configure environment variables).

Add necessary files like requirements.txt and Procfile for deployment.

Frontend:

Design the UI with HTML, CSS, JavaScript, and frameworks.

Build interactive pages for login, data input, and result display.

Backend:

Set up Flask routes for handling user requests.

Integrate machine learning models for site evaluation.

Machine Learning:

Train models with historical data.

Validate and save models for integration.

Database:

Design and set up tables for storing user and evaluation data.

Deployment:

Host the application on a cloud platform for global accessibility.

1.6 Algorithm

1.6.1 Ensemble Learning

Ensemble learning is a machine learning technique where multiple models (often referred to as "weak learners" or "base models") are trained and combined to solve a single problem. The goal is to improve the overall predictive performance compared to any single model. Ensembles can reduce overfitting, handle complex data relationships, and provide better generalization.

Types of Ensemble Methods:

Bagging (Bootstrap Aggregating):

Involves training multiple models on different subsets of the data (created via bootstrapping, i.e., sampling with replacement).

Boosting:

Trains models sequentially, where each model corrects the errors of the previous one.

Assigns higher weights to misclassified instances to make subsequent models focus more on them.

Stacking:

Combines predictions from multiple models (often of different types) using a meta-model that learns how to best combine these predictions.

1.6.2 Random Forest

Random Forest is an ensemble learning method based on bagging, specifically using decision trees as the base learners. It is a versatile algorithm used for both classification and regression tasks.

Data Sampling:

Randomly selects subsets of the data (bootstrapping) to train each decision tree.

Each tree is trained on a different sample, ensuring diversity among trees.

Feature Selection:

At each split in a tree, Random Forest considers only a random subset of features rather than all features.

Tree Construction:

Each decision tree is grown to its full depth without pruning, making it a strong learner on its subset.

Prediction Aggregation:

For classification: The final prediction is the majority vote from all the trees.

For regression: The final prediction is the average of predictions from all the trees.

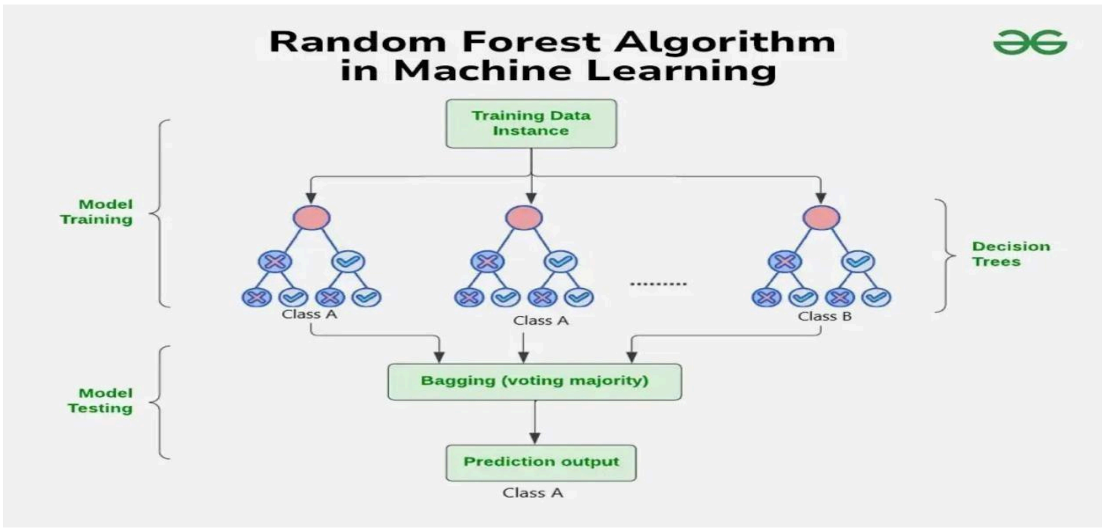


Figure 5.2: Random Forest Algorithm.

1.7 Backend code

```
from flask import Flask, render_template, request, session,Response

import pandas as pd

import mysql.connector

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier,
RandomForestRegressor

from sklearn.preprocessing import LabelEncoder

from reportlab.lib.pagesizes import letter

from reportlab.lib.styles import getSampleStyleSheet

from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer

from io import BytesIO

from reportlab.lib.pagesizes import letter

from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle

from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer,
Image

from reportlab.lib.units import

inch app = Flask(___name___)

app.secret_key = 'your_secret_key'


# Configure MySQL connection

db_config = {

    'user': 'root',

    'password': 'root',

    'host': 'localhost',

    'database': 'health_data'

}



# Load dataset

df = pd.read_csv("heart_attack_prediction_dataset.csv")


# Drop unnecessary columns
```

```
df1 = df.drop(['Patient ID', 'Income', 'Country', 'Continent', 
'Hemisphere', 'Cholesterol'], axis=1)


# Convert 'Sedentary Hours Per Day' to integer

df1['Sedentary Hours Per Day'] = df1['Sedentary Hours Per 
Day'].astype(int)


# Encode categorical variables

le = LabelEncoder()

df1['Sex'] = le.fit_transform(df1['Sex'])

df1['Diet'] = le.fit_transform(df1['Diet'])


# Split 'Blood Pressure' into 'BP1' and 'BP2'

def split_blood_pressure(blood_pressure):

    return pd.Series(blood_pressure.split('/', 1))


df1[['BP1', 'BP2']] = df1['Blood 
Pressure'].apply(split_blood_pressure)

df1 = df1.drop(['Blood Pressure', 'Triglycerides', 'Sedentary Hours 
Per Day'], axis=1)


# Convert 'BP1' and 'BP2' to numeric

df1['BP1'] = pd.to_numeric(df1['BP1'], errors='coerce')

df1['BP2'] = pd.to_numeric(df1['BP2'], errors='coerce')


# Define weights for features

weights = {

    'Age': 0.1,

    'Sex': 0.05,

    'Heart Rate': 0.15,

    'Diabetes': 0.2,

    'Family History': 0.15,

    'Smoking': 0.25,

    'Obesity': 0.2,
    'Alcohol Consumption': 0.1,

    'Exercise Hours Per Week': 0.1,

    'Diet': 0.15,

    'Previous Heart Problems': 0.3,

    'Medication Use': 0.05,

    'Stress Level': 0.2,

    'BMI': 0.15,

    'Physical Activity Days Per Week': 0.1,

    'Sleep Hours Per Day': 0.1,

    'Heart Attack Risk': 0.4, # This is a composite measure, not a
factor itself

    'BP1': 0.25, # Systolic Blood Pressure

    'BP2': 0.25   # Diastolic Blood Pressure

}




# Modify weights based on

conditions for index, row in

df1.iterrows():

    if row['Age'] >= 45:

        weights['Age'] = 0.2

    if row['Sex'] == 0:

        weights['Sex'] = 0.1

    if row['Heart Rate'] < 60:

        weights['Heart Rate'] = 10 + (row['Heart Rate'] - 1) * 0.02

    elif row['Heart Rate'] > 100:

        weights['Heart Rate'] = 0.2 + (row['Heart Rate'] - 100) *
0.02

    if row['BP1'] > 150:

        weights['BP1'] = 0.2 + (row['BP1'] - 150) * 0.02

    if row['BP2'] > 90:

        weights['BP2'] = 0.2 + (row['BP2'] - 90) * 0.02
```

```
# Calculate total weighted sum

total_weighted_sum = df1.apply(lambda row: sum(row[col] *
weights[col] for col in df1.columns), axis=1)


# Normalize total weighted sum

max_weighted_sum = total_weighted_sum.max()

min_weighted_sum = total_weighted_sum.min()

df1['percentage'] = ((total_weighted_sum - min_weighted_sum) /
(max_weighted_sum - min_weighted_sum)) * 100


# Features and target variable

X = df1.drop(columns=['Heart Attack Risk', 'percentage'])

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Initialize and train Random Forest Classifier

random_forest_classifier = RandomForestClassifier(n_estimators=100,
random_state=42)

random_forest_classifier.fit(X_scaled, df1['Heart Attack Risk'])


# Initialize and train Random Forest Regressor

random_forest_regressor = RandomForestRegressor(n_estimators=100,
random_state=42)

random_forest_regressor.fit(X_scaled, df1['percentage'])


# Function to predict manually

def predict_manually(age, sex, heart_rate, diabetes, family_history,
smoking, obesity, alcohol_consumption,

                     exercise_hours_per_week, diet,
previous_heart_problems, medication_use, stress_level,

                     bmi, physical_activity_days_per_week,
sleep_hours_per_day, bp1, bp2):

    input_data = pd.DataFrame([[age, sex, heart_rate, diabetes,
family_history, smoking, obesity, alcohol_consumption,

                                exercise_hours_per_week, diet,
previous_heart_problems, medication_use, stress_level,
                                bmi,
physical_activity_days_per_week, sleep_hours_per_day, bp1, bp2]],
                              columns=X.columns)
    input_scaled = scaler.transform(input_data)
    predicted_heart_attack_risk =
random_forest_classifier.predict(input_scaled)[0]
    predicted_percentage =
random_forest_regressor.predict(input_scaled)[0]
    return predicted_heart_attack_risk, predicted_percentage


# Function to get user details from the database
def get_user_details(email):
    conn = mysql.connector.connect(**db_config)
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT age, sex, diabetes, famhistory, smoking,
obesity, alcohol, exercise_hours, diet, heart_problem, bmi,
physical_activity, sleep_hours, blood_pressure_systolic,
blood_pressure_diastolic FROM user_details WHERE email = %s",
(email,))
    user_details = cursor.fetchone()
    cursor.close()
    conn.close()
    return user_details


@app.route('/')
def index():
    email = session.get('email')
    user_details = None
    if email:
        user_details = get_user_details(email)
    return render_template('index.html', user_details=user_details)


@app.route('/predict', methods=['POST'])
def predict():
    if request.method == 'POST':
        email =
        session.get('email') if not
        email:
            return render_template('index.html', error="Session
expired. Please log in again.")

        user_details = get_user_details(email)
        if not user_details:
            return render_template('index.html', error="User details
not found in the database.")

        # Extract user data from
        database age =
        int(user_details['age']) sex =
        int(user_details['sex'])
        diabetes = int(user_details['diabetes'])
        family_history = int(user_details['famhistory'])
        smoking = int(user_details['smoking'])
        obesity = int(user_details['obesity'])
        alcohol_consumption = int(user_details['alcohol'])
        exercise_hours_per_week =
float(user_details['exercise_hours'])
        diet = int(user_details['diet'])
        previous_heart_problems = int(user_details['heart_problem'])
        bmi = float(user_details['bmi'])
        physical_activity_days_per_week =
int(user_details['physical_activity'])
        sleep_hours_per_day = float(user_details['sleep_hours'])
        bp1 = int(user_details['blood_pressure_systolic'])
        bp2 = int(user_details['blood_pressure_diastolic'])

        # Debugging: Print the form data
        print("Form data received:", request.form)
        try:

                                                    # Extract heart rate, stress level,
form data                                           HRV, and SpO2 from

                                                    heart_rate =
                                                    int(request.form['heart_
                                                    rate']) stress_level =
                                                    int(request.form['stress
                                                    _level']) hrv =
                                                    int(request.form['hrv'])
                                                    spo2 = int(request.form['spo2'])
        except KeyError as e:
            return render_template('index.html', error=f"Missing
form data: {e}")


        # Predict using user data and input heart rate and stress
level

        predicted_heart_attack_risk, predicted_percentage =
predict_manually(
            age, sex, heart_rate, diabetes, family_history, smoking,
obesity,
            alcohol_consumption, exercise_hours_per_week, diet,
previous_heart_problems,
            0, stress_level, bmi, physical_activity_days_per_week,
sleep_hours_per_day, bp1, bp2
        )

        messages = []
        # Adjust predicted percentage based on HRV and SpO2
        if hrv < 45 :
            messages.append("Your HRV is too low ")
            predicted_percentage += 25.8
        if hrv > 200:
            messages.append("Your HRV is too high ")
            predicted_percentage += 5.65
        if spo2 < 90:
            messages.append("Your SpO2 level is below normal.")
            predicted_percentage += 4.5

        # Generate messages based on user's health data


        if age > 50:
            messages.append("Age greater than 50.")
        if heart_rate < 60 or heart_rate > 100:
            messages.append("Your heart rate is not good.")
        if diabetes:
            messages.append("You have diabetes.")
        if family_history:
            messages.append("You have a family history of heart
problems.")
        if smoking:
            messages.append("You are a smoker.")
        if obesity:
            messages.append("You are obese.")
        if alcohol_consumption:
            messages.append("You consume alcohol.")
        if previous_heart_problems:
            messages.append("You have had previous heart problems.")
        if stress_level > 6:
            messages.append("Stress Level is High, Take rest or
Hangout.")
        if bp1 > 140 or bp2 > 80:
            messages.append("Your blood pressure is high.")



        # Add HRV and SpO2 messages
```

```
< 50:
                        # Determine the user's category

                        if predicted_heart_attack_risk == 0
                        and predicted_percentage

                                    category = "Heart
                        attack risk 0, heart attack
                        risk percentage less than 50.
                        You are safe. Please take
                        care of yourself."

                                elif
                        predicted_heart_attack_risk
                        == 0 and predicted_percentage
                        >= 50:

                                    category = "Heart
                        attack risk 0, heart attack
                        risk percentage greater than
                        50. Please consult the doctor
                        by sharing the report."

                                elif
                        predicted_heart_attack_risk
                        == 1 and predicted_percentage
                        < 50:

                                    category = "Heart
                        attack risk 1, heart attack
                        risk percentage less than 50.
                        Something unpredictable.
                        Please consult a doctor."

                                else:

                                    category = "Heart
                        attack risk 1, heart attack
                        risk percentage greater than
                        50. Don't be afraid. Just
                        contact the nearest hospital.
                        That's all."


                                return
                        render_template('result.html',
                        heart_attack_risk=predicted_heart_a
                        ttack_risk,

                                                               perc
                        entage=predicted_percentage,
                        messages=messages,
                        category=category)

                            else:

                                return
```

```
        render_template('index.html
        ')
```

```
@app.route('
/download_re
port',
methods=['PO
ST']) def
download_rep
ort():
    i
        f
        r
        e
        q
        u
        e
        s
        t
        .
        m
        e
        t
        h
        o
        d
        =
        =
        '
        P
        O
```

```
S
T
'
:
e
m
a
i
l
=
s
e
s
s
i
o
n
.
g
e
t
(
'
e
m
a
i
l
'
)
i
f
n
```

```
o                                     d
t                                     e
e                                     t
m                                     a
a                                     i
i                                     l
l                                     s
:                                     (
    return "Session                   e
    expired. Please log in            m
    again."                           a
                                      i
u                                     l
s                                     )
e                                     i
r                                     f
_                                     n
d                                     o
e                                     t
t                                     u
a                                     s
i                                     e
l                                     r
s                                     _
=                                     d
g                                     e
e                                     t
t                                     a
_                                     i
u                                     l
s                                     s
e                                     :
r
_
```

```
            return "User details not found in the database."

        # Retrieve report content from form data
        report_content = request.form['report_content']

        # Create a PDF buffer
        pdf_buffer = BytesIO()
        doc = SimpleDocTemplate(pdf_buffer, pagesize=letter)

        # Define the styles
        styles = getSampleStyleSheet()
        title_style = ParagraphStyle(
            'TitleStyle',
            parent=styles['Title'],
            fontSize=18,
            spaceAfter=14,
        )


        body_style = ParagraphStyle(
            'BodyStyle',
            parent=styles['BodyText'],
            fontSize=12,
            leading=14,
            spaceAfter=10,
        )



        # Create the report
        elements elements = []

        # Add title and user details
        elements.append(Paragraph("Heart Attack Risk Report",
title_style))
        elements.append(Spacer(1, 12))
        user_info = f"""
        <b>User Details:</b><br/>
        Age: {user_details['age']}<br/>
        Sex: {user_details['sex']}<br/>
        Diabetes: {user_details['diabetes']}<br/>
        Family History: {user_details['famhistory']}<br/>
        Smoking: {user_details['smoking']}<br/>
        Obesity: {user_details['obesity']}<br/>
        Alcohol Consumption: {user_details['alcohol']}<br/>
        Exercise Hours Per Week:
{user_details['exercise_hours']}<br/>
        Diet: {user_details['diet']}<br/>
        Previous Heart Problems:
{user_details['heart_problem']}<br/>
        BMI: {user_details['bmi']}<br/>
        Physical Activity Days Per Week:
{user_details['physical_activity']}<br/>
        Sleep Hours Per Day: {user_details['sleep_hours']}<br/>
        Blood Pressure (Systolic/Diastolic):
{user_details['blood_pressure_systolic']}/{user_details['blood_press
ure_diastolic']}<br/>
        """


        elements.append(Paragraph(user_info, body_style))
        elements.append(Spacer(1, 12))

        # Add the report content
        elements.append(Paragraph(report_content, body_style))
        elements.append(Spacer(1, 12))

        # Add the image
        image_path = 'static/final_graph.png'
                                              elements.append(Image(image_pa
inch))
                                      th, width=4 * inch, height=4 *

                                      elements.append(Spacer(1, 12))

                                      #
                                      Bui
                                      ld
                                      the
                                      PDF
                                      doc
                                      .bu
                                      ild
                                      (el
                                      eme
                                      nts
                                      )

        # Set up response to initiate download
        pdf_buffer.seek(0)
        response = Response(pdf_buffer, mimetype='application/pdf')
        response.headers.set("Content-Disposition", "attachment",
filename="heart_attack_report.pdf")

        return response
    else:
        return "Method not
allowed." if _____name__ == '____
main_':
    app.run(debug=True, port=3005)
```

5.7 Front-end code

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>BlusteryTech Site Prediction</title>
    <link rel="icon" type="image/png"
href="static/images/apple-touch-icon.png"
>
    <style>
        /* General Styles */
        body {
            font-family: 'Poppins', sans-serif;
            background: linear-gradient(to bottom, #e0f7fa,
#ffffff);
                                            }
        }
        h1 {
```

```
color: #333;
text
-ali
gn:
cent
er;
marg
in:
0;
padding: 0;
h
e
i
g
h
t
:
1
0
0
%
;
d
i
s
p
l
a
y
:
f
l
e
```

```
x
;
just
ify-
cont
ent:
cent
er;
alig
n-it
ems:
cent
er;
flex
-dir
ecti
on:
colu
mn;


f
o
n
t
-
s
i
z
e
:
2
.
```

```
5
r
e
m
;
m
a
r
g
i
n
-
t
o
p
:
2
0
p
x
;
c
o
l
o
r
:
#
0
0
6
9
5
```

```
c
;
animation: fadeIn 1s ease-out;
```

```
form {
    width: 80%;
    max-width: 700px;
    background: rgba(255, 255, 255, 0.9);
    padding: 30px;
    border-radius: 15px;
    box-shadow: 0 8px 20px rgba(0, 0, 0, 0.2);
    animation: slideIn 1s ease-out;
}

label {
    display: block;
    margin-top: 15px;
    font-size: 1.1rem;
    color: #00695c;
}
input, select, button {
    width: 100%;
    padding: 12px;
    margin-top: 10px;
    font-size: 1rem;
    border-radius: 10px;
    border: 1px solid #ccc;
    outline: none;
    transition: all 0.3s ease;
}

input:focus, select:focus {
    border-color: #00796b;
}

button {
    background-color: #00796b;
    color: white;
    font-weight: bold;
    border: none;
    cursor: pointer;
    margin-top: 20px;
    padding: 10px; /* Reduced size */
    transition: transform 0.2s;
}
button:hover {
    background-color: #004d40;
    transform: scale(1.05);
}

/* Animations */
@keyframes slideIn {
    from {
        transform: translateY(-50px);
        opacity: 0;
    }
    to {                transform:
                        translateY(0);
                        opacity: 1;
    }
}

@keyframes fadeIn {
    from {
        opacity: 0;
    }                                   }
    to {                            }
```

```
                            opacity: 1;

/* Toggle Switch Styles */
.toggle-container {
    display: inline-flex;
    align-items: center;
    position: relative;
    width: 80px;
    height: 40px;
    margin-top: 10px;
}
.toggle {
    display: none;
}
.toggle-label {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-color: #ccc;
    border-radius: 50px;
    transition: background-color 0.3s;
}
.toggle-label:before {
    content: '';
    position: absolute;
    top: 4px;
    left: 4px;
    width: 30px;
    height: 30px;
    background-color: white;
    border-radius: 50%;
    transition: left 0.3s;
}
.toggle:checked + .toggle-label {
    background-color: #00796b;
}
.toggle:checked + .toggle-label:before {
    left: 46px;
}


/*  Slider  Styles  */
input[type="range"] {
    width:  100%;
    height: 10px;
    background: #e0f2f1;
    border-radius: 10px;
    outline: none;
    transition: background 0.3s;
}
input[type="range"]::-webkit-slider-thumb {
    -webkit-appearance: none;
    appearance: none;
    width: 20px;
    height: 20px;
    background: #00796b;
    border-radius: 50%;
    cursor: pointer;
}
input[type="range"]:hover {
    background-color: #00796b;
```

```
}


/* Input Sliders Value Display */
.slider-value {
    font-weight: bold;
    margin-top: 5px;
}


.slider-container {
    margin-top: 10px;
    width: 100%;
    margin: 0 auto;
}


/* Yes/No labels */
.yes-no-labels {
    display: flex;
    justify-content: space-between;
    color: #00796b;
    font-weight: bold;
    margin-left: 10px; /* Place next to the toggle */
    margin-top: 0;
    padding-left: 5px;
    padding-right: 5px;
}


/* Dropdown with Placeholder */
select {
    background-color: #e0f7fa;
    color: #00796b;
```

```
}


select option:first-child {
    color: #aaa;
}


/* Logout Button */
.logout-btn {
    margin-top: 30px;
    padding: 12px 24px;
    background-color: #00796b;
    color: white;
    font-weight: bold;
    border: none;
    cursor: pointer;
    border-radius: 10px;
    transition: transform 0.2s;
}
.logout-btn:hover {
    background-color: #00796b;
    transform: scale(1.05);
}



/* Flash message styles */
.message-alert {
    display: flex;
    align-items: center;
    padding: 15px;
    border-radius: 5px;
    margin: 20px 0;
    width: fit-content;
```

```
            position: fixed;
            top: 10px;
            left: 50%;
            transform: translateX(-50%);
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
            font-size: 18px;
        }

        .message-alert.success {
            background-color: #28a745;
            color: white;
        }


        .message-alert.error {
            background-color: #e74c3c;
            color: white;
        }


        .message-alert .icon {
            margin-right: 10px;
            font-size: 30px;
        }


        .message-alert .message {
            font-size: 18px;
        }
        footer {
            text-align: center;
            padding: 1rem;
            background: rgba(0, 122, 140, 0.9); /* Dark Green
*/
            color: white;
            position: relative;

            width: 100%;

            bottom: 0;

            animation: fadeIn 1s ease-out; /* Footer fade-in
animation */

        }

    </style>

</head>

<body>

    <h1>Enter Site Factors for Wind Turbine Suitability
Prediction</h1>


    <!-- Flash Message for errors or success -->

    {% with messages =
get_flashed_messages(with_categories=true) %}

        {% if messages %}

            {% for category, message in messages %}

                <div class="message-alert {{ category }}">

                    <span class="icon">{{ '✔' if category ==
'success' else '✖' }}</span>

                    <span class="message">{{ message }}</span>

                </div>

            {% endfor %}

        {% endif %}

    {% endwith %}


    <form action="/dashboard" method="POST">

        <h2>

        <div class="slider-container">

        <label for="wind_speed">Wind Speed (m/s):</label>

        <input type="range" name="wind_speed" min="0" max="25"
step="0.1" oninput="updateSliderValue('wind_speed',
this.value)" required>

            <div class="slider-value"
id="wind_speed_value">12.5</div><br>
```

```
        <label for="wind_direction">Wind Direction
(degrees):</label>

        <input type="range" name="wind_direction" min="0"
max="360" step="1" oninput="updateSliderValue('wind_direction',
this.value)" required>

            <div class="slider-value"
id="wind_direction_value">180</div><br>


        <label for="elevation">Elevation (m):</label>

        <input type="range" name="elevation" min="0" max="3000"
step="10" oninput="updateSliderValue('elevation', this.value)"
required>

            <div class="slider-value"
id="elevation_value">1500</div><br>


        <label for="humidity">Humidity (%):</label>

        <input type="range" name="humidity" min="0" max="100"
step="1" oninput="updateSliderValue('humidity', this.value)"
required>

            <div class="slider-value"
id="humidity_value">50</div><br>

        </div>

        <label for="accessibility">Accessibility:</label>

        <select name="accessibility" id="accessibility"
required>

            <option disabled selected>Click Here</option>

            <option>Good</option>

            <option>Moderate</option>

            <option>Poor</option>

        </select><br>

        <label for="seasonal_variation">Seasonal
Variation:</label>

        <select name="seasonal_variation"
id="seasonal_variation" required>

            <option disabled selected>Click Here</option>

            <option>High</option>

            <option>Moderate</option>
            <option>Low</option>
        </select><br>

        <label for="terrain_type">Terrain Type:</label>
        <select name="terrain_type" id="terrain_type" required>
            <option disabled selected>Click Here</option>
            <option>Flat</option>
            <option>Hilly</option>
            <option>Mountainous</option>
        </select><br>

        <label for="land_use">Land Use:</label>
        <select name="land_use" id="land_use" required>
            <option disabled selected>Click Here</option>
            <option>Agricultural</option>
            <option>Commercial</option>
            <option>Residential</option>
            <option>Forest</option>
        </select><br>

        <label for="environmental_impact (1-10)">Environmental
Impact:</label>
        <input type="range" name="environmental_impact" min="1"
max="10" step="1"
oninput="updateSliderValue('environmental_impact', this.value)"
required>
            <div class="slider-value"
id="environmental_impact_value">5</div>

        <label for="wildlife_habitats_nearby">Wildlife Habitats
Nearby(No/Yes):</label>
        <div class="toggle-container">
            <input type="checkbox"
id="wildlife_habitats_nearby" class="toggle"
name="wildlife_habitats_nearby">
            <label for="wildlife_habitats_nearby"
```

```
class="toggle-label"></label>
            </div>
        <div class="slider-container">
        <label for="economic_viability">Economic Viability (1-
10):</label>
        <input type="range" name="economic_viability" min="1"
max="10" step="1"
oninput="updateSliderValue('economic_viability', this.value)"
required>
            <div class="slider-value"
id="economic_viability_value">5</div>


        <label for="permit_status">Permit Status:</label>
        <select name="permit_status" id="permit_status"
required>
            <option disabled selected>Click Here</option>
            <option>Approved</option>
            <option>Pending</option>
            <option>Rejected</option>
        </select><br>


        <button type="submit">Predict</button>
        </h2>
    </form>
    {% if efficiency is not none %}
        <div class="result">
            <h2>Efficiency Score: {{ efficiency }}%</h2>
            <h2>Prediction Result: {{ result }}</h2>
        </div>
    {% endif %}
    <br>
    <!-- Logout Button -->
    <form action="/" method="GET">
        <button type="submit" class="logout-btn">Logout
</button>
```

```
    </form>

    <script>
        function updateSliderValue(id, value) {
            document.getElementById(id + '_value').innerText =
value;

        }

    </script>

     <script>

        // Dismiss the message after 5 seconds

        setTimeout(function() {

            var messageAlert =
document.querySelector('.message-alert');

            if (messageAlert) {

                messageAlert.style.display = 'none';

            }

        }, 5000);

    </script>

<footer>

    <p>&copy; 2024 BlusteryTech. All rights reserved.</p>

</footer>

</body>

</html>
```

Chapter 6

TESTING

Testing can be stated as the process of verifying and validating whether a software or application is bug-free, meets the technical requirements as guided by its design and development, and meets the user requirements effectively and efficiently by handling all the exceptional and boundary cases.

The process of software testing aims not only at finding faults in the existing software but also at finding measures to improve the software in terms of efficiency, accuracy, and usability. It mainly aims at measuring the specification, functionality, and performance of a software program or application. Test Case is a set of actions executed to verify a particular feature or functionality of your software application.

A Test Case contains test steps, test data, precondition, post condition developed for specific test scenario to verify any requirement. The test case includes specific variables or conditions, using which a testing engineer can compare expected and actual results to determine whether a software product is functioning as per the requirements.

6.1 Unit Testing

Unit testing focuses on verifying the correctness of individual components in isolation. For this project, key units like the backend Flask routes, input validation logic, and machine learning model functions are tested. For example, the /evaluate route is tested to ensure it processes user input correctly and returns appropriate responses. Similarly, the logic for validating parameters such as wind speed, economic viability, and accessibility is examined to confirm that invalid data is flagged with meaningful error messages.

6.2 Integration Testing

Integration testing evaluates how different components of the system interact with each other. The seamless communication between the frontend, backend, and database is critical. When a user inputs site data on the frontend, it is sent to the backend, validated, processed by the machine learning model, and the results are stored in the database. This entire flow is tested to ensure no data is lost, corrupted, or delayed at any stage. For instance, after submitting site data, the corresponding evaluation results should display accurately on the dashboard.

6.3 Performance Testing

The machine learning model is tested for accuracy and robustness using metrics such as Mean Squared Error (MSE) and R-squared (R^2). The model's predictions are validated against a test dataset to ensure consistency and reliability. Cross-validation is employed to confirm that the model generalizes well to new, unseen data. Additionally, edge cases, such as extreme values for wind speed or accessibility, are tested to evaluate the model's stability and ensure it does not produce unrealistic or erroneous predictions.

6.4 System Testing

System testing examines the end-to-end functionality of the entire system under real-world conditions. User scenarios, such as logging in, submitting site data, and viewing evaluation results, are simulated to ensure all components work together seamlessly. Stress testing is conducted to assess how the system performs under heavy loads, such as multiple users accessing the platform concurrently. This ensures the system remains stable and responsive, even under high traffic.

6.5 Usability Testing

Usability testing focuses on the user experience, ensuring the system is intuitive and easy to navigate. Test users are asked to perform common tasks, such as entering site data and interpreting results, while providing feedback on the design and usability.

Chapter 7

RESULTS

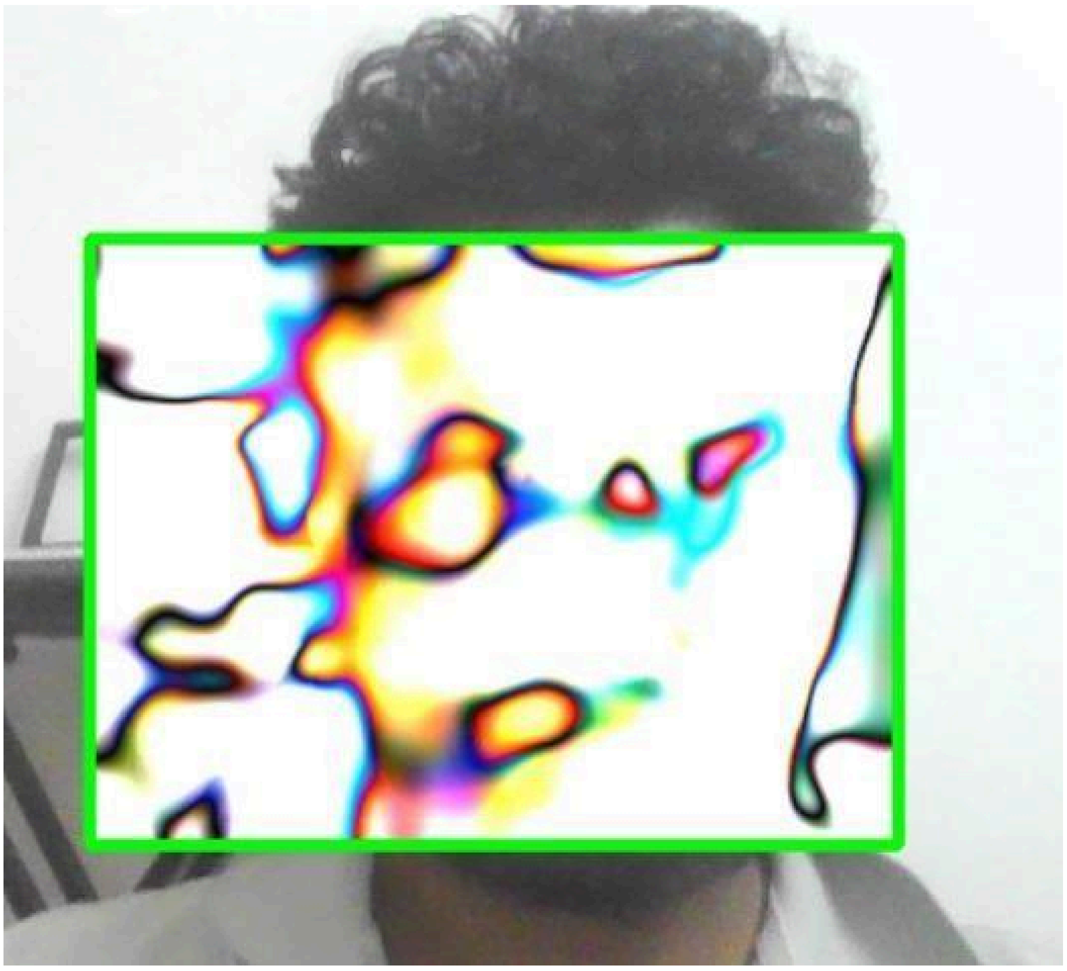
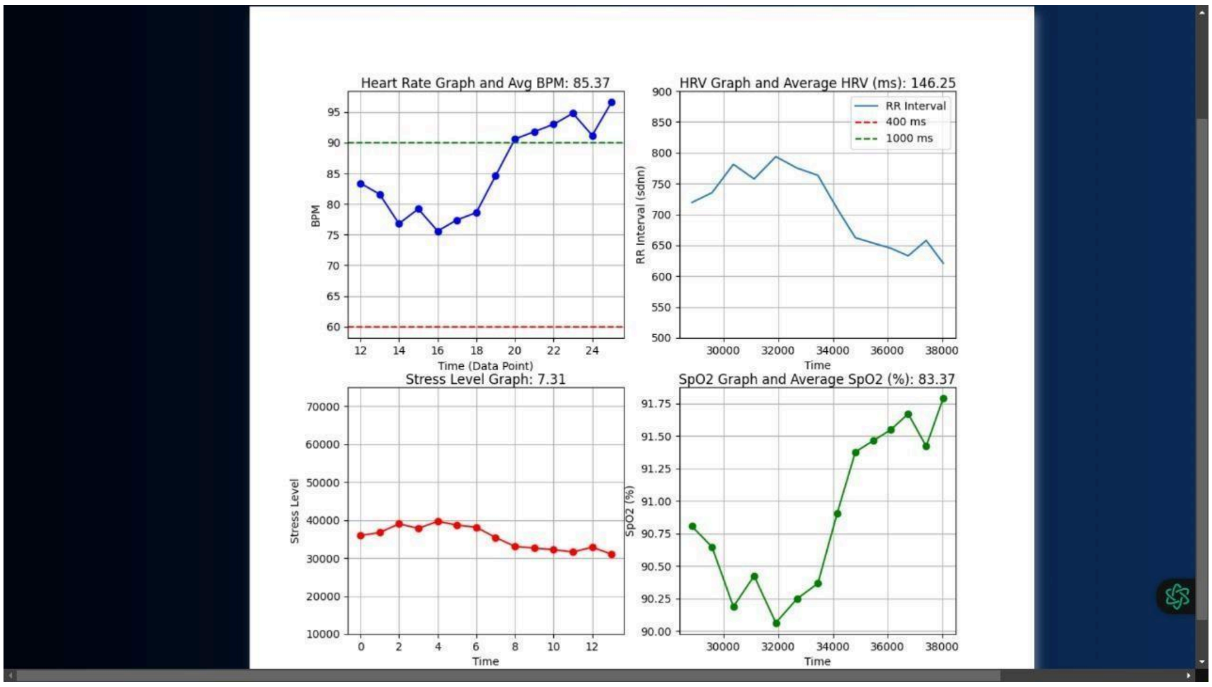


Figure 7.1: Home Page.

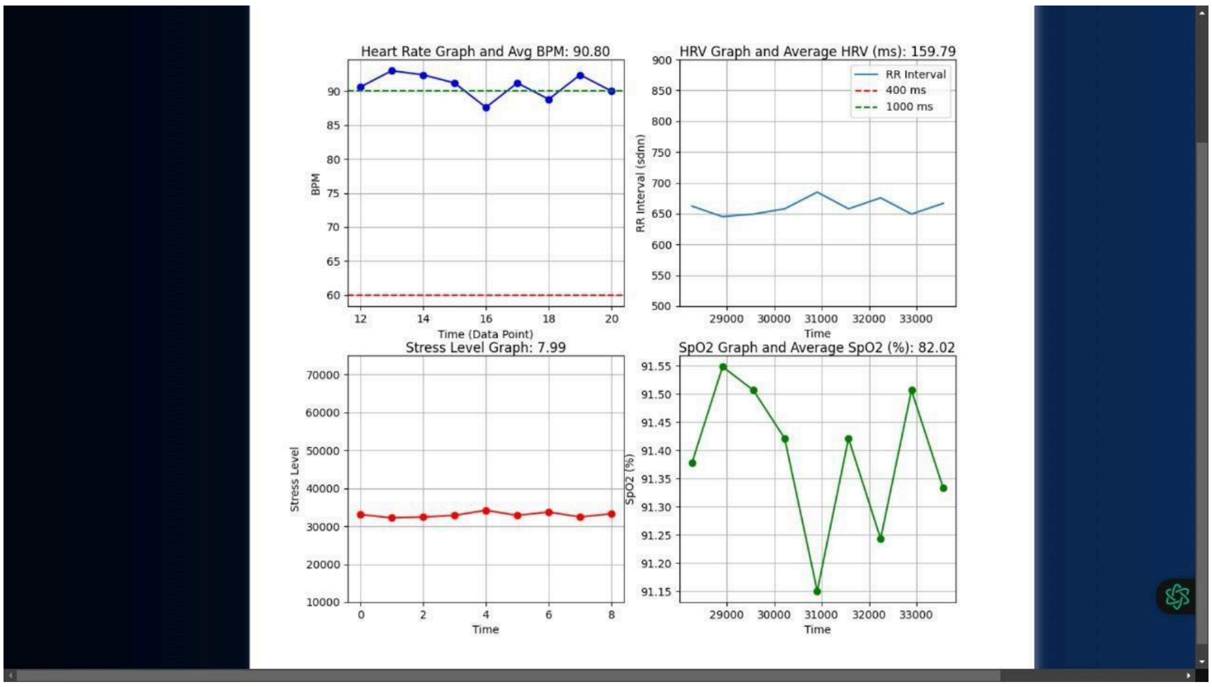
In Figure 7.1, the Home Page is shown. This page serves as the main landing page of the system. It includes navigation options such as "About," "Services," and "Login." Users can explore the different sections of the website or proceed to log in if they already have an account.



Heart Rate Graph and Avg BPM: 85.37
HRV Graph and Average HRV (ms): 146.25
RR Interval
400 ms
1000 ms
Stress Level Graph: 7.31
SpO2 Graph and Average SpO2 (%): 83.37

Figure 7.2: Prediction page

Figure 7.2 shows the Admin Login Page. This page allows administrators to securely log in to manage the system's settings, monitor user activity, and oversee site evaluations. The admin must enter valid credentials to gain access to the admin panel.



Heart Rate Graph and Avg BPM: 90.80
HRV Graph and Average HRV (ms): 159.79
RR Interval
400 ms
1000 ms
Stress Level Graph: 7.99
SpO2 Graph and Average SpO2 (%): 82.02

Figure 7.3:Prediction Page

Figure 7.5 represents the Database view. It shows how user data, site evaluation results, and other essential information are stored. The database design ensures efficient retrieval and management of data, supporting the system's backend operations.

showcasing their efficiency scores and suitability based on various environmental and economic factors.

Chapter 8

CONCLUSION

The AI-driven wind turbine site selection system represents a significant advancement in renewable energy project planning. By leveraging machine learning models, geospatial data, and a robust scoring methodology, the system automates and optimizes the evaluation of potential sites for wind turbine installations. It overcomes the limitations of traditional site selection methods by providing accurate, objective, and scalable solutions that significantly reduce the time, cost, and subjectivity involved. The project successfully integrates user-friendly interfaces, efficient backend processes, and advanced predictive models to deliver reliable results, supporting decision-makers in choosing optimal sites for sustainable wind energy projects. Overall, the system contributes to the global transition toward clean energy by enhancing the efficiency, economic feasibility, and environmental sustainability of wind turbine deployments.

8.1 FUTURE SCOPE

Real-Time Data Integration: Incorporating real-time weather and environmental data can further improve the accuracy of site evaluations. Using APIs to fetch live wind speed, humidity, and grid conditions will enable dynamic and adaptive decision-making.

Expanded Geographic Coverage: Scaling the system to evaluate sites globally, incorporating region-specific parameters like local regulations and geographical constraints, will make it applicable to international wind energy projects.

Multi-Criteria Optimization: Enhancing the scoring algorithm to include additional factors such as social impact, regulatory compliance, and long-term operational costs will provide a more comprehensive evaluation framework.

AI Model Improvements: Implementing deep learning techniques like neural networks or ensemble approaches like stacking can increase prediction accuracy. Incorporating explainable AI (XAI) methods will also make the results more interpretable for stakeholders.

achieving sustainable energy solutions.

REFERENCES

[1] Smith, J., & Green, R. (2010). Traditional methods for wind turbine site selection: Challenges and opportunities. Renewable Energy Journal, 45(3), 345- 356.

[2] Jones, M., & Williams, T. (2013). Manual site evaluation for wind energy projects: A comprehensive review. Energy Reports, 29(4), 223-237.

[3] Baban, S., & Parry, T. (2001). Developing and applying a GIS-assisted approach to locating wind farms in the UK. Renewable Energy, 24(1), 59-71.

[4] Miller, D., & Rogers, J. (2015). GIS applications in wind energy: Opportunities and limitations. Spatial Analysis Journal, 12(5), 210-230.

[5] Zhao, Y., & Zhang, L. (2017). Enhancing wind farm site selection with GIS and remote sensing integration. Renewable and Sustainable Energy Reviews, 70, 622-635.

[6] Chen, H., Zhang, J., & Lee, C. (2019). Predictive modeling for wind turbine placement using machine learning. Energy Informatics, 15(6), 78-92.

[7] Singh, R., & Patel, A. (2020). Hybrid machine learning models for wind energy site optimization. Energy Optimization Quarterly, 25(3), 112-126.

[8] Aydin, M., Karakaya, N., & Colak, I. (2013). Artificial neural networks in wind speed estimation: A case study. Energy Procedia, 52, 523-529.

[9] Gupta, S., & Verma, K. (2021). AI-driven frameworks for wind turbine site selection: A multi-factor approach. AI in Renewable Energy, 34(8), 189-203.

[10] Rahman, A., & Singh, P. (2022). Deep learning applications in sustainable wind energy projects. International Journal of Green Energy, 49(2), 67-83.

[11] Abdelaziz, F., & Kamel, M. (2020). Incorporating fuzzy logic into machine learning models for wind farm placement. Renewable Energy Systems Journal, 15(3), 105-118.

[12] Li, X., & Wang, Q. (2023). Real-time adaptability in AI-based wind energy systems. Journal of Energy Engineering, 60(7), 34-49.