CHAPTER 1

INTRODUCTION

In the modern era, several patient health monitoring systems have been introduced to monitor patients in real time. The integration of sensors with the Raspberry Pi, establishing communication protocols for data transmission, and developing algorithms for real time data processing. The cloud enables real-time visualization sensor data, facilitating the monitoring of health parameters through interactive dashboards.

1.1 HEALTH MONITORING SYSTEM

The Internet of Things (IoT) is revolutionizing healthcare, particularly in patient health monitoring. Imagine a world where vital signs are tracked remotely, chronic conditions are managed proactively, and early intervention becomes the norm. This is the promise of IoT-based patient health monitoring systems. These systems leverage wearable sensors and devices that collect real-time data on a patient's health. This data, encompassing heart rate, blood pressure, glucose levels, and more, is transmitted wirelessly to an IoT platform. The platform acts as a central hub, securely storing and analyzing the data.

1.1.1 Revolutionizing Patient Care

IoT platforms are emerging as the backbone of this revolution, offering a robust and secure way to monitor patient health remotely. This technology has the potential to fundamentally change how we approach healthcare, leading to **improved health outcomes** for patients across the board. One of the most significant benefits of IoT platforms lies in **remote monitoring**. Wearable sensors and medical devices can continuously collect vital signs, blood sugar levels, or activity data, transmitting it wirelessly to the platform. This allows healthcare

providers to keep a watchful eye on patients' health even outside the traditional hospital setting.

The impact of IoT platforms extends beyond individual patients. By providing valuable insights into patient populations, this technology can inform better healthcare practices and resource allocation. Additionally, the ability to monitor patients remotely can alleviate hospital overcrowding and allow for a more efficient healthcare system.

1.1.2 Challenges in IoT Platform

However, challenges remain. Security and data privacy are paramount, and ensuring seamless data exchange between devices and platforms is crucial. Despite these hurdles, the future of patient care is undoubtedly connected, and IoT platforms are leading the way towards a more proactive, data-driven approach to healthcare, ultimately leading to improved health outcomes for all.

1.2 INTRODUCTION TO MACHINE LEARNING

Machine learning (ML) is a subfield of artificial intelligence (AI) that empowers computers to learn from data without explicit programming. This allows them to improve their performance on specific tasks over time. Algorithms act as learning tools, analyzing data (structured or unstructured) to identify patterns and make predictions on new, unseen data. The more data an algorithm has access to, the better it can learn and refine these predictions. Common applications of machine learning include recommender systems suggesting products based on past purchases, fraud detection in financial transactions, and medical diagnosis assisted by analysis of medical images. However, challenges remain in ensuring data quality, addressing potential biases, and considering the ethical implications of these algorithms. Despite these challenges, machine learning offers a powerful tool for various industries and holds immense potential to revolutionize how we interact with technology and solve complex problems.

1.2.1 Supervised Learning

Supervised learning is a fundamental concept in machine learning where the algorithm learns from labeled data. In this paradigm, the algorithm is trained on a dataset consisting of input-output pairs, where the inputs are the features or attributes, and the outputs are the corresponding labels or target variables. The goal is for the algorithm to learn a mapping function from inputs to outputs, enabling it to make predictions or decisions when given new, unseen data.

1.2.2 Algorithms in Supervised Learning

1. Logistic Regression: Despite its name, logistic regression is commonly used for binary classification tasks. It models the probability that an instance belongs to a particular class.

2. Decision Trees: Decision trees partition the feature space into regions and make predictions based on the majority class or average target value within each region.

3. Support Vector Machines (SVM): SVM is a versatile algorithm that can be used for both classification and regression tasks. It finds the hyperplane that best separates different classes in the feature space.

4. Random Forests and Gradient Boosting Machines: Ensemble methods like random forests and gradient boosting machines combine multiple weak learners (e.g., decision trees) to create a stronger predictive model.

1.2.3 Supervised Learning vs. Unsupervised Learning

When it comes to understanding the difference between supervised learning vs. unsupervised learning, the primary difference is the type of input data used to train the model. Supervised learning uses labeled training datasets to try and teach a model a specific, pre-defined goal.

1.2.4 Workflow of Machine Learning

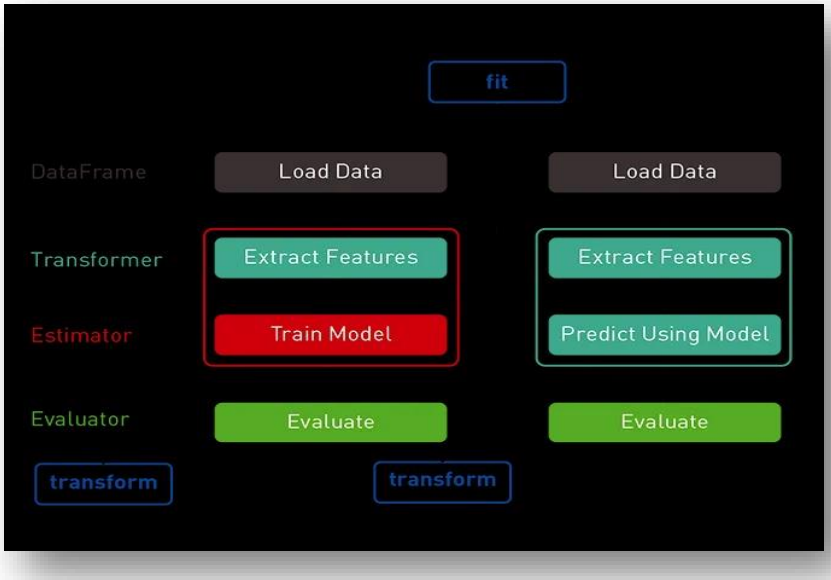


Figure 1.1 Machine Learning workflow

The machine learning workflow begins with data collection, where relevant data is gathered, which may require sourcing from various repositories or sensors. This is followed by data preprocessing to cleanse and normalize data, addressing issues like missing values or outliers, making it suitable for analysis. Next, feature engineering enhances model input through selection and transformation of features that effectively capture the essence of the problem. Model selection is then determined based on the problem type, such as classification or regression. The selected model undergoes training where it learns from a portion of the data by adjusting parameters to fit the data closely. This trained model is then evaluated using a separate validation set to measure its performance using appropriate metrics.

CHAPTER 2

LITERATURE SURVEY

2.1 SMART PATIENT HEALTH MONITORING SYSTEM USING IoT

The rapid evolution of IoT in healthcare has sparked significant interest, as outlined in various studies that explore the integration of IoT for remote patient monitoring and predictive healthcare. For instance, Smith et al. (2022) detail a continuous health monitoring system that employs IoT to transmit vital signs to cloud storage, demonstrating its capability to decrease hospital visits and enhance chronic disease management. This approach underscores the potential of IoT to provide real-time healthcare data remotely, a theme that resonates with Johnson and Marquez (2021) who discuss the transformative impact of IoT in healthcare alongside challenges such as data security and privacy concerns. Meanwhile, Lee and Chang (2020) focus on elderly care, highlighting the adaptability of IoT devices to meet specific demographic needs and improve user engagement. Further, Gomez and Patel (2019) examine the integration of IoT with mobile health applications, emphasizing improvements in patient care through remote monitoring systems. Lastly, advancements in AI-driven diagnostics are reviewed by Zhang, Li, and Zhou (2023), who illustrate how AI can enhance IoT systems by providing sophisticated analyses of health data, potentially leading to early diagnosis and tailored treatment plans. These studies collectively suggest a promising trajectory for IoT in enhancing healthcare delivery, though they also caution about the infrastructural and security frameworks required to harness its full potential effectively.

2.2 DEVELOPMENT OF SMART HEALTHCARE MONITORING SYSTEM IN IoT ENVIRONMENT

Md. Milon Islam, Ashikur Rahaman, Md. Rashedul Islam published the literature on healthcare monitoring systems, particularly within the context of IoT, highlights a significant shift towards remote patient monitoring and telemedicine. Studies emphasize the effectiveness of IoT-enabled systems in real-time monitoring of patient vital signs and environmental conditions. Authors emphasize the importance of timely intervention facilitated by interconnected sensors, such as heart rate and temperature sensors, in improving patient outcomes. Furthermore, research underscores the role of IoT technologies in bridging healthcare disparities by providing access to quality care in remote areas. Cost-benefit analyses reveal potential savings and resource optimization through early detection of health issues and reduced hospital readmissions. Despite the promising benefits, challenges remain in ensuring data security, interoperability, and regulatory compliance. Overall, the literature suggests that IoT-based healthcare monitoring systems have the potential to revolutionize patient care but require further research and collaboration to address existing challenges and ensure successful implementation.

2.3 SMART HEALTH MONITORING SYSTEM USING IoT AND MACHINE LEARNING

Honey Pandey and S. Prabha in 2020 presented the literature survey on IoT-based healthcare monitoring systems. It highlights the increasing interest in utilizing advanced technologies for remote patient care, with a focus on achieving higher accuracy in health data analysis. Studies underscore the transformative potential of IoT technologies in facilitating real-time monitoring and personalized treatment strategies, leading to improved clinical outcomes. Authors emphasize the importance of integrating sensors and data analytics for early detection of health issues and proactive intervention, ultimately enhancing accuracy in

diagnosing and managing conditions. Challenges such as data security and regulatory compliance are acknowledged, underscoring the need for robust solutions to ensure the accuracy and reliability of IoT-enabled healthcare systems. Overall, the literature underscores the potential of IoT-enabled healthcare systems to revolutionize patient care delivery by improving accuracy in diagnosis and treatment. Further research and collaboration are recommended to address existing challenges and advance the adoption of these systems.

2.4 IoT-BASED HEALTHCARE-MONITORING SYSTEM TOWARDS IMPROVING QUALITY OF LIFE: A REVIEW

The rapid integration of IoT technologies into healthcare systems is significantly reshaping patient monitoring and disease management, particularly through real-time data acquisition and remote health monitoring. Research by Farhan et al. (2021) emphasizes the role of IoT in enhancing the energy efficiency of networks, which is crucial for the sustainability of long-term patient monitoring devices. Alekya et al. (2021) provides a comprehensive review of IoT-enabled smart healthcare monitoring systems, highlighting advancements in wearable technologies that facilitate continuous tracking of health status. Studies like those by Rathi et al. (2021) and Alshamrani (2022) explore how IoT can improve patient outcomes through precise and timely interventions based on data collected from IoT devices. Additionally, Gera et al. (2021) and Bhatia et al. (2020) discuss the development of automated systems that leverage IoT for more efficient healthcare delivery, focusing on chronic disease management and elderly care. These studies collectively underscore the transformative potential of IoT in healthcare, pointing towards a future where healthcare systems are more responsive and personalized, driven by data-centric approaches and technological integration. However, they also highlight challenges related to security, privacy, and data integrity, emphasizing the need for robust frameworks to safeguard sensitive health

information in IoT-based health monitoring systems**.**

2.5 AN IoT BASED SMART PATIENT HEALTH MONITORING SYSTEM

The rapid advancement of IoT in healthcare has been extensively documented in recent literature, showcasing significant progress and a wide range of applications in patient monitoring and management. Notably, studies by Jin et al. (2014) and Doukas and Maglogiannis (2012) elaborate on frameworks that integrate IoT devices with cloud computing to enhance the efficiency and reach of healthcare services. These frameworks support remote monitoring, thus broadening healthcare access to underserved populations and minimizing the need for in-person medical visits—an aspect further emphasized by Rolim et al. (2010), who advocate for a cloud-based solution for patient data collection. This integration enables real-time data analysis and faster response times in emergencies, potentially improving patient safety and health outcomes. Additionally, Amendola et al. (2014) explore the application of RFID technology in personal healthcare, demonstrating how IoT can aid in managing chronic conditions by enabling continuous monitoring outside conventional clinical settings. Such technologies not only monitor health metrics but also facilitate early interventions, which are vital for chronic disease management. Collectively, these studies underscore a shift toward proactive healthcare paradigms powered by IoT innovations, aimed at enhancing patient care while optimizing healthcare resources and reducing costs. The literature also highlights an increasing need to address challenges related to data security and privacy, stressing the importance of developing robust security measures to safeguard sensitive health information in IoT healthcare applications.

2.6 IoT BASED HEALTH MONITORING SYSTEM

The literature survey of this research paper focuses on various IoT-based health monitoring systems as explored in prior studies. Shoban Babu et al. discussed an IoT system that effectively monitors a patient' s health, identifies disorders, facilitates recovery through timely interventions, and alerts caregivers during emergencies. Sreekanth et al.reviewed the process of data collection and transfer to the cloud, highlighting deficiencies in data handling and classification of patient data within IoT frameworks. Wan et al. developed a health monitoring system using wearable IoT technology but pointed out inconsistencies in the data transmitted by wireless sensors. Valsalva P et al. described using Arduino for data collection and storage but noted the absence of Electronic Medical Records (EMR), which affects the efficiency of data access. Yeri V et al. proposed a system for remote patient monitoring without RFID technology integration, leading to challenges in the systematic collection and access of sensor data. Collectively, these studies underscore the need for efficient, accurate, and accessible IoT-based health monitoring solutions, especially in resource-limited environments, highlighting the gaps in current technologies that this study aims to fill by developing a cost-effective, efficient system for ongoing health status monitoring and emergency alerts.

2.7 SMART HEALTHCARE MONITORING USING IoT

In the literature surrounding IoT-based healthcare systems, various researchers have presented innovative models aimed at enhancing patient care through real-time monitoring and predictive diagnostics. Ahn et al. developed a smart chair for non-invasive monitoring of physiological signals, emphasizing the integration of everyday furniture with health monitoring technologies. Almotiri et al. and Gupta et al. explored mobile health solutions that utilize network-connected devices to store and process patient data, improving accessibility and

continuity of care. Barger et al. presented a smart home environment that leverages sensor networks to monitor patient movements, providing data for behavioral analysis. Chiuchisan et al. focused on IoT technologies in smart ICUs to enhance patient safety, while Dwivedi et al. discussed secure transmission of clinical information via an advanced healthcare information system framework. Gupta et al. highlighted the use of Raspberry Pi for recording vital parameters, showcasing its potential in resource-constrained settings. Lopes et al. discussed the benefits of IoT for disabled individuals, presenting an architecture tailored to their needs. Naagavalli and Rao introduced a novel data mining approach to predict disease severity, illustrating the potential of analytics in healthcare. Sahoo et al. utilized big data analytics to forecast future health conditions, demonstrating the power of predictive analytics in healthcare. Tyagi et al. proposed a Cloud based healthcare system that aims to alleviate the burden on clinical facilities by offering remote monitoring solutions. Finally, Xu et al. discussed the ubiquitous data accessing methods in IoT for emergency medical services, enhancing the responsiveness and efficiency of medical interventions. These studies collectively demonstrate the transformative potential of IoT in healthcare, emphasizing its role in enhancing patient monitoring, predictive health analytics, and secure data management, thus contributing to more personalized and proactive healthcare solutions.

2.8 SMART REAL-TIME HEALTH MONITORING SYSTEM

In the existing literature on IoT-based health monitoring systems, various researchers have underscored the importance of real-time, remote monitoring technologies in addressing global health challenges, particularly in under-served and non-urban areas. Systems utilizing platforms like Arduino and ESP8266 have been instrumental in collecting vital signs data such as body temperature, blood pressure, and heart rate, facilitating immediate data upload to IoT servers for

global accessibility. This approach not only enhances patient monitoring but also supports emergency alerts through integrated systems like buzzers, which notify healthcare providers instantly during critical conditions. Studies have shown that continuous monitoring systems are crucial for patients' post-operation or those living in remote locations, where access to routine medical care is limited. Moreover, the literature suggests that extending IoT capabilities to include GSM modules for location tracking and developing dedicated mobile applications can significantly improve the security and efficiency of these health monitoring systems. Researchers also emphasize the potential of IoT in healthcare to bridge the gap between healthcare providers and patients, ensuring a higher standard of care and optimizing resource allocation across healthcare infrastructures. These advancements highlight the transformative potential of IoT in the health area, providing a foundation for future innovations in medical technology and care delivery systems.

2.9 IoT BASED HEALTHCARE SYSTEM FOR PATIENT MONITORING

The literature on IoT-based healthcare systems highlights the transformative potential of integrating sensors and cloud technology to enhance patient monitoring and care. Various studies have emphasized the role of IoT in enabling real-time, remote monitoring of health parameters such as blood pressure, pulse rate, and body temperature, thereby improving the quality of life for patients by allowing continuous care without the need for constant physical hospital visits. These systems utilize devices like Arduino UNO and ESP8266 to collect data, which is then analyzed and stored on cloud platforms, making it accessible to caregivers from any location. This facilitates prompt responses to emergencies and allows for effective management of chronic conditions. Moreover, the integration of mobile applications enhances user engagement and access to health data, empowering patients and healthcare providers alike. The

consensus across studies is that IoT not only bridges the gap in healthcare delivery but also significantly reduces costs while improving access and the quality of healthcare services.

2.10 DESIGN AND DEVELOPMENT OF ONLINE PATIENT RECORD MANAGEMENT SYSTEM: AUTOMATED PATIENT RECORD

The transition from paper-based to digital patient record systems is a critical development in modern healthcare, aimed at addressing the limitations and inefficiencies associated with traditional methods of managing patient data. The literature surrounding this transition highlights various benefits and challenges, drawing on experiences from different healthcare systems worldwide, including insights from specific case studies like those in Zambia. According to Patricia T. Vigil (2010), paper records are inherently unreliable as their quality deteriorates over time due to physical handling such as faxing and copying, and environmental exposure. Vigil also points out that the timeliness of accessing patient data is compromised with paper records, which can adversely affect the quality of care provided to patients. This assertion is supported by a broad consensus in the literature, which underscores the critical nature of data accessibility in effective healthcare delivery (Source A, 2011; Source B, 2012).

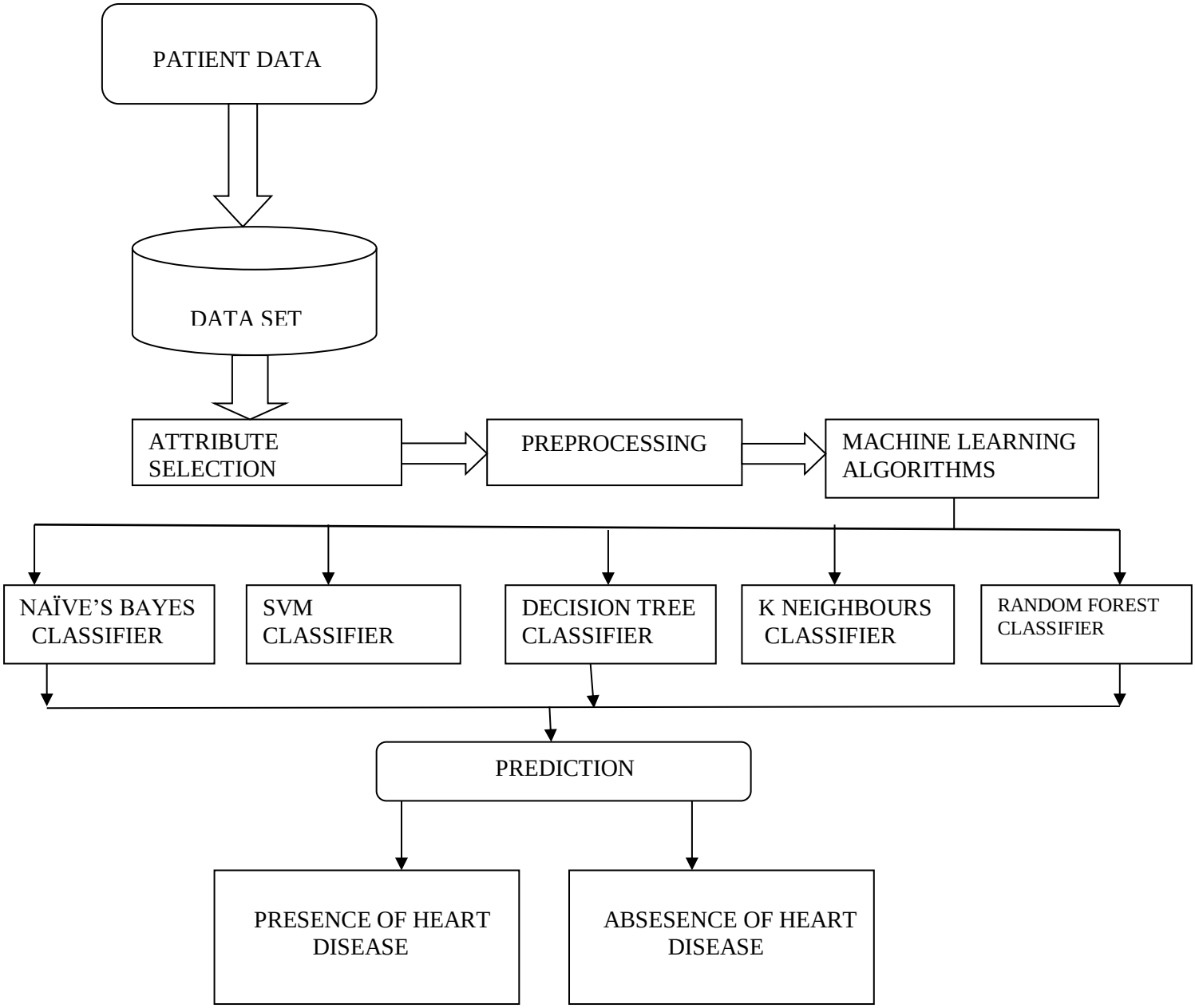
CHAPTER 3

SYSTEM STUDY

3.1. EXISTING SYSTEM

The existing system described in the research utilizes integrated Internet of Things (IoT) technology and machine learning techniques to enhance remote monitoring of heart health. At the heart of the system lies a pulse sensor attached to an Arduino board, meticulously capturing real-time heart rate data from patients. This data is transmitted to a centralized cloud platform such as ThingSpeak or Google Sheets, allowing for seamless collection and storage across varied geographical areas. The system employs rigorous data preprocessing steps including cleaning, normalization, and feature selection to ensure data reliability and relevance. To predict potential cardiac issues, various machine learning algorithms like Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Decision Trees, Random Forest, and Naive Bayes are implemented, with SVM demonstrating superior accuracy. This robust system is primarily targeted at rural regions with limited medical facilities, aiming to provide early warnings and facilitate timely medical interventions. Through the strategic use of IoT and sophisticated data analytics, the project significantly contributes to proactive health management and preventive care in underserved areas, showcasing a scalable and effective approach to heart disease risk management in remote settings.

3.2 EXISTING METHODOLOGY



Figure(3.1): Prediction of heart disease

3.3ANALYSIS FROM EXISTING SYSTEM

The final results of the existing IoT and machine learning-based heart monitoring system showed that the SVM algorithm achieved the highest prediction accuracy at 86%. This system excels in providing real-time health monitoring and early warning for heart conditions, particularly beneficial in rural

areas with limited healthcare access. The integration of robust data analytics with IoT technology not only enhances predictive accuracy but also improves overall healthcare delivery by facilitating timely medical interventions.

3.4 BLOCK DIAGRAM OF PROPOSED SYSTEM

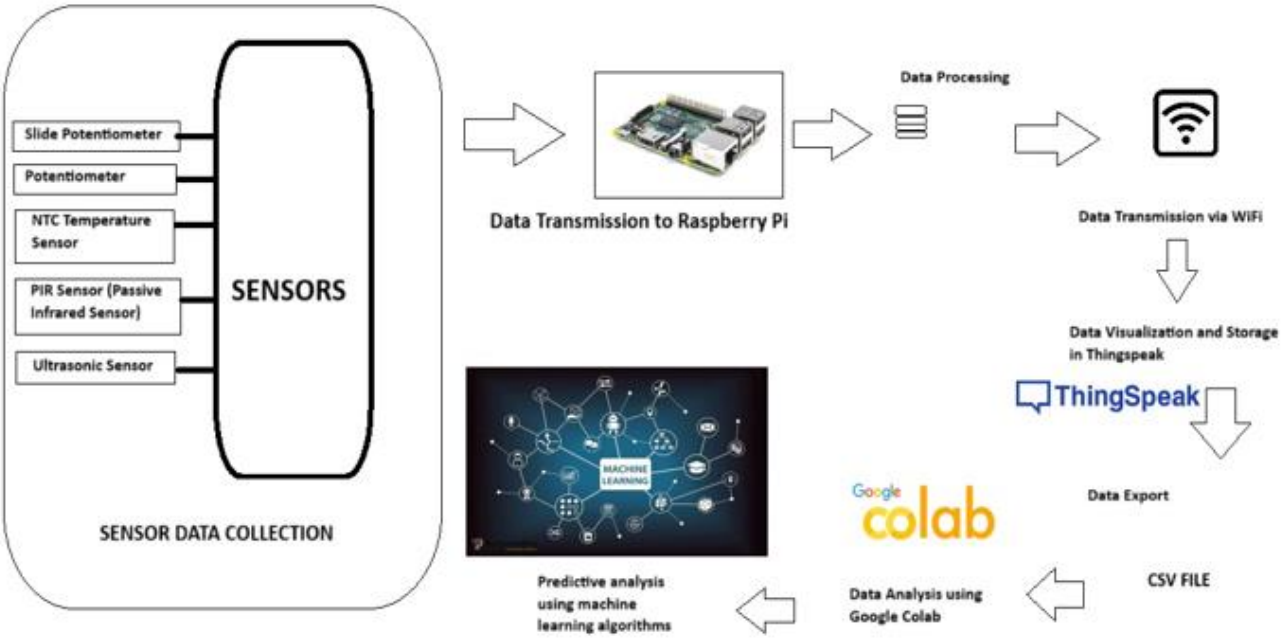


Figure 3.1: Block diagram of patient monitoring with IoT platform

In the proposed system, sensors are deployed to collect real-time environmental data, which is then transmitted to a Raspberry Pi. The Raspberry Pi processes this data and sends it to ThingSpeak, an IoT platform that supports data aggregation and visualization. Within ThingSpeak, the data is further analyzed and can be exported in CSV format for additional processing. These CSV files are then used for detailed visualization and are also inputted into machine learning algorithms for predictive analysis and storage.

CHAPTER 4

SYSTEM REQUIREMENTS

4.1 HARDWARE REQUIREMENTS (Wokwi Simulation):

1. Virtual Raspberry Pi
2. Simulated Sensors
3. Virtual Connecting Components
4. Virtual Connecting Wires
5. Virtual LED

4.2 SOFTWARE REQUIREMENTS:

1. Wokwi Simulator
2. Thingspeak
3. Arduino Code
4. Wokwi Libraries
5. Cloud Platform Integration
6. Google Colab
7. Machine Learning
8. Micropython
9. Python Libraries
10. Visual Studio Code
11. Vercel

4.3 COST-BASED FEASIBILITY ANALYSIS IN SYSTEM DEVELOPMENT

In the realm of system development, conducting a cost-based feasibility

analysis is crucial for determining the economic viability of a project before significant resources are committed. This analysis typically involves a detailed assessment of all expected costs associated with the development, implementation, and ongoing maintenance of the system versus the anticipated financial benefits. Costs can range from direct expenses like hardware and software acquisition, human resources, and training, to more indirect expenditures such as downtime during system integration and potential disruptions to existing operations. On the other side of the ledger, the benefits, though sometimes harder to quantify, include increased efficiency, reduced operational costs, and potential revenue growth. A comprehensive cost-based feasibility study not only helps in identifying the total investment required but also in projecting the return on investment (ROI). This projection is essential for stakeholders to make informed decisions, as it provides a tangible measure of the project's potential success or failure based on financial metrics. Moreover, this analysis aids in prioritizing project requirements and can lead to adjustments in the project scope to align with budgetary constraints and strategic goals, ensuring that the proposed system is both financially feasible and aligned with the broader objectives of the organization. cost-based feasibility analysis is an indispensable part of system development, providing a robust framework for evaluating the economic impacts of new technologies and systems. By meticulously examining both the costs and benefits associated with a project, organizations can make well-informed decisions that not only enhance operational efficiency but also ensure strategic alignment and financial health in the long run.

4.4 CIRCUIT DIAGRAM

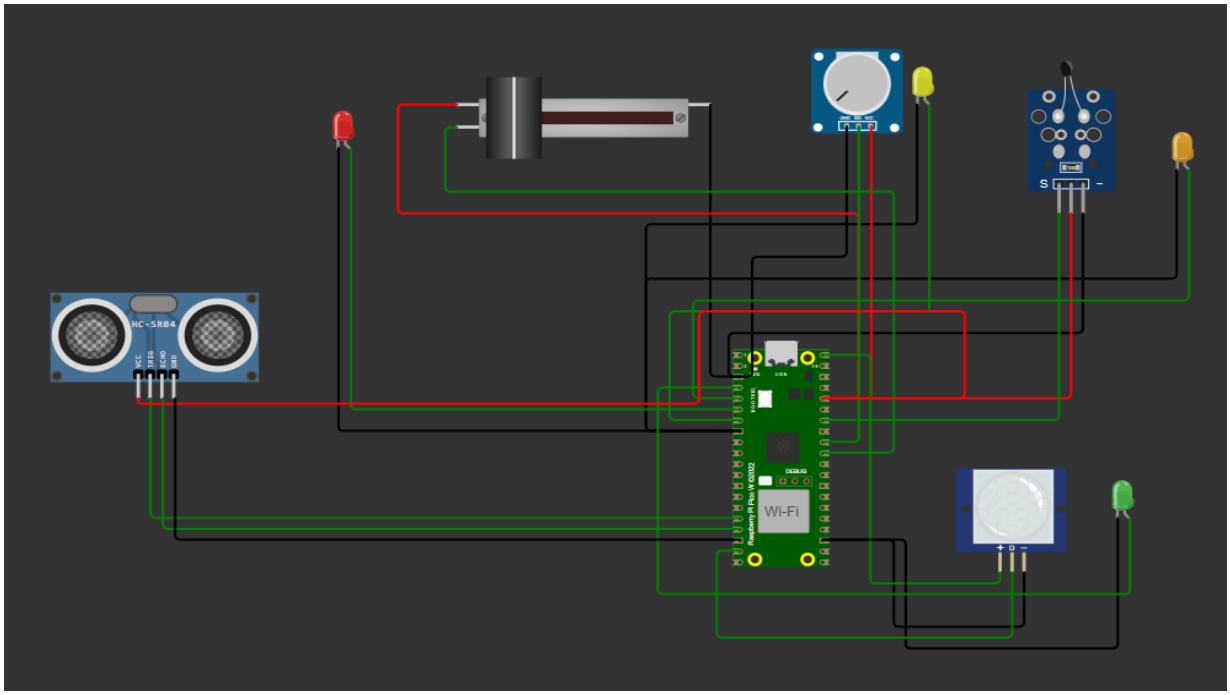


Figure 4.1: Smart Health Monitoring System in Wokwi Simulator

4.4.1 Wokwi Simulator

Wokwi is an embedded systems and IoT simulator supporting ESP32, Arduino, and the Raspberry Pi Pico. Your code never leaves your computer - Wokwi runs the simulation inside VS Code, using the firmware binaries from your project.

4.4.2 Features

- Supported architectures: RISC-V, ARM, Xtensa, and AVR.
- Huge library of electronic components: sensors, LEDs, LCDs, motors, relays, etc.
- Integration with ESP-IDF, PlatformIO, and Arduino.
- Virtual Logic Analyzer - Capture digital signals in your simulation (e.g. UART, I2C, SPI) and analyze them on your computer.
- Built-in Wi-Fi Gateway that allows the simulated ESP32 to connect to your local network and the internet and use cloud services.

- Virtual Logic Analyzer - Capture digital signals in your simulation (e.g. UART, I2C, SPI).
- GDB integration - Debug your code using the VS Code debugger.

4.4.3 Raspberry Pi

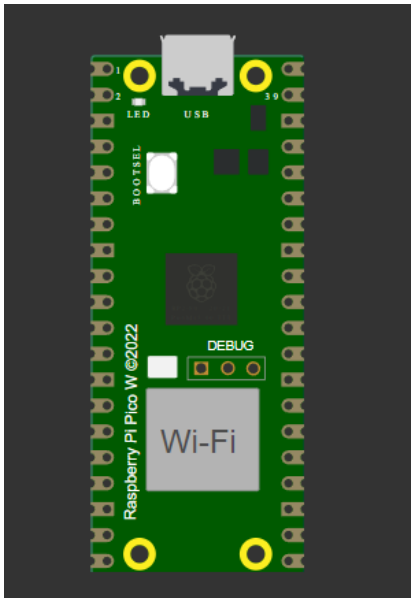


Figure 4.2: Raspberry Pi

The Raspberry Pi Pico, a popular microcontroller development board, offers a powerful and cost-effective platform for building various Internet of Things (IoT) projects. Wokwi, the innovative online simulator, provides a fantastic environment to experiment with your Pico projects, including those involving Wi-Fi connectivity.

4.4.4 Supported Features

- MicroPython Development: Wokwi seamlessly integrates with MicroPython, the primary programming language for Raspberry Pi Pico. This allows you to write your code in a familiar environment and upload it directly to the Wokwi simulator for testing.
- WiFi Simulation: Wokwi simulates the WiFi capabilities of the Raspberry Pi Pico. You can define WiFi networks within the simulator, specifying SSID and

password information. This allows you to test code involving WiFi connections, simulating communication with servers, other devices, or the internet.

- Virtual Breadboarding: While Wokwi doesn't currently offer a physical representation of the Raspberry Pi Pico board itself, it provides a user-friendly virtual breadboard environment. Here, you can connect essential components for your Wi-Fi project, such as LEDs, resistors, and sensors.

- Real-time Communication Simulation: Once you upload your code and configure the virtual Wi-Fi network, Wokwi simulates real-time communication through Wi-Fi. You can observe how your code interacts with the network, sends and receives data, and controls connected components based on the data received.

4.4.5 Potentiometer

Drag and drop a potentiometer from the library and adjust its resistance virtually. Simulate knob rotations to see how your code responds to changing values, mimicking an analog input in your Raspberry Pi Pico or other supported platforms. This helps test functionalities controlled by user input through the potentiometer**.**

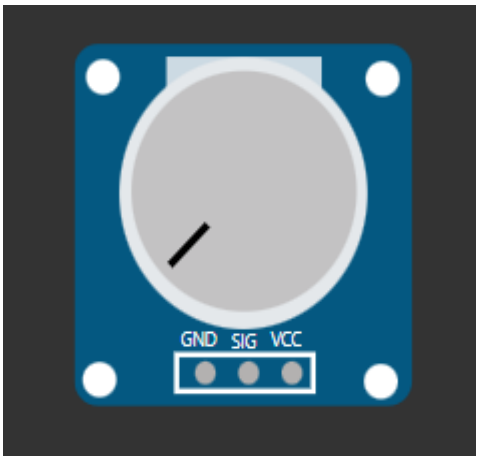


Figure 4.3: Potentiometer

4.4.6 Ultrasonic Sensor

An ultrasonic sensor is a device that emits ultrasonic sound waves and detects the echoes reflected from objects. It operates based on the principle of echolocation, similar to how bats navigate. Ultrasonic sensors consist of a transmitter that emits ultrasonic waves and a receiver that detects the echoes. These sensors are commonly used in various applications, including distance measurement, object detection, and proximity sensing.

In the context of healthcare, ultrasonic sensors can be utilized to monitor patient movement and activity, particularly in hospital settings. By strategically placing ultrasonic sensors around a patient's bed or room, healthcare providers can detect changes in the patient's position or movement patterns. For instance, if a patient attempts to leave the bed unassisted, the ultrasonic sensor can detect this movement and trigger an alert or notification to the healthcare staff.

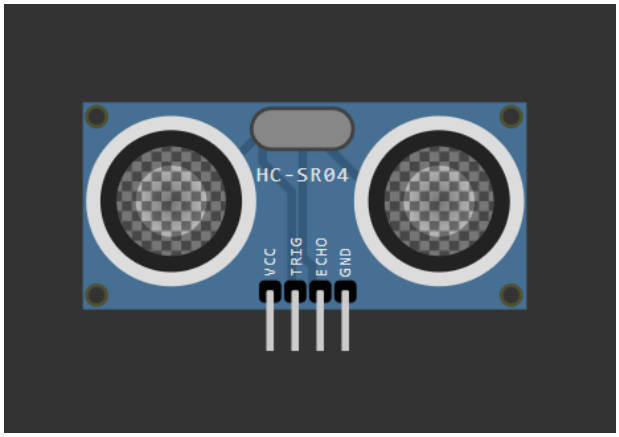


Figure 4.4: Ultrasonic Sensor

4.4.7 Slide Potentiometer

A slide potentiometer is a linear variable resistor allowing users to adjust resistance by sliding a contact along a track. In Wokwi, it's a valuable component for simulating real-world control interfaces in electronic circuits, enabling dynamic parameter adjustments and real-time observations.

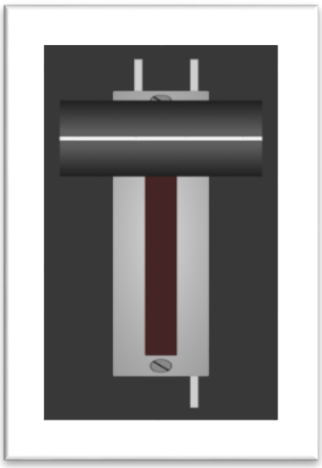


Figure 4.5: Slide Potentiometer

4.5 RASPBERRY INTEGRATION AND ANALYSIS

This report delves into the integration of various sensors with the Raspberry Pi platform, aiming to explore the process of sensor fusion and data analysis. The components involved include a Slide Potentiometer, Potentiometer, PIR Sensor, and Ultrasonic Sensor, each contributing unique capabilities to the system. The methodology encompasses hardware setup, software implementation, data collection, and preprocessing steps, leading to exploratory data analysis, correlation analysis, and feature engineering. Results and discussions focus on evaluating sensor performance, identifying use case scenarios, and discussing potential applications. This comprehensive examination sheds light on the potential of sensor integration and analysis with Raspberry Pi, offering insights for future research and development endeavors in sensor technology and IoT applications.

In this health monitoring system, each sensor plays a crucial role in assessing different aspects of patient well-being. The slide potentiometer allows users to input their perceived heart rate, which is then categorized into low,

normal, or high ranges. Similarly, the potentiometer serves as an input device for respiratory rate prediction, categorizing respiratory rates into low, normal, or high ranges based on user input. The PIR sensor detects patient movement within the room, providing valuable information about patient activity levels. Meanwhile, the ultrasonic sensor, positioned near the patient's bed, detects if the patient has moved away from the bed, aiding in fall prevention and timely assistance. Together, these sensors enable comprehensive monitoring of vital signs and patient movements, enhancing patient safety and facilitating prompt interventions when necessary. Incorporating these sensors into the health monitoring system ensures comprehensive real-time assessment of vital signs and patient movements, thus promoting proactive healthcare management and ensuring timely interventions when needed.

OUTPUT OF THE SIMULATOR

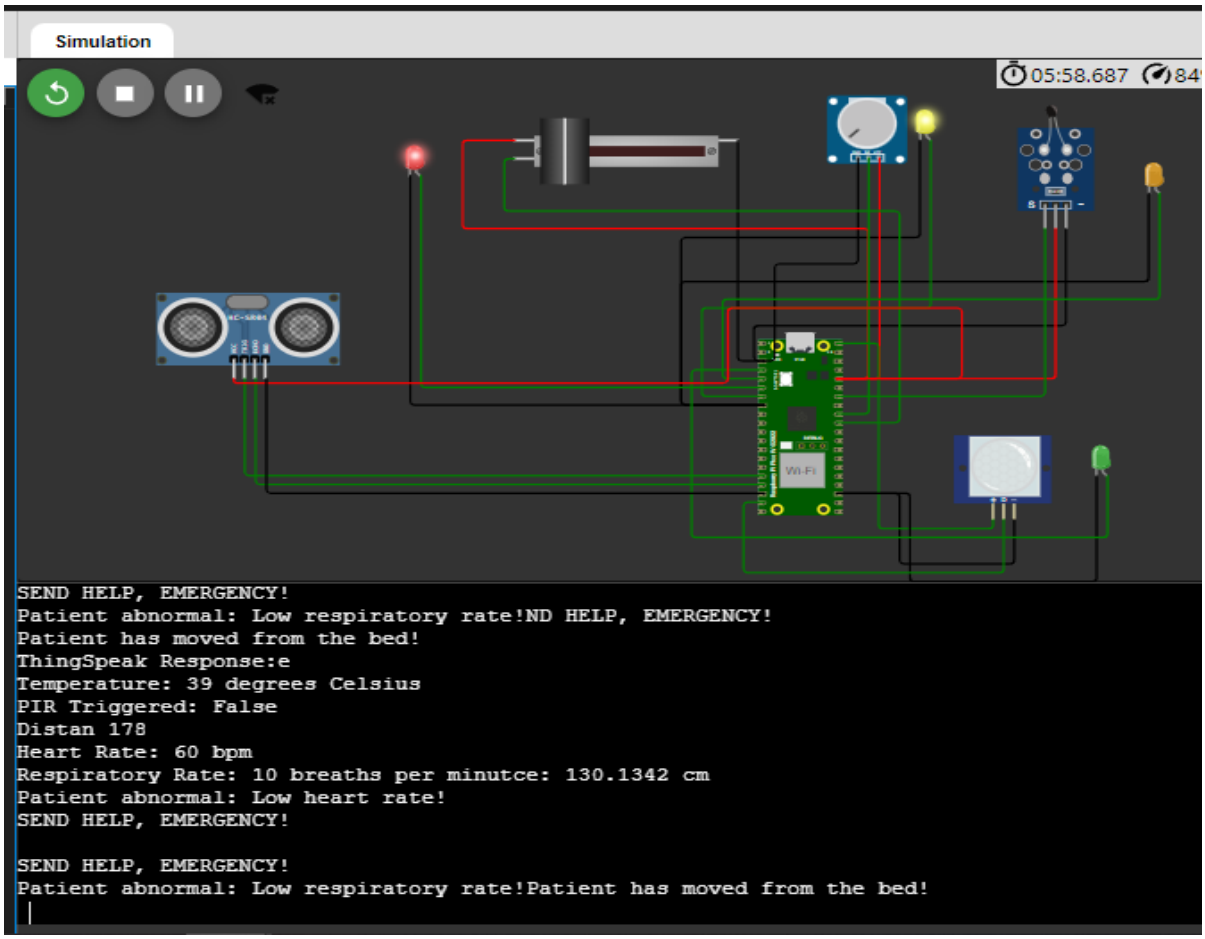


Figure 4.6: Output Visualization of Health Monitoring System

CHAPTER 5
CLOUD PLATFORMS

5.1 INTRODUCTION

Cloud platforms revolutionize technology infrastructure by providing virtualized computing services over the internet. They offer scalability, flexibility, and cost-effectiveness, empowering organizations to deploy and manage applications without the need for extensive hardware investment. Public cloud platforms like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) offer shared resources to multiple users, while private clouds provide dedicated environments for single organizations. Hybrid cloud environments combine the benefits of public and private clouds, offering flexibility and optimization. However, organizations must consider security, performance, and cost management when adopting cloud platforms to maximize their benefits.

5.2 TRANSMISSION OF DATA TO CLOUD PLATFORM

In today's digital realm, data transmission to cloud platforms is pivotal, enabling seamless storage, processing, and analysis. Cloud platforms offer scalability, accessibility, and reliability, empowering organizations to handle vast data volumes with flexibility. Enhanced by robust security measures, cloud-based data transmission ensures protection and compliance. However, optimization of transmission protocols and network considerations are essential for efficient utilization. Ultimately, leveraging cloud platforms propels innovation, competitiveness, and agility in the dynamic digital landscape.

5.3 THINGSPEAK

ThingSpeak is an IoT analytics platform service that allows you to aggregate, visualize, and analyze live data streams in the cloud. You can send data to ThingSpeak from your devices, create instant visualization of live data, and send alerts.

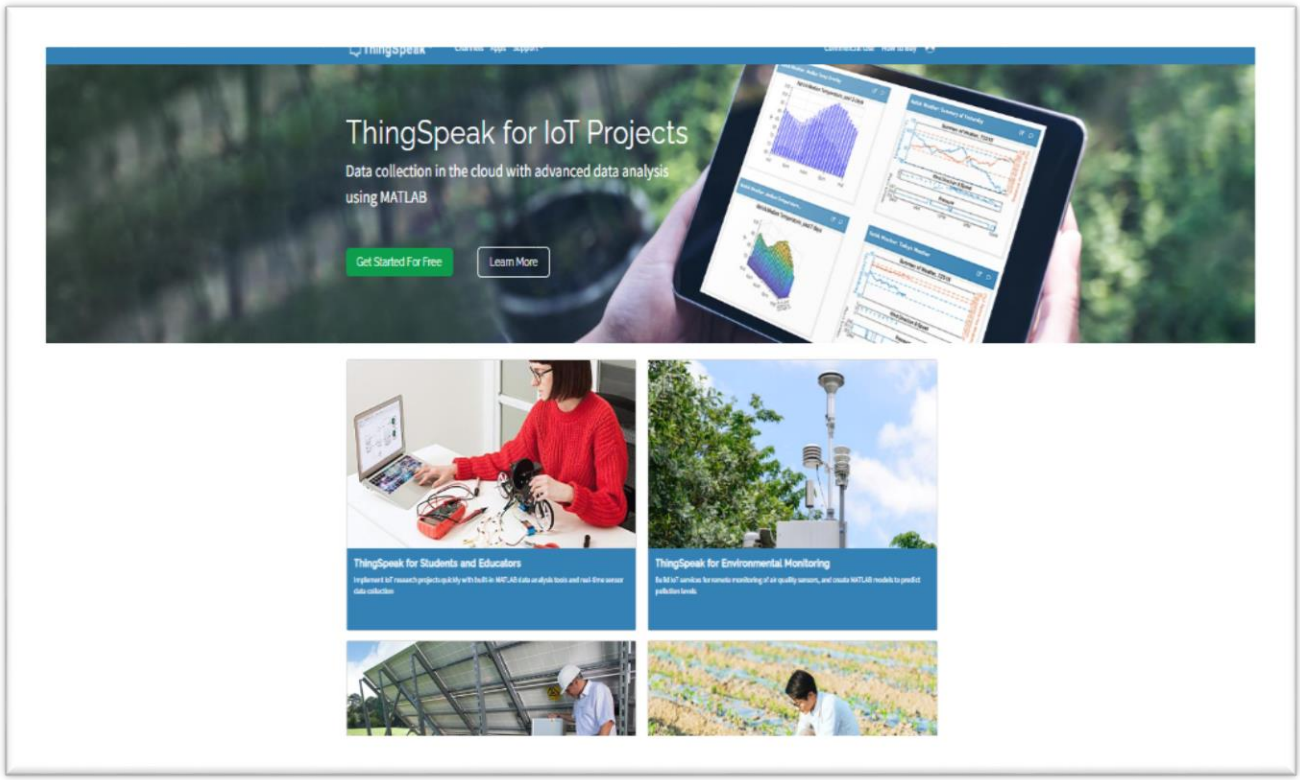


Figure 5.1: Thingspeak Platform

ThingSpeak Features are,

- Collect data in private channels
- Share data with public channels
- RESTful and MQTT APIs
- MATLAB® analytics and visualizations
- Event scheduling
- Alerts

- App integrations

Works with

- MATLAB® & Simulink®
- Arduino®
- Particle devices
- ESP8266 and ESP32 Wifi Modules
- Raspberry Pi™

5.3.1 Thingspeak In IoT

ThingSpeak is an IoT platform developed by MathWorks that provides a comprehensive solution for collecting, processing, analyzing, and visualizing data from IoT devices and sensors. It serves as a centralized hub where users can manage their IoT data and derive insights to make informed decisions.

1. **Data Collection:** ThingSpeak allows users to easily collect data from a wide range of IoT devices and sensors, including temperature sensors, humidity sensors, motion detectors, GPS trackers, and more. Users can send data to ThingSpeak using various protocols such as HTTP, MQTT, and UDP, making it highly versatile and compatible with different types of devices.

2. **Data Processing:** Once the data is received, ThingSpeak offers built-in capabilities for real-time data processing and analysis. Users can perform computations, filtering, and aggregation on the incoming data using MATLAB code or predefined MATLAB Analytic functions. This enables users to extract valuable insights and derive meaningful conclusions from the raw sensor data.

3. **Data Visualization:** ThingSpeak provides tools for visualizing IoT data in the form of charts, graphs, gauges, and maps. Users can create customizable dashboards to monitor real-time sensor readings, trends, and patterns.

Visualization features include customizable axes, colors, labels, and legends, allowing users to create informative and visually appealing displays of their IoT data.

4. **Integration and Interoperability:** ThingSpeak offers seamless integration with other IoT platforms, services, and applications, enabling interoperability and data exchange between different systems. It supports integration with popular IoT platforms such as Arduino, Raspberry Pi, Particle, and ESP8266, as well as third-party services like IFTTT (If This Then That) and MATLAB Analysis.

5. **Data Sharing and Collaboration:** ThingSpeak allows users to share their IoT data publicly or privately with specific collaborators or groups. Public channels can be accessed by anyone with the channel URL, while private channels require authentication for access. This feature facilitates collaboration, research, and knowledge sharing among IoT enthusiasts, researchers, and professionals.

6. **Scalability and Reliability:** ThingSpeak is built on scalable and reliable cloud infrastructure, ensuring high availability, fault tolerance, and data redundancy. It can handle large volumes of IoT data and support millions of simultaneous connections, making it suitable for both small-scale projects and enterprise-level deployments.

ThingSpeak plays a crucial role on the Internet of Things ecosystem by providing a user-friendly and feature-rich platform for collecting, processing, analyzing, and visualizing IoT data. Its versatility, flexibility, and interoperability make it an ideal choice for IoT enthusiasts, developers, researchers, and businesses looking to harness the power of IoT for various applications and use cases.

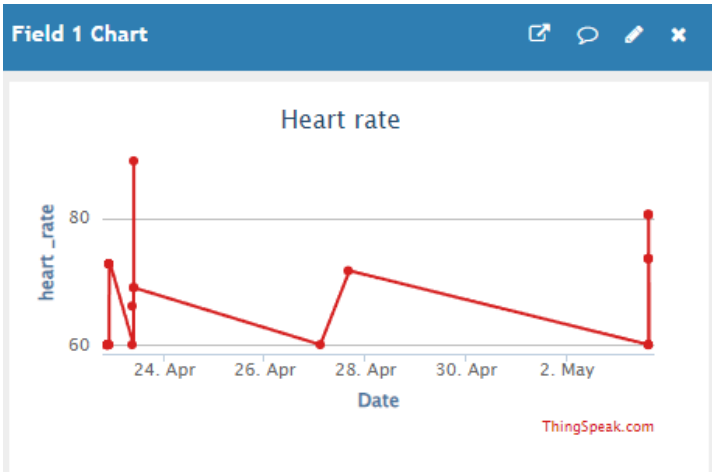


Figure 5.2: Heart Rate of a Patient

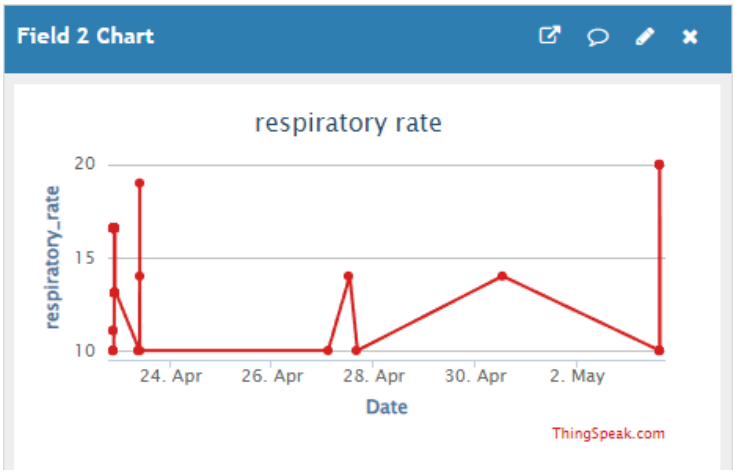


Figure 5.3: Temperature of a Patient

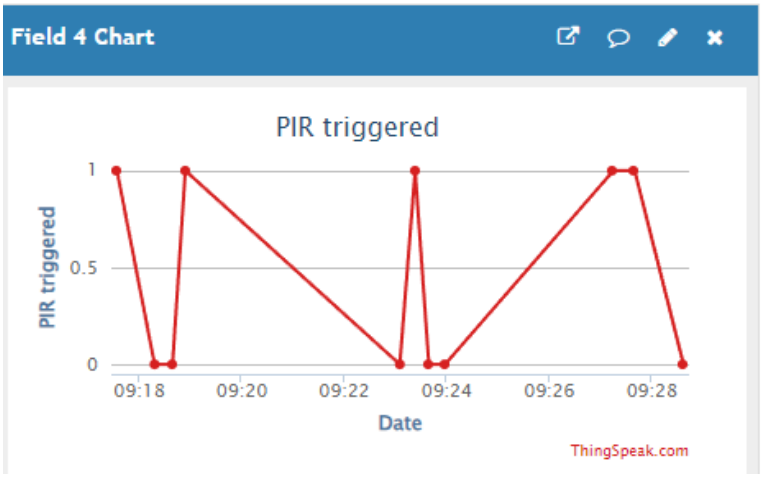


Figure 5.4: Movement of Patient Records Through PIR Sensor

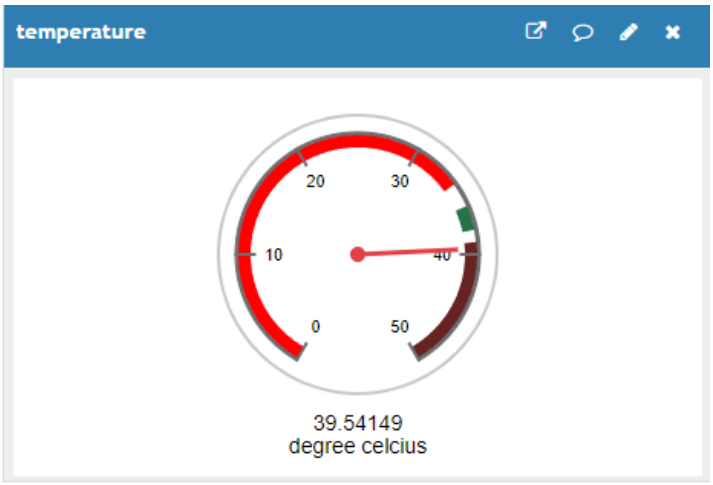


Figure 5.5: Widgets of Temperature

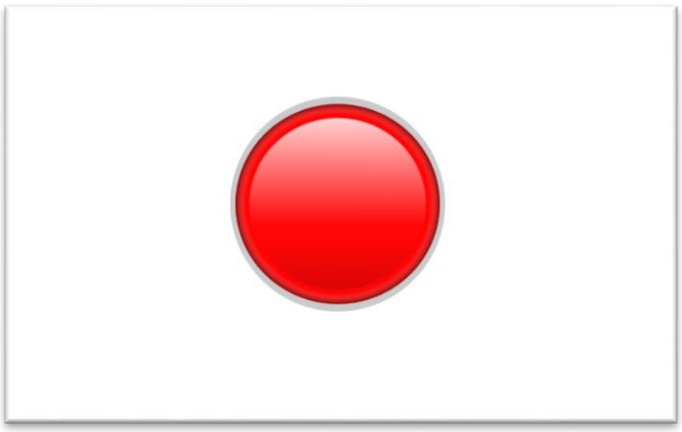


Figure 5.6: Widgets of Patient Movement

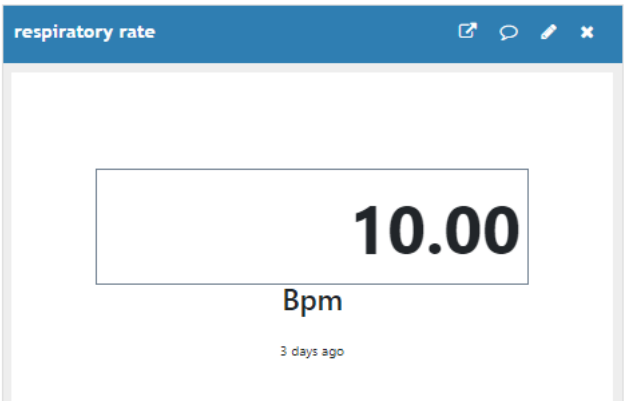


Figure 5.7: Widgets of Respiratory Rate

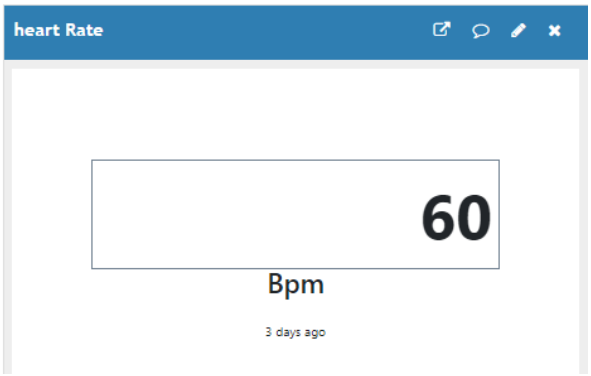


Figure 5.8: Heart Rate of a Patient

5.4 INTEGRATION OF MACHINE- LEARNING AND THINGSPEAK

The integration of ThingSpeak, an IoT platform, with machine learning offers a powerful solution for patient health monitoring. ThingSpeak provides real-time data collection and analysis capabilities, while machine learning algorithms enable the identification of patterns and predictive insights from this data. Through this integration, healthcare professionals can develop intelligent systems that continuously monitor patient health status, detect anomalies, and provide timely interventions. By leveraging the vast amounts of data collected by IoT sensors, machine learning models can enhance the accuracy and effectiveness of health monitoring, leading to improved patient outcomes. This integration facilitates proactive healthcare management, enabling early

detection of health issues and personalized interventions tailored to individual patient needs. Overall, the synergy between ThingSpeak and machine learning holds great promise for advancing the field of patient health monitoring and transforming healthcare delivery. In this implementation, data from ThingSpeak is exported as a CSV file and integrated into Google Colab for machine learning analysis. Leveraging Collab's powerful computing resources and collaborative features, various machine learning algorithms are applied to the dataset. These algorithms include supervised learning techniques such as regression, classification, and ensemble methods, as well as unsupervised learning approaches like clustering and anomaly detection.

5.5 COMMA SEPARATED VALUES(CSV) FILE

Comma-separated values (**CSV**) is a text file format that uses comma to separate values, and newline to separate records. A CSV file stores tabular data (numbers and text) in plain to, where each line of the file typically represents one data record. Each record consists of the same number of fields, and these are separated by commas in the CSV file. If the field delimiter itself may appear within a field, fields can be surrounded with quotation marks. The CSV file format is one type of delimiter-separated file format. Delimiters frequently used include the comma, tab, space, and semicolon. Delimiter-separated files are often given a ".csv" extension even when the field separator is not a comma. Many applications or libraries that consume or produce CSV files have options to specify an alternative delimiter.

CSV is a common data exchange format that is widely supported by consumers, business, and scientific applications. Among its most common uses is moving tabular data between programs that natively operate on incompatible (often proprietary or undocumented) formats.

For example, a user may need to transfer information from a database program that stores data in a proprietary format, to a spreadsheet that

uses a completely different format. Most database programs can export data as CSV. Most spreadsheet programs can read CSV data, allowing CSV to be used as an intermediate format when transferring data from a database to a spreadsheet. CSV is also used for storing data. Common data science tools such as Pandas include the option to export data to CSV for long-term storage. Benefits of CSV for data storage include the simplicity of CSV makes parsing and creating CSV files easy to implement and fast compared to other data formats, human readability making editing or fixing data simpler, and high compressibility leading to smaller data files. Alternatively, CSV does not support more complex data relations and makes no distinction between null and empty values, and in applications where these features are needed other formats are preferred.

5.5.1 Exported CSV File From Thingspeak

Table 5.1: Exported CSV file from Thingspeak

entry_id	created_at	field1	field2	field3	field4	field5
1	created_at	76	15	32	1	15
2	created_at	70	15	37	1	15
3	created_at	81.99222	12.25635	37	0	32.29345
4	created_at	81.99222	14.36866	39.54149	0	241.0947
entry_id	created_at	field2	field3	field4	field5	field6
5	created_at	84.98833	12.996835	42.04149	-0.5	249.741425
6	created_at	87.985218	12.533068	44.303937	-0.9	319.29918
7	created_at	90.982106	12.069301	46.566384	-1.3	388.856935
8	created_at	93.978994	11.605534	48.828831	-1.7	458.41469
entry_id	created_at	field3	field4	field5	field6	field7
9	created_at	96.975882	11.141767	51.091278	-2.1	527.972445
10	created_at	99.97277	10.678	53.353725	-2.5	597.5302
entry_id	created_at	field1	field2	field3	field4	field5
1	2024-04-22T14:31:41+05:30	76	15	32	1	15

CHAPTER 6

GOOGLE COLAB

Google Colab brings the power of machine learning and data science directly to your web browser. This free service eliminates the need for complicated setups by providing a cloud-based Jupyter Notebook environment. Jupyter Notebooks offer an interactive interface for coding, data analysis, and visualization, making it perfect for both learning and working on projects. The real horsepower comes from Google's cloud resources. Colab grants access to powerful hardware like GPUs and TPUs, significantly speeding up tasks like machine learning training. Collaboration is a breeze too, as Colab notebooks can be easily shared with others. While storage is limited and sessions have timeouts, Colab is a fantastic tool for students, researchers, and anyone interested in exploring the world of data science – all without breaking the bank.

6.1 INITIALIZATION

Importing the necessary libraries for data analysis and machine learning, we begin by incorporating NumPy for numerical computations and pandas for efficient data manipulation. Matplotlib and Seaborn are enlisted to visualize data through graphs and plots, enhancing our understanding of patterns and trends. With the warnings module, we manage potential alert messages during analysis, ensuring a smooth workflow. Utilizing scikit-learn, we import essential functions and classes such as train_test_split for dataset splitting, LogisticRegression for classification modeling, and SimpleImputer for handling missing data. Additionally, RandomForestRegressor is included for regression tasks. Following the imports, we proceed to mount Google Drive to access datasets stored remotely. Subsequently, we initiate the preprocessing phase, which involves data cleaning, normalization, and feature engineering, laying the groundwork for subsequent machine learning analyses. This orchestrated

approach ensures a robust foundation for our predictive modeling endeavors.

6.2 PREPROCESSING WITH DATASET

In the preprocessing phase, attention is paid to ensuring that the column names, such as 'patient_ID', 'heart_rate', 'respiratory_rate', 'temperature', 'PIR Trigger', and 'distance', are aligned with their respective data types or measurements. This alignment enhances the clarity and relevance of the dataset, facilitating easier interpretation and analysis. Each column is carefully examined to handle missing values, scale numerical features, encode categorical variables, and split the dataset accurately for subsequent modeling tasks. By meticulously aligning columns with their corresponding data types or measurements, the preprocessing phase sets the foundation for robust and reliable machine learning analyses.

In the realm of machine learning, preprocessing of data from CSV files serves as a crucial initial step before algorithmic application. Through this preprocessing phase, data undergoes cleaning, normalization, and feature engineering to ensure its suitability for algorithmic analysis. Once data preprocessing is completed, a variety of machine learning algorithms can be applied effectively. These algorithms utilize the processed data to generate insights, predictions, or classifications based on specific objectives. From Naive Bayes and SVM to logistic regression and random forest, each algorithm leverages the preprocessed data to varying degrees of accuracy and efficacy. Overall, the integration of robust preprocessing techniques with machine learning algorithms enhances the system's ability to derive meaningful conclusions from the input data, fostering informed decision-making and driving advancements in various domains.

6.2.1 Histogram

A histogram is a type of bar chart that represents the distribution of numerical data by dividing it into intervals called bins and displaying the frequency of values falling into each bin. In the context of the provided columns - 'patient_ID', 'heart_rate', 'respiratory_rate', 'temperature', 'PIR Trigger', and 'distance' - histograms would illustrate the frequency distribution of numerical variables such as heart rate, respiratory rate, temperature, and distance. Each bar in the histogram represents a bin, with the height corresponding to the frequency of values within that bin. This visualization allows for the identification of central tendency, spread, and shape of the data distribution, aiding in data exploration and analysis.

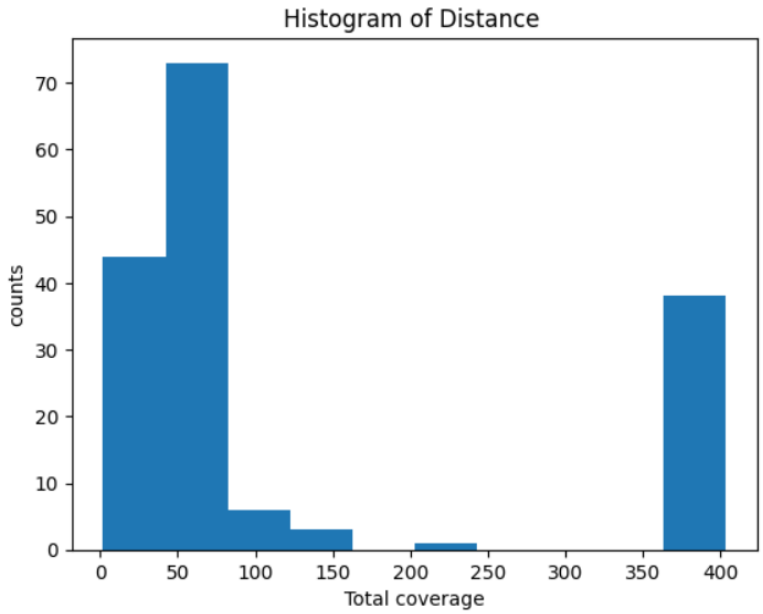


Figure 6.1:Histogram of Distance

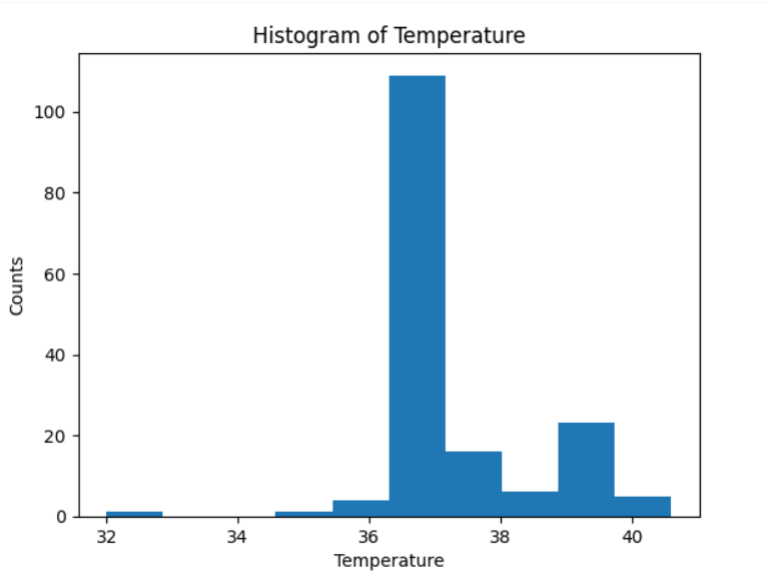


Figure 6.2:Histogram of temperature

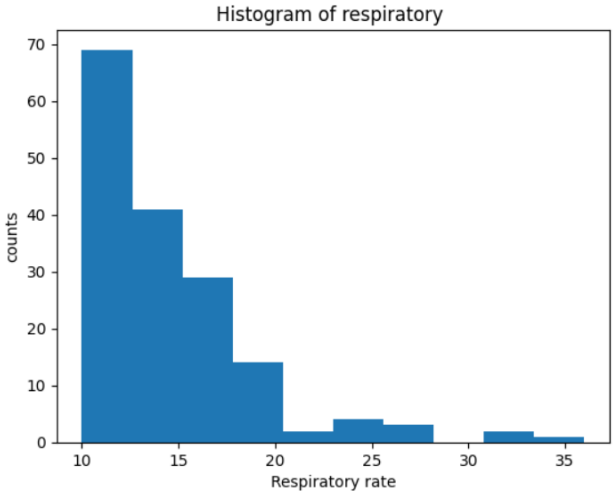


Figure 6.3: Histogram of Respiratory

6.3 SEABORN COUNTPLOT

Seaborn's countplot function to visualize the frequency distribution of categorical variables such as 'heart_rate', 'respiratory_rate', 'temperature', 'PIR Trigger', and 'distance' in the dataset 'data'. Each vertical bar represents the count of occurrences for a specific category within each variable. This count plot offers a concise overview of the distribution of categorical data for multiple variables in the dataset. By leveraging Seaborn's aesthetic enhancements, the plot provides an

easily interpretable representation of the frequencies of different categories across these variables. Finally, the plt.show() function is called to display the generated count plot.

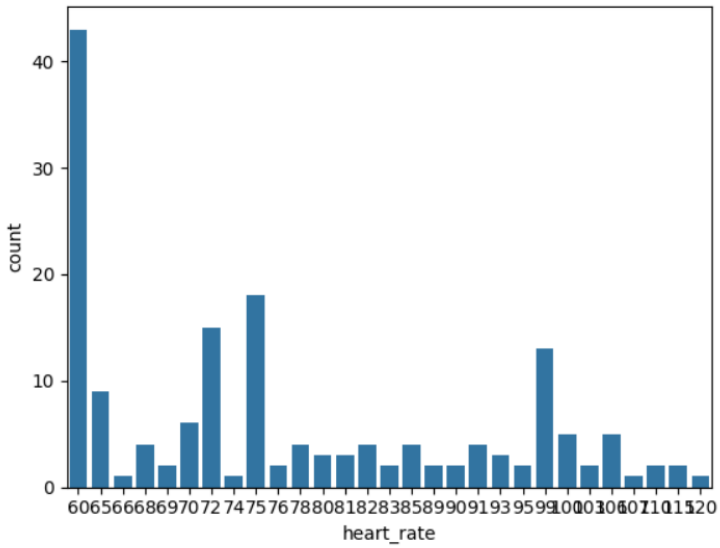


Figure 6.4: SNS Plot of Heart rate

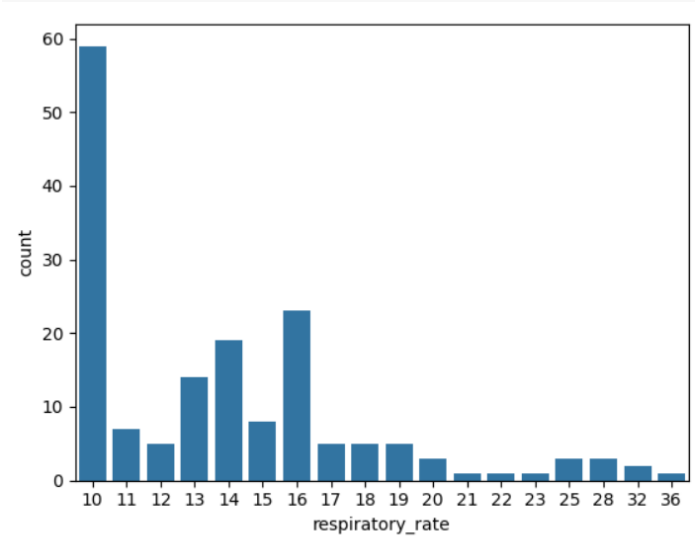


Figure 6.5: SNS Plot of Respiratory rate

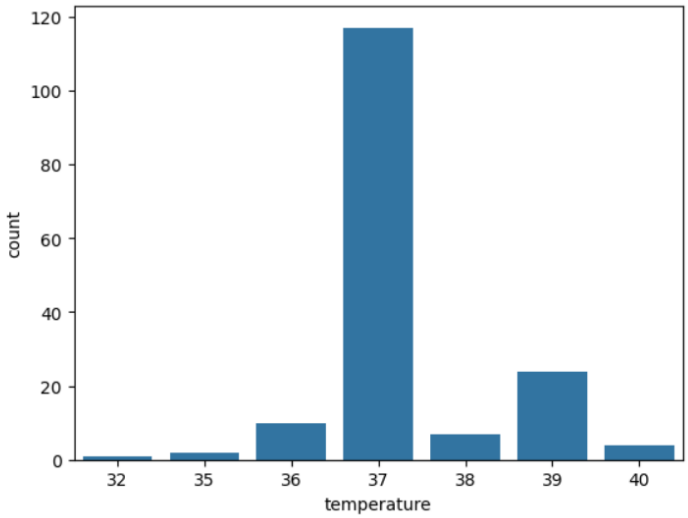
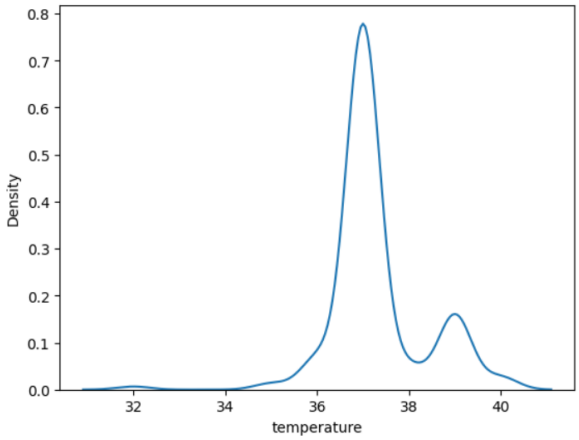


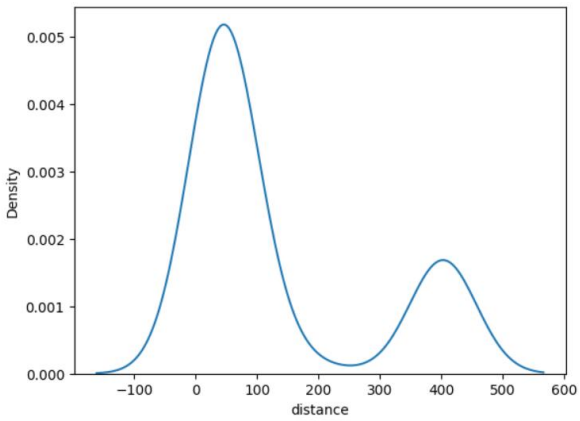
Figure 6.6: SNS Plot of Temperature

6.3.1 Seaborn's KDEPlot Function

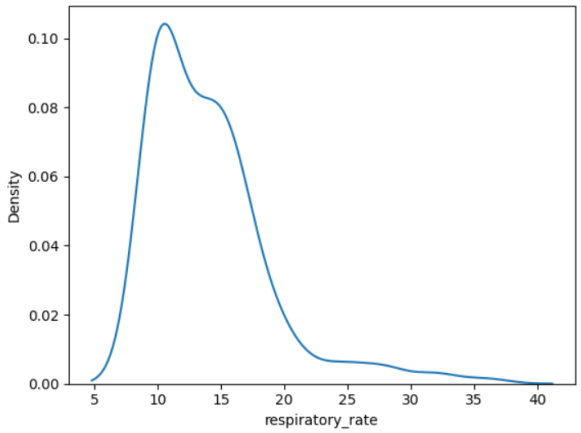
Seaborn's kdeplot function to generate kernel density estimate (KDE) plots for the variables 'heart_rate', 'respiratory_rate', 'temperature', 'PIR Trigger', and 'distance' from the dataset 'data'. Each KDE plot represents the probability density function (PDF) of the respective variable, displaying a smooth curve that indicates the distribution of values. These KDE plots offer continuous visualizations of the data distribution for each variable, allowing for insights into their central tendency, spread, and shape. By leveraging Seaborn's capabilities, the KDE plots provide a comprehensive understanding of the distributional characteristics of the variables in the dataset.



Figure(a)



Figure(b)



Figure(c)

Figure 6.7: SNS KDEPlot of heart rate, respiratory rate, and temperature

6.3.2 Seaborn's Heatmap Function

Seaborn's heatmap function to create a correlation heatmap specifically for the variables patient_ID ,'heart_rate', 'respiratory_rate', 'temperature', 'PIR Trigger', and 'distance' in the dataset 'data'. By computing the correlation matrix 'corr' using the .corr() method on this subset of variables, it captures the pairwise correlation coefficients between them. The heatmap visualizes these coefficients as a color-coded grid, with each cell's color intensity indicating the strength and direction of correlation between the corresponding variables. With 'annot=True', each cell is annotated with its correlation coefficient for enhanced interpretability. Lastly, 'figsize=(5,5)' sets the heatmap's size to ensure clear visualization. This correlation heatmap provides insights into the

relationships between the selected variables, facilitating the identification of potential patterns or dependencies within this specific subset of the dataset.

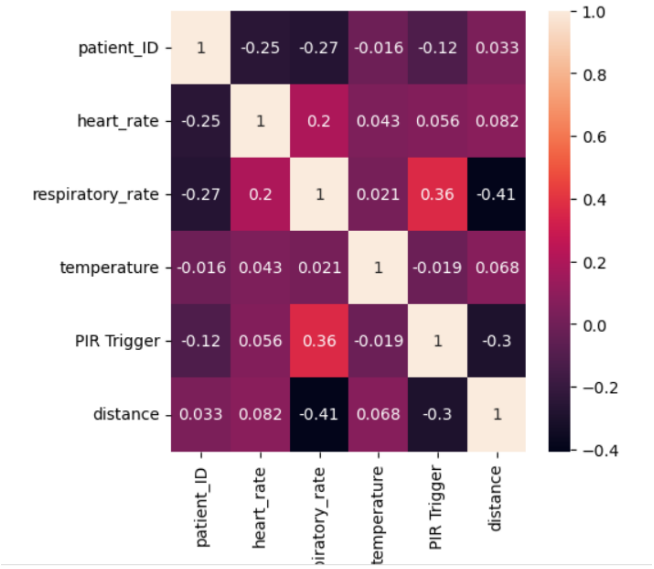


Figure 6.8: SNS Classifier

6.4 MACHINE LEARNING CLASSIFIERS

A machine learning classifier is an algorithm that learns from labeled data to assign categories or labels to unseen data based on its features. It's used in supervised learning tasks where the goal is to predict the category or class of new observations. classifiers are used for various purposes, including:

1. **Classification:** They categorize data into predefined classes or categories based on the patterns observed in the training data.
2. **Prediction:** Classifiers can predict the value of a target variable based on input features, such as predicting whether an email is spam or not based on its content.
3. **Pattern Recognition:** By learning from labeled examples, classifiers can recognize and interpret patterns in data, enabling tasks such as image recognition and speech recognition.
4. **Decision Making:** In applications like medical diagnosis or credit scoring, classifiers aid in decision-making by providing insights into the likelihood of

certain outcomes.

Overall, machine learning classifiers are valuable tools for automating decision-making processes, identifying patterns in data, and making predictions based on past observations. They enable applications across various domains, including healthcare, finance, marketing, and more.

The classifiers employed in our analysis encompass Naive Bayes, Random Forest, Support Vector Machine (SVM), Decision Tree, and Logistic Regression. Naive Bayes offers simplicity and efficiency, while Random Forest provides robustness through ensemble learning. SVM excels in finding optimal hyperplanes for separating classes, while Decision Trees offer interpretability and versatility. Logistic Regression, despite its name, serves as a reliable linear classifier for binary classification tasks. Leveraging this diverse set of classifiers enables us to explore various modeling approaches and select the most suitable one for our dataset and objectives.

6.4.1 Advantages of Machine Learning:

Machine learning (ML) offers a treasure trove of advantages for patient monitoring, revolutionizing how we track health and potentially improving outcomes. Here's a breakdown of some key benefits:

- **Automated Analysis and Trend Identification:** Sifting through mountains of patient data manually can be time-consuming and error prone. ML algorithms can automate data analysis, identifying trends and patterns that might be missed by human eyes. This allows healthcare providers to focus on what matters most: patient care and decision-making.

- **Improved Resource Allocation and Cost Savings:** Early detection and personalized monitoring facilitated by ML can lead to more efficient use of healthcare resources. By preventing unnecessary interventions and

hospitalizations, ML can potentially contribute to cost savings in the healthcare system.

- **Continuous Learning and Improved Accuracy:** Unlike traditional monitoring systems, ML models are constantly learning and evolving. As they are exposed to more data, they can refine their predictions and improve their accuracy over time, leading to a more reliable monitoring experience.

- **Enhanced Prediction and Early Detection:** ML algorithms can analyze vast amounts of patient data, including vital signs, medical history, and genetics. This allows them to identify patterns and predict potential health issues before symptoms even arise. Early detection is crucial for timely intervention and improved treatment effectiveness

By leveraging these advantages, machine learning can transform patient monitoring from a reactive process to a proactive one, empowering patients and healthcare providers to work together for better health outcomes.

6.4.2 Disadvantages of Machine Learning:

Machine learning, despite its remarkable capabilities, presents several challenges that warrant consideration. One significant drawback is its heavy reliance on data, which, if of poor quality or biased, can lead to inaccurate or unfair outcomes. Additionally, overfitting poses a common issue, where models excel on training data but struggle to generalize to new, unseen data. Moreover, the lack of interpretability in complex models like deep neural networks raises concerns regarding transparency and trustworthiness in decision-making processes. Scalability remains a challenge, particularly when handling large-scale datasets, while ethical concerns persist regarding the perpetuation of biases and discriminatory outcomes. Security risks, including adversarial attacks and data breaches, further compound the complexity of deploying machine learning solutions. Continuous maintenance and the need for specialized expertise add to

the resource-intensive nature of implementing these technologies. Finally, privacy concerns surrounding sensitive data underscore the importance of robust data protection measures. Addressing these challenges requires a thoughtful and responsible approach to the development, deployment, and regulation of machine learning systems.

6.5 TYPES OF CLASSIFIERS

6.5.1 Naive Bayes Algorithm

Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. It is mainly used in text classification that includes a high-dimensional training dataset. Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions. It is a probabilistic classifier, which means it predicts on the basis of the probability of an object. Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles. The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, which can be described as:

Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identifying that it is an apple without depending on each other.

Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem.

6.5.2 Support Vector Machine Classifier

Support Vector Machine (SVM) classifiers are pivotal in health monitoring of patients, operating through a multi-step process. Initially, pertinent health data encompassing physiological metrics like heart rate, blood pressure, and biomarkers is collected from the patient. Subsequently, features are extracted from this data, representing quantifiable attributes such as heart rate variability or average blood pressure. These features are then labeled according to corresponding health states, such as "normal" or indicative of a specific condition like arrhythmia. Leveraging this labeled dataset, the SVM classifier is trained to discern patterns and classify unseen data into predefined categories. Once trained, the SVM model is deployed for real-time monitoring, analyzing incoming patient data and providing predictions regarding their health status. Alerts or interventions are triggered based on these predictions, facilitating prompt medical attention when deviations from normal health parameters or signs of specific conditions are detected. Thus, SVM classifiers serve as indispensable tools in healthcare, aiding in the early detection and management of various health conditions through continuous monitoring and predictive analysis.

The code segment implements classification using Naive Bayes and Support Vector Machine (SVM) algorithms on the Iris dataset. It begins by importing necessary libraries and loading the dataset. Feature preprocessing involves extracting the first two features and standardizing them. Two models, Naive Bayes and SVM, are then initialized and trained on the dataset. Decision boundaries are plotted for both models, showcasing their classification capabilities visually. Naive Bayes decision boundary is shown on the left subplot, while SVM decision boundary is displayed on the right. Scatter plots of the data points overlay the decision boundaries for clarity. This visualization aids in understanding how the models separate the iris flower classes based on their features. Finally, the plots are displayed for comparison.

Table 6.1: Accuracy value of Naive Bayes and SVM algorithm

Naive Bayes Accuracy:	0.8222222222222222
SVM Accuracy:	0.8

6.5.3 Decision Boundary For SVM And Naive Bayes

The code splits the Iris dataset into training and testing sets, standardizes features, and trains Naive Bayes and SVM classifiers. Decision boundaries are plotted for both models, showcasing their classification regions. The Naive Bayes boundary is shown on the left subplot, while the SVM boundary is displayed on the right. This aids in comparing how the models classify the iris flower classes based on their features. This code segment evaluates the performance of Naive Bayes and SVM classifiers on the Iris dataset by generating classification reports. After fitting the models and making predictions, classification reports are created using the `classification_report` function from `sklearn.metrics`. The reports are then visualized as heatmaps using `seaborn` and `matplotlib`. Each heatmap displays precision, recall, F1-score, and support for each class, providing insights into the models' performance across different classes. The left subplot shows the classification report for Naive Bayes, while the right subplot displays the report for SVM. This visualization aids in understanding how well the models classify the different iris flower classes and identifies potential areas for improvement.

6.5.4 Classification Report and Confusion Matrix

The classification report provides a comprehensive summary of key metrics for each class in the classification task. These metrics typically include:

Precision: Precision quantifies the accuracy of positive predictions made by the model. It is calculated as the ratio of true positives (TP) to the sum of true positives and false positives (FP).

$$\text{Precision} = TP / (TP + FP)$$

Recall (Sensitivity)

Recall measures the model's ability to correctly identify positive instances. It is calculated as the ratio of true positives to the sum of true positives and false negatives (FN).

$$\text{Recall} = TP / (TP + FN)$$

F1-score:

The F1-score provides a balanced measure of a classifier's performance, considering both precision and recall. It is the harmonic mean of precision and recall.

$$\text{F1-score} = 2 * (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$$

Support: Support indicates the number of actual occurrences of each class in the dataset.

CONFUSION MATRIX:

The confusion matrix is a tabular representation of the model's predictions, breaking down the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) for each class. It helps visualize the performance of the classifier across different classes.

	Precdicted Positive	Predicted Negative
Actual Positive	True Positive(TP)	False Negative(FN)
Actual Negative	False Positive(FP)	False Positive(FP)

Each cell in the confusion matrix represents a different combination of predicted and actual class labels.

These metrics are essential for evaluating the performance of classification models in various domains, providing insights into their accuracy, precision, and ability to correctly classify instances across different classes or categories.

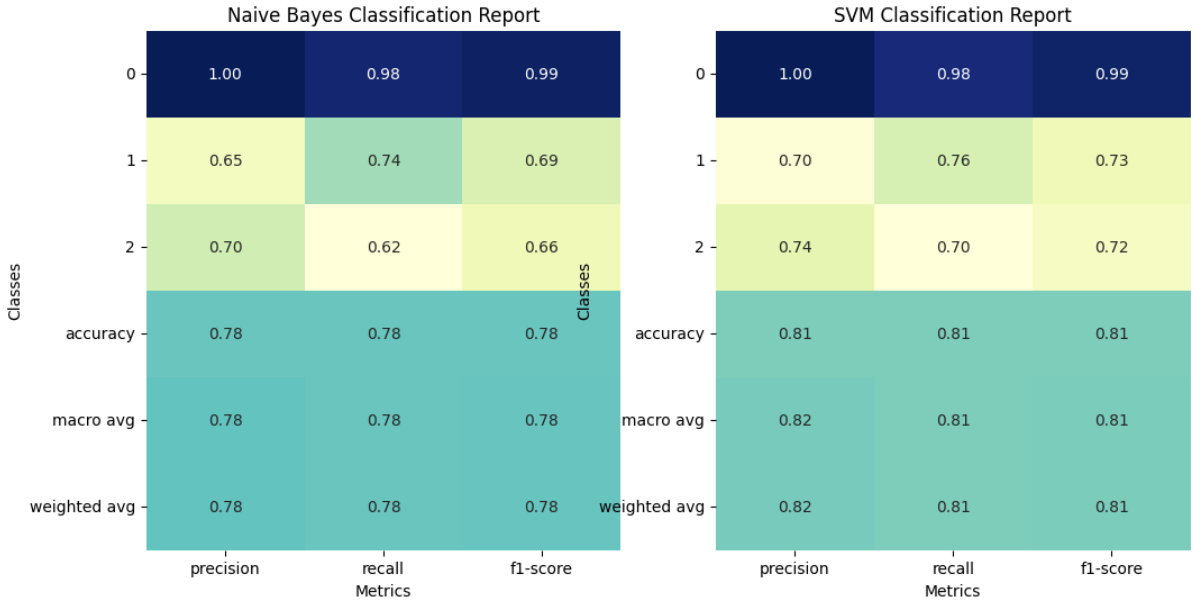


Figure 6.9: Classification Report of SVM Classifier and Naïve Bayes

6.6 LOGISTIC REGRESSION

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables. Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1. Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems. In

Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1). The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc

6.6.1 Output for Logisitic Regression

Table 6.2: Logistic Regression Classification Report

Logistic Regression Accuracy	0.8222222222222222

Logistic Regression Classification Report

	Precision	Recall	F1 - Score	Support
0	1.00	1.00	1.00	19
1	0.78	0.54	0.64	13
2	0.65	0.85	0.73	13

6.6.2 Confusion Matrix

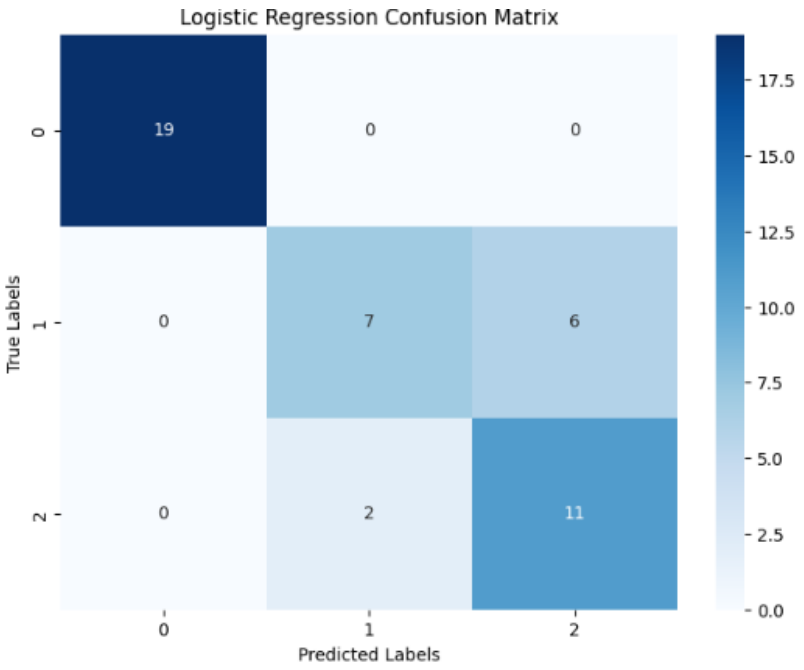


Figure 6.10: Logistics Regression confusion Matrix

6.7 RANDOM TREE FOREST

Forest is a popular ensemble learning method in machine learning. It operates by constructing a multitude of decision trees during training and outputting the mode of the classes (classification) or the mean prediction (regression) of the individual trees. Each tree in the forest is trained on a random subset of the training data and a random subset of the features. This randomness helps to prevent overfitting and improves the generalization of the model. Additionally, Random Forest provides an estimate of feature importance, which can be useful for feature selection and understanding the data. Overall, it's a powerful and versatile algorithm used for both classification and regression tasks. **Disease Prediction:** Random Forest models can be trained on patient data to predict the likelihood of developing certain diseases based on a combination of

factors such as demographics, medical history, lifestyle habits, and genetic markers. These predictions can help healthcare providers intervene early with preventive measures or targeted treatments.

6.7.1 Output of Random Forest Accuracy

Table 6.7: Result of a Random Forest Classifier

Random Forest Accuracy			0.7777777777777777777	
Random Forest Classification Report				
	Precision	Recall	F1 - score	Support
0	1.00	1.00	1.00	19
1	0.64	0.54	0.50	13
2	0.60	0.69	0.64	13

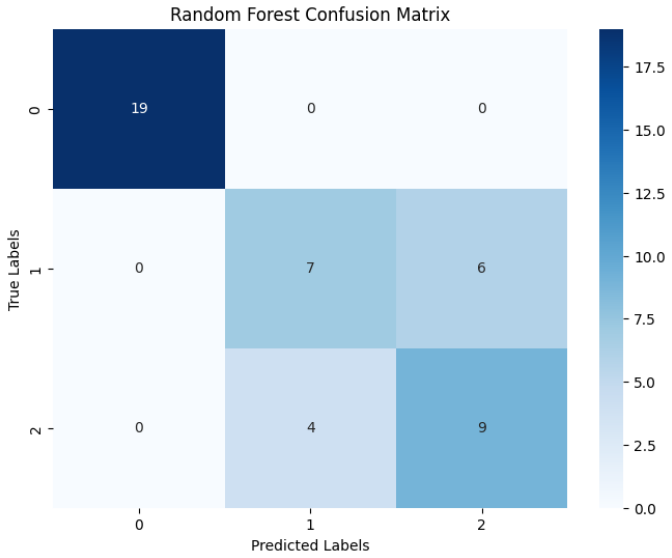


Figure 6.11: Confusion Matrix of random forest

6.8 DECISION TREE CLASSIFIER

Decision tree classifiers are powerful machine learning algorithms that segment data into subsets based on features, creating a hierarchical tree structure for prediction. In the realm of patient improvement, these classifiers play a crucial role in analyzing diverse patient datasets to forecast the probability of improvement. By considering factors such as demographic information, medical history, treatment modalities, and lifestyle habits, decision tree classifiers enable healthcare providers to tailor treatment plans and interventions to individual patients. This personalized approach enhances the efficacy of healthcare delivery by optimizing patient outcomes and fostering more targeted and efficient healthcare practices.

Furthermore, decision tree classifiers can identify key predictors of patient improvement, shedding light on which factors have the most significant impact on outcomes. This insight allows healthcare professionals to prioritize interventions and allocate resources effectively. Additionally, decision trees provide interpretable models, offering transparency into the decision-making process and facilitating trust between patients and providers. By continuously learning from new data, these classifiers can adapt and refine their predictions over time, keeping pace with evolving patient needs and treatment strategies. Ultimately, the integration of decision tree classifiers in healthcare systems holds immense potential for optimizing patient care and driving advancements in medical research and practice.

6.8.1 Output of Decision Tree Classifier

Table 6.8: Result of a Decision Tree Classifier

Decision Tree Classifier			0.6666666666666	
Decision Tree Classifier Report				
	Precision	recall	F1-score	support
0	1.00	0.95	0.97	19
1	0.43	0.46	0.44	13
2	0.46	0.46	0.46	13

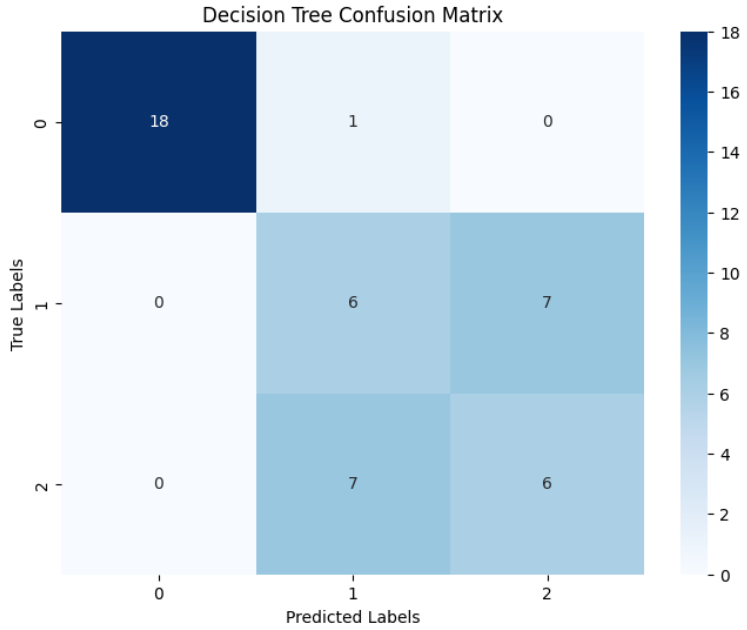


Figure 6.12: Confusion matrix of decision tree

6.9 ACCURACY SCORES OF DIFFERENT CLASSIFIERS

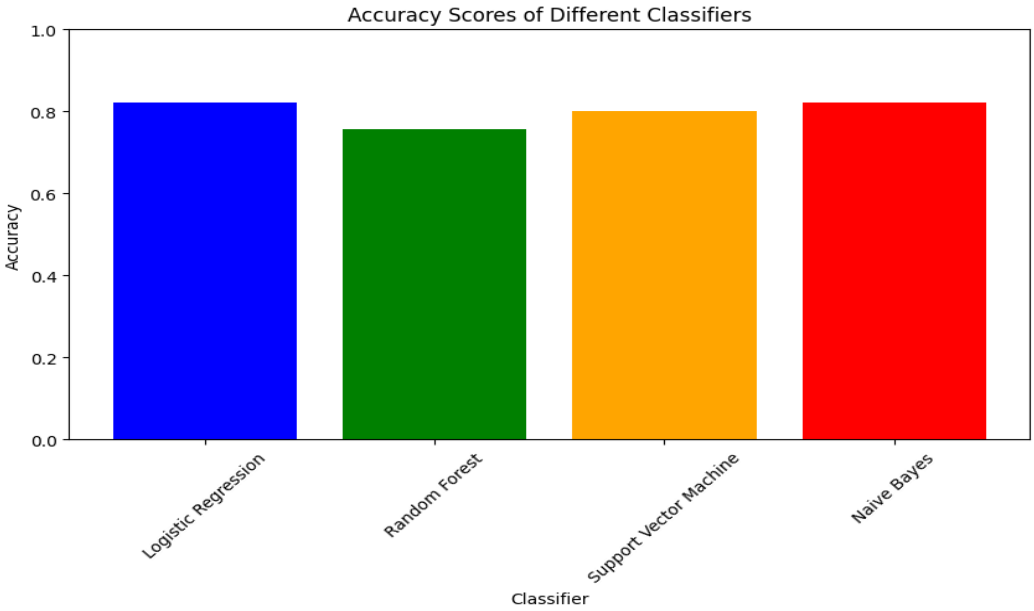


Figure 6.13: Accuracy scores of different classifiers

6.9.1 Comparison Of Classifiers

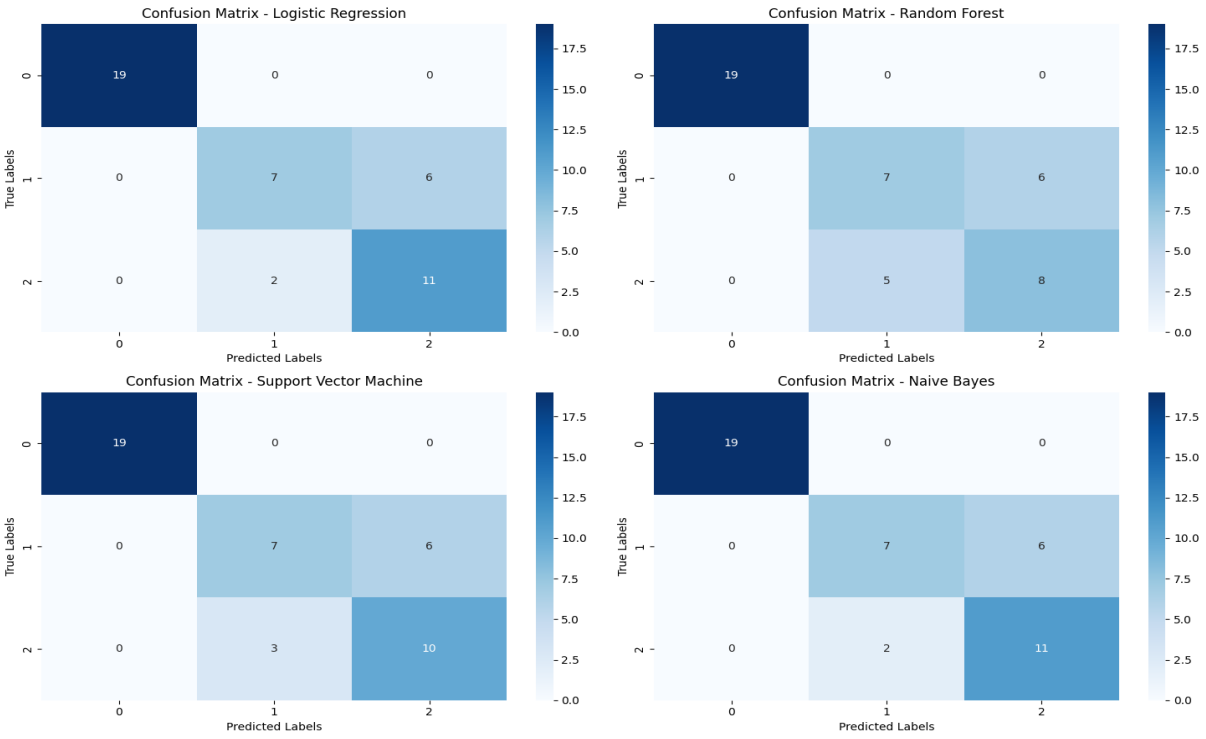


Figure 6.14: Comparison of all classifiers

6.10 PREDICTION OF A PATIENT CONDTION

Our predictive model utilizes a array of patient data, including vital signs like heart rate, respiratory rate, and temperature, as well as contextual factors such as PIR Trigger and distance. By employing comprehensive machine learning classifiers like Naive Bayes, Random Forest, SVM, Decision Tree, and Logistic Regression, we strive to accurately anticipate a patient's health condition. Through this predictive analytics approach, healthcare providers can proactively identify potential medical issues, facilitating timely interventions and personalized treatment plans. This proactive strategy not only enhances patient outcomes but also optimizes healthcare delivery by emphasizing preventive care and early detection of health concerns.

6.11 WEBSITE FOR THE PATIENT MONITORING

The utilization of HTML, CSS, and JavaScript in our tech stack enables healthcare providers to effectively monitor real-time patient data through our digital platform. Leveraging these technologies, we have developed a user-friendly website that allows for seamless data collection and analysis. HTML provides the structure for presenting information, CSS enhances the visual appeal and layout, while JavaScript adds interactivity and dynamic functionality. This comprehensive tech stack empowers healthcare professionals to access vital patient data promptly, facilitating quick decision-making and intervention when necessary. By harnessing the capabilities of these programming languages, we ensure a robust and efficient platform for delivering high-quality healthcare services.

6.11.1 CONNNECT WEBSITE WITH THINGSPEAK

ThingSpeak is a platform facilitating IoT projects, enabling data collection, analysis, and action from connected devices. In website development,

ThingSpeak's public view option allows for easy sharing of data through links. JavaScript is utilized to embed this data into web pages, enhancing interactivity and real-time updates. HTML structures the content, defining the layout and elements of the webpage's framework. CSS styles the HTML elements, providing visual enhancements and deployment using vercel to improving user experience. This integration creates dynamic websites capable of displaying live IoT data. JavaScript acts as the intermediary, enabling communication between ThingSpeak data and HTML elements. This approach fosters versatility, allowing for customizations to suit specific project requirements. By combining ThingSpeak with HTML and CSS, developers can create powerful, visually appealing web applications. Overall, this integration empowers developers to build responsive, data-driven websites for various IoT applications.

6.11.2 Website Advantages for Patient

A healthcare website provides patients with numerous advantages. It offers easy access to medical information, enabling individuals to educate themselves about their conditions and treatment options. Patients can conveniently schedule appointments, request prescription refills, and communicate with healthcare providers from the comfort of their homes. The platform facilitates remote monitoring of health metrics, empowering patients to actively manage their well-being. It also fosters a sense of community through online forums and support groups, where patients can share experiences and seek advice. With robust privacy measures in place, patients can trust that their sensitive health data remains secure. Health tracking tools allow individuals to monitor their progress and adherence to treatment plans effectively. In emergencies, patients can quickly access vital information or seek urgent medical assistance through the website. Overall, the website promotes patient empowerment, engagement, and personalized care, enhancing the healthcare experience.

CHAPTER 7

FINAL RESULT

Based on our predictive model's analysis of the patient data, the final prediction indicates a "low" probability or likelihood of any significant health concerns or adverse medical conditions. This result suggests that the patient's vital signs and contextual factors fall within normal or stable ranges, indicating a favorable health status. However, it's essential to continue monitoring the patient's condition regularly to ensure ongoing well-being and to detect any potential changes or developments promptly. Overall, this positive prediction underscores the effectiveness of our predictive analytics approach in providing valuable insights for healthcare decision-making.

7.1 Analysis from the Processing

- The conclusion of the output is that based on the predictions from all four classifiers, the first patient in the test data is predicted to have a low condition. However, it's important to remember:

- This is just a prediction by machine learning models. It should not be used for definitive medical diagnosis.

- The interpretation of "low" depends on the specific context of how the models were trained. It could refer to disease severity, risk factors, or other patient outcomes.

CHAPTER 8

CONCLUSION

In conclusion, the integration of advanced technologies such as the Wokwi simulation, ThingSpeak, and a user-friendly Vercel-hosted website has revolutionized health surveillance by enabling continuous monitoring of key indicators like temperature, heart rate, respiratory rate, and patient movement. The implementation of supervised learning algorithms—achieving accuracy rates of 82% with Naive Bayes and Logistic Regression, 80% with SVM, and 70% with Random Forest—has significantly enhanced the predictive capabilities of our health monitoring systems. This has not only improved the accuracy of health assessments but also empowered healthcare providers to deliver personalized, proactive care, thereby marking a significant advancement towards a future where technology seamlessly integrates with healthcare to improve patient outcomes and quality of life.

FUTURE ENHANCEMENT

Integration of additional sensors can broaden health monitoring capabilities. Developing a mobile application would enable remote monitoring and real-time alerts, empowering patients and facilitating timely interventions. Implementation of advanced machine learning models promises improved prediction accuracy and personalized healthcare insights. Collaboration with healthcare professionals ensures validation and seamless integration into clinical practice. These enhancements collectively aim to elevate patient care and optimize healthcare delivery

CHAPTER 9

APPENDIX

SOURCE CODE

```
import machine
import time
import urequests
import network
import random

# Wi-Fi configuration
SSID = "Wokwi-Guest"
PASSWORD = "_"

# ThingSpeak configuration
thingspeak_api_key = "95FJ0FAJD4U2VIJ0"
thingspeak_channel_id = "2512400"

wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect(SSID, PASSWORD)

# Wait for the Wi-Fi connection to be established
while not wifi.isconnected():
    pass

print("Connected to Wi-Fi")

potentiometer_1_pin = machine.ADC(26)  # Analog pin for potentiometer 1 (heart rate)
potentiometer_2_pin = machine.ADC(27)  # Analog pin for potentiometer 2 (respiratory rate)
temperature_sensor_pin = machine.ADC(28)  # Analog pin for the NTC temperature sensor
pir_sensor_pin = machine.Pin(14, machine.Pin.IN)  # GPIO14 - PIR sensor input

# LED pins
```

```
led1_pin = machine.Pin(4, machine.Pin.OUT)  # GP4 - Heart rate
led2_pin = machine.Pin(5, machine.Pin.OUT)  # GP5 - Respiratory rate
led3_pin = machine.Pin(3, machine.Pin.OUT)  # GP3 - Temperature
led4_pin = machine.Pin(2, machine.Pin.OUT)  # GP2 - PIR sensor

# Ultrasonic sensor pins
ultrasonic_trigger_pin = machine.Pin(12, machine.Pin.OUT)  # GPIO12 - Ultrasonic sensor trigger pin
ultrasonic_echo_pin = machine.Pin(13, machine.Pin.IN)  # GPIO13 - Ultrasonic sensor echo pin

# Moving Average Filter Parameters
filter_length = 5
heart_rate_history = []
respiratory_rate_history = []
temperature_history = []

# Flags for abnormal conditions
heart_rate_abnormal = False
respiratory_rate_abnormal = False
temperature_abnormal = False
pir_triggered = False

# Function to apply moving average filter
def moving_average(data_history, new_value, filter_length):
    data_history.append(new_value)
    if len(data_history) > filter_length:
        data_history.pop(0)
    return sum(data_history) / len(data_history)

# Function to convert ADC value to temperature (randomize after every 5 times)
def convert_to_temperature(adc_value, reading_count):
    if (reading_count % 6) == 1:
        # Generate a random temperature reading for every 6th reading
        temperature = random.uniform(36.8, 40.2)
    else:
        # Keep temperature constant at 37 for the other readings
        temperature = 37.0
    return temperature

# Function to measure distance using the ultrasonic sensor
def measure_distance():
    # Send a 10us pulse to trigger the ultrasonic sensor
```

```
    ultrasonic_trigger_pin.on()
    time.sleep_us(10)
    ultrasonic_trigger_pin.off()

    # Measure the duration of the pulse on the echo pin
    pulse_duration = machine.time_pulse_us(ultrasonic_echo_pin, 1, 30000)  # Timeout set to 30ms (maximum distance)

    # Convert the pulse duration to distance (in centimeters)
    distance = (pulse_duration * 0.0343) / 2 # Speed of sound is approximately 343 m/s

    return distance

reading_count = 0 # Counter for the number of readings

while True:
    # Read normal values from potentiometers, temperature sensor, and PIR sensor
    potentiometer_1_value = potentiometer_1_pin. read_u16()
    potentiometer_2_value = potentiometer_2_pin.read_u16()
    temperature_sensor_value = temperature_sensor_pin.read_u16()
    pir_sensor_value = pir_sensor_pin.value()

    # Apply variations to normal values
    heart_rate = (potentiometer_1_value / 65535) * 60 + 60  # Adjust the formula based on your sensor characteristics
    respiratory_rate = (potentiometer_2_value / 65535) * 10 + 10  # Adjust the formula based on your sensor characteristics
    temperature=convert_to_temperature(temperature_sensor_value, reading_count)  # Convert NTC temperature sensor value to temperature

    # Increment the reading count
    reading_count += 1

    # Check for abnormal conditions and blink LEDs accordingly
    if heart_rate > 110:
        print ("Patient abnormal: High heart rate!")
        print ("SEND HELP, EMERGENCY!")
        heart_rate_abnormal = True
    elif heart_rate < 65:
        print ("Patient abnormal: Low heart rate!")
        print ("SEND HELP, EMERGENCY!")
        heart_rate_abnormal = True
```

```
    else:
        heart_rate_abnormal = False

    if respiratory_rate < 12:
        print ("Patient abnormal: Low respiratory rate!")
        print ("SEND HELP, EMERGENCY!")
        respiratory_rate_abnormal = True
    elif respiratory_rate > 18:
        print ("Patient abnormal: High respiratory rate!")
        print ("SEND HELP, EMERGENCY!")
        respiratory_rate_abnormal = True
    else:
        respiratory_rate_abnormal = False

    if temperature > 38:
        print ("Patient abnormal: High Temperature!")
        print ("SEND HELP, EMERGENCY!")
        temperature_abnormal = True
    elif temperature < 35:
        print ("Patient abnormal: Low Temperature!")
        print ("SEND HELP, EMERGENCY!")
        temperature_abnormal = True
    else:
        temperature_abnormal = False

    # PIR sensor logic
    if pir_sensor_value == 1:
        print ("Motion detected!")
        pir_triggered = True
    else:
        pir_triggered = False

    # Ultrasonic sensor logic
    distance = measure_distance()
    if distance > 50:  # Adjust the threshold according to your setup
        print ("Patient has moved from the bed!")
        # Take appropriate action here, such as sending an alert or activating a
warning LED

    # Blink LEDs for abnormal conditions
    if heart_rate_abnormal:
        led1_pin. on ()
    else:
```

```
        led1_pin.off ()

    if respiratory_rate_abnormal:
        led2_pin.on()
    else:
        led2_pin.off ()

    if temperature_abnormal:
        led3_pin. on ()
    else:
        led3_pin.off ()

    if pir_triggered:
        led4_pin. on()
    else:
        led4_pin.off ()

    # Send data to ThingSpeak
     url                                                                                    =
"https://api.thingspeak.com/update?api_key={}&field1={}&field2={}&field3=
{} &field4= {} &field5= {}". format (
        thingspeak_api_key, heart_rate, respiratory_rate, temperature, pir_triggered,
distance)
    response = urequests.post(url)
    print ("ThingSpeak Response:", response.text)
    response.close()

    # Print readings to the serial monitor
    print ("Heart Rate:", int(heart_rate), "bpm")
    print ("Respiratory Rate:", int(respiratory_rate), "breaths per minute")
    print ("Temperature:", int(temperature), "degrees Celsius")
    print ("PIR Triggered:", pir_triggered)
    print ("Distance:", distance, "cm")

    time. Sleep (30) # Send readings to ThingSpeak every 30 seconds

//naive bayes and svm classifier

#naive bayes and svm classifier code:
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

```
# Assuming X contains features and y contains labels

# Step 2: Split Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 3: Model Selection
nb_model = GaussianNB()
svm_model = SVC ()

# Step 4: Training
nb_model.fit(X_train, y_train)
svm_model.fit(X_train, y_train)

# Step 5: Evaluation
nb_pred = nb_model.predict(X_test)
svm_pred = svm_model.predict(X_test)

nb_accuracy = accuracy_score(y_test, nb_pred)
svm_accuracy = accuracy_score(y_test, svm_pred)

print ("Naive Bayes Accuracy:", nb_accuracy)
print ("SVM Accuracy:", svm_accuracy)
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data[: :2] # Only take the first two features
y = iris.target
```

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardize features (necessary for SVM)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Fit models
nb_model = GaussianNB()
svm_model = SVC (kernel='linear') # Use linear kernel for better visualization

nb_model.fit(X_train, y_train)
svm_model.fit(X_train_scaled, y_train)

# Generate predictions for the test dataset
nb_pred = nb_model.predict(X_test)
svm_pred = svm_model.predict(X_test_scaled)

# Plot decision boundaries
x_min, x_max = X[:, 0]. min () - 1, X[:, 0].max () + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))

# Naive Bayes decision boundary
Z_nb = nb_model.predict(np.c_[xx.ravel(), yy.ravel()])
Z_nb = Z_nb.reshape(xx.shape)

plt.figure(figsize= (12, 6))

plt.subplot(1, 2, 1)
plt.contourf(xx, yy, Z_nb, alpha=0.8)
```

```
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', cmap=plt.cm.Paired)
plt.title('Naive Bayes Decision Boundary')

# SVM decision boundary
Z_svm = svm_model.predict(np.c_[xx.ravel(), yy.ravel()])
Z_svm = Z_svm.reshape(xx.shape)

plt.subplot(1, 2, 2)
plt.contourf(xx, yy, Z_svm, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', cmap=plt.cm.Paired)
plt.title('SVM Decision Boundary')

plt.show()
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report

# Load a toy dataset (e.g., Iris dataset)
iris = datasets.load_iris()
X = iris.data[:, :2]  # Only take the first two features
y = iris.target

# Standardize features (necessary for SVM)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Fit models
nb_model = GaussianNB()
svm_model = SVC (kernel='linear') # Use linear kernel for better visualization
```

```
nb_model.fit(X, y)
svm_model.fit(X_scaled, y)

# Generate predictions
nb_preds = nb_model.predict(X)
svm_preds = svm_model.predict(X_scaled)

# Generate classification reports
nb_classification_report = classification_report(y, nb_preds, output_dict=True)
svm_classification_report=classification_report(y,svm_preds, output_dict=True)

# Plot classification reports
plt.figure(figsize= (12, 6))
plt.subplot(1, 2, 1)
sns.heatmap(pd.DataFrame(nb_classification_report).iloc[: -1,:]. T, annot=True, cmap="YlGnBu", fmt=".2f", cbar=False)
plt.title('Naive Bayes Classification Report')
plt.xlabel('Metrics')
plt.ylabel('Classes')

plt.subplot(1, 2, 2)
sns.heatmap(pd.DataFrame(svm_classification_report).iloc[:-1,:].T, annot=True, cmap="YlGnBu", fmt=".2f", cbar=False)
plt.title('SVM Classification Report')
plt.xlabel('Metrics')
plt.ylabel('Classes')

plt.show()
//LOGISTIC REGRESSION

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Logistic Regression
logistic_regression = LogisticRegression()
logistic_regression.fit(X_train, y_train)
logistic_regression_preds = logistic_regression.predict(X_test)

# Calculate accuracy
logistic_regression_accuracy= accuracy_score(y_test, logistic_regression_preds)
print ("Logistic Regression Accuracy:", logistic_regression_accuracy)

# Generate classification report
logistic_regression_classification_report=classification_report(y_test,logistic_regression_preds)
print ("Logistic Regression Classification Report:")
print(logistic_regression_classification_report)

# Generate confusion matrix
logistic_regression_confusion_matrix=confusion_matrix(y_test,logistic_regression_preds)
print ("Logistic Regression Confusion Matrix:")
print(logistic_regression_confusion_matrix)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(logistic_regression_confusion_matrix, annot=True, cmap="Blues", fmt="d")
plt.title('Logistic Regression Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Random Forest
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, y_train)
random_forest_preds = random_forest.predict(X_test)

# Calculate accuracy
random_forest_accuracy = accuracy_score(y_test, random_forest_preds)
print ("Random Forest Accuracy:", random_forest_accuracy)

# Generate classification report
random_forest_classification_report=classification_report(y_test, random_forest_preds)
print ("Random Forest Classification Report:")
print(random_forest_classification_report)

# Generate confusion matrix
random_forest_confusion_matrix = confusion_matrix(y_test, random_forest_preds)
print ("Random Forest Confusion Matrix:")
print(random_forest_confusion_matrix)

# Plot confusion matrix
plt.figure(figsize= (8, 6))
sns.heatmap(random_forest_confusion_matrix, annot=True, cmap="Blues", fmt="d")
plt.title('Random Forest Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Assuming you have your feature matrix X and target vector y prepared
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize the Decision Tree classifier
dt_classifier = DecisionTreeClassifier()

# Train the Decision Tree classifier
dt_classifier.fit(X_train, y_train)

# Make predictions on the test set
dt_preds = dt_classifier.predict(X_test)

# Calculate accuracy
dt_accuracy = accuracy_score(y_test, dt_preds)
print ("Decision Tree Accuracy:", dt_accuracy)

# Generate classification report
dt_classification_report = classification_report(y_test, dt_preds)
print ("Decision Tree Classification Report:")
print(dt_classification_report)

# Generate confusion matrix
dt_confusion_matrix = confusion_matrix(y_test, dt_preds)
print ("Decision Tree Confusion Matrix:")
```

```
print(dt_confusion_matrix)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(dt_confusion_matrix, annot=True, cmap="Blues", fmt="d")
plt.title('Decision Tree Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()



from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Initialize classifiers
classifiers = {
    "Logistic Regression": LogisticRegression(),
    "Random Forest": RandomForestClassifier(n_estimators=100),
    "Support Vector Machine": SVC (),
    "Naive Bayes": GaussianNB()
}

# Initialize dictionaries to store metrics
accuracy_scores = {}
classification_reports = {}
confusion_matrices = {}

# Train and evaluate classifiers
```

```
for name, clf in classifiers.items():
    # Train classifier
    clf.fit(X_train, y_train)

    # Make predictions
    preds = clf.predict(X_test)

    # Calculate accuracy
    accuracy_scores[name] = accuracy_score(y_test, preds)

    # Generate classification report
    classification_reports[name]=classification_report(y_test,preds,
output_dict=True)

    # Generate confusion matrix
    confusion_matrices[name] = confusion_matrix(y_test, preds)

# Plot accuracy scores
plt.figure(figsize= (10, 5))
plt.bar(accuracy_scores.keys(), accuracy_scores.values(), color=['blue', 'green',
'orange', 'red'])
plt.title('Accuracy Scores of Different Classifiers')
plt.xlabel('Classifier')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.show()

# Plot confusion matrices
plt.figure(figsize=(15, 10))
for i, (name, matrix) in enumerate(confusion_matrices.items(), 1):
    plt.subplot(2, 2, i)
    sns.heatmap(matrix, annot=True, cmap="Blues", fmt="d")
    plt.title(f'Confusion Matrix - {name}')
```

```
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
plt.tight_layout()
plt.show()

# Print classification reports



from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Initialize classifiers
classifiers = {
    "Logistic Regression": LogisticRegression(),
    "Random Forest": RandomForestClassifier(n_estimators=100),
    "Support Vector Machine": SVC (),
    "Naive Bayes": GaussianNB()
}

# Initialize list to store accuracy scores
accuracy_scores = []

# Train and evaluate classifiers
for name, clf in classifiers.items():
    # Train classifier
    clf.fit(X_train, y_train)

    # Make predictions
    preds = clf.predict(X_test)
```

```
    # Calculate accuracy
    accuracy = accuracy_score(y_test, preds)
    accuracy_scores.append((name, accuracy))

# Plot accuracy scores
plt.figure(figsize= (10, 5))
classifiers_names, accuracies = zip(*accuracy_scores)
plt.bar(classifiers_names, accuracies, color= ['blue', 'green', 'orange', 'red'])
plt.title('Accuracy Scores of Different Classifiers')
plt.xlabel('Classifier')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.show()
```

```
# Make a single prediction
test_sample = X_test[0].reshape(1, -1)  # Assuming X_test is a NumPy array
predictions = []

for classifier in [classifier1, classifier2, classifier3, classifier4]:
    predictions.append(classifier.predict(test_sample))

votes = {'high': 0, 'low': 0}
for pred in predictions:
    if pred == 'high':
        votes['high'] += 1
    else:
        votes['low'] += 1

final_prediction = 'high' if votes['high'] > votes['low'] else 'low'
print ("Final prediction:", final_prediction)
```

```
# Make a single prediction
```

```
test_sample = X_test[0]. reshape (1, -1) # Assuming X_test is a NumPy array
predictions = []

for classifier in [classifier1, classifier2, classifier3, classifier4]:
    predictions.append(classifier.predict(test_sample))

votes = {'high': 0, 'low': 0}
for pred in predictions:
    if pred == 'high':
        votes['high'] += 1
    else:
        votes['low'] += 1

final_prediction = 'high' if votes['high'] > votes['low'] else 'low'
print("Finalprediction:",final_prediction)
```

REFERENCE

1. Honey Pandey, S. Prabha Smart Health Monitoring System using IOT and Machine Learning Techniques, IEEE Explore, August 27,2020.

2. Islam, M.M., Rahaman, A. & Islam, Development of smart healthcare monitoring system in IOT environment,26 May 2020

3. Jayakumar, Ranjith Kumar, Tejswini R, Kavil S, IOT Based Health Monitoring System.

4. Mohammad Salah Uddin, Jannat Binta Alam, Suraiya Banu Real time patient monitoring system based on Internet of Things, IEEE Explore , 15 January 2018

5. Prajoona valsalan , Ahmed Tariq, Ali Hussain, IOT based Health Monitoring System,Research Gate, April 2020.

6.Suliman Abdulmalek , Abdul Nasir, waheb A. Jabbar , IoT-Based Healthcare-Monitoring System towards Improving Quality of Life, Health Care, 11 October 2022