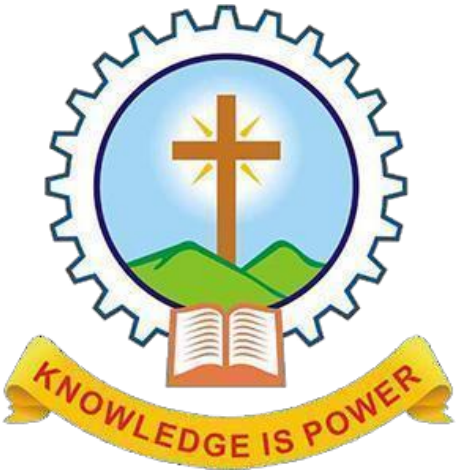
MAR ATHANASIUS COLLEGE OF ENGINEERING

(Affiliated to APJ Abdul Kalam Technological University, TVM)
KOTHAMANGALAM



Department of Computer Applications

Main Project Report

Elderly Caregiver Matchmaking System

Done by

SANDRA S SANTHOSH

Reg No: MAC23MCA-2048

Under the guidance of

Prof. Manu John

2023-2025

MAR ATHANASIUS COLLEGE OF ENGINEERING

(Affiliated to APJ Abdul Kalam Technological University, TVM)
KOTHAMANGALAM

CERTIFICATE

Elderly Caregiver Matchmaking System

Certified that this is the bonafide record of project work done by

SANDRA S SANTHOSH
Reg No: MAC23MCA-2048

During the academic year 2024-2025, in partial fulfillment of requirements for award of the degree,

Master of Computer Applications
of
APJ Abdul Kalam Technological University, Thiruvananthapuram

Faculty Guide
Prof. Manu John

Head of the Department
Prof. Biju Skaria

Project Coordinator
Prof. Sonia Abraham

External Examiner

Acknowledgement

With heartfelt gratitude, I extend my deepest thanks to the Almighty for his unwavering grace and blessings that have made this journey possible. May his guidance continue to illuminate our path in the years ahead.

I'm immensely thankful to Prof. Biju Skaria, head of the department of Computer Applications, Prof. Sonia Abraham, our project coordinator and Prof. Manu John, our dedicated faculty guide , for their invaluable guidance and timely advice, which played a pivotal role in shaping this project. Their guidance, constant supervision, and provision of essential information were instrumental in the successful completion of the project.

I extend my profound thanks to all the professors in the department and the entire staff at MACE for their unwavering support and inspiration throughout my academic journey. My sincere appreciation goes to our beloved parents, whose guidance has been a beacon in every step of our path.

I'm also grateful to my friends and individuals who generously shared their expertise and assistance, contributing significantly to the fulfillment of this endeavor.

Sandra S Santhosh

Abstract

The Elderly Caregiver Matchmaking System is designed to provide a seamless and efficient platform connecting elderly individuals with caregivers who meet their specific needs and preferences. By integrating skill-based search, emergency support, health monitoring, and secure financial transactions, the platform aims to bridge the gap in quality caregiving services. The system ensures ease of access and reliability by leveraging advanced technologies such as predictive analytics and chatbots. It is developed to improve the quality of life for elderly individuals by offering a holistic and user-friendly solution.

The system consists of four primary actors: elderly users, family members, caregivers, and administrators. Elderly users primarily engage with the platform for emergency support through a dedicated mobile application. They can trigger an emergency alert, which sends their location to designated family members and emergency contacts. Family members play a crucial role in searching for caregivers, reviewing services, making payments, utilizing chatbot assistance, and managing health assistance features, including monitoring health conditions and predictive analytics. Caregivers register their profiles, update their availability, and provide services while tracking earnings. Administrators oversee all operations, including caregiver approvals and maintaining service quality. This structured approach ensures smooth interactions and efficient service delivery.

The platform comprises multiple functional modules, each catering to specific needs. The User Management Module enables role-based account creation and management. The Skill-Based Search Module helps families find suitable caregivers based on experience, certifications, and availability. The Review and Rating Module maintains service quality by allowing feedback and ratings. A Chatbot Assistance Module offers instant support for navigation and queries. The Emergency Support Module ensures that elderly users can quickly alert family members or emergency services in case of distress, with their location automatically shared. Additionally, the Health Assistance Module allows family members to monitor health conditions and receive predictive insights. The Payment Gateway Module facilitates secure financial transactions between users and caregivers.

The Elderly Caregiver Matchmaking System serves as a comprehensive solution to the increasing demand for quality elderly care. By integrating essential features such as skill-based caregiver search, emergency support, predictive health analytics, and a secure payment system, the platform enhances accessibility, safety, and convenience. With a focus on user-friendly interactions and AI-driven health insights, the system ensures that elderly individuals receive personalized and high-quality caregiving services. This project represents a significant step towards improving elderly care infrastructure, ensuring a safer and more comfortable aging experience.

List of Tables

List of Figures

Contents

1 INTRODUCTION

The Elderly Caregiver Matchmaking System is designed to connect elderly individuals with caregivers who meet their specific needs, preferences, and skill requirements. This platform addresses the growing demand for personalized caregiving services while ensuring accessibility, reliability, and high-quality service. It integrates key features such as skill-based search, emergency support, predictive health monitoring, and secure payment processing.

The system includes four primary user roles: elderly individuals, family members, caregivers, and administrators. Elderly users interact mainly with the emergency support app to trigger alerts and share their location in emergencies. Family members play an active role in searching for caregivers, making payments, utilizing chatbot assistance, and managing health monitoring. Caregivers register on the platform, manage their availability, and provide services, while administrators oversee platform operations, caregiver approvals, and dispute resolution.

The platform includes essential modules such as User Management for role-based access, Skill-Based Search for efficient caregiver matching, and a Review and Rating System to ensure service quality. A Chatbot Assistance Module provides automated support, while the Emergency Support Module allows elderly users to quickly notify family or emergency services in critical situations. Additionally, a Health Assistance Module helps monitor and predict chronic conditions, and a Payment Gateway Module facilitates secure transactions.

The integration of emergency support features ensures that elderly users can easily access assistance when needed, offering peace of mind to their families. The predictive health analytics system helps identify potential health risks, allowing timely medical intervention. With secure financial transactions and caregiver verification processes in place, the platform enhances trust and reliability for all users.

The Elderly Caregiver Matchmaking System is a comprehensive solution that enhances elderly care by combining technology-driven caregiver matching, health monitoring, emergency support, and financial security. By addressing critical caregiving gaps, the platform ensures a safe, efficient, and user-friendly experience for elderly individuals and their families, ultimately improving the quality of care and well-being for aging individuals.

2 SUPPORTING LITERATURE

2.1 Literature Review

Paper 1:"Rahman, Senjuti, Mehedi Hasan, and Ajay Krishno Sarkar. "Prediction of brain stroke using machine learning algorithms and deep neural network techniques." ***European Journal of Electrical Engineering and Computer Science 7.1 (2023): 23-30.***

The paper titled " Prediction of brain stroke using machine learning algorithms and deep neural network techniques " by Rahman, Senjuti, Mehedi Hasan, and Ajay Krishno Sarkar presents a study to forecast the possibility of a brain stroke occurring at an early stage using deep learning and machine learning techniques. To gauge the effectiveness of the algorithm, a reliable dataset for stroke prediction was taken from the Kaggle website. Several classification models, including XGBoost, Ada Boost, Light Gradient Boosting Machine, Random Forest, Decision Tree, Logistic Regression, K Neighbors, SVM, Naive Bayes, and deep neural networks were successfully used in this study for classification tasks. The Random Forest classifier has 99% classification accuracy, which was the highest (among the machine learning classifiers). The three layer deep neural network has produced a higher accuracy of 92.39% than the three-layer ANN method utilizing the selected features as input. The research's findings showed that machine learning techniques outperformed deep neural networks.

Title	Prediction of brain stroke using machine learning algorithms and deep neural network techniques.
Area of Work	Brain stroke prediction using machine learning and deep neural networks.
Dataset	Kaggle dataset with 5110 rows and 12 columns.
Methodology/Strategy	Data preprocessing, feature selection, PCA for dimensionality reduction, followed by model training.
Algorithm	XGBoost, Ada Boost, Light Gradient Boosting Machine, Random Forest, Decision Tree, Logistic Regression, K Neighbors, SVM, Naive Bayes, and deep neural networks.
Result/Precision	The Random Forest classifier achieved 99% classification accuracy.
Advantages	High accuracy in stroke prediction; comprehensive comparison of multiple algorithms.
Future Proposal	Explore more advanced deep learning models and real-time prediction systems for clinical use.

Table 2.1: Summary of Brain Stroke Prediction Study 1

Paper 2: Puranjay Savar Mattasa. "Brain Stroke Prediction Using Machine Learning." *International Journal of Research Publication and Reviews*, vol. 3, no. 12, Dec. 2022, pp. 711-722.

The paper titled " Brain Stroke Prediction Using Machine Learning, " by Puranjay Savar Mattasa focuses on predicting brain strokes using machine learning models trained on electronic health records (EHRs). The authors used a dataset of 43,400 patients with 12 attributes to train and test the model. The study used a stacking ensemble method that combined multiple algorithms to improve prediction accuracy. The research highlights the importance of accurate feature selection and preprocessing in improving model performance. Stacking, which consists of two-layer estimators, is a method of assembling classification or regression models. All baseline models that are used to forecast the results in the test datasets are contained in the first layer. The second layer is made up of a Meta-Classifier or Regressor that creates new predictions by using all of the predictions from baseline models as input. They can increase the accuracy that is currently demonstrated by individual models. As different algorithms capture distinct trends in training data, we can obtain the majority of stacked models by selecting a variety of algorithms for the first layer of the architecture. By combining both models, we can obtain more precise and superior outcomes.

Title	Brain Stroke Prediction Using Machine Learning
Area of Work	Healthcare, Machine Learning
Dataset	EHR dataset with 43,400 records and 12 attributes.
Methodology/Strategy	Stacking ensemble method combining multiple machine learning models.
Algorithm	K-Nearest Neighbors, Support Vector Classifier, Decision Tree, Random Forest, MultiLayer Perceptron, Logistic Regression
Result/Precision	The accuracy achieved by this model is: 98.48% on the training set, 98.49% on the testing set
Advantages	High accuracy in stroke prediction; comprehensive comparison of multiple algorithms.
Future Proposal	Explore more advanced deep learning models and real-time prediction systems for clinical use.

Table 2.2: Summary of Brain Stroke Prediction Study 2

Paper 3: Muktevi Srivenkatesh . "Prediction of Cardiovascular Disease using Machine Learning Algorithms" ***International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-9 Issue-3, February, 2020.***

This research explores the application of machine learning techniques for predicting cardiovascular disease (CVD), a leading cause of global mortality. The study utilizes the Cardiovascular Disease Dataset from Kaggle, which contains 68,975 patient records with various features categorized into objective, examination, and subjective attributes. Objective features include age, gender, height, and weight, while examination features consist of blood pressure, cholesterol, and glucose levels. Subjective features cover lifestyle factors such as smoking, alcohol intake, and physical activity. Additionally, Body Mass Index (BMI) is introduced as a new feature, derived from height and weight, to assess its impact on prediction accuracy.

To develop a robust prediction model, the dataset is divided into 80% training and 20% testing sets, and multiple machine learning algorithms are applied. The models tested include Logistic Regression, Decision Tree, k-Nearest Neighbors (k-NN), Naïve Bayes, and a Neural Network (MLP Classifier). Their respective accuracies are 72.5% for Logistic Regression, 76.3% for Decision Tree, 70.8% for k-NN, and 68.2% for Naïve Bayes. Among these, the Neural Network (MLP Classifier) achieved the highest accuracy of 81.5%, making it the best-performing model. The Decision Tree algorithm also produced competitive results, but its tendency to overfit made the Neural Network a better choice for generalization.

The findings indicate that incorporating BMI as a feature enhances prediction accuracy, reinforcing the role of lifestyle and physical attributes in assessing cardiovascular risk. This study highlights the effectiveness of machine learning in early detection and risk assessment of CVD, demonstrating its potential to support timely medical interventions. By leveraging machine learning, healthcare professionals can improve diagnostic precision and enhance patient outcomes through predictive analytics.

This study highlights the effectiveness of machine learning in cardiovascular disease prediction. The results indicate that incorporating BMI as an additional feature improves accuracy, emphasizing the significance of lifestyle factors in CVD assessment. The Neural Network model outperformed other classifiers, making it a promising tool for early disease detection and risk assessment in healthcare.

Title	Machine Learning-Based Prediction of Cardiovascular Disease Using Clinical and Lifestyle Features
Area of Work	Healthcare informatics and predictive analytics for cardiovascular disease (CVD) prediction using machine learning.
Dataset	The study utilizes the Cardiovascular Disease Dataset from Kaggle, consisting of 68,975 patient records. The dataset includes three main feature categories:Additionally, Body Mass Index (BMI) is introduced as a derived feature to evaluate its contribution to model accuracy. The dataset is split into 80% training and 20% testing for model evaluation.
Methodology/Strategy	The research follows a machine learning-based approach involving several steps:Data Preprocessing: Handling missing values, feature scaling, and deriving BMI as a new feature. Feature Selection: Identifying key variables that influence cardiovascular disease prediction.Model Training: Applying different machine learning algorithms for classification.Evaluation: Comparing model performance using accuracy as the primary metric.
Algorithms	• Logistic Regression • Decision Tree • k-Nearest Neighbors (k-NN) • Naïve Bayes • Neural Network (MLP Classifier)
Results/Accuracy	• **Neural Network (MLP Classifier)**: 81.5% (Highest Accuracy) • Decision Tree: 76.3% • Logistic Regression: 72.5% • k-Nearest Neighbors (k-NN): 70.8% • Naïve Bayes: 68.2%
Advantages	• Early detection of cardiovascular disease • Improved accuracy with feature engineering

Table 2.3: Summary of CVD Prediction Study

2.2 Literature Summary

The first paper, ”Prediction of Brain Stroke Using Machine Learning Algorithms and Deep Neural Network Techniques,” explores the use of machine learning and deep neural networks for early stroke prediction. The study employs various classification models, including Random Forest, XGBoost, AdaBoost, and deep learning-based artificial neural networks (ANNs). A dataset from Kaggle containing 5110 records and 12 features was used for training and evaluation. Among the machine learning classifiers, the Random Forest algorithm achieved the highest accuracy of 99%, whereas the three-layer deep neural network (4-Layer ANN) attained 92.39%. The study concludes that machine learning classifiers outperform deep neural networks in stroke prediction and suggests future research on advanced deep learning models for real-time clinical applications.

The second paper, ”Brain Stroke Prediction Using Machine Learning,” focuses on predicting brain strokes using electronic health records (EHRs) from a dataset containing 43,400 patient records with 12 attributes. The study emphasizes the significance of feature selection and data preprocessing in improving model performance. A stacking ensemble method, consisting of multiple machine learning models, was used to enhance accuracy. The first layer of the model contained base classifiers such as k-Nearest Neighbors, Support Vector Classifier, Decision Tree, Random Forest, Multilayer Perceptron, and Logistic Regression. The second layer was a meta-classifier that combined predictions from these base models. The stacking model achieved an impressive accuracy of 98.49% on the testing dataset. The research highlights the effectiveness of ensemble methods in improving stroke prediction accuracy and suggests further exploration of deep learning techniques and real-time implementation for clinical decision-making.

The third paper, ”Machine Learning-Based Prediction of Cardiovascular Disease Using Clinical and Lifestyle Features,” investigates the application of machine learning techniques for predicting cardiovascular disease (CVD). The study uses a Kaggle dataset comprising 68,975 patient records with objective, examination, and subjective features. The dataset was divided into 80% training and 20% testing sets, and various machine learning algorithms, including Logistic Regression, Decision Tree, k-Nearest Neighbors, Naïve Bayes, and a Neural Network (MLP Classifier), were applied. The results showed that the MLP Classifier achieved the highest accuracy of 81.5%, followed by the Decision Tree model with 76.3%. The study emphasizes the importance of feature engineering, particularly incorporating Body Mass Index (BMI) as a derived feature, which improved prediction accuracy. It concludes that machine learning models can effectively support early detection and risk assessment of CVD, enabling timely medical intervention.

2.3 Findings and Proposals

This study aims to develop a machine learning-based predictive model for stroke and cardiovascular disease (CVD) using efficient classification algorithms. For stroke prediction, the Random Forest classifier will be employed because of its superior accuracy demonstrated in existing literature. It offers robustness against overfitting and effectively handles large datasets with multiple features. The model will be trained on a dataset containing relevant clinical parameters such as age, gender, smoking status, hypertension, heart disease history, and BMI.

For cardiovascular disease prediction, Logistic Regression will be used because of its strong interpretability and effectiveness in binary classification problems. Logistic Regression has been widely applied in medical predictive modeling, offering a balance between simplicity and accuracy. The model will be trained on a dataset incorporating clinical measurements and lifestyle features, including blood pressure, cholesterol levels, glucose levels, smoking habits, physical activity, and BMI.

Data preprocessing techniques such as feature scaling, missing value imputation, and feature selection will be implemented to enhance model performance. Both models will be evaluated using accuracy, precision, recall, and F1-score metrics to ensure their reliability in medical diagnosis. The study aims to provide an efficient, data-driven approach for early disease detection, contributing to improved healthcare decision-making and patient outcomes.

3 SYSTEM ANALYSIS

3.1 Actors and roles

The system involves actors: **elderly users**, **Family users**,**caregivers**, and **administrators**.

- **Elderly Users**:
  - Have a simplified role focused on emergency support.
  - Can trigger an emergency notification from an android app, which alerts their family members or designated emergency contacts.
  - Can call local emergency contacts like fire/ambulence/police from within the app in case of emergency.
- **Family Members**:
  - Search for caregivers based on specific requirements.
  - View caregiver profiles and ratings.
  - Provide reviews and make payments.
  - Use a chatbot for assistance.
  - Utilize health assistance features for monitoring health and predicting risks of diseases like cardiovascular conditions and strokes.
- **Caregivers**:
  - Register on the platform and create profiles showcasing their skills.
  - Manage their availability.
  - Respond to service requests from elderly users.
  - Track their earnings through the payment gateway.
- **Administrators**:
  - Oversee the platform's operations to ensure smooth functioning.
  - Approve caregiver profiles.
  - Resolve disputes and address user queries.
  - Manage the payment gateway to ensure secure financial transactions.

3.2 Module Description

- **User Management Module:** Handles account creation, login, and profile management for all user roles (elderly individuals, caregivers, and administrators). Families can update preferences and personal details, caregivers can add skills, certifications, and availability, while administrators manage user accounts, approve caregiver profiles, and address queries. This ensures secure access and role-based functionalities.

- **Skill-Based Search Module:** Provides a search feature that allows families to filter caregivers based on experience, certifications, languages spoken, and availability.

- **Review and Rating Module:** Enables families to rate and review caregivers. Caregivers can view feedback for improvement. Administrators monitor reviews for inappropriate content and ensure quality control.

- **Chatbot Assistance Module:** Implements a rule-based chatbot to assist users with common queries, such as platform navigation, feature usage, and emergency support. This enhances user experience and reduces dependency on customer support.

- **Emergency Support Module:** Allows elderly users to contact emergency services or their emergency contacts during critical situations. The elderly will use an Android app to trigger emergency support, which sends a notification along with their location to all emergency contacts. They can also directly contact local emergency services (police, ambulance, fire department) from the app.

- **Health Assistance Module:** Provides health monitoring and predictive analytics for chronic diseases such as stroke and cardiovascular conditions using machine learning algorithms.

- **Payment Gateway Module:** Integrates secure and convenient payment options, enabling families to pay for services and caregivers to receive payments seamlessly.

- **Scheduling and Availability Module:** Allows caregivers to manage their availability and schedule appointments with families, ensuring efficient service coordination.

3.3 Bussiness Rules

Business rules are specific guidelines, conditions, or constraints that govern the operations of a business or system. They define how processes and activities should be conducted to ensure consistency, fairness, and alignment with business objectives.Business rules help ensure that the system behaves as expected and meets the needs of users or stakeholders.

- Each user (Admin, Family, Caregiver, Elderly) must register with a unique username and password.
- Admins can approve or reject caregiver registrations before they appear in the system.
- Only approved caregivers will be listed and available for booking.
- Caregivers must provide verified details such as skills, certifications, experience, and availability.
- Caregivers are matched to elderly users based on skills, location, and availability preferences.
- A caregiver cannot accept overlapping bookings for the same time slot.
- A family can send a booking request specifying date, start time, and end time.
- Elderly users can send service requests to caregivers, with requests automatically expiring after 48 hours if unanswered.
- The SilverSupport app is for elderly users only.
- The SilverSupport app does not require an account on the matchmaking website.
- Elderly users receive a confirmation when an emergency alert is sent via SilverSupport.
- The chatbot can only handle predefined queries.

3.4 Analysis of dataset

Dataset analysis is a fundamental step in machine learning that involves examining and understanding the data before model development. This process helps identify patterns, relationships, inconsistencies, and potential biases within the dataset. A thorough analysis ensures data quality, enhances model performance, and improves decision-making.

3.4.1 About the Dataset

3.4.1.1 Stroke Dataset

The dataset for this brain stroke prediction project is sourced from Kaggle and includes 5,000 samples with 12 columns: 1 patient ID, 1 binary class label (stroke), and 10 numeric or Boolean features. Missing values are handled during preprocessing.

The patient ID is a unique identifier, and the features—age, gender, hypertension, heart disease, marital status, work type, residence type, average glucose level, BMI, and smoking status—are chosen for their relevance to stroke risk.

The class variable (stroke) is binary, indicating the occurrence (1) or absence (0) of a stroke. This binary classification allows the model to predict the likelihood of a stroke based on patient health indicators.

3.4.1.2 Cardiovascular Dataset

The dataset for this cardiovascular disease prediction project is sourced from Kaggle and contains 70,000 samples with 13 columns: 1 ID, 1 class variable (cardio), and 11 numerical features.

The ID column is a unique identifier and excluded from analysis. Features include age (in days), gender, height, weight, blood pressure, cholesterol, glucose, smoking, alcohol intake, and physical activity—factors known to influence cardiovascular health.

The target variable (cardio) is binary, indicating the presence (1) or absence (0) of cardiovascular disease, supporting binary classification based on health metrics. The dataset's large size and fully numerical attributes make it well-suited for training machine learning models while simplifying preprocessing.

3.4.2 Explore the dataset

3.4.2.1 Exploring Stroke Dataset

The Stroke dataset, sourced from Kaggle, contains 5,000 patient health records with 12 columns: 1 identifier (‘patient_id‘), 1 binary class label (‘stroke‘), and 10 numeric or Boolean

features. These include age, gender, hypertension, heart disease, marital status, work type, residence type, average glucose level, BMI, and smoking status—key attributes linked to stroke risk.

The dataset includes some missing values to be handled during preprocessing. The class variable (‘stroke‘) indicates stroke occurrence (1) or absence (0), enabling binary classification to predict stroke likelihood based on health indicators.

Initial analysis involves exploring the distribution of variables, checking for missing data, identifying outliers, and examining feature correlations with stroke occurrence to highlight significant risk factors.

Dataset:https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

Figure 3.1: Snapshot of healthcare-dataset-stroke-data.csv

```
df.describe()
```

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	4909.000000	5110.000000
mean	36517.829354	43.226614	0.097456	0.054012	106.147677	28.893237	0.048728
std	21161.721625	22.612647	0.296607	0.226063	45.283560	7.854067	0.215320
min	67.000000	0.080000	0.000000	0.000000	55.120000	10.300000	0.000000
25%	17741.250000	25.000000	0.000000	0.000000	77.245000	23.500000	0.000000
50%	36932.000000	45.000000	0.000000	0.000000	91.885000	28.100000	0.000000
75%	54682.000000	61.000000	0.000000	0.000000	114.090000	33.100000	0.000000
max	72940.000000	82.000000	1.000000	1.000000	271.740000	97.600000	1.000000

Figure 3.2: Description of healthcare-dataset-stroke-data.csv

The dataset used for brain stroke prediction comprises 5,110 entries across 12 columns, providing a diverse set of features related to patient demographics and health status. The data types include:

```
df.shape

(5110, 12)
```

Figure 3.3: Shape of healthcare-dataset-stroke-data.csv

- **Integer (int64)**: Columns like id, hypertension, heart_disease, and stroke are stored as integers. These variables represent categorical binary values (e.g., presence or absence of hypertension and heart disease) and identifiers.
- **Float (float64)**: The columns age, avg_glucose_level, and bmi are stored as floats, accommodating decimal values for continuous variables such as age and health measurements.
- **Object (object)**: Categorical variables such as gender, ever_married, work_type, Residence_type, and smoking_statused as objects (textual data). These variables require encoding before they can be used in machine learning models.

```
[ ] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5110 non-null   int64
 1   gender             5110 non-null   object
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64
 4   heart_disease      5110 non-null   int64
 5   ever_married       5110 non-null   object
 6   work_type          5110 non-null   object
 7   Residence_type     5110 non-null   object
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object
 11  stroke             5110 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

Figure 3.4: Datatypes of healthcare-dataset-stroke-data.csv

This mix of data types necessitates specific preprocessing steps, such as handling missing values in the bmi column and encoding categorical variables for model training. The dataset structure and data types are key factors that will shape the feature engineering process for accurate brain stroke prediction.

The target class contains results as stroke(1) and non stroke(0).

The target variable, labeled as stroke, is a binary class representing whether a person has experienced a stroke. It is encoded with values 1 and 0, where:

- 1 indicates that the person has had a stroke, marking them as part of the positive class.
- 0 indicates no history of stroke, marking them as part of the negative class.

3.4.2.2 Exploring Cardiovascular Dataset

The Cardiovascular Disease (CVD) dataset from Kaggle includes 70,000 entries with 13 columns: 1 identifier (id), 11 numerical features, and 1 class variable (cardio). The features represent medical and lifestyle factors closely associated with cardiovascular health. These include age (in days), gender (1 = female, 2 = male), height (cm), weight (kg), systolic and diastolic blood pressure, cholesterol and glucose levels, smoking status, alcohol intake, and physical activity.

The id column serves as a unique identifier for each individual and is excluded from model training, as it holds no predictive value. All feature columns are numerical, simplifying preprocessing and making the dataset suitable for machine learning models.

In the exploratory phase, we analyze feature distributions, identify outliers, and handle any missing or inconsistent values. We also assess the significance of each feature in predicting the binary outcome (cardio: 1 = presence, 0 = absence of disease), aiding in feature selection and model refinement. These steps help reveal underlying patterns and correlations critical to understanding CVD risk.

Blood pressure readings are recorded through two variables:

- systolic blood pressure (ap_hi), which represents the pressure in the arteries when the heart beats
- diastolic blood pressure (ap_lo), which represents the pressure when the heart is at rest.

These are critical indicators of cardiovascular health, as abnormal blood pressure levels can be linked to heart disease and stroke. The dataset also captures cholesterol and glucose levels, each categorized as 1 (normal), 2 (above normal), and 3 (well above normal). Elevated levels of these indicators are established risk factors for cardiovascular disease.

Lifestyle habits such as smoking, alcohol consumption, and physical activity are represented as binary variables: 1 (Yes) and 0 (No). These behavioral attributes significantly impact cardiovascular health and are critical in risk prediction.

The target variable, cardio, is binary—0 indicating no cardiovascular disease and 1 indicating its presence. This classification enables supervised machine learning models to predict disease

risk based on an individual's health profile.

Overall, the dataset offers a well-rounded set of medical and lifestyle features, making it suitable for building effective predictive models. Analyzing these variables allows for pattern recognition and risk assessment, aiding early diagnosis and supporting preventive healthcare initiatives.

Dataset:https://www.kaggle.com/code/edameral/cardiovascular-disease-ml

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0

Figure 3.5: Snapshot of cardio_train.csv

```
[8] df.describe()
```

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
count	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000
mean	49972.419900	19468.865814	1.349571	164.359229	74.205690	128.817286	96.630414	1.366871	1.226457	0.088129	0.053771	0.803729	0.499700
std	28851.302323	2467.251667	0.476838	8.210126	14.395757	154.011419	188.472530	0.680250	0.572270	0.283484	0.225568	0.397179	0.500003
min	0.000000	10798.000000	1.000000	55.000000	10.000000	-150.000000	-70.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000
25%	25006.750000	17664.000000	1.000000	159.000000	65.000000	120.000000	80.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000
50%	50001.500000	19703.000000	1.000000	165.000000	72.000000	120.000000	80.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000
75%	74889.250000	21327.000000	2.000000	170.000000	82.000000	140.000000	90.000000	2.000000	1.000000	0.000000	0.000000	1.000000	1.000000
max	99999.000000	23713.000000	2.000000	250.000000	200.000000	16020.000000	11000.000000	3.000000	3.000000	1.000000	1.000000	1.000000	1.000000

Figure 3.6: Description of cardio_train.csv

```
[7]  df.shape

     (70000, 13)
```

Figure 3.7: Shape of cardio_train .csv

The dataset used for Cardiovascular Disease Prediction consists of 70,000 entries across 13 columns, offering a comprehensive set of features related to patient demographics and health indicators. The data types in the dataset include:

- **Integer (int64):** Most columns, including id, age, gender, height, ap_hi, ap_lo, cholesterol, gluc, smoke, alco, active, and cardio, are stored as integers. These

variables represent categorical and numerical data, such as blood pressure measurements, cholesterol levels, and binary indicators for smoking status, alcohol consumption, and physical activity.

- **Float (float64):**The weight column is stored as a float to accommodate decimal values for continuous variables like body weight.

The target variable is cardio.The cardio column serves as the dependent variable in this dataset. It is a binary classification variable:

- **1 (Positive Class):** Indicates the presence of cardiovascular disease.
- **0 (Negative Class):** Indicates the absence of cardiovascular disease

```
[ ] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5110 non-null   int64
 1   gender             5110 non-null   object
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64
 4   heart_disease      5110 non-null   int64
 5   ever_married       5110 non-null   object
 6   work_type          5110 non-null   object
 7   Residence_type     5110 non-null   object
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object
 11  stroke             5110 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

Figure 3.8: Datatypes of cardio_train .csv

This binary classification setup facilitates the development of predictive models that can analyze patterns and relationships within the dataset, helping to identify individuals at risk of cardiovascular disease based on health metrics such as age, cholesterol level, and blood pressure. The dataset's structure, with numerical features and a clear binary target, is ideal for training and evaluating machine learning models for cardiovascular disease prediction.

3.5 Data Preprocessing

Data preprocessing is a crucial step in machine learning and data analysis that involves transforming raw data into a clean and structured format before it is used for modeling. Raw datasets often contain missing values, inconsistencies, duplicate records, and noise, which can negatively impact the performance of predictive models. Preprocessing ensures that the data is accurate, consistent, and suitable for analysis, ultimately leading to more reliable and efficient machine learning models.

3.5.1 Data Cleaning

3.5.1.1 Data Cleaning of Stroke Dataset

Remapping Target Class

The proposed system will classify brain stroke risk into two classes: ”stroke” and ”no stroke.” The target variable is mapped to these labels, identifying individuals either at risk of stroke or not at risk of stroke.

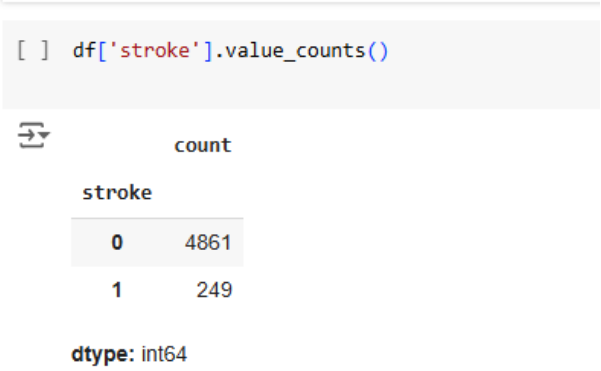


Figure 3.9: Mapping target class values

Missing Values
There are missing values in the dataset.

```
df.isna().sum()
```

	0
id	0
gender	0
age	0
hypertension	0
heart_disease	0
ever_married	0
work_type	0
Residence_type	0
avg_glucose_level	0
bmi	201
smoking_status	0
stroke	0

dtype: int64

Figure 3.10: Check for missing values

In this dataset, missing values were identified in the ”bmi” column, a critical feature used in predicting stroke risk.

To address these missing values, the K-Nearest Neighbors Imputer (KNN Imputer) method from the sklearn library was employed. This imputation method leverages the similarity between data points to estimate and fill in missing values based on their nearest neighbors. The detailed steps of this imputation process are outlined below:

- Selecting the KNN Imputer: The KNN Imputer was chosen for its effectiveness in handling continuous data by considering the values of neighboring samples. In this project, the number of neighbors was set to 5, meaning the imputer calculates each missing ”bmi” value by averaging the ”bmi” values of the 5 nearest (most similar) records

in the dataset. This number of neighbors was selected to provide a balanced imputation, considering enough data points while avoiding potential bias from outliers.

- Applying Imputation: The imputer was applied specifically to the ”bmi” column. This transformation replaced missing values with imputed values directly within the dataset, resulting in a more complete dataset that retains all instances for training and analysis.
- Verification of Missing Values: After the imputation, it was necessary to verify that no missing values remained in the dataset. A summary of missing values for each column confirmed that all missing values in the ”bmi” column were successfully imputed, with no remaining missing values in the entire dataset. This ensures that the data is complete and ready for further analysis and model training.

```
[ ]  imputer = KNNImputer(n_neighbors = 5)
     df['bmi'] = imputer.fit_transform(df[['bmi']])
```

Figure 3.11: Handling missing values

Handling class imbalance

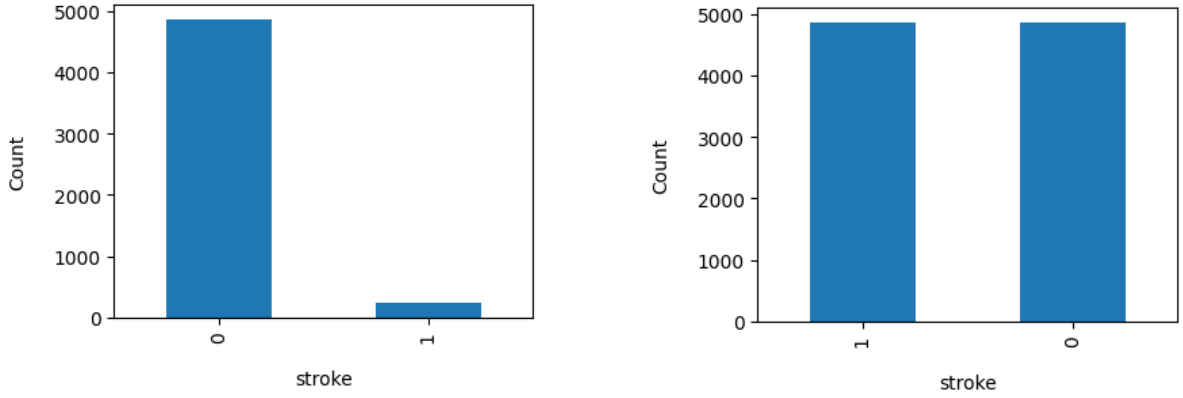
Target class is highly imbalanced. There are large number of people with no stroke and comparatively small number of people with stroke .

```
from imblearn.over_sampling import RandomOverSampler

X = df.drop('stroke', axis=1)
y = df['stroke']
ros = RandomOverSampler(random_state=42)
X_res, y_res = ros.fit_resample(X, y)
df = pd.concat([X_res, y_res], axis=1)
```

Figure 3.12: Random oversampling

Handling class imbalance with oversampling is a technique often used in classification tasks where one class has significantly fewer samples than others. Oversampling techniques create synthetic samples for the minority class to balance the dataset, making the model less biased toward the majority class.



(a) Imbalanced target class (b) Balanced target class

Figure 3.13: Handling class imbalance

3.5.1.2 Data Cleaning of Cardiovascular Dataset

Converting age from days to years

Age in days is unnecessarily detailed and hard to interpret. Converting it into years improves readability and aligns with common medical practices where age is discussed in years.

```
In [4]: df['age']=df['age'].apply(lambda x: x/365).astype(int)
```

Figure 3.14: Age convertion

Converting the height from centimeters to meters

The attribute height from centimeters to meters, for ease in bmi feature creation

```
In [5]: df['height']=df['height'].apply(lambda x: x/100)
```

Figure 3.15: Height convertion

Converting existing attributes

The process of converting existing attributes, such as cholesterol and gluc, into categorical values or other forms involves aligning the dataset with domain knowledge and improving its usability for machine learning models.

Attributes like cholesterol and gluc have ordinal properties, so lets convert them to categorical:

```
[20] df['cholesterol']=df['cholesterol'].apply(lambda x: 'normal' if x==1 else('above_normal' if x==2 else 'well_above_normal'))
     df['gluc']=df['gluc'].apply(lambda x: 'normal' if x==1 else('above_normal' if x==2 else 'well_above_normal'))
```

Figure 3.16: Categorical convertion of cholestrol and gluc

- Both cholesterol and gluc have ordinal properties (ordered levels: $1 < 2 < 3$), but when represented as integers, their meaning is not clear.
- Converting them into descriptive categories (normal, above_normal, well_above_normal) improves interpretability.
- Gender is categorical (male and female), but numerical encoding can confuse models that treat it as continuous data.
- Using one-hot encoding or keeping it as a category ensures models handle it correctly as a binary attribute.

Box-Cox Transformation on Age

Age is a critical predictor of cardiovascular disease, but if its distribution is skewed, the model might have difficulty capturing the relationship properly. Box-Cox transformation normalizes the age distribution, making the feature more suitable for models sensitive to data distributions (e.g., regression-based models).

```
[ ]  age=np.array(df['age'])
     _, opt_lambda=boxcox(age)
     print(opt_lambda)

     2.0568229424754394

[ ]  df['age_box']=boxcox(df['age'], lmbda=opt_lambda)
```

Figure 3.17: Box-Cox transformation of age

By normalizing age, the model can better utilize it to detect subtle patterns in cardiovascular risk across different age groups.

3.5.2 Analysis of Feature Variables

3.5.2.1 Analysis of Feature Variables of Stroke Dataset

Feature variables in the dataset includes:

- **id** – Unique identifier for each patient.
- **gender** – Categorical variable: ”Male”, ”Female”, or ”Other”.
- **age** – Age of the patient.
- **ever_married** – Categorical variable: ”No” or ”Yes”.
- **work_type** – Categorical variable: ”Children”, ”Govt_job”, ”Never_worked”, ”Private”, or ”Self-employed”.
- **Residence_type** – Categorical variable: ”Rural” or ”Urban”.
- **avg_glucose_level** – Average glucose level in the blood.
- **bmi** – Body Mass Index (BMI).
- **smoking_status** – Categorical variable: ”Formerly smoked”, ”Never smoked,” ”Smokes”, or ”unknown (unknown means information is unavailable).
- **stroke (Target Variable)** – Binary outcome(1: patient had a stroke, 0: no stroke).

Note: ”Unknown” in smoking_status means that the information is unavailable for this patient.

3.5.2.2 Analysis of Feature Variables of Cardiovascular Dataset

Feature variables in the dataset includes:

- **id**: Personal identification number (usually not used in analyses).
- **age**: Age (given in days).
- **gender**: Sex (1: female, 2: male).
- **height**: Height (cm).
- **weight**: Weight (kg).
- **ap_hi**: Systolic blood pressure.
- **ap_lo**: Diastolic blood pressure.
- **cholesterol**: Cholesterol level (1: normal, 2: above normal, 3: very high).
- **gluc**: Glucose level (1: normal, 2: above normal, 3: very high).

- **smoke**: Smoking status (1: yes, 0: no).
- **alco**: Alcohol consumption (1: yes, 0: no).
- **active**: Physically active status (1: yes, 0: no).
- **cardio**: Cardiovascular disease status (1: yes, 0: no).

3.5.3 Analysis of Class Variables

3.5.3.1 Analysis of Class Variables of Stroke dataset

In this stroke prediction project, the class variable is structured as a binary classification, with two distinct labels: 0 (no stroke risk) and 1 (stroke risk). Here, label 0 represents individuals who are not at risk of a stroke, while label 1 denotes those who are at risk. This binary structure provides a streamlined and targeted approach to predict stroke likelihood, enabling the model to focus on differentiating at-risk patients from those who are not.This classification setup not only simplifies the predictive task but also facilitates clearer, actionable insights. By reducing the classification process to a binary outcome, the model efficiently identifies patients who require closer medical attention. This is especially critical in healthcare scenarios, where a binary outcome aids in prioritizing cases for preventive measures or further assessment. Additionally, the binary setup allows for the application of various machine learning classification algorithms—such as logistic regression, decision trees, and support vector machines—which are well-suited to handling binary outcomes, thereby optimizing model accuracy and interpretability.

3.5.3.2 Analysis of Class Variables of Cardiovascular dataset

The class variable in the Cardiovascular Disease (CVD) dataset, labeled as ”cardio,” serves as the binary target for prediction: 0 indicates no CVD, and 1 indicates the presence of CVD. This classification supports machine learning models in predicting CVD risk based on health-related attributes. As CVD is a major global cause of mortality, early detection through predictive modeling can assist in preventive healthcare and timely intervention. The class variable is vital in supervised learning, especially in classification tasks. The dataset is imbalanced, with more non-CVD cases, so techniques like oversampling, undersampling, or SMOTE are used to prevent bias toward the majority class. Accurate classification allows analysis of risk factors like age, cholesterol, blood pressure, and lifestyle habits (smoking, alcohol intake, physical activity), enabling better CVD risk assessment and prevention strategies.

3.6 Data Visualization

3.6.1 Data Visualization of Stroke Dataset

• Gender

Females are more likely to be affected by Brain stroke.

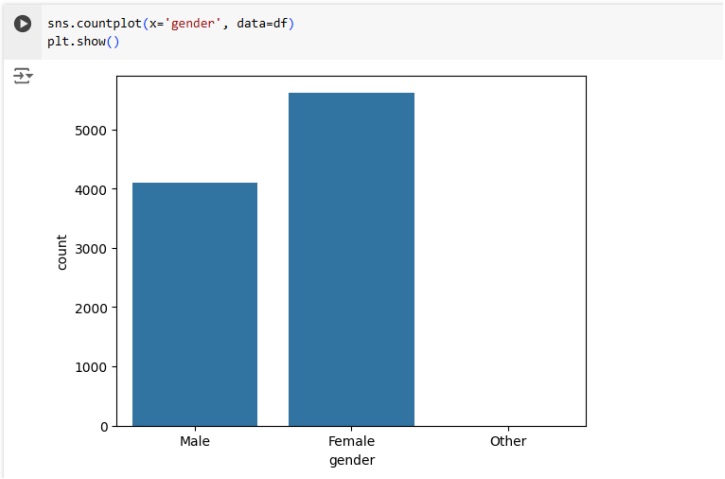


Figure 3.18: Distribution of Gender

• Ever_married

Those who married have the probability of higher chance of stroke .

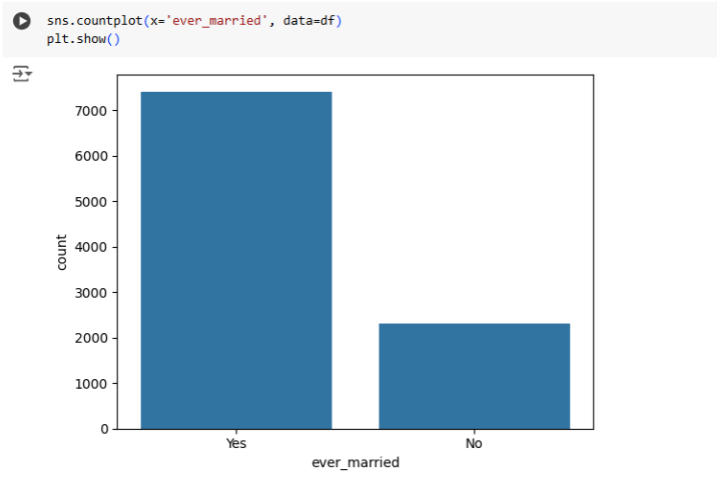


Figure 3.19: Distribution of married and non married

• Work_type

Work type classified into 5 and those who are doing private job are most likely to be affected by brain stroke.

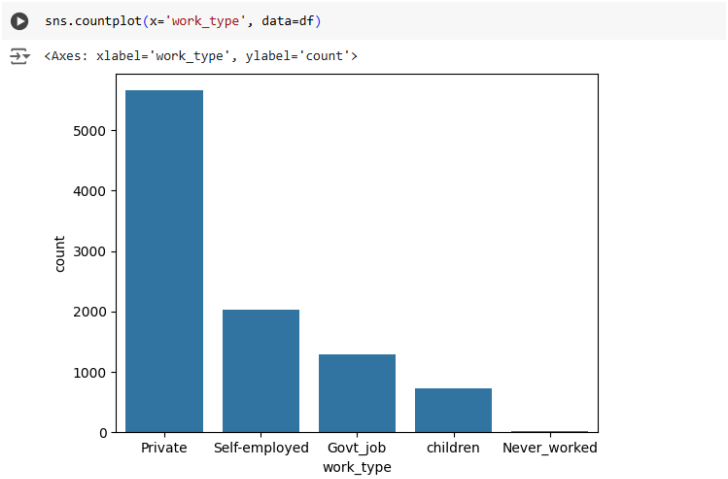


Figure 3.20: Distribution of work_type

•Smoking_status

Smoking status classified into formerly smoked,never smoked,smokes, and unknown.

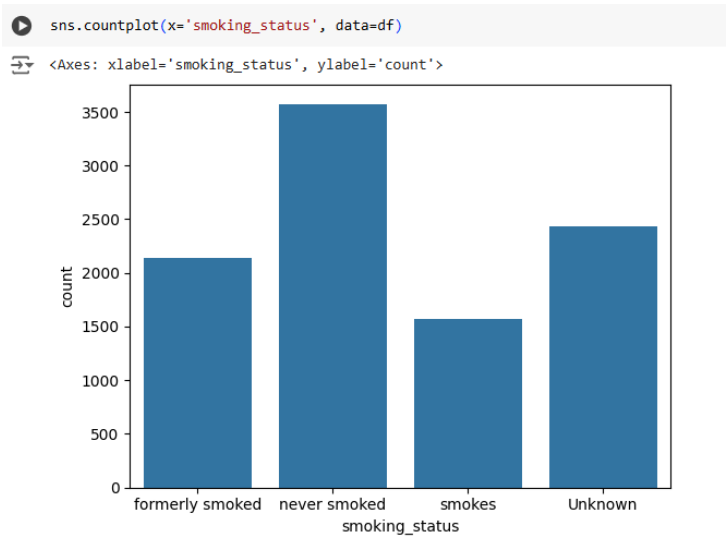


Figure 3.21: Distribution of smoking_status

•Residence_Type
Residence type is similarly distributed for stroke occurrence.

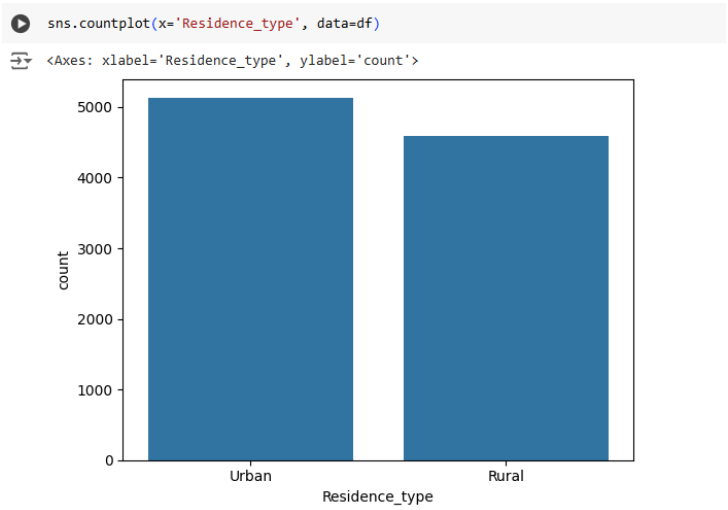


Figure 3.22: Distribution of Residence_Type

3.6.2 Data Visualization of Cardiovascular Dataset

•Gender

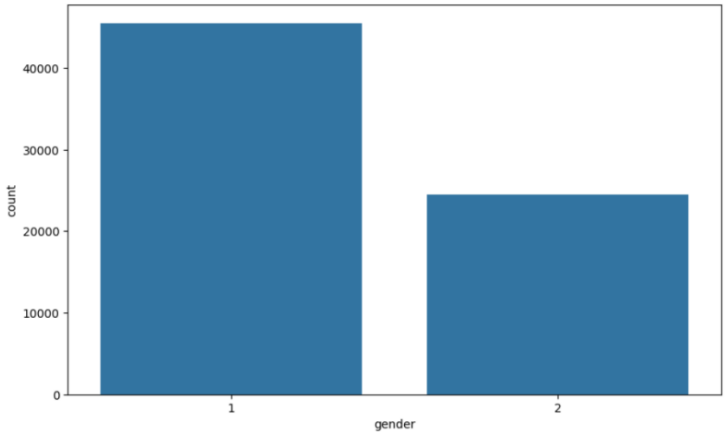


Figure 3.23: Distribution of gender in cvd dataset

•Cholestrol

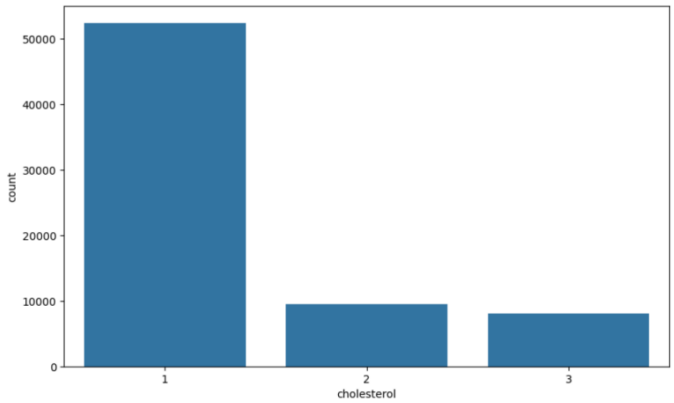


Figure 3.24: Distribution of cholesterol in cvd dataset

Violin chart of age and cardio

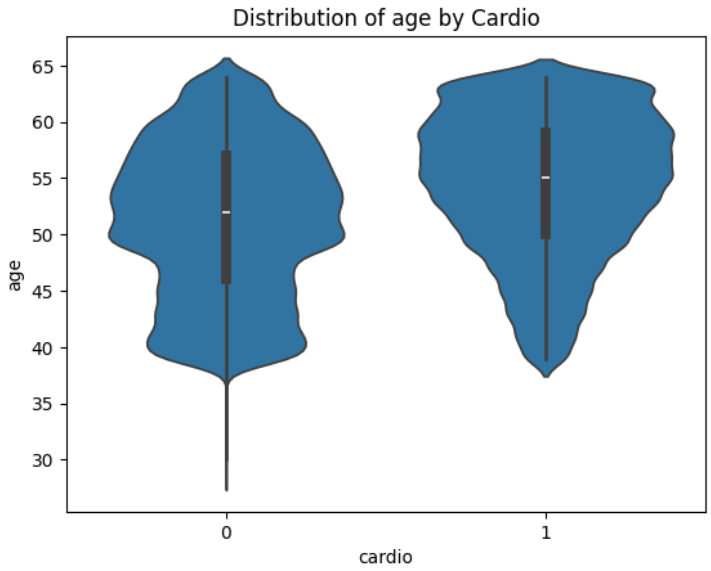


Figure 3.25: Violin chart of age and cardio

Age plays a critical role in the likelihood of cardiovascular disease:Individuals with cardiovascular disease (cardio = 1) tend to be older.Younger individuals are more likely to be in the ”no cardiovascular disease” group (cardio = 0).This supports the hypothesis that age is a significant risk factor for cardiovascular disease, as older individuals are more prone to it.

Data Distribution

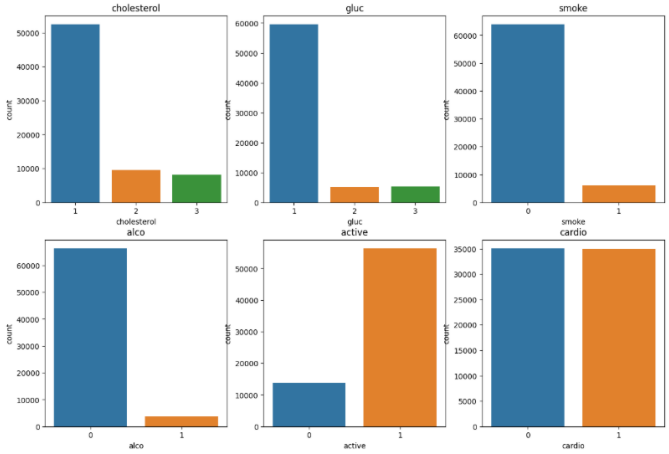


Figure 3.26: Data Distribution in CVD dataset

There is significant imbalance within the categories of each attribute. To avoid any issues , upsampling is .Upsampling (also known as oversampling) involves increasing the number of samples in the minority class by duplicating or synthetically generating data. This balances the class distribution and improves the model's ability to detect minority class patterns.

3.7 Analysis of Algorithms

3.7.1 Analysis of Algorithms of Stroke prediction

Algorithms used in 'Brain Stroke Using Machine Learning' Random forest Classifier.

Random Forest

Random Forest is a powerful supervised machine learning algorithm that can be used for both classification and regression tasks. It belongs to the family of ensemble learning methods, which aim to combine the predictive power of multiple models to produce a stronger overall model.

At the core of Random Forest is the Decision Tree—a model that splits the dataset based on feature values to make predictions. However, relying on a single decision tree can lead to overfitting, especially when the model becomes too complex. Random Forest addresses this by creating a "forest" of decision trees, each trained on a different subset of the training data.

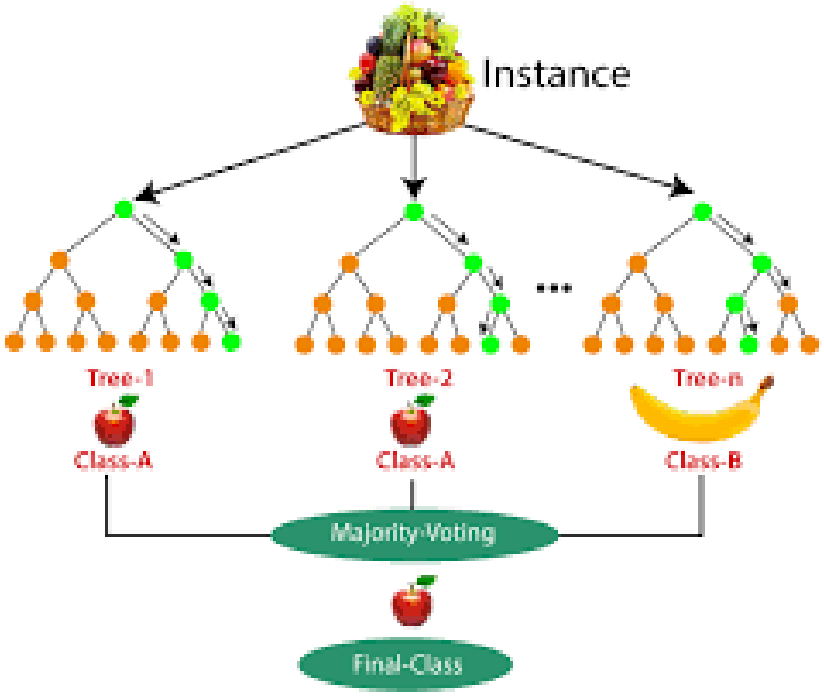


Figure 3.27: Random forest diagram

Working

Step-1: Select random K data points from the training set.
Step-2: Build the decision trees associated with the selected data points (Subsets).
Step-3: Choose the number N for decision trees that you want to build.
Step-4: Repeat Step 1 & 2.
Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

Pseudocode

1. Input:

   - Training dataset with N samples and M features.
   - Number of trees to create: T.
   - Number of features to randomly select at each split: F.

2. Initialize an empty list called ‘forest‘ to store the trees.

3. For each tree (from 1 to T):

   a. Create a random sample of the training data (some samples may repeat).

   b. Build a decision tree:

      i. At each decision point in the tree:
         * Randomly pick F features.
         * Find the best feature and split point among the F features to split the node.
         * Split the node into two child nodes based on the selected split point.

      ii. Keep splitting until the tree is fully grown or meets some stopping condition (like max depth or minimum samples).

   c. Add this decision tree to the ‘forest‘.

4. To make a prediction for a new sample x:

   a. Let each tree in the forest make a prediction.

   b. The final prediction is the one that most trees agree on (majority vote).

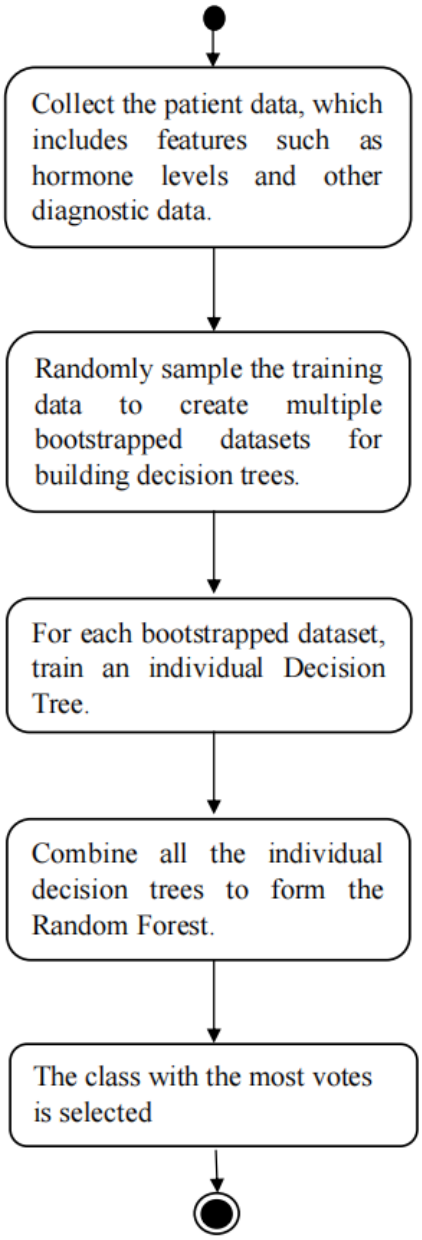


Figure 3.28: Activity Diagram of Random Forest

Random Forest is an ensemble learning technique that constructs multiple decision trees during training and combines their outputs to improve classification accuracy. By training each tree on a random subset of the data, Random Forest captures diverse patterns and reduces the risk of overfitting that can occur with individual trees. The final prediction is made based on the majority vote from all trees, ensuring robustness and stability. Random Forest is particularly effective for analyzing large datasets with high dimensionality, as it can identify complex relationships between features. In healthcare applications like cardiovascular disease prediction, it can analyze attributes such as age, blood pressure, cholesterol levels, and lifestyle factors to predict disease risk accurately. Additionally, it provides insights into feature importance, which can help identify critical variables influencing the predictions, making it both a powerful and interpretable model.

3.7.2 Analysis of Algorithms of Cardiovascular disease prediction

The algorithm used in 'Cardiovascular disease prediction using Machine Learning' is Logistic Regression.

Logistic Regression

Logistic regression comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables. Logistic regression has several variants, including binary logistic regression, multinomial logistic regression, and ordinal logistic regression. Logistic regression is used for solving the classification problems.

Logistic Function (Sigmoid Function)

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.

Working

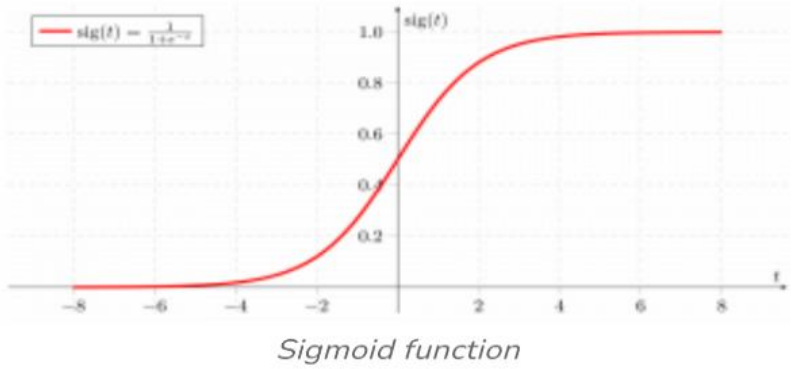
- Logistic regression works by creating a linear combination of input features, which involves multiplying each feature by a coefficient and summing the results. Applies the multi-linear function to the input variables:

$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b \tag{3.1}$$

x_i is the ith observation of X. $\vec{w} = [w_1, w_2, w_3, \cdots, w_m]$ is the weight vector or coefficient. b is the bias term, also known as the intercept.

- This value is then passed through the logistic $sigmoid$ function, which transforms it into a probability value between 0 and 1. Z will be input to the sigmoid function and find the probability between 0 and 1.

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



Sigmoid function

Figure 3.29: Logistic Regression diagram

- Sigmoid function converts the continuous variable data into the probability i.e. between 0 and 1.
- For binary classification, this probability is compared to a threshold (usually 0.5) to decide the class label.

Multinomial logistic regression, also known as softmax regression, is used for multi-class classification tasks, where there are more than two possible outcomes for the output variable.In this case, the softmax function is used in place of the sigmoid function.
Softmax function for K classes will be:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

- The calculated probabilities will be in the range of 0 to 1.
- The sum of all the probabilities is equals to 1.
- The class with highest probability will be the final output.

Pseudocode:

1. **Input:**
   - Training data with N samples and M features.
   - Labels (0 or 1) for each sample.
   - Learning rate (how big of a step to take when updating weights).
   - Number of iterations (how many times to update weights).
2. **Initialize:**
   - Set weights and bias to 0 (or small random numbers).
3. **For a set number of iterations:**
   a. For each sample in the training data:
      i. Calculate the weighted sum (z):

$$z = (\text{weight}_1 \times \text{feature}_1) + (\text{weight}_2 \times \text{feature}_2) + ... + (\text{weight}_M \times \text{feature}_M) + \text{bias}$$

      ii. Apply the sigmoid function to get a probability:

$$\text{probability} = \frac{1}{1 + e^{-z}}$$

      iii. Calculate the error:

$$\text{error} = \text{probability} - \text{actual label}$$

      iv. Update each weight:

$$\text{weight}_j = \text{weight}_j - (\text{learning rate} \times \text{error} \times \text{feature}_j)$$

      v. Update the bias:

$$\text{bias} = \text{bias} - (\text{learning rate} \times \text{error})$$

4. **After finishing all iterations, use the weights and bias to make predictions:**
   a. For a new sample:
      i. Calculate the weighted sum (z) using the final weights and bias.
      ii. Apply the sigmoid function to get a probability.
      iii. If the probability is 0.5 or higher, output class 1. Otherwise, output class 0.

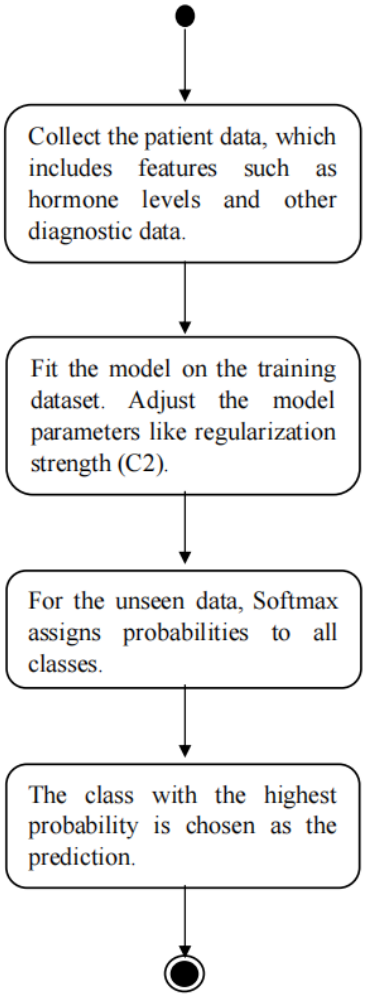


Figure 3.30: Activity Diagram of Logistic Regression

The flowchart outlines the step-by-step process of a machine learning-based medical prediction system. It begins with the collection of patient data, including diagnostic features such as hormone levels and other relevant health indicators. This data is then used to train a predictive model, where parameters like regularization strength (C2) are fine-tuned to optimize performance. Once the model is trained, it processes unseen data using the Softmax function, which calculates the probability of each possible class. Finally, the class with the highest probability is selected as the predicted outcome. This structured approach ensures accurate and data-driven predictions in medical diagnosis applications.

3.8 Project Pipeline

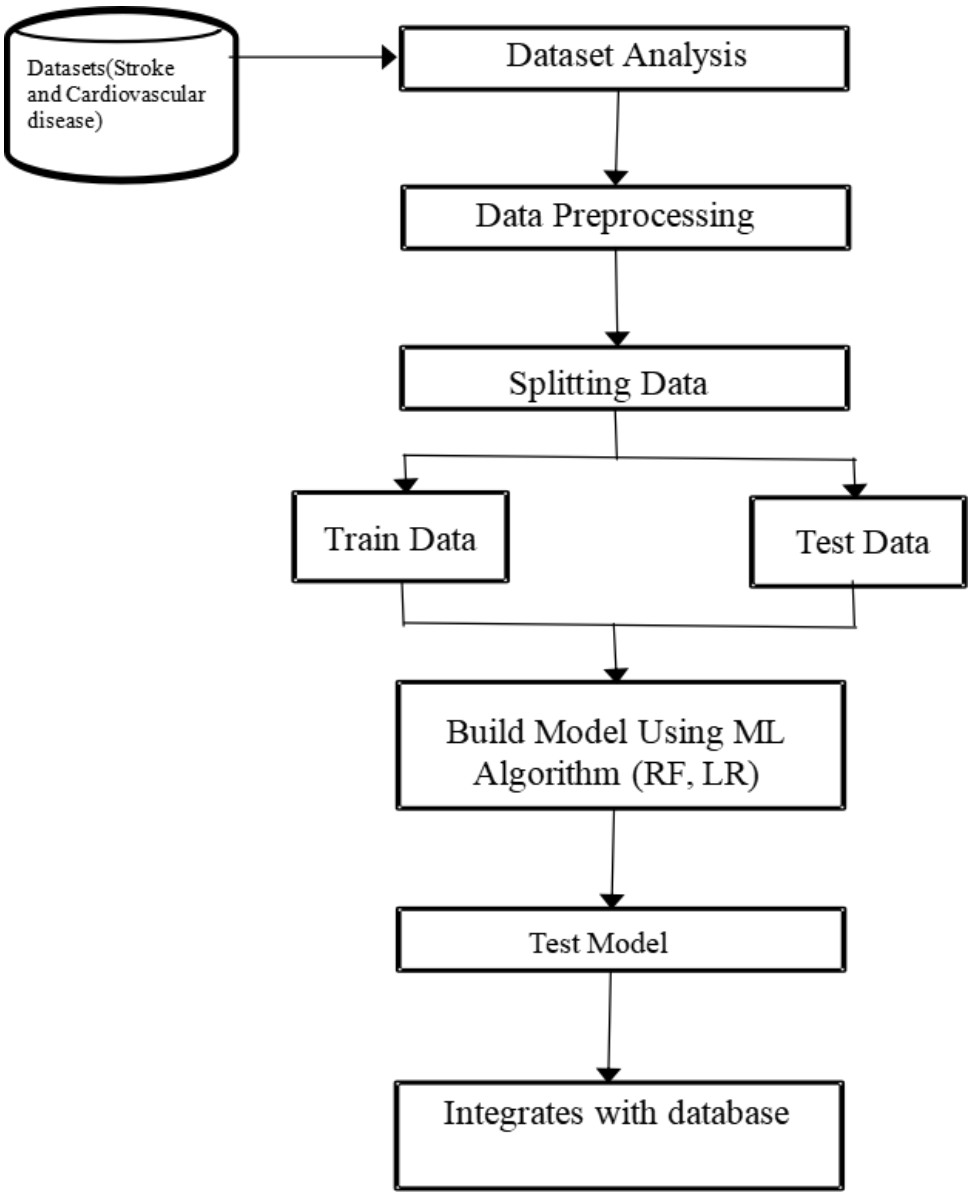


Figure 3.31: Project pipeline

3.9 Feasibility Analysis

A feasibility study aims to objectively and rationally uncover the strengths and weaknesses of an existing system or proposed system, opportunities and threats present in the natural environment,the resources required to carry through, and ultimately the prospects for success. Evaluated the feasibility of the system in terms of the following categories:

- Technical Feasibility
- Economic Feasibility
- Operational Feasibility

3.9.1 Technical Feasibility

Technical Feasibility assesses whether the required technologies are accessible and if the development team can effectively implement them. The ”Elderly Caregiver Matchmaking System” demonstrates strong technical feasibility through the use of widely adopted tools and frameworks.The front-end is built using HTML, CSS, and JavaScript, ensuring compatibility and responsiveness across devices. The back-end leverages Python Flask, a lightweight yet powerful framework that supports modular development and seamless integration. For the database, SQLite is used due to its simplicity, zero-configuration setup, and suitability for rapid development and prototyping.

Chatbot Integration enhances usability, particularly for elderly users needing navigation assistance. It will be implemented using Python libraries like ChatterBot or third-party APIs such as Dialogflow. Flask’s flexibility facilitates smooth integration, allowing the chatbot to deliver real-time, rule-based or conversational support efficiently. Payment Gateway Integration is achieved using platforms such as Razorpay or Stripe. These APIs are Flask-compatible and support secure financial transactions in compliance with standards like PCI-DSS. The use of encryption protocols (e.g., SSL/TLS) further ensures the protection of sensitive data during transactions.The use of Python also enables integration of machine learning functionalities using libraries like Scikit-learn, Pandas, and TensorFlow. This allows implementation of intelligent features such as health prediction and caregiver matching. Flask’s routing and component control, combined with Python’s versatility, ensure a robust back-end workflow.

The system is designed to be accessible across smartphones, tablets, and desktops, offering a consistent user experience. The front-end delivers intuitive interaction, while the back-end ensures efficient data handling and prediction capabilities.Moreover, the architecture supports future scalability and expansion, allowing for the addition of new modules or handling higher user loads without significant redesign. Overall, the project is technically feasible, backed by mature technologies, strong community support, and secure, scalable design principles.

3.9.2 Economical Feasibility

Economic Feasibility of the ”Elderly Caregiver Matchmaking System” is strongly supported by the use of open-source technologies, which significantly reduce development and operational costs. HTML, CSS, and JavaScript enable the creation of a responsive front-end without the need for proprietary tools. On the back-end, Python Flask offers a flexible, lightweight framework that accelerates development while keeping resource usage low.SQLite serves as the database engine, offering a free, embedded, zero-configuration solution with ACID compliance. It is included by default with Python, eliminating licensing and setup costs—making it ideal for small to medium-scale applications.Machine learning capabilities are integrated using free Python libraries like NumPy, Pandas, and Scikit-learn. These tools allow the implementation of advanced features such as caregiver matching and health prediction without incurring additional expenses.Hosting can be managed via free or low-cost platforms such as Render, Heroku (free tier), or PythonAnywhere. These services provide adequate resources for development and early-stage deployment, helping minimize infrastructure expenses.

Overall, by leveraging free, open-source tools and efficient platforms, the system maintains low upfront and ongoing costs. This makes the project not only technically achievable but also economically sustainable and scalable.

3.9.3 Operational Feasibility

Operational Feasibility assesses whether the project fits practical usage scenarios and integrates smoothly with day-to-day processes. The Elderly Caregiver Matchmaking System addresses real-world challenges by offering a user-friendly, accessible platform tailored to elderly users and caregivers.The system prioritizes ease of use through an intuitive interface, responsive design, and simple navigation, making it suitable even for users with limited technical skills. A built-in chatbot further supports operational efficiency by assisting users with navigation and answering queries, reducing the need for external support. This feature is especially valuable for elderly users who may prefer conversational interaction.Secure and streamlined financial transactions are ensured through payment gateway integration with trusted providers like Razorpay or Stripe. This allows in-app payments without redirecting users to third-party platforms, enhancing trust and simplifying the process. The platform’s modular and scalable architecture supports efficient management of key functionalities—such as user registration, caregiver search, skill-based matching, and health assistance. Integration of machine learning models enables proactive health support and disease risk prediction, offering users meaningful health insights.With full compatibility across desktops, tablets, and smartphones, the responsive design ensures users can access the platform anytime, anywhere. This enhances convenience and encourages broader adoption.In summary, the system’s accessible design, operational efficiency, secure processes, and device compatibility make it highly feasible for real-world use, aligning well with user needs and supporting long-term sustainability.

3.10 System Environment

3.10.1 Software Environment

Various software tools and frameworks were utilized in the development of this application to ensure its functionality and performance. By leveraging these software tools and technologies, the development team aimed to create a robust, scalable, and user-friendly application that meets the needs and expectations of its users. The primary technologies employed include:
Python: Python is a high-level programming language that lets developers work quickly and integrate systems more efficiently. This model is developed by using many of the Python libraries and packages such as:

- Pandas: Pandas is an open-source library that is made mainly for working with relational or labelled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series. This library is built on top of the NumPy library.
- Matplotlib: Matplotlib is a cross-platform, data visualization and graphical plotting library for Python. One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. In this application, its used for plotting the graph.
- NumPy: NumPy is a Python library used for working with arrays. NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. In this application, its used for handling arrays
- Scikit-learn: Scikit-learn is a machine learning library used for implementing predictive analytics in the caregiver-matching system, helping to recommend the most suitable caregiver based on user preferences and historical data.

Visual studio code: Visual Studio Code is a streamlined code editor with support for development operations like debugging, task running, and version control. It aims to provide just the tools a developer needs for a quick code-build-debug cycle and leaves more complex workflows to fuller featured IDEs, such as Visual Studio IDE.

HTML and CSS: Hyper Text Markup Language is used for creating web pages.HTML describes the structure of the web page. Here, the user interface of my project is done using HTML. Cascading Style Sheet is used with HTML to style the web pages.

Android Studio: Android Studio is the official Integrated Development Environment (IDE) for developing the SilverSupport Android app. The app is built using Java/Kotlin and incorporates Room Database for locally storing emergency contact details.

Firebase: Firebase Authentication is used for handling user login and registration in the SilverSupport app. It ensures secure authentication and allows users to access their emergency contact details across multiple devices.

JavaScript: JavaScript is frequently employed to validate user input on web forms, ensuring that data submitted by users meets specified criteria and preventing erroneous or malicious submissions.

Flask: Flask is a web framework, it's a Python module that lets you develop web applications easily. It's has a small and easy-to-extend core: it's a microframework that doesn't include an ORM (Object Relational Manager) or such features. It does have many cool features like URL routing, template engine. It is a WSGI web app framework.

SQLite: The database system is designed using SQLite (for web-based components and for mobile app storage). It stores user details, caregiver profiles, caregiver ratings, and emergency contacts, ensuring structured and efficient data management.

Web Browser: Google Chrome (Version 135.0.7049.42/52) is used for testing and accessing the web-based components of the Elderly Caregiver Matchmaking System.

GitHub: GitHub is a web-based platform that uses Git for version control, enabling developers to host, manage, and track changes to their code repositories. It supports collaboration through features like issue tracking, pull requests, and wikis, making it ideal for team-based software development. Developers also use GitHub to contribute to open-source projects, manage project workflows, and showcase their work. As a central tool in modern development, GitHub streamlines collaboration, code review, and project management.

3.10.2 Hardware Environment

Hardware configuration significantly impacts software development, influencing performance and scalability. Choices regarding CPU, RAM, storage, and network infrastructure directly affect application execution and data handling. Moreover, considerations extend to system architecture and compatibility, ensuring efficiency and reliability in modern software solutions.

- Processor: 11th Gen Intel(R) Core(TM) 3.00 GHz
- Memory: 512 GB SSD
- RAM : 8GB
- System Type : 64-bit operating system, x64-based processor
- Good internet connectivity.

Android App:

- Device: Android 8.0+
- RAM: 4GB
- Processor: Snapdragon 845 or better
- Screen Resolution: 1080p+

4 SYSTEM DESIGN

4.1 Use case model

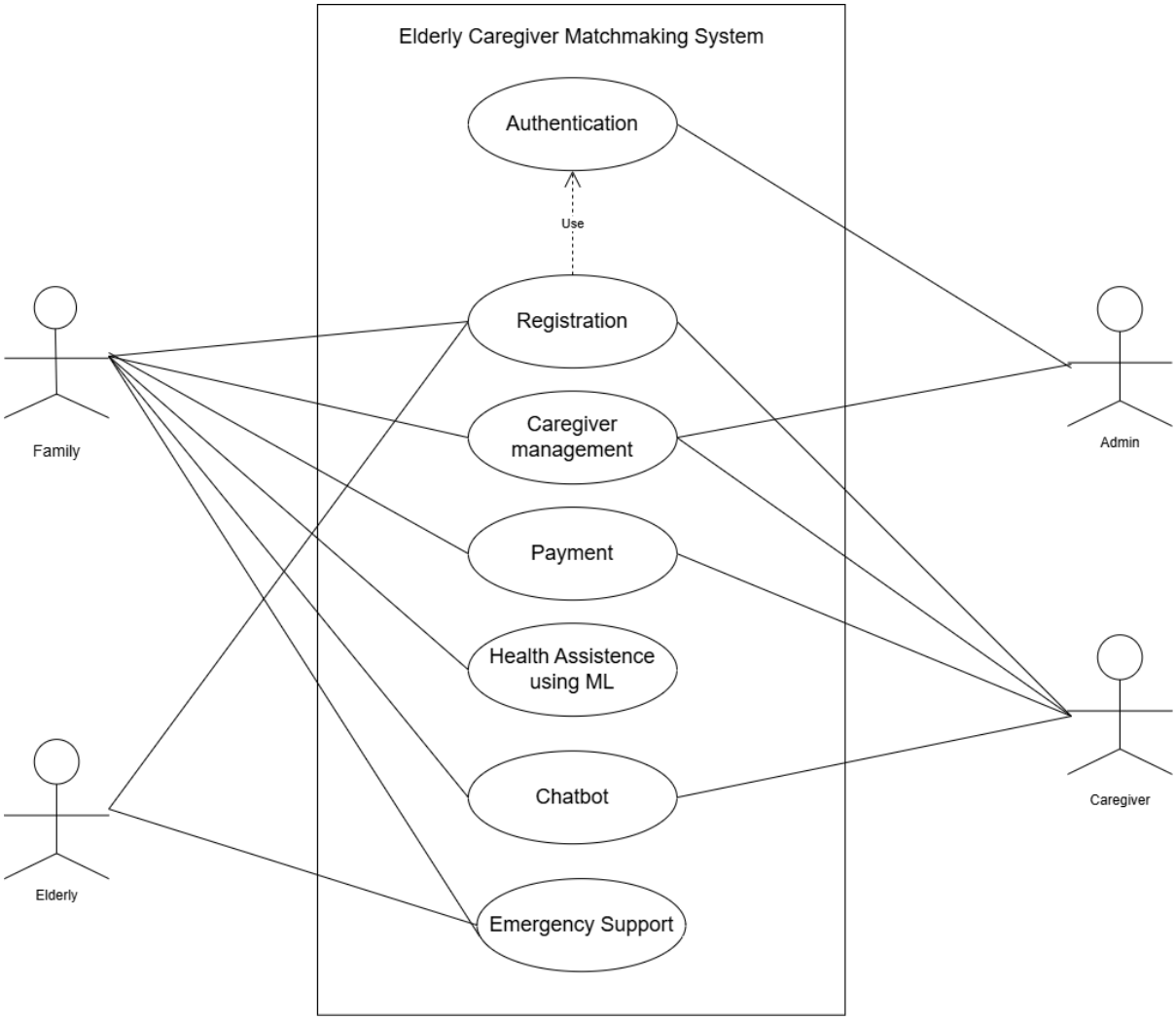


Figure 4.1: Use case diagram

The use case diagram for the Elderly Caregiver Matchmaking System provides a clear representation of how different user roles interact with the system's various functionalities. It consists of four main actors: Family, Elderly, Caregiver, and Admin, each having distinct roles and interactions with the system.

Both the Family and Elderly users can access core functionalities such as Registration and Authentication to create and manage their accounts securely. Authentication ensures that only registered users can use the system.A key function for families is Caregiver Management, where they can search for caregivers based on specific preferences such as skills, experience, availability, and location. After finding a suitable caregiver, they can proceed with the Payment process to finalize the booking.Another critical feature for elderly users is Emergency Support, allowing them to send alerts to family members or emergency contacts in case of distress. This ensures immediate assistance when required.The Health Assistance using ML (Machine Learning) module may provide recommendations or predictive insights related to the elderly user's health based on input data.

Caregivers register on the platform and interact with Caregiver Management, where they can manage their profiles, update availability, and respond to booking requests from families.Once a booking request is received, caregivers have the choice to accept or decline. If they accept, they will proceed with the Payment module, where they receive payments for their services.

The Admin has an overarching role in managing the platform, primarily handling User Registration and Authentication to ensure that only verified users can access the system.They also oversee Caregiver Management, likely approving caregiver profiles and ensuring quality service providers are available on the platform.

4.2 Activity Diagram

Booking and payment

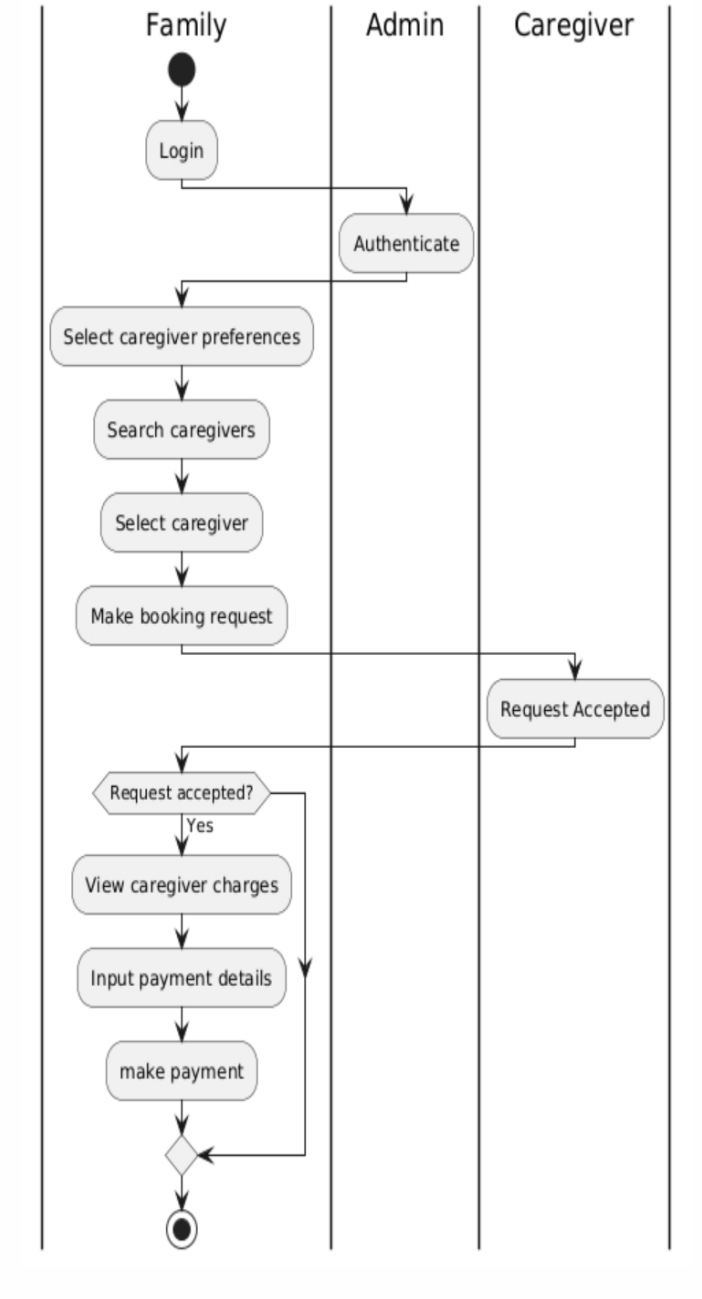


Figure 4.2: Booking and payment activity diagram

This activity diagram outlines the process of booking a caregiver and making a payment. The process starts with a family member logging in to the system. The family then sets preferences, searches for caregivers, and selects a suitable caregiver. A booking request is sent, and the caregiver has the option to accept the request. If accepted, the family views the caregiver's charges, inputs payment details, and completes the payment to finalize the booking.

Emergency Support

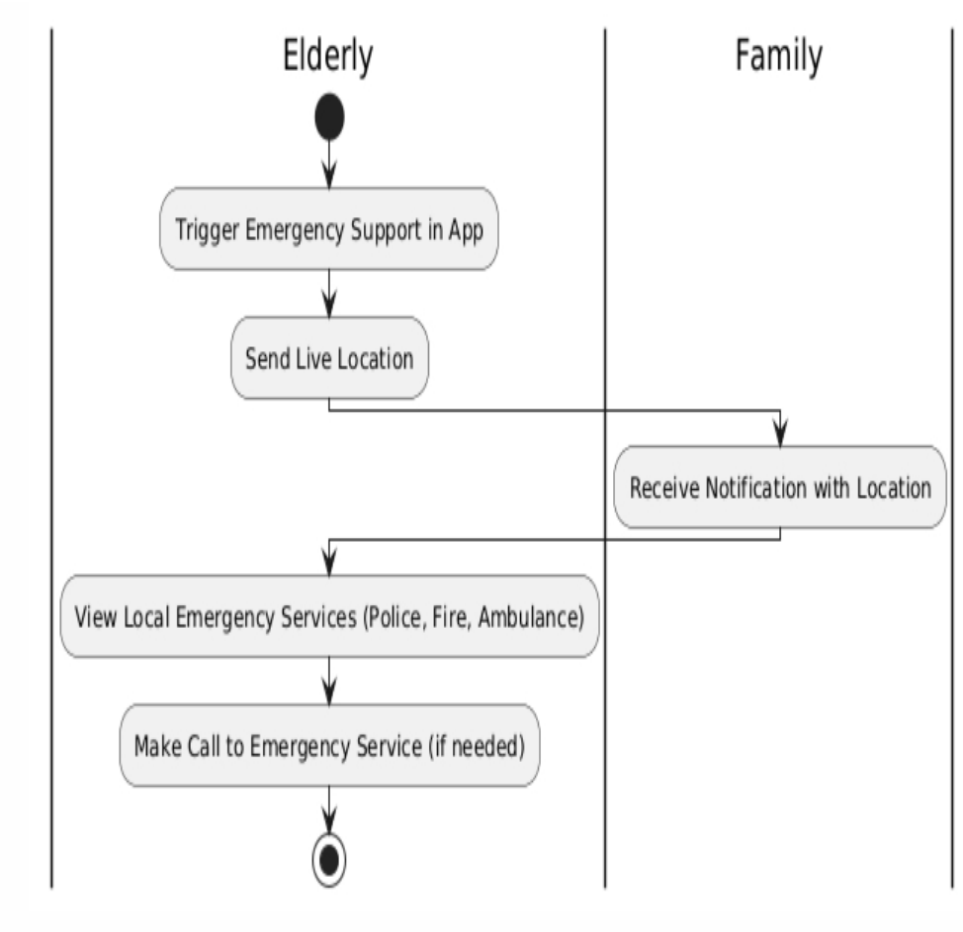


Figure 4.3: Emergency Support activity diagram

This activity diagram represents the emergency support system in the SilverSupport app. The process begins when an elderly user triggers an emergency alert within the app. This sends the user's live location to their registered family contact. The family member receives a notification with the location details. Meanwhile, the elderly user can also view local emergency services (police, fire, ambulance) and has the option to make direct emergency calls if necessary.

Feedback

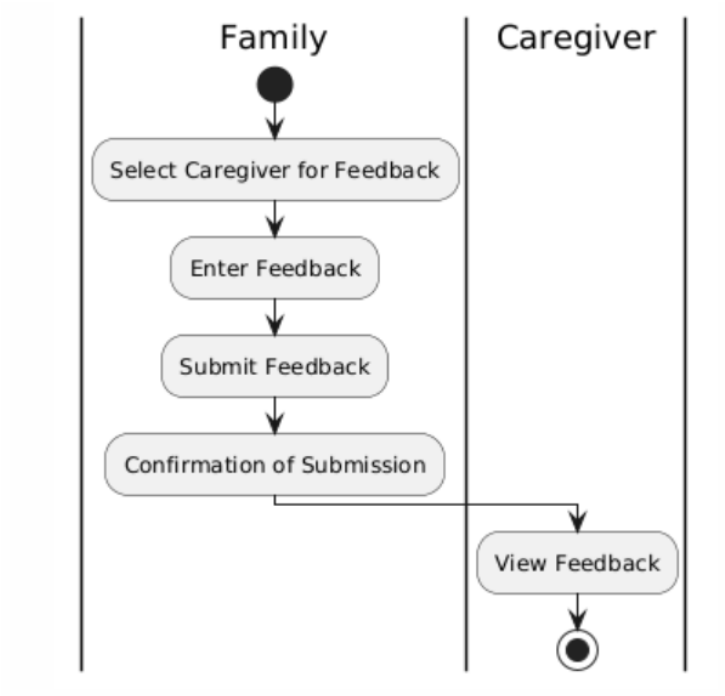


Figure 4.4: Feedback activity diagram

This activity diagram illustrates the feedback system for caregivers. A family member selects the caregiver for whom they wish to provide feedback. They then enter their feedback and submit it. Upon successful submission, a confirmation message is displayed. The caregiver can later view the feedback provided by the family, ensuring transparency and service quality improvement.

4.3 Sequence Diagram

Caregiver Booking and Payment

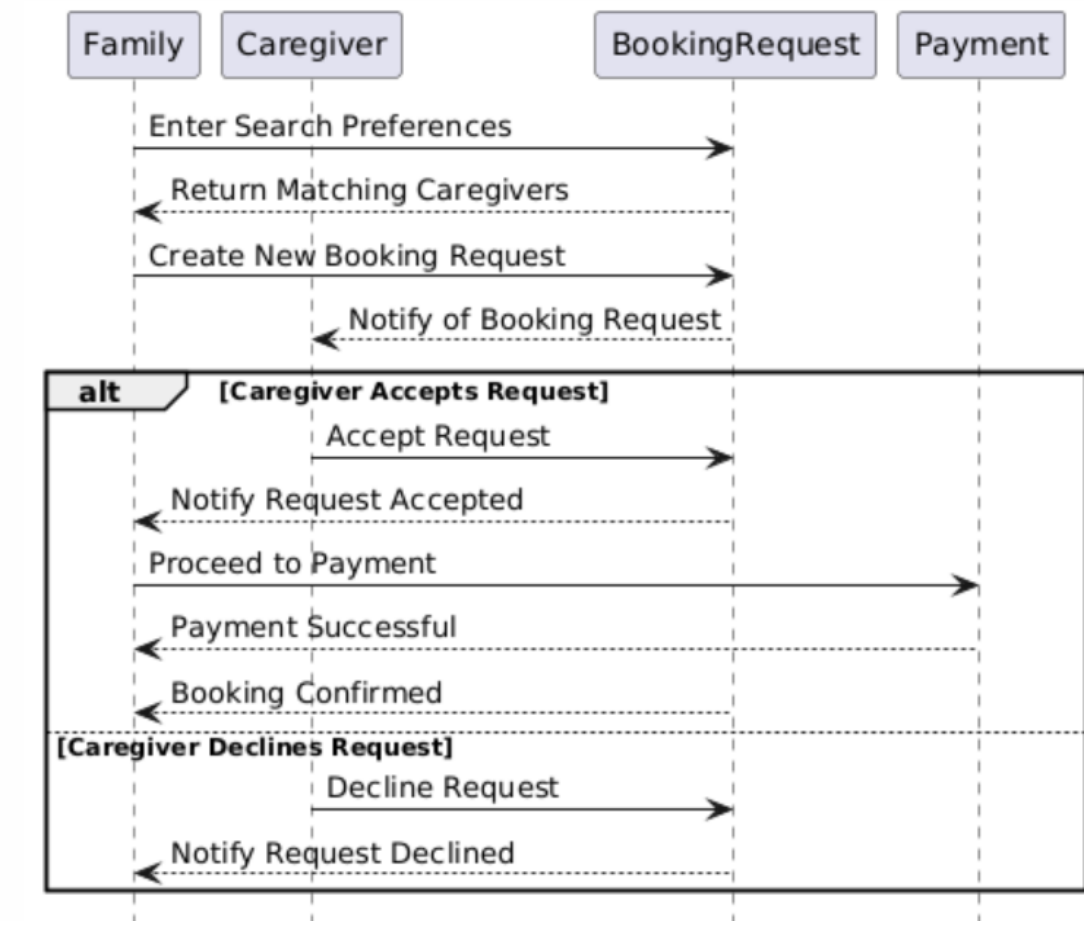


Figure 4.5: Caregiver Booking and Payment Sequence Diagram

The sequence diagram for Caregiver Booking and Payment represents the step-by-step process involved in booking a caregiver. The process starts when a Family user enters search preferences, and the system returns a list of matching caregivers. The user selects a caregiver and creates a booking request, which is then sent to the caregiver. The caregiver can either accept or decline the request. If the caregiver accepts, the system notifies the family, and they proceed to make a payment. Once the payment is successfully processed, the booking is confirmed. However, if the caregiver declines, the system informs the family that the request has been rejected. This sequence ensures a structured booking process, incorporating service confirmation and payment handling, making the system efficient and user-friendly.

Emergency Support

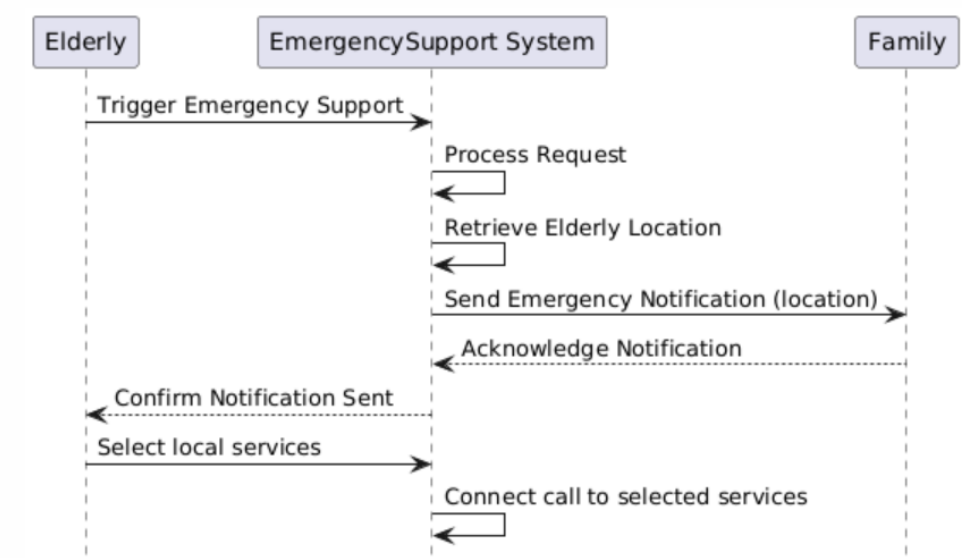


Figure 4.6: Emergency Support Sequence Diagram

The process begins when the elderly user initiates an emergency alert within the SilverSupport app by pressing the emergency button.The system retrieves the live location of the elderly user using GPS functionality.The retrieved location and an emergency alert message are sent as a notification to the registered family contact in the database.The family member's device receives the emergency alert, including the elderly's live location.The elderly user is given the option to view emergency services (police, fire, ambulance) within the app.If necessary, the elderly user can directly call one of the emergency services (e.g., police, fire, ambulance) using predefined numbers.

Health Assistence

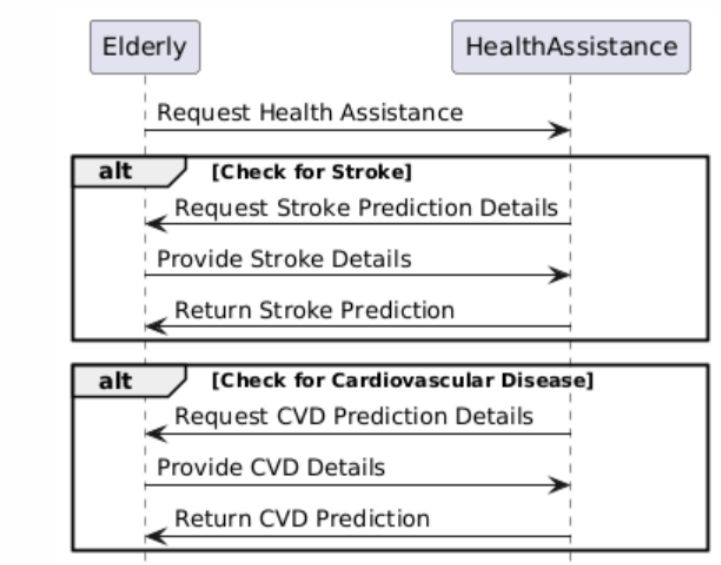


Figure 4.7: Health Assistance Sequence Diagram

The user enters health-related parameters.The system validates and preprocesses the input data, ensuring it matches the expected format.The preprocessed data is sent to the trained ML model for prediction.The ML model analyzes the data and returns a prediction indicating the likelihood of stroke or cardiovascular disease.

Feedback

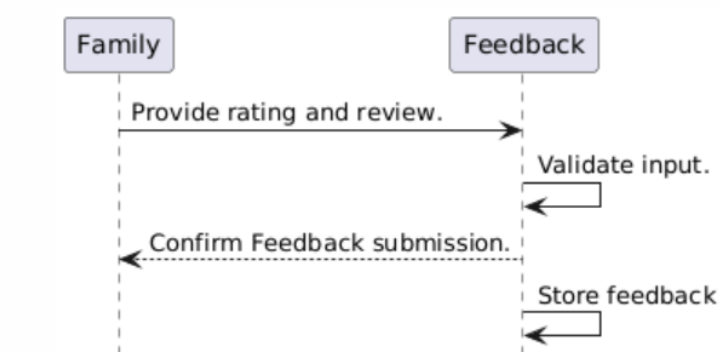


Figure 4.8: Feedback Sequence Diagram

The family member can search for the caregiver and provide rating and review and submit it. The caregiver and family users will be able to see this review in the caregiver profile.

4.4 List of identified classes, attributes and their relationships

Classes are an important mechanism for classifying objects. The primary role of a class is to define the attributes, methods, and applicability of its instances.

4.4.1 Identified Classes

- User
- Admin
- Elderly
- Caregiver
- Family
- EmergencySupport
- HealthAssistence
- BookingRequest
- Payment
- Feedback

4.4.2 Identified Attributes

The Admin class is an extension of the User class and is responsible for managing the platform. In addition to inheriting attributes from User, it introduces adminId, a unique identifier for each administrator. Admins have special privileges, including manageUsers(), which enables them to approve, block, or remove users, ensuring that only verified caregivers and family members participate in the platform.

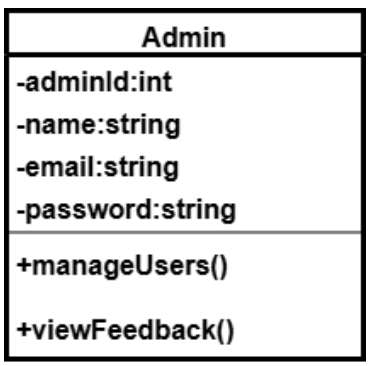


Figure 4.9: Admin Class

The User class serves as the foundation for all users in the system, including Admins, Elderly users, Caregivers, and Family members. This class ensures that all users share common attributes and functionalities. The attributes include userId, which uniquely identifies each user, name to store the full name, email for authentication, and password for secure access. Additionally, phoneNumber is included to allow communication between users, and role is specified to determine whether a user is an Admin, Elderly, Caregiver, or Family member. The methods provided in this class include register() for creating an account, login() for user authentication, updateProfile() for modifying user details, and deleteAccount() for removing a user from the system permanently.

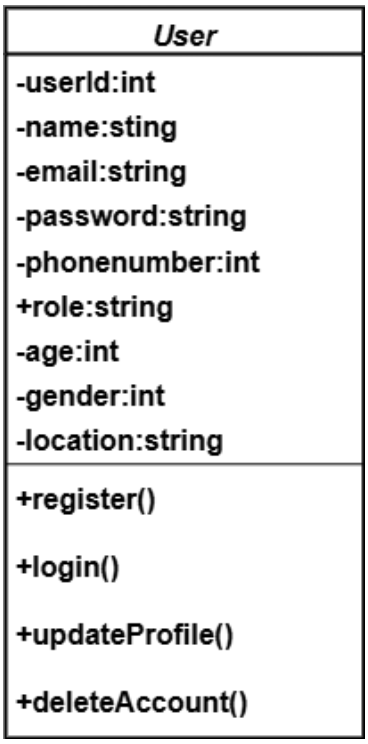


Figure 4.10: User Class

The Caregiver class represents individuals who provide caregiving services to elderly users. Along with inheriting attributes from User, it introduces caregiverId, a unique identifier for each caregiver. The class also includes skills, which stores a list of caregiving skills such as palliative care, mobility assistance, or dementia care. Additionally, yearsOfExperience keeps track of the number of years the caregiver has been active in the field, while certifications store professional qualifications like CPR certification or nursing degrees. The availability attribute contains time slots indicating when the caregiver is free to work. Another essential attribute is hourlyRate, which defines the caregiver's cost per hour for services. The caregiver can respond to service requests using the respondToRequest() method, modify their working hours through updateAvailability(), and view feedback from previous clients using viewFeedback().

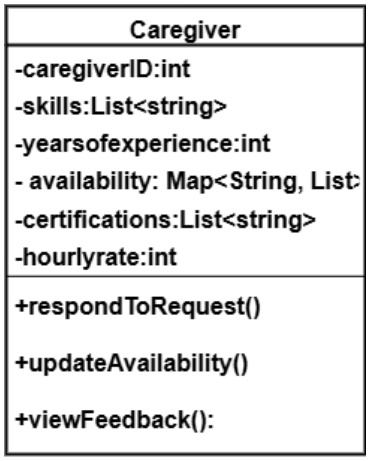


Figure 4.11: Caregiver Class

The Elderly class represents senior citizens who require caregiving services. In addition to inheriting attributes from User, it has healthConditions, which stores a list of medical conditions (e.g., diabetes, hypertension, or mobility issues). Another critical attribute is emergencyContact, which holds the contact details of family members or emergency services that can be notified in case of an emergency. The elderly user can trigger the triggerEmergencySupport() method, which sends an alert to their family contact and emergency responders. Additionally, the callEmergencyServices() method enables them to quickly dial pre-configured emergency numbers such as 911, ambulance services, or their registered family member.

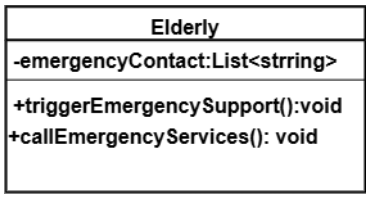


Figure 4.12: Elderly Class

The Family class represents the relatives or guardians responsible for managing care for elderly users. This class extends the User class and introduces familyId, a unique identifier for each family member. Additionally, the elderlyId attribute links the family member to the specific elderly user they are caring for. Families play a crucial role in selecting caregivers, and the searchCaregiver() method allows them to find suitable caregivers based on skills, availability, and pricing. Once a caregiver is selected, the requestService() method is used to send a booking request specifying the date and time. After the service is completed, families can provide feedback using giveFeedback(), which enables them to rate and review caregivers based on their performance.

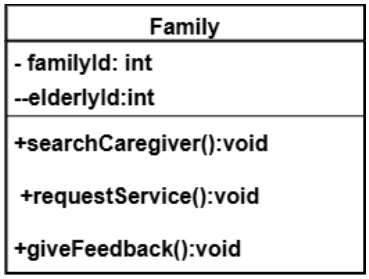


Figure 4.13: Family Class

The BookingRequest class manages service requests between families and caregivers. It has a bookingId attribute, which serves as a unique identifier for each booking. The caregiverId and elderlyId attributes establish the connection between the caregiver and the elderly user receiving the service. The requestedDate, startTime, and endTime attributes specify when the service is scheduled to occur. A crucial attribute in this class is status, which tracks whether the booking is Pending, Accepted, Completed, or Cancelled. The createRequest() method enables a family to initiate a new service request, while cancelRequest() allows them to cancel it before it is accepted. Caregivers can approve requests using the acceptRequest() method, and once the service is completed, the completeRequest() method marks the booking as finished.

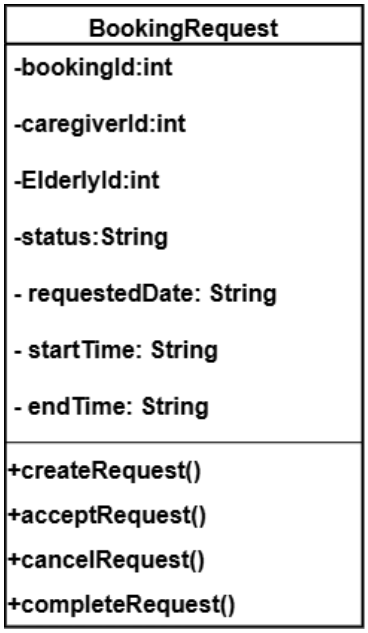


Figure 4.14: BookingRequest Class

The Payment class handles transactions for caregiving services. The key attributes include paymentId, which uniquely identifies each transaction, familyId to track the payer, and caregiverId to track the recipient of the payment. The amount attribute stores the total cost of caregiving services. Payments can be made through various methods stored in paymentMethod, such as Credit Card, PayPal, or Bank Transfer. A transactionID ensures secure tracking of payments, and the status attribute indicates whether the payment was Successful, Pending, Failed, or Refunded. The initiatePayment() method begins the payment process, while

updatePaymentStatus() updates the transaction status. If necessary, the refundPayment() method allows issuing refunds due to service cancellations.

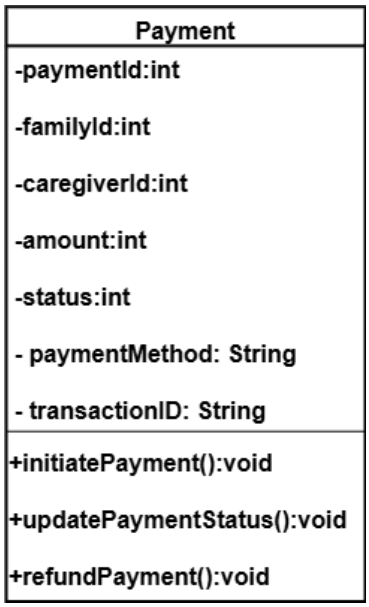


Figure 4.15: Payment Class

The Feedback class allows families to leave ratings and reviews for caregivers. The primary attributes include feedbackId, a unique identifier for each review, familyId to link the reviewer, and caregiverId to associate the review with a specific caregiver. The review attribute stores the textual feedback, while ratings representing service quality. The timestamps createdAt track when the feedback was posted or modified. Families can submit a review using submitFeedback(), view existing feedback with viewFeedback(), update their review via updateFeedback(), and remove a review using deleteFeedback(). The editFeedback() method enables users to modify both the rating and review content.

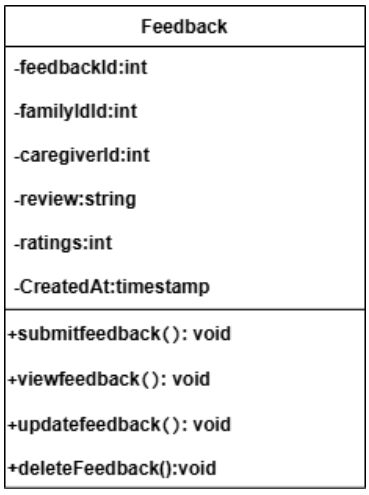


Figure 4.16: Feedback Class

The EmergencySupport class is dedicated to handling emergency situations for elderly users. The supportId uniquely identifies each emergency case, while elderlyId links the emergency incident to a specific user. The emergencyServices attribute stores predefined contacts such as 911, ambulance services, or a family member. The location attribute retrieves the real-time GPS coordinates of the elderly user during emergencies. The sendNotification() method automatically alerts the registered emergency contact, while dialEmergencyService() initiates a one-click call to emergency responders. Additionally, fetchLocation() helps retrieve and transmit location data to ensure quick assistance.

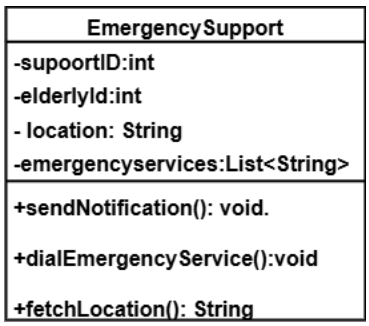


Figure 4.17: EmergencySupport Class

The HealthAssistance class integrates medical monitoring capabilities to assess the elderly user's health status. The bloodPressure attribute tracks systolic and diastolic levels, while heartRate stores the user's pulse rate. The age attribute is included since age is a critical factor in medical risk assessment. Additionally, cholesterolLevel monitors cholesterol readings, which are important indicators of cardiovascular health. The class includes two major health prediction methods: checkStroke(), which analyzes the risk of a stroke based on blood pressure, heart rate, and age, and checkCVD(), which evaluates the likelihood of cardiovascular disease using cholesterol and blood pressure levels.

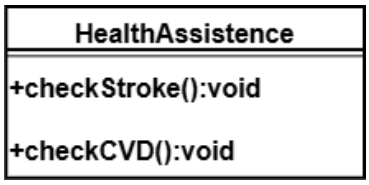


Figure 4.18: Health Assistence Class

The ‘Message‘ class represents the internal communication system between family members and caregivers within the Elderly Caregiver Matchmaking System. It contains attributes such as ‘id‘ to uniquely identify each message, ‘sender_id‘ and ‘receiver_id‘ to specify the sender and recipient of the message, ‘message_text‘ to hold the content of the communication, and ‘timestamp‘ to record the time the message was sent. These attributes ensure that every message is traceable, user-specific, and time-stamped. The class includes key methods like ‘send_message()‘ for sending a message, ‘receive_message()‘ for retrieving messages sent to a user. Through this structure, the ‘Message‘ class facilitates a secure and organized way for families and caregivers to communicate within the platform.

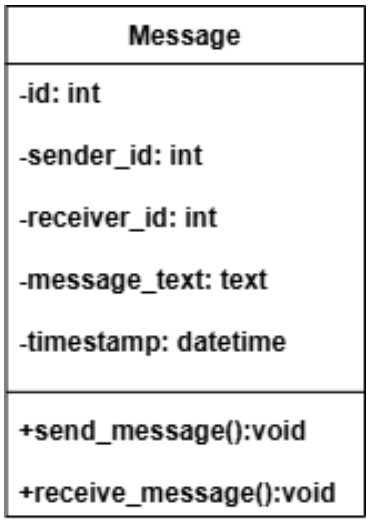


Figure 4.19: Message Class

4.4.3 Identified Relationship

Class	Related Class	Association Name	Cardinality
Admin	User	Manages user accounts	1 - *
Family	Caregiver	Requests caregiving services	1 - *
Family	BookingRequest	Creates service request	1 - *
Caregiver	BookingRequest	Responds to requests	1 - *
BookingRequest	Payment	Triggers payment if accepted	1 - 1
Family	Feedback	Provides caregiver feedback	1 - *
Caregiver	Feedback	Receives feedback	1 - *
Caregiver	Availability	Updates availability details	1 - 1
Family	Availability	Selects start and end time	1 - 1
Elderly	EmergencySupport	Triggers emergency help	1 - 1
EmergencySupport	Emergency Contacts	Notifies emergency contact	1 - *
HealthAssistance	User	Checks health status	* - 1

Table 4.1: Identifies Relationships

4.5 Class Diagram

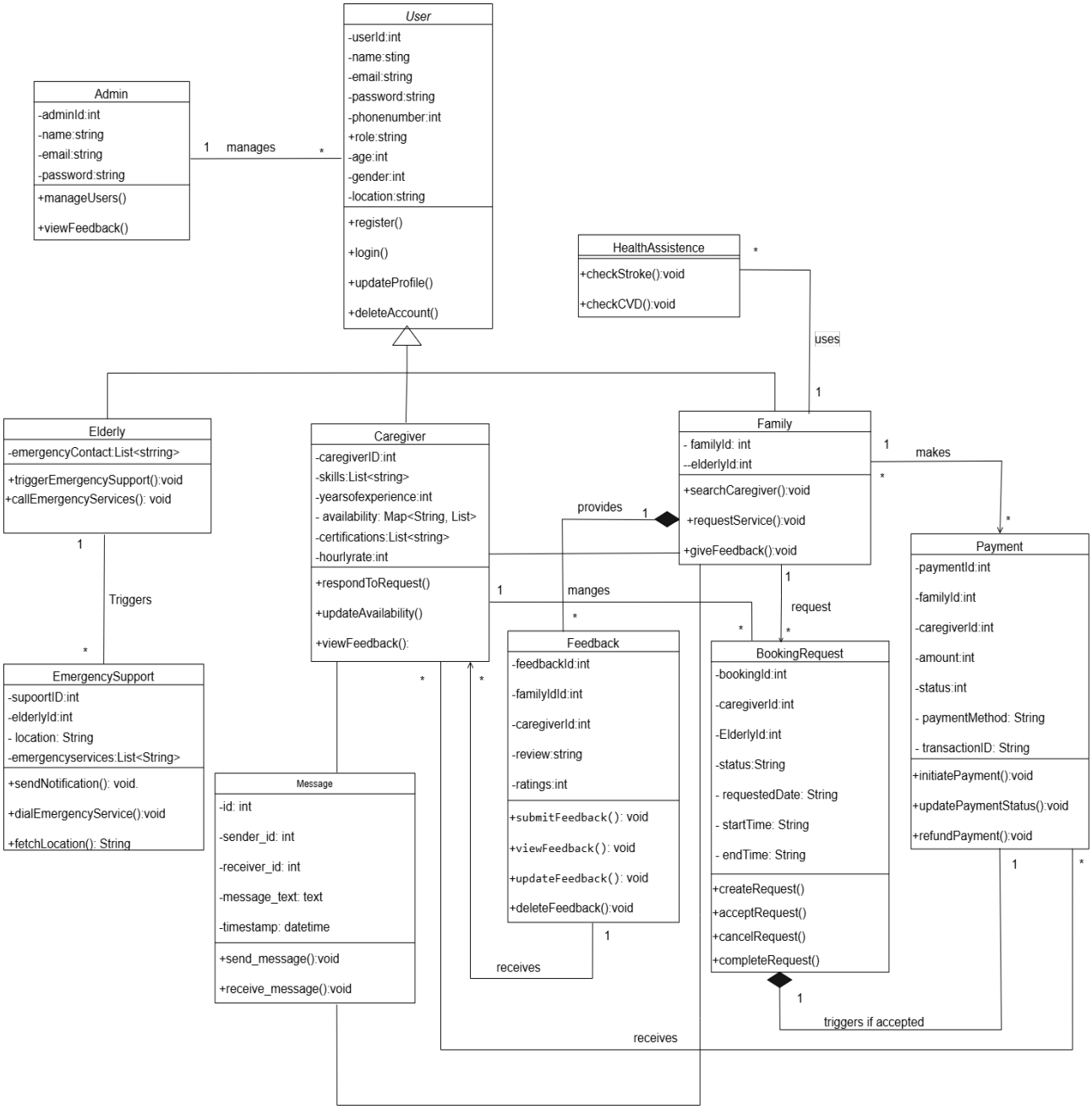


Figure 4.20: Class diagram

4.6 Database Design

Table name : User
Primary Key : id

Attribute	Type	Size	Constraints	Description
id	Integer	-	Primary Key, Auto-increment	Unique identifier for the user
username	String	150	Unique, Not Null	Username for authentication
password	String	150	Not Null	Hashed password for authentication
role	String	50	Not Null	Role of the user
status	String	50	Default=”pending”	Account status
phone	String	20	Nullable	User’s phone number
location	String	200	Nullable	User’s address/location
gender	String	50	Nullable	Gender of the user
age	Integer	-	Nullable	Age of the user
profile_picture	String	300	Nullable	Profile picture path
experience	Integer	-	Nullable	Years of experience (for caregivers)
certifications	String	500	Nullable	Certifications of the caregiver
experience_file	String	300	Nullable	Path to experience-related documents
specialized_skills	String	500	Nullable	Caregiver’s special skills
availability	String	100	Nullable	Available working hours
gender_preference	String	50	Nullable	Preferred gender for caregiving
hourly_rate	Float	-	Nullable	Caregiver’s hourly rate

Table 4.2: User Table

Table name : Review
Primary Key: id
Foreign Keys: caregiver_id , family_user_id

Attribute	Type	Size	Constraints	Description
id	Integer	-	Primary Key, Auto-increment	Unique identifier for the review
caregiver_id	Integer	-	Foreign Key (user.id), Not Null	ID of the caregiver being reviewed
family_user_id	Integer	-	Foreign Key (user.id), Not Null	ID of the family providing the review
rating	Integer	-	Not Null	Rating score (1 to 5)
review_text	Text	-	Nullable	Review comments
timestamp	DateTime	-	Default=current timestamp	Time when review was created

Table 4.3: Review Table

Table name : Schedule
Primary Key: id
Foreign Keys: caregiver_id , family_user_id

Attribute	Type	Size	Constraints	Description
id	Integer	-	Primary Key, Auto-increment	Unique identifier for the schedule
caregiver_id	Integer	-	Foreign Key (user.id), Not Null	ID of the assigned caregiver
family_user_id	Integer	-	Foreign Key (user.id), Not Null	ID of the requesting family
start_time	DateTime	-	Not Null	Scheduled start time
end_time	DateTime	-	Not Null	Scheduled end time
status	String	50	Default=”scheduled”	Status of the schedule

Table 4.4: Scheduling Table

Table name : Message
Primary Key: id
Foreign Keys: sender_id , receiver_id

Attribute	Type	Size	Constraints	Description
id	Integer	-	Primary Key	Unique identifier for each message
sender_id	Integer	-	Foreign Key (user.id), Not Null	ID of the user who sent the message
receiver_id	Integer	-	Foreign Key (user.id), Not Null	ID of the user who receives the message
message_text	Text	-	Not Null	Content of the message
timestamp	DateTime	-	Default: Current Timestamp	Time when the message was sent

Table 4.5: Structure of the Message Table

Table name : Payment
Primary Key: id
Foreign Keys: schedule_id , caregiver_id , family_id

Attribute	Type	Size	Constraints	Description
id	Integer	-	Primary Key, Auto-increment	Unique identifier for the payment
schedule_id	Integer	-	Foreign Key (schedule.id), Not Null	Associated schedule ID
caregiver_id	Integer	-	Foreign Key (user.id), Not Null	ID of the caregiver receiving payment
family_id	Integer	-	Foreign Key (user.id), Nullable	ID of the paying family
amount	Float	-	Not Null	Payment amount
status	String	20	Default=”pending”	Payment status (pending, approved, paid)
timestamp	DateTime	-	Default=current timestamp	Time when the payment was made

Table 4.6: Payment Table

Tables in SilverSupport app

Table name : User_auth
Primary Key: email

Attribute	Type	Size	Constraints	Description
email	TEXT	50	Primary Key, Unique	Unique email ID for login
username	VARCHAR	150	Not Null	The display name of the user
password	VARCHAR	150	Not Null	Encrypted user password

Table 4.7: User_auth Table in SilverSupport App

Table name: Emergency_Contact
Primary key: id
Foreign key: email

Attribute	Type	Size	Constraints	Description
id	INTEGER	-	Primary Key, Auto-increment	Unique ID for each contact
email	TEXT	50	Foreign Key (User_auth.email)	Links contact details to a registered user
family_contact	VARCHAR	10	Unique, 10-digit number	The user's emergency family contact number
ambulance	VARCHAR	3	Default: "112"	Emergency ambulance number
police	VARCHAR	3	Default: "100"	Emergency police number
fire	VARCHAR	3	Default: "911"	Emergency fire number

Table 4.8: Emergency Contact Table in SilverSupport App

4.7 UI Design

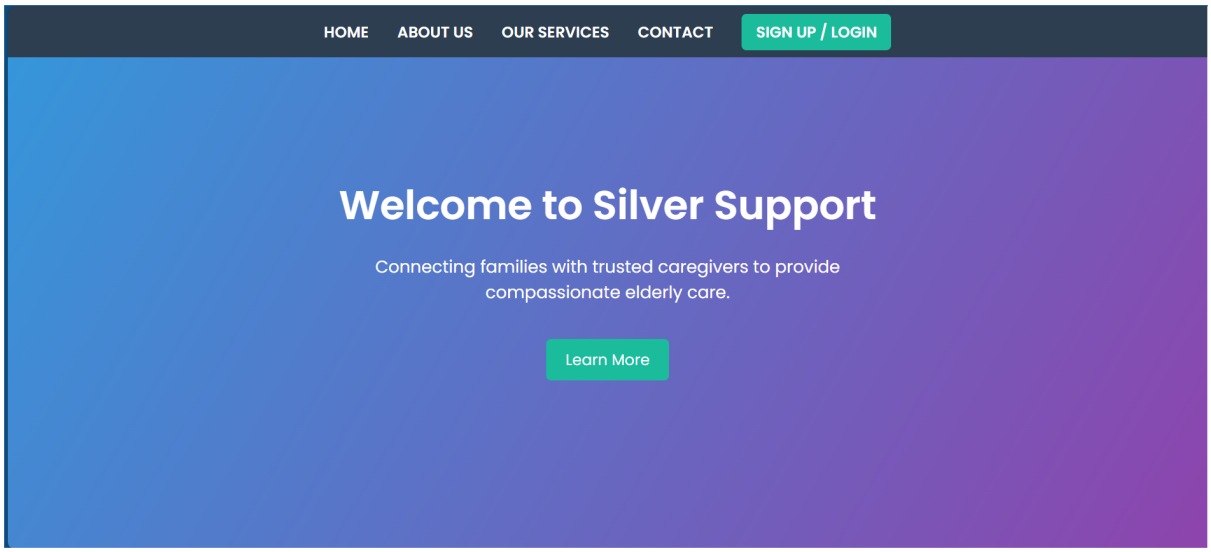


Figure 4.21: Main page of Silver Support

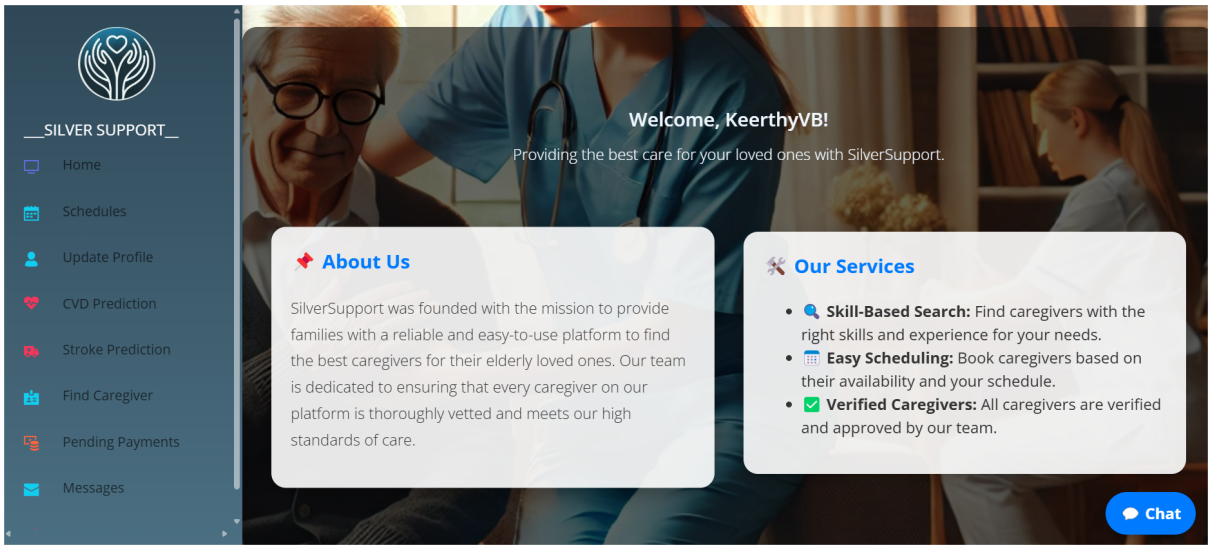


Figure 4.22: Main Dashboard of family

This image displays the Family Dashboard of the SilverSupport platform. On the left, a side navigation panel offers quick access to features like scheduling, health predictions, and payments. The main area includes an ”About Us” section, which highlights the platform’s mission to connect families with reliable caregivers for their elderly loved ones.

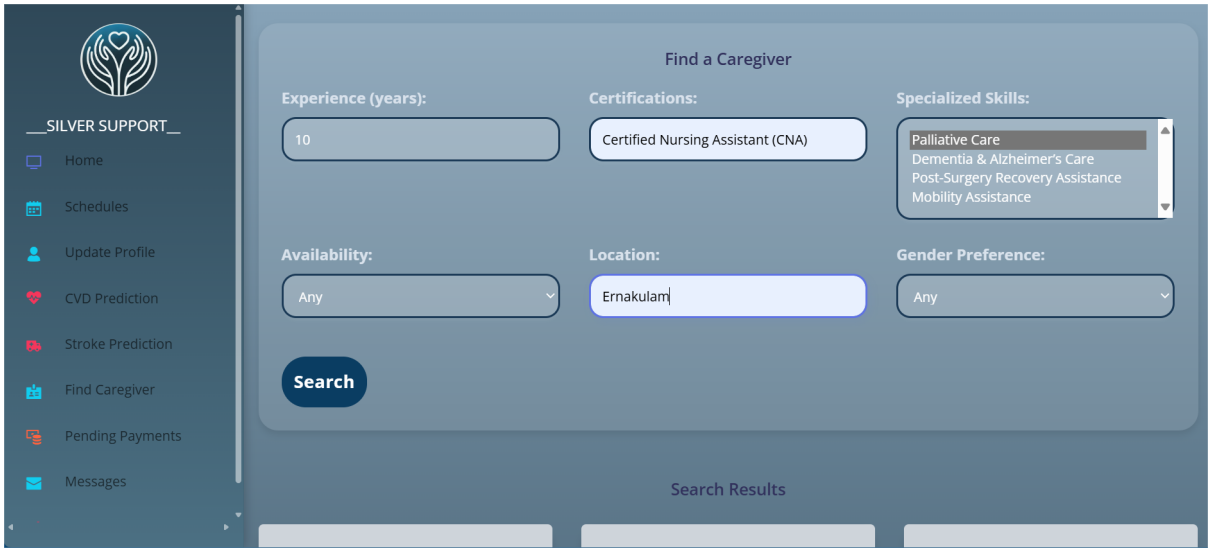


Figure 4.23: Skill Based Searching

The image shows the Caregiver Search interface for family users.At the top, a search form allows filtering by experience, certifications, skills (e.g., palliative or dementia care), availability, and location. Clicking "Search" displays matching caregivers below, with each card showing experience, skills, and availability. The side-by-side layout helps families easily compare options, supporting well-informed decisions based on their specific needs.

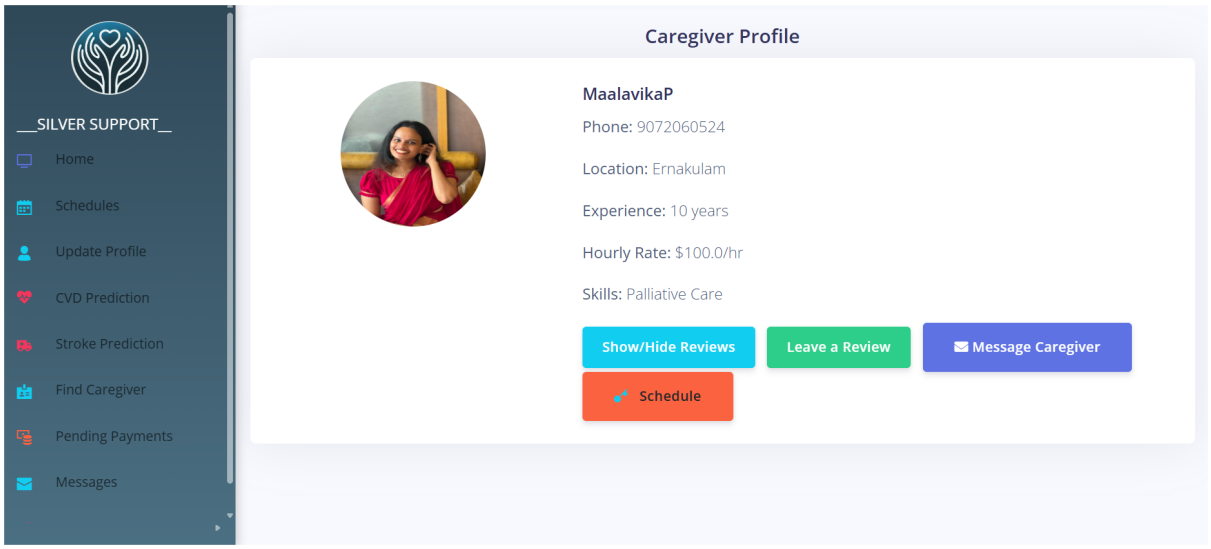


Figure 4.24: Caregiver profile viewing page

The image displays the Caregiver Profile View. After selecting a caregiver, families can view detailed info such as name, contact, location, experience, hourly rate, and key skills (e.g., Mobility Assistance). Action buttons allow them to check availability, read or write reviews. The streamlined layout helps families quickly assess caregiver suitability before booking.

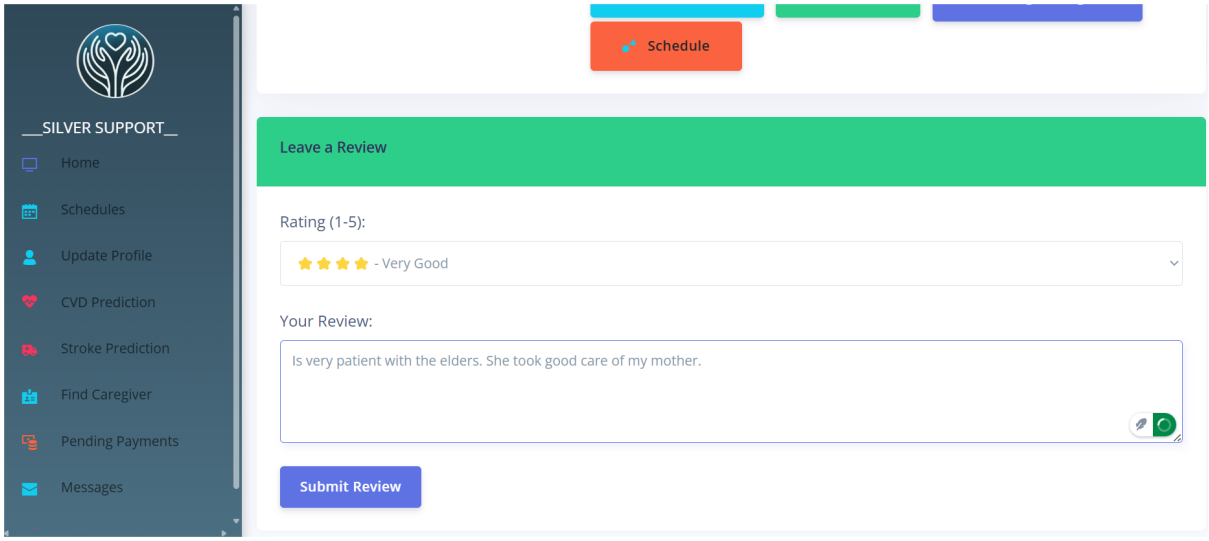


Figure 4.25: Rating and review providing section

The image shows the Reviews Section below the caregiver's profile.It lists feedback from other families, including reviewer names, star ratings, comments, and submission dates.

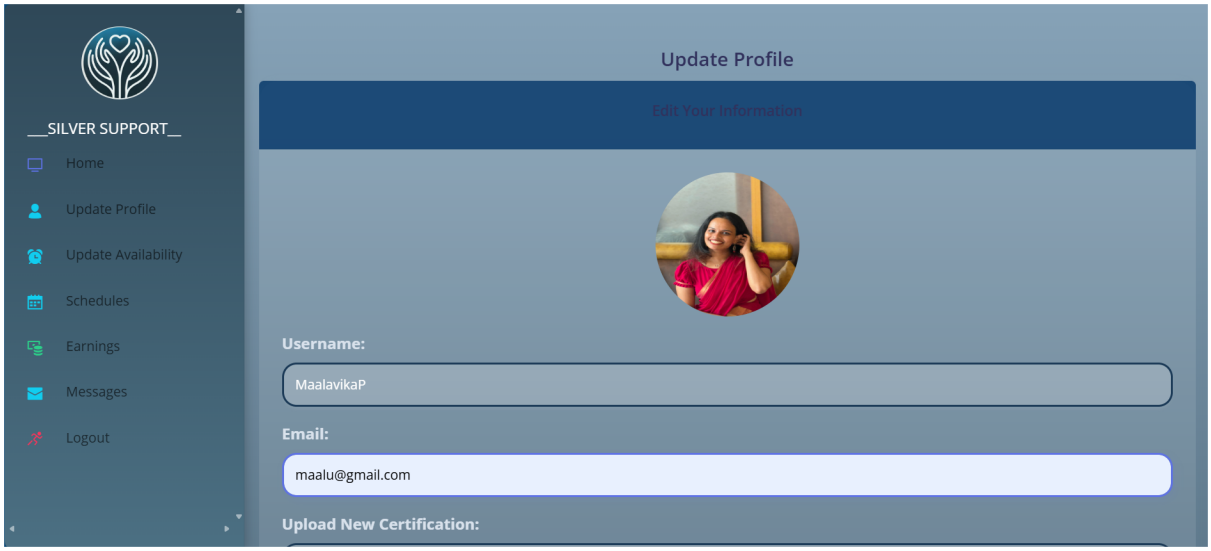


Figure 4.26: Profile updating page of caregiver

Allows caregivers to update their personal details, including username, email, certifications, experience documents, and profile picture.

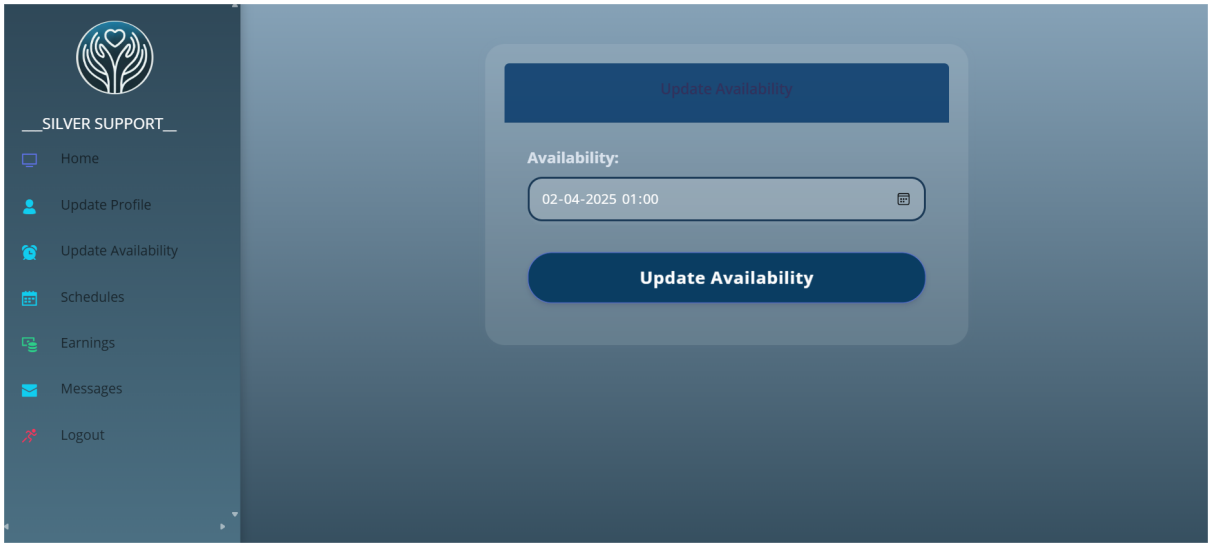


Figure 4.27: Availability Updating page

Enables caregivers to set or modify their availability for caregiving sessions.

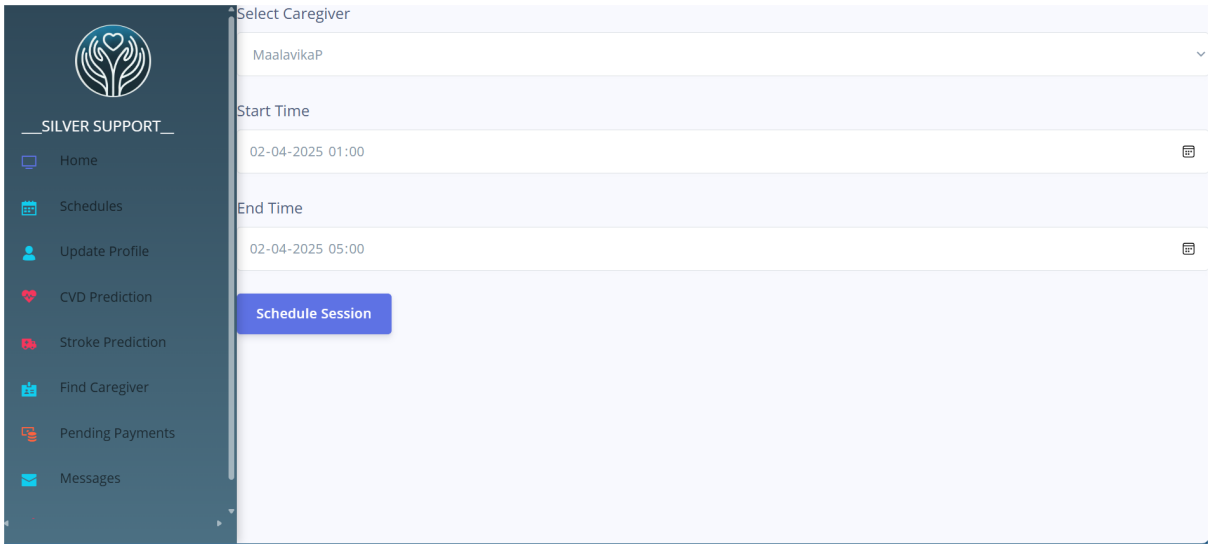


Figure 4.28: Scheduling page of family

The family members can schedule the session with caregiver by providing the start date and time of session.

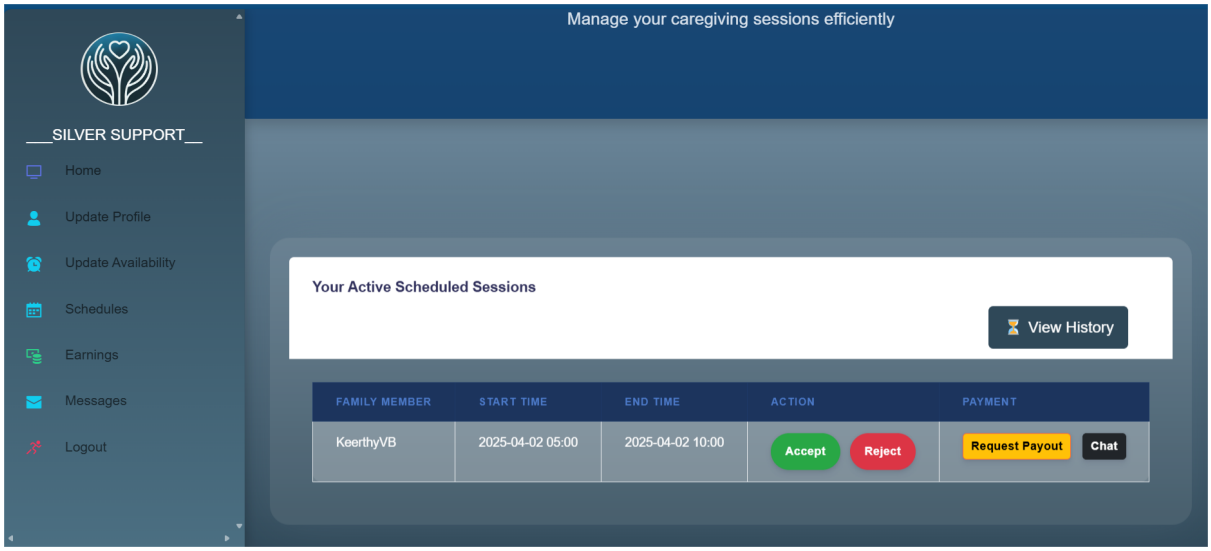


Figure 4.29: Schedules page of caregiver

Displays request for booking session page where caregiver can accept or reject, and provides an option to request payment for completed services.

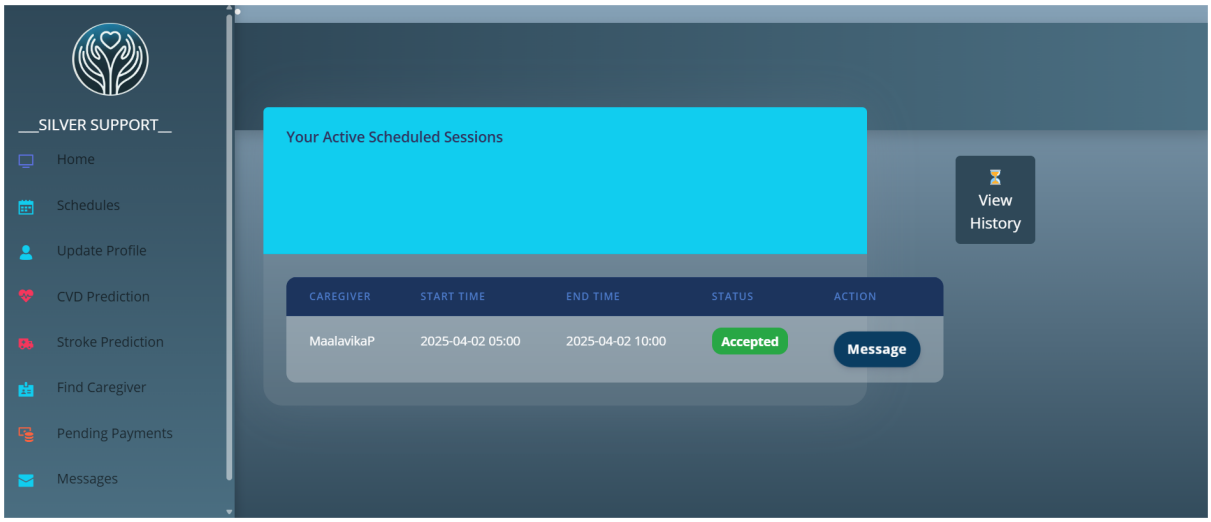


Figure 4.30: View the scheduled session

The image showcases the Booked Sessions View, where caregivers can view their scheduled sessions with elderly clients.

Figure 4.31: Payment Payout page of Family

This screen displays pending caregiver payments. Each entry lists the caregiver's name, requested amount, and request date. Families can approve and initiate payments using the "Approve Pay" button. Upon clicking, users are redirected to Razorpay for secure and seamless transaction processing, ensuring reliable financial interactions between families and caregivers.

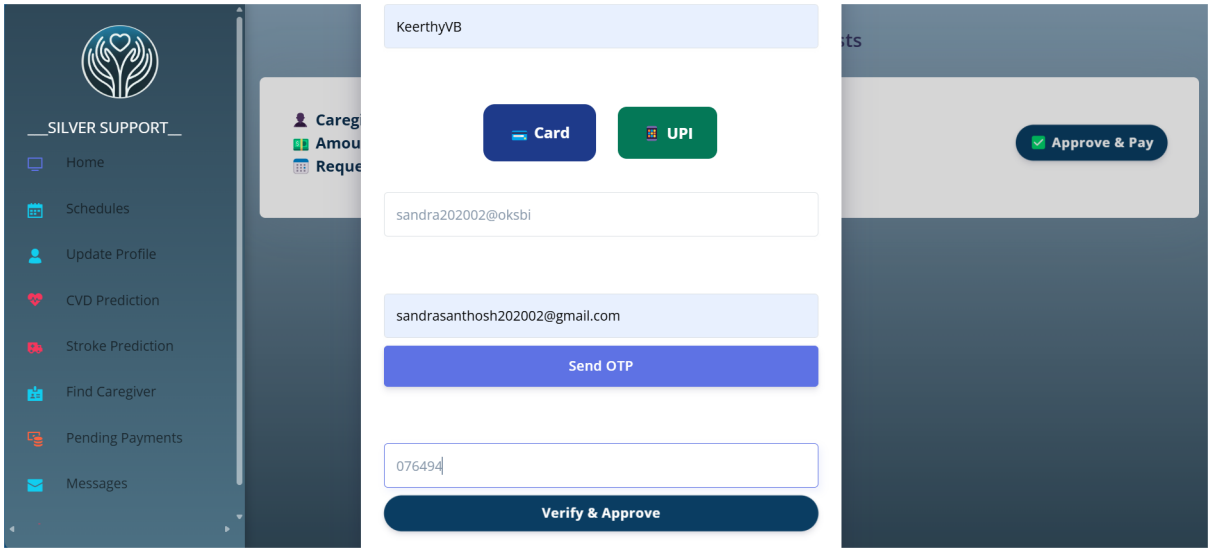


Figure 4.32: Razorpay Payment page of Family

The UI is the Razorpay Payment Page, designed for family members users to make payments. It features input fields for the payer's name, email, and payment amount, with options to pay via Card or UPI.

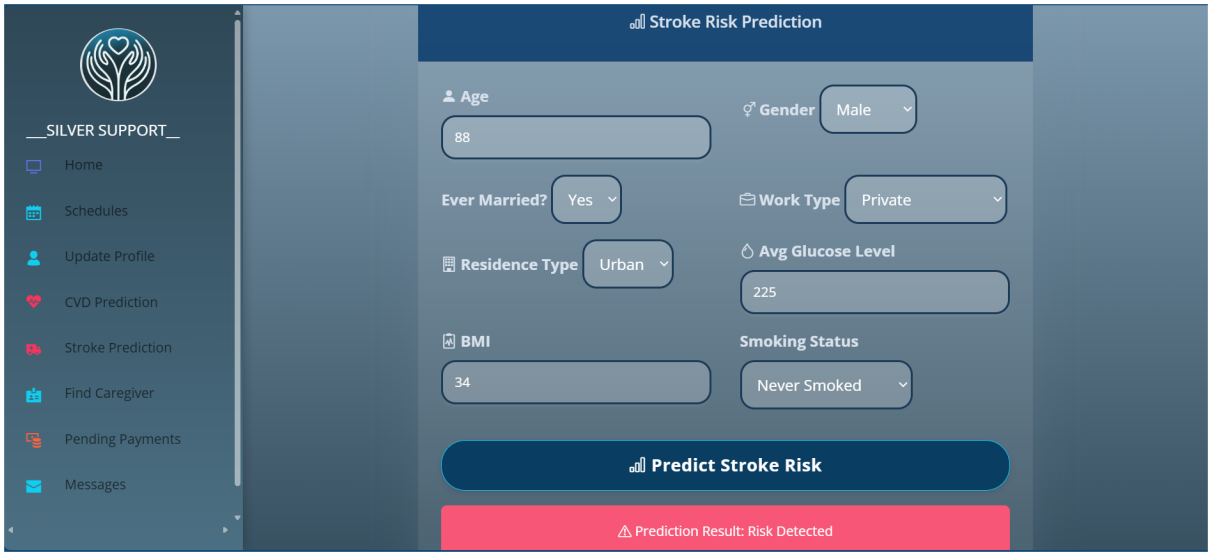


Figure 4.33: Stroke prediction page of family

Users enter personal health details (age, BMI, glucose level, etc.), and the system predicts their stroke risk.

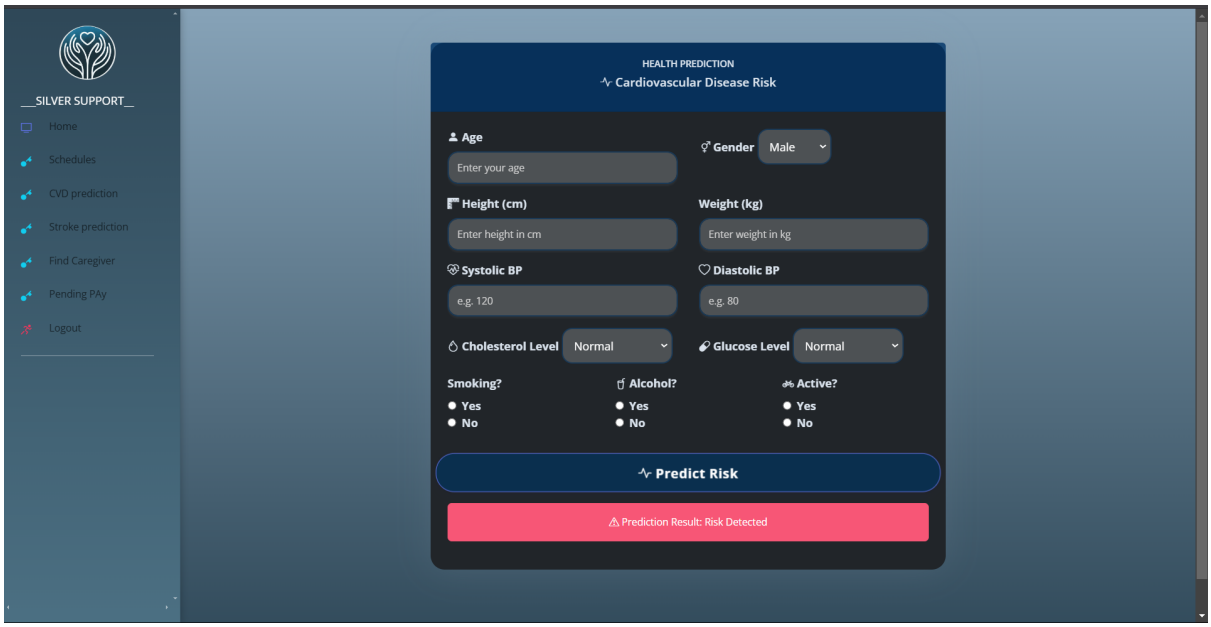


Figure 4.34: Cardiovascular prediction page of family

Users input various health metrics like blood pressure, cholesterol, and activity levels to determine their risk of cardiovascular disease.

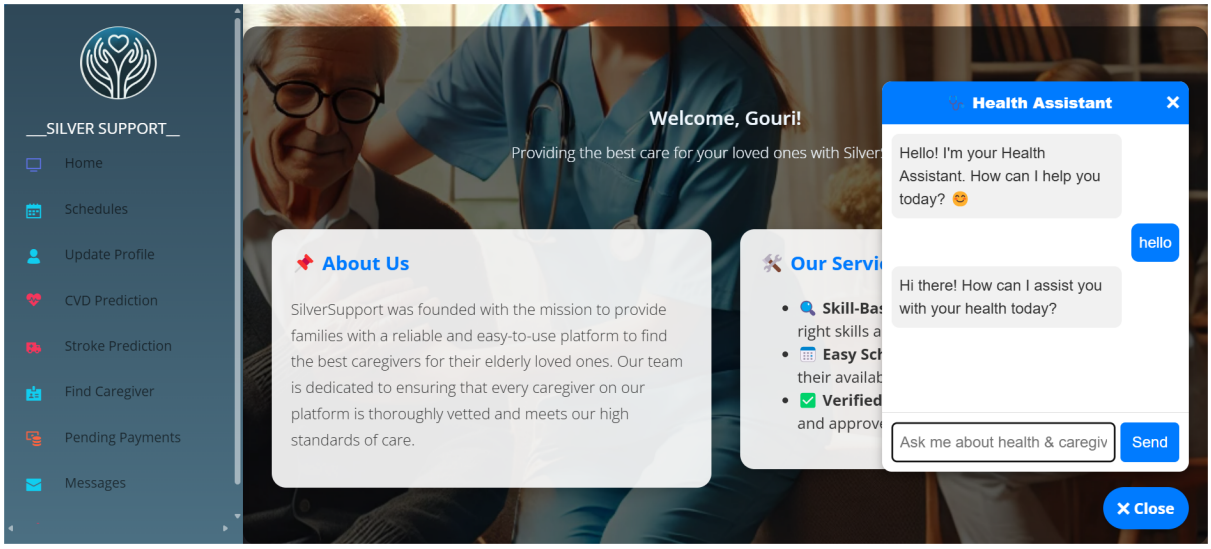


Figure 4.35: Chatbot assistence

There is a small icon for help, which is a rule-based chatbot. It will answer queries of users.

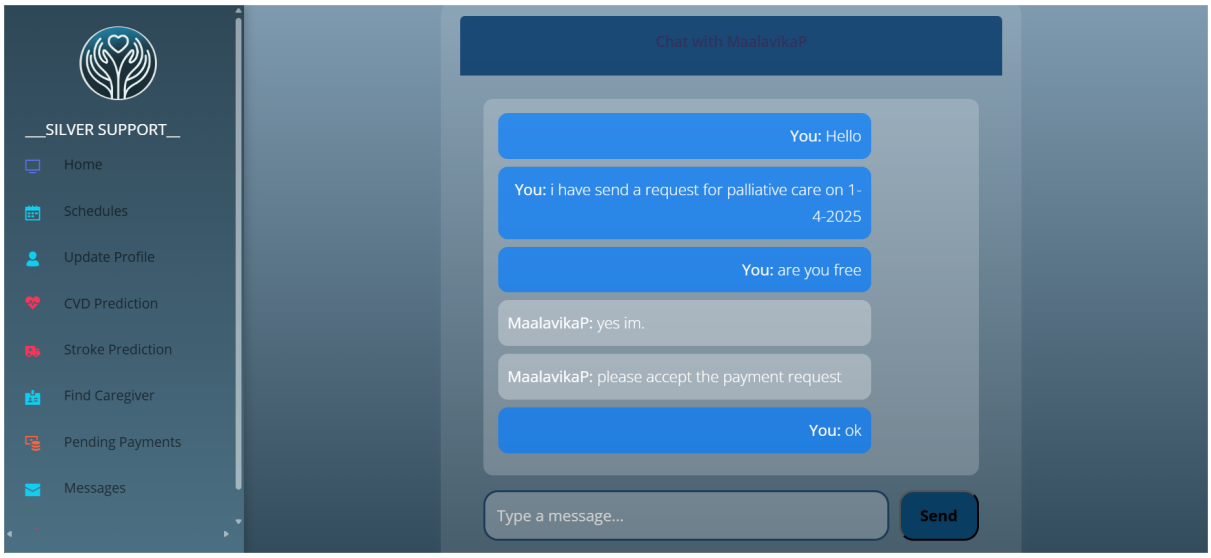
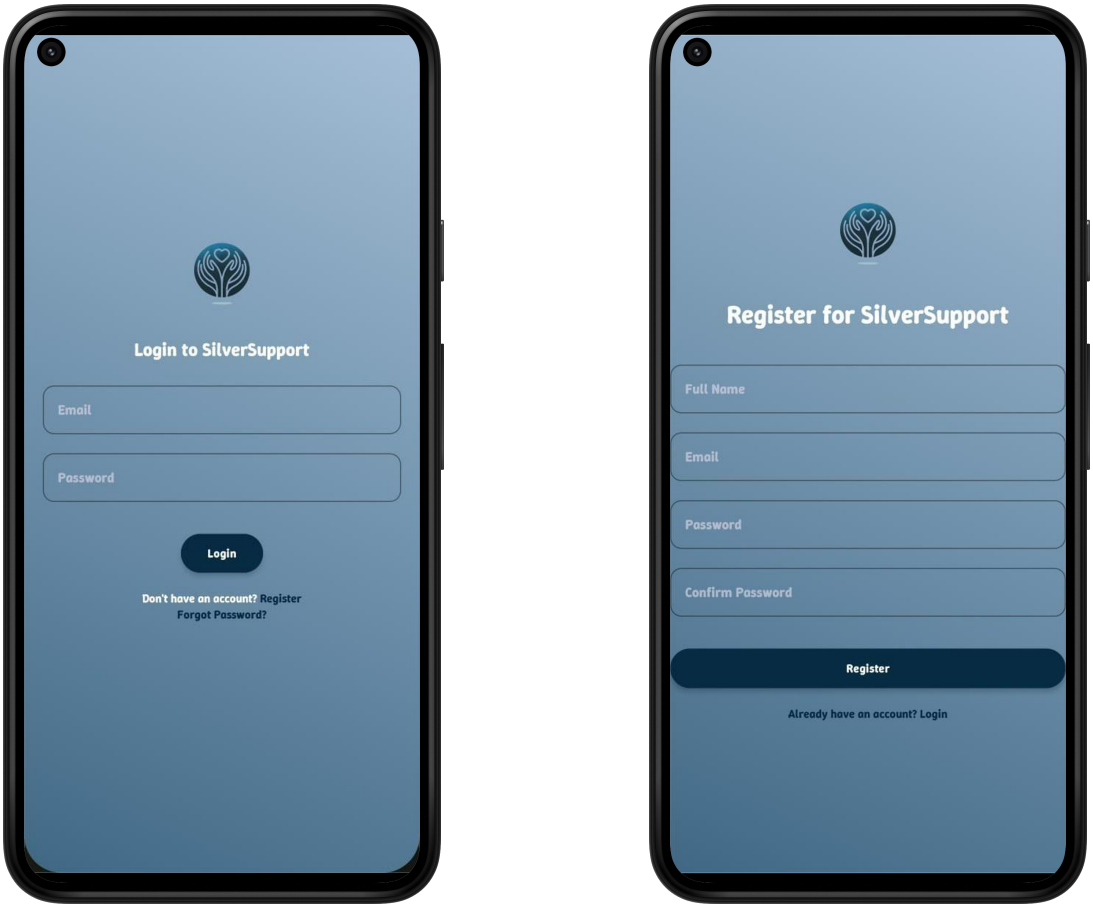


Figure 4.36: Comminication between family and caregiver

The family can communicate with the caregiver by selecting the message option in the caregiver profile or can communicate from the scheduled sessions page.

Emergency Support App



(a) Login page (b) Registration page

Figure 4.37: Registration and Login

Enter the necessary user credential for registering and use those credentials for logging in.

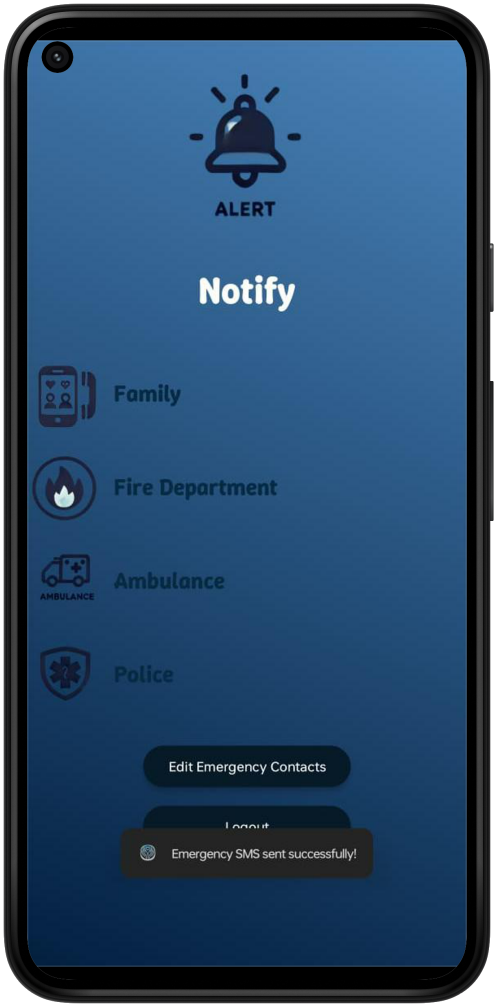


Figure 4.38: Main Emergency Activity page

When clicking the notification image, the location istion istion istioSMSs feSMShed SMS an SMS is send to the contact,contact,contact,contact, and a toast is shown saying that ”Emergency SMS sent successfully”.

Figure 4.39: Emergency details storing page

You can enter the emergency contacts and save them by clicking on on the ”save contact”.

4.8 Model Building

Model building in Machine Learning (ML) is the process of developing a predictive model using historical data. It involves data pre-processing,, selecting the right algorithm, training the model, evaluating its performance, and optimizing it for better accuracy. In healthcare, ML models help predict diseases such as stroke and cardiovascular disease (CVD) by analyzing patient data, identifying risk factors, and providing early warnings.

4.8.1 Implementation Code

4.8.1.1 Implemetation code of stroke dataset

```
[ ]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.linear_model import LogisticRegression
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
     from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

     # Define features and target variable
     X = df.drop(columns=["cardio"])  # Replace "cardio" with actual target column name
     y = df["cardio"]

     X = pd.get_dummies(X, columns=['cholesterol', 'gluc'])  # One-hot encode categorical variables

     # Split dataset
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

Figure 4.40: Implemetation code of Stroke Dataset

The stroke prediction model is built using a Random Forest Classifier in Python. The dataset is first loaded, and missing values in the bmi column are handled by filling them with the median value. Categorical variables such as gender, ever_married, work_type, Residence_type, and smoking_status are encoded using one-hot encoding to convert them into numerical values. The dataset is then split into features (X) and target variable (y), where stroke is the target. The dataset is further divided into training and testing sets (80% for training and 20% for testing). Feature scaling is applied using StandardScaler to normalize the data. The Random Forest Classifier is trained on the preprocessed data, and performance is evaluated using metrics such as accuracy score, confusion matrix, and classification report.

4.8.1.2 Implemetation code of Cardiovascular dataset

For the CVD prediction model, Logistic Regression is used as the primary machine learning algorithm. The dataset is preprocessed by encoding categorical variables like cholesterol and gluc using one-hot encoding. The features (X) are separated from the target variable (y), which corresponds to cardiovascular disease presence. The dataset is split into training and testing sets (80% training, 20% testing). Standardization is applied using StandardScaler to normalize the feature values. The Logistic Regression model is then trained using the training dataset. Performance evaluation is carried out using

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load the dataset
df = pd.read_csv("stroke_data.csv")

# Handling missing values
df['bmi'].fillna(df['bmi'].median(), inplace=True)

# Encoding categorical variables
df = pd.get_dummies(df, columns=['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status'], drop_first=True)

# Define features and target variable
X = df.drop(columns=['stroke', 'id'])
y = df['stroke']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Standardizing the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Figure 4.41: Implemetation code of stroke

accuracy score, confusion matrix, and classification report, with additional visualization of the confusion matrix using Seaborn and Matplotlib.

4.8.2 Model Planning

4.8.2.1 Model Planning of Stroke Prediction

- Objective
  Develop a Brain Stroke prediction system using Random Forest (RF) to predict the likelihood of a stroke based on patient health indicators.

- Approach
  Implement Random Forest (RF) due to its ability to handle complex and non-linear relationships in medical data.Train and test the model using structured health data, including features such as *age, hypertension, heart disease, BMI, glucose levels, and smoking status*.

- Data Preparation
  Import and preprocess the dataset containing patient information.Handle missing values using imputation techniques.Encode categorical variables (e.g., gender, work type) into numerical form. Normalize numerical features to improve model performance.Split the dataset into 80% training and 20% testing.

- Exploratory Data Analysis (EDA)
  Analyze feature distributions and their impact on stroke occurrence.Identify correlations between risk factor and stroke probability.Visualize data using histograms, box plots,and heatmaps.

- Model Building
  Train a Random Forest Classifier to predict stroke risk.Fine-tune hyperparameters (*number of trees, depth, minimum samples per split*) for best performance.Evaluate using accuracy, confusion matrix, and F1-score.

- Model Evaluation & Deployment
  Validate using precision, recall, and AUC-ROC score. Optimize model performance through

feature selection and hyperparameter tuning. Deploy the model via Flask API for real-time predictions in healthcare applications.

4.8.2.2 Model Planning of Cardiovascular Disease prediction

- Objective
  Develop a cardiovascular disease (CVD) prediction system using Logistic Regression (LR) to classify whether a person has CVD based on medical attributes.

- Approach
  Implement Logistic Regression (LR) as it is well-suited for binary classification problems in structured health data. Train and evaluate the model using clinical features such as *cholesterol, blood pressure, smoking habits, glucose levels, BMI, and heart disease history*.

- Data Preparation
  Load and preprocess the cardiovascular dataset. Handle missing values using mean or median imputation. Convert categorical variables (e.g., smoking status, work type) into numerical form. Apply feature scalin (standardization) to improve model performance. Split the dataset into 80% training and 20% testing.

- Exploratory Data Analysis (EDA)
  Identify significant risk factor contributing to CVD. Analyze trends using histograms, correlation heatmaps, and box plots. Check for any class imbalance and apply resampling techniques if needed.

- Model Building
  Train a Logistic Regression model for CVD classification. Ensure proper feature scaling for efficient gradient descent optimization. Evaluate using accuracy, confusion matrix, and AUC-ROC curve.

- Model Evaluation & Deployment
  Measure model performance using precision, recall, and F1-score. Optimize using hyperparameter tuning and feature selection. Deploy the CVD risk prediction model via Flask API or integrate it into a health monitoring system.

4.8.3 Training

4.8.3.1 Training of Stroke Prediction Model

The dataset was divided into two parts. X representing the input features, and y representing the target variable. The training set consists of 67% of the data and is used to train the model.

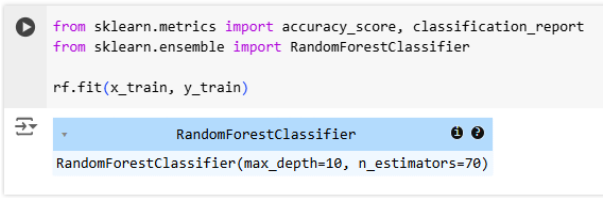


Figure 4.42: Random forest algorithm implementation

A Random Forest Classifier, which is an ensemble learning method based on multiple decision trees is used to train the model. The model consists of 70 decision trees. The maximum depth of each tree is limited to 10 levels. The number of features considered for splitting at each node is the square root of the total number of features.

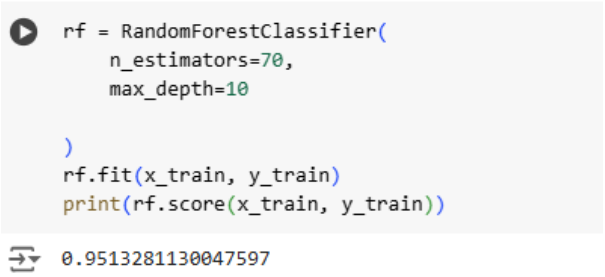


Figure 4.43: Random forest training

The model achieved an accuracy of 95.13% on the training data. To train a Random Forest model using Python,you typically start by importing the neccesary libraries, such as pandas for data manipulation and RandomForestClassifier from sklearn.ensemble. First, you load and preprocess the dataset, handling any missing values and encoding categorical features. After preparing the data, you can split it into training and testing sets using train_test_split. The model can then be instantiated with desired parameters (like the number of trees) and fit to the training data using the fit method.

4.8.3.2 Training of Cardiovascular Disease Prediction Model

A Logistic Regression model is initialized with a maximum iteration limit of 1000, which ensures that the model converges properly.Dividing the dataset into 80% training and 20% testing.fit() trains the model on the scaled training dataset (X_train_scaled, y_train), allowing it to learn the patterns in the data.

```
# Train Logistic Regression model
log_reg = LogisticRegression(max_iter=1000, random_state=42)
log_reg.fit(X_train_scaled, y_train)
```

Figure 4.44: Logistic Regression training

The trained model makes predictions on the training data (y_train_pred). The accuracy_score() function calculates how well the model performed on the training dataset.This accuracy helps assess if the model is overfitting (too good on training but bad on new data) or underfitting.

```
# Evaluate the model on training data
y_train_pred = log_reg.predict(X_train_scaled)
training_accuracy = accuracy_score(y_train, y_train_pred)
print(f"Logistic Regression Training Accuracy: {training_accuracy}")
```

Figure 4.45: Evaluating model on training data

```
Logistic Regression Training Accuracy: 0.7196607142857143
```

Figure 4.46: Accuracy on training data

Logistic regression got an accuracy of 71.96% in cardiovascular dataset.

4.8.4 Testing

4.8.4.1 Testing of Stroke Prediction Model

```
[43] from sklearn.metrics import accuracy_score
     y_pred = model.predict(x_test)
     accuracy_score(y_test,y_pred)

     0.9892030848329049
```

Figure 4.47: Evaluation on testing data

```
print(classification_report(y_test, y_pred))

              precision    recall  f1-score   support

           0       0.98      0.88      0.92      1604
           1       0.89      0.98      0.93      1605

    accuracy                           0.93      3209
   macro avg       0.93      0.93      0.93      3209
weighted avg       0.93      0.93      0.93      3209
```

Figure 4.48: Classification report

```
[15] from sklearn.metrics import mean_squared_error
     mean_squared_error(y_test, y_pred)

     0.07136179495169835
```

```
from sklearn.metrics import log_loss

y_pred_proba = rf.predict_proba(x_test)
loss = log_loss(y_test, y_pred_proba)
print(f"Log Loss: {loss}")

Log Loss: 0.22270991052920233
```

Figure 4.49: Loss log

The classification model demonstrates excellent performance based on the provided evaluation metrics. It achieves a high accuracy of approximately 98.92%, indicating that it correctly classifies the vast majority of test samples. The detailed classification report shows strong precision and recall scores for both classes. Specifically, Class 0 has a precision of 0.98 and a recall of 0.88, while Class 1 shows a precision of 0.89 and an impressive recall of 0.98. The overall F1-score is balanced for both classes, with a macro and weighted average of 0.93, reflecting the model's robustness across class distributions. Additionally, the model achieves a very low mean squared error (MSE) of 0.0713, indicating minimal difference between predicted and actual labels. The log loss value of 0.2227 further confirms that the model is confident and reliable in its probabilistic predictions. Overall, these results suggest that the model is highly effective for binary classification tasks and performs consistently across different evaluation metrics.

Confusion Matrix

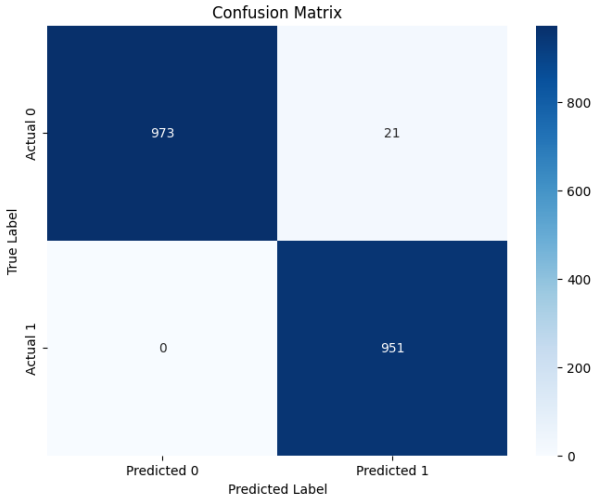


Figure 4.50: Confusion metric of stroke model

973 True Negatives (TN) → Correctly identified non-stroke cases.
951 True Positives (TP) → Correctly identified stroke cases.
21 False Positives (FP) → Non-stroke cases wrongly classified as stroke.
0 False Negatives (FN) → No stroke cases were misclassified as non-stroke.

The confusion matrix for the stroke prediction model demonstrates exceptional performance. Out of all the cases, the model correctly identified 973 individuals as not having a stroke and 951 individuals as having a stroke, with no false negatives and only 21 false positives. This means the model did not miss a single actual stroke case, which is critically important in medical scenarios where early detection can save lives. The recall (sensitivity) is 100%, indicating the model successfully flags all stroke cases, and the precision is also high at 97.82%, meaning most of the positive predictions were accurate. With an overall accuracy of around 99%, this model is highly effective.

This performance is especially significant in the context of stroke detection, where failing to identify a patient who is actually at risk (a false negative) can lead to severe complications or even death. By completely eliminating false negatives, the model ensures that every person who may need urgent medical attention is identified, even at the cost of a few false alarms. In preventive healthcare, this trade-off is highly favorable, making the model both a safe and effective tool for aiding in early stroke intervention and potentially saving lives.

4.8.4.2 Testing of Cardiovascular Prediction Model

The trained logistic regression model (log_reg) is used to predict outcomes for the test dataset (X_test_scaled).These predictions (y_pred) represent the model's best guesses on whether a person has cardiovascular disease (CVD) or not.

```
# Evaluate model on test data
y_pred = log_reg.predict(X_test_scaled)
test_accuracy = accuracy_score(y_test, y_pred)
print(f"Logistic Regression Test Accuracy: {test_accuracy}")
```

Figure 4.51: Evaluation on testing data

```
Logistic Regression Test Accuracy: 0.7222857142857143
```

Figure 4.52: Accuracy on testing data

The accuracy_score() function compares the true labels (y_test) with the predicted labels (y_pred).The output gives the test accuracy, which shows how well the model performs on unseen data. If test accuracy is significantly lower than training accuracy, the model may be overfitting (memorizing the training data but failing to generalize). The testing accuracy is 72.22%. **ROC curve of cardiovascular model**

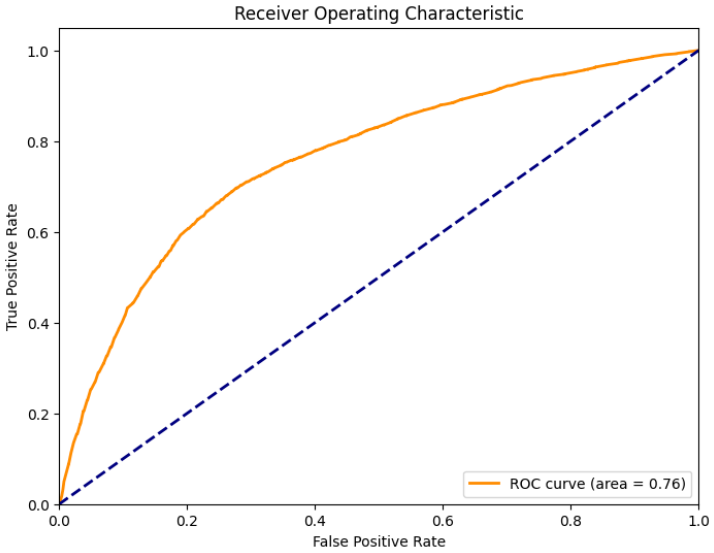


Figure 4.53: ROC curve of Cardiovascular model

- The y-axis represents the True Positive Rate (TPR) i.e., how well the model detects actual positive cases (CVDpatients).
- The x-axis represents the False Positive Rate (FPR) i.e., how many non-CVD cases were wrongly classified as CVD.
- An AUC of 0.79 indicates a moderate level of discrimination between CVD and non-CVD cases.

The Receiver Operating Characteristic (ROC) curve shown illustrates the performance of a classification model by plotting the True Positive Rate (sensitivity) against the False Positive Rate at various threshold levels. The orange curve represents how well the model distinguishes between the two classes. A key metric derived from this curve is the Area Under the Curve (AUC), which in this case is 0.76. This indicates that the model has a good ability to differentiate between positive and negative classes, with a 76% chance of correctly identifying a randomly chosen positive instance over a negative one. The blue dashed diagonal line represents the performance of a random classifier (AUC = 0.5). Since the ROC curve is significantly above this line, it confirms that the model performs better than random guessing. Overall, the ROC curve and AUC value demonstrate that the model exhibits a fair to good classification capability.

Confusion metrix of cardiovascular model

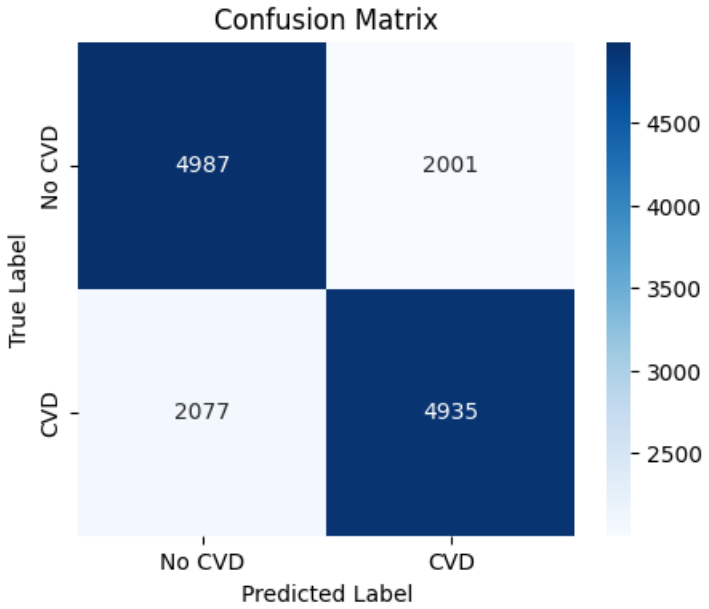


Figure 4.54: Confusion metrix of Cardiovascular model

TN (No CVD correctly classified) = 4,987
FP (Incorrectly classified as CVD when No CVD) = 2,001
FN (Incorrectly classified as No CVD when actually CVD) = 2,077
TP (CVD correctly classified) = 4,935

The model accurately identifies a high number of both CVD and non-CVD cases. However, there is a notable number of false negatives (2077), which means some actual CVD cases are being missed. This is a critical consideration, as failing to identify someone with CVD could result in missed preventive or urgent care.
Despite this, the true positive count (4935) and true negative count (4987) show that the model performs well overall, especially in identifying those who actually have CVD. The recall for CVD (sensitivity) is around 70.4%, and the precision is approximately 71.2%. These metrics suggest that the model is reliable for large-scale screening but could be further improved, particularly in reducing false negatives.

5 TESTING

In the context of apps, software testing remains a critical component of ensuring high quality and reliability. It involves thoroughly reviewing the app's specifications, design, and code generation to identify and rectify any errors. Before release to customers, rigorous testing procedures are employed to systematically uncover and address any issues. This process is carried out using disciplined techniques to design and execute tests aimed at verifying the app's functionality, usability, performance, and security, ultimately enhancing the overall user experience. Software testing offers numerous advantages in ensuring the quality and reliability of an app before its release to customers. Firstly, it helps identify and rectify defects early in the development cycle, reducing the cost and effort associated with fixing issues later on. By systematically reviewing the app's specifications, design, and code, testers can uncover bugs and inconsistencies that may otherwise go unnoticed, thus enhancing the overall stability of the app. Additionally, rigorous testing procedures ensure that the app functions as intended, meeting the requirements and expectations of end users. This, in turn, boosts user satisfaction and trust in the app's reliability, leading to increased adoption and retention rates.

Test Cases: Testing is based on test cases. It describes which feature or service test attempts to cover.

5.1 Unit Testing

Unit testing is a vital software development practice that involves testing individual components or functions in isolation to ensure correct behavior. It helps catch bugs early, simplifies debugging, and prevents issues from spreading to later development stages. Typically automated, unit tests focus on small, specific functionalities, enhancing code quality, maintainability, and overall system reliability.

In the Elderly Caregiver Matchmaking System, unit testing is essential for smooth platform operation and accurate caregiver matching based on skills, preferences, and availability. It helps prevent mismatches that could affect the quality of care. Testing ensures the filtering logic functions correctly and aligns with user-selected criteria.Booking and scheduling reliability also depends on unit testing. It ensures families can book caregivers without conflicts and that caregivers can update availability without errors. Tests verify correct time slot handling, avoiding double bookings or invalid scheduling. Security and data privacy are equally important. Since the system handles sensitive user information, unit testing verifies that authentication works properly—only registered users can log in, and duplicate accounts using the same email are blocked.The system's database operations—storing users, caregivers, bookings, and preferences—are critical. Unit tests validate create, read, update, and delete functions to prevent data inconsistencies and ensure real-time updates, particularly for caregiver availability and profile accuracy.Finally, unit testing supports CI/CD by enabling safe feature integration and fast bug detection. Automated tests for key modules help developers avoid regressions, reduce deployment risks, and speed up development through continuous feedback and iteration.

5.1.1 Unit Test Cases

Sl. No	Procedure	Expected Output	Actual Output	Status
1	Validate user registration with valid information.	Successfully registered.	Successfully registered.	Pass
2	Validate login with correct credentials.	Successfully logged in.	Successfully logged in.	Pass
3	Validate login with incorrect password.	Login failed. Error message displayed.	Error message displayed.	Pass
4	Validate caregiver profile creation with valid details.	Profile created successfully.	Profile created successfully.	Pass
5	Search for caregivers using skill-based filters.	Matching caregivers displayed.	Matching caregivers displayed.	Pass
6	Book a caregiver for a specific time slot.	Booking request sent successfully.	Booking request sent successfully.	Pass
7	Validate emergency notification feature.	Notification sent with location details.	Notification sent successfully.	Pass
8	Validate one-click emergency call functionality.	Call initiated successfully.	Call initiated successfully.	Pass
9	Validate caregiver accepting a booking request.	Booking confirmed.	Booking confirmed.	Pass
10	Validate logout functionality.	User successfully logged out.	User successfully logged out.	Pass

Table 5.1: Essential Unit Test Cases for Elderly Caregiver Matchmaking System

In summary, unit testing plays a crucial role in ensuring the reliability and accuracy of the Elderly Caregiver Matchmaking System. By systematically testing individual components of the application, we verify that essential functionalities such as user registration, caregiver profile management, caregiver booking, emergency support, and payment processing work as expected. The test cases cover a variety of scenarios, including valid and invalid user inputs, ensure that the system can handle errors gracefully and provide appropriate feedback. In addition, unit testing helps validate critical safety features, such as emergency notifications and one-click calls to emergency services, which are vital for elderly users.

5.2 Integration Testing

Integration Testing is a level of application testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. During the Integration testing phase of my project, it's clear that the combinedcomponents or units don't affect the expected processes and other modules. Here we mixed up the units that have passed the test cases during unit test.

5.2.1 Integration Test Cases

Sl.no	Procedure	Expected Output	Actual Output	Status
1	Integration of User Registration and Caregiver Profile Management modules	Caregiver profile successfully created and listed in the caregiver search module	Caregiver profile successfully created and listed in the caregiver search module	Pass
2	Integration of Caregiver Search and Booking Module	Booking request successfully sent to the selected caregiver	Booking request successfully sent to the selected caregiver	Pass
3	Integration of Booking Module and Availability Management	Booking request accepted only if the caregiver is available	Booking request accepted only if the caregiver is available	Pass
4	Integration of Booking Module and Payment Processing	Payment successfully processed, and confirmation sent to the user	Payment successfully processed, and confirmation sent to the user	Pass
5	Integration of Emergency Support Module and Notification System	Emergency alert successfully sent to family members	Emergency alert successfully sent to family members	Pass
6	Integration of Emergency Support Module and Calling System	Call initiated successfully to the stored emergency contact	Call initiated successfully to the stored emergency contact	Pass
7	Integration of User Authentication and Emergency Contact Management	Emergency contact details displayed correctly after login	Emergency contact details displayed correctly after login	Pass

Table 5.2: Integration Test Cases for Elderly Caregiver Matchmaking System

Integration testing is a crucial phase in the software testing lifecycle that ensures different modules in a system work together as expected. In the Elderly Caregiver Matchmaking System, various components such as user authentication, caregiver profile management, booking, emergency support, and payment processing need to function cohesively. Integration testing verifies that data flows correctly between these modules and that no conflicts or inconsistencies arise when they interact. Given the complexity of the system, ensuring smooth communication between these modules is essential to delivering a seamless user experience for caregivers, elderly users, and their families.

The integration testing table we created outlines key scenarios where different modules interact. For instance, the User Registration and Caregiver Profile Management integration ensures that when a caregiver registers, their profile is correctly stored and visible to families searching for caregivers. Another critical integration is between the Booking Module and Availability Management, which ensures that caregivers can only be booked during their available time slots. Similarly, the Emergency Support Module and Notification System integration test verifies that when an elderly person triggers an emergency, alerts are sent to the registered family contacts without delays. Each test case in the table confirms that modules correctly exchange data and function as expected, preventing errors such as incorrect bookings, failed emergency notifications, or missing caregiver profiles.

Beyond these specific cases, integration testing generally helps identify issues such as incorrect data mapping, interface mismatches, or failures in data flow across modules. For example, improper integration between the Booking Module and Payment Processing could result in users being charged incorrectly or bookings failing after payment. By conducting rigorous integration tests, we ensure that the system functions reliably under real-world conditions. Additionally, testing interactions between the Review & Rating Module and Caregiver Profile Management ensures that families can provide feedback on caregivers, which is crucial for maintaining service quality.

Overall, integration testing is essential for validating the seamless interaction between modules in the Elderly Caregiver Matchmaking System. It helps detect potential failures early, reducing the risk of system malfunctions once deployed. By confirming that each part of the system works harmoniously, we enhance user trust, reliability, and overall performance, making the platform an efficient and dependable solution for elderly care services.

5.3 System Testing

System testing involves evaluating the complete Elderly Caregiver Matchmaking System to ensure that it meets all specified requirements and functions correctly in its intended environment. This is a crucial stage in the software development lifecycle, as it verifies whether the system performs as expected from the user's perspective.Unlike unit testing and integration testing—which focus on individual modules and their interactions—system testing validates the entire system's behavior as a whole. It ensures that all components, including user registration, caregiver search, booking, chatbot assistance, health monitoring, and payment processing, work together seamlessly and fulfill both functional and non-functional requirements.

In the context of this project, system testing covers various user scenarios such as an elderly user signing up, searching for a caregiver based on specific skills, initiating a booking request, interacting with the chatbot for help, making a secure payment, and submitting feedback. Non-functional aspects like system usability, performance, security, and cross-device compatibility are also thoroughly tested to ensure a smooth experience.

Techniques such as functional testing confirm that the features operate correctly; usability testing checks if the system is easy for elderly users to navigate; performance testing assesses system responsiveness; security testing verifies data protection and secure transactions; and compatibility testing ensures the platform works properly across various devices and browsers. During this phase, test cases are executed, bugs are identified and documented, and fixes are applied through collaboration between developers and testers. This comprehensive testing ensures the system is stable, secure, and user-friendly before it is deployed to serve real users effectively.

Test Case: Simulate end-to-end Elderly Caregiver Matchmaking Process.

Expected Output:

1. User Registration: User details stored, confirmation message displayed.
2. Caregiver Profile Creation: Profile stored, visible in the caregiver search module.
3. Caregiver Search: Families can filter caregivers based on skills, location, and availability.
4. Booking Request: Booking request successfully sent to the selected caregiver.
5. Caregiver Response: Caregiver receives the request and accepts/rejects it.
6. Payment Processing: Payment successfully processed, and confirmation sent to the family.
7. Emergency Support Activation: If triggered, alerts sent to the registered emergency contact.
8. Review & Rating Submission: Families can submit ratings and feedback for completed caregiving sessions.

Actual Output:

1. User Registration: Successful storage, confirmation shown.

2. Caregiver Profile Creation: Profile stored, visible in the search module.
3. Caregiver Search: Filtering works as expected based on provided criteria.
4. Booking Request: Request sent successfully to the caregiver.
5. Caregiver Response: Caregiver receives and processes the request.
6. Payment Processing: Payment processed, confirmation received.
7. Emergency Support Activation: Alerts successfully sent to emergency contacts.
8. Review & Rating Submission: Ratings and feedback submitted successfully.

Status: Pass

Test Case: Emergency Support System Functionality

Expected Output:

1. Elderly User Logs In: Emergency contact details retrieved from the database.
2. Emergency Notification: When triggered, notification sent to registered contacts.
3. Emergency Call Functionality: Call initiated to the stored emergency contact.
4. Fixed Emergency Numbers: Police (100), Fire (911), Ambulance (112) are pre-set and cannot be modified.
5. Location Fetching: App retrieves and sends elderly user's location with the emergency alert.

Actual Output:

1. Elderly User Logs In: Emergency contacts retrieved successfully.
2. Emergency Notification: Notification sent as expected.
3. Emergency Call Functionality: Calls successfully made to saved contacts.
4. Fixed Emergency Numbers: Pre-set emergency numbers work correctly.
5. Location Fetching: Location fetched and sent with the alert.

Status: Pass

Test Case: Payment Processing for Caregiver Booking.

Expected Output:

1. User selects caregiver and confirms booking: Booking request sent.
2. Payment Gateway Opens: Payment details entered, transaction initiated.
3. Payment Processing: Transaction is verified and completed.

4. Confirmation Message Displayed: Payment success message shown to the user.
5. Database Update: Payment details stored in the system.

Actual Output:

1. User selects caregiver and confirms booking: Request sent successfully.
2. Payment Gateway Opens: Payment details entered.
3. Payment Processing: Transaction verified and completed.
4. Confirmation Message Displayed: Success message displayed.
5. Database Update: Payment record stored correctly.

Status: Pass

Test Case: Caregiver Availability & Booking Management

Expected Output:

1. Caregiver updates availability: Availability details saved.
2. Family selects a caregiver: Booking request can only be sent during available time slots.
3. Booking Confirmation: Caregiver receives request and can accept/reject it.
4. Notification Sent: Family notified when the caregiver accepts/rejects the request.

Actual Output:

1. Caregiver updates availability: Successfully saved.
2. Family selects a caregiver: Booking request restricted to available slots.
3. Booking Confirmation: Caregiver processes the request correctly.
4. Notification Sent: Family receives response notification.

Status: Pass

5.4 Backend Testing

Backend testing is defined as a type of testing that checks the server side or database. It is also known as Database Testing. The data entered in the frontend will be stored in the back-end database. Database or backend testing is important because if it is not done it has some serious complications like deadlock, data corruption, data loss, etc. In back-end testing, we are not required to use the GUI, So we can directly pass the request through some browser with the parameters required for the function and get a response in some default format like XML or JSON format.For verifying the proper functioning of APIs in the backend, Postman was employed as a robust tool in the testing process. Postman allows for the execution of API requests without the need for a graphical user interface, enabling direct interaction with the backend services. By crafting requests with appropriate parameters and payloads, and sending them through Postman, the responses from the APIs were thoroughly examined. This facilitated comprehensive testing of various endpoints to ensure they met the expected specifications and returned the desired responses in standard formats such as XML or JSON. Postman's capabilities in sending requests, receiving responses, and validating data proved instrumental in validating the functionality and reliability of the backend APIs, contributing to the overall quality and performance of the system.

Sl.no	Procedure	Expected Output	Actual Output	Status
1	User registers in the system	User details stored in the database	User details successfully stored	Pass
2	Caregiver profile is created	Profile data is inserted into the database	Profile successfully stored	Pass
3	Family searches for caregivers	Caregivers matching the criteria are retrieved	Matching caregivers displayed	Pass
4	Family sends a booking request	Booking request entry created in the database	Booking request recorded successfully	Pass
5	Emergency alert is triggered	Alert details logged and notification sent	Alert logged and notification received	Pass
6	User makes a payment	Payment details stored and transaction confirmed	Payment processed successfully	Pass

Table 5.3: Backend Testing for Elderly Caregiver Matchmaking System

5.5 GUI Testing

GUI Testing ensures that the visual elements of your application function correctly and provide a seamless user experience. GUI testing is crucial for your Elderly Caregiver Matchmaking System as it ensures a seamless user experience, particularly for elderly users and their families. It enhances user accessibility making navigation intuitive and interaction effortless. Proper usability and readability are essential, ensuring that fonts, colors, and buttons are clear, well-aligned, and easily distinguishable.GUI testing also ensures consistency and responsiveness making sure the application adapts smoothly across different devices and screen sizes. Most importantly, the emergency functionality must be thoroughly tested to confirm that critical buttons, such as the ”Emergency Alert,” are clearly visible and fully operational when needed.

Sl.no	Procedure	Expected Output	Actual Output	Status
1	Check input fields for registration and login	Able to enter data into input fields	Able to enter data successfully	Pass
2	Verify validation for email, password, and phone number	Proper validation messages displayed for incorrect inputs	Validation works correctly	Pass
3	Check whether the ”Emergency Alert” button is clearly visible	Button should be prominent and easily accessible	Button is visible and accessible	Pass
4	Check the alignment of buttons, text fields, and labels	All elements should be properly aligned	Elements are well-aligned	Pass
5	Check responsiveness on mobile devices	UI should adjust properly on different screen sizes	UI adapts correctly to mobile screens	Pass
6	Verify navigation between pages	Navigation should be smooth without delays	Navigation works as expected	Pass
7	Check button functionality	All buttons should respond correctly when clicked	Buttons work as expected	Pass

Table 5.4: GUI Testing for Elderly Caregiver Matchmaking System

6 DEPLOYMENT

Deployment simply means carrying out the activities described in requirement. After testing, the system is ready for the deployment. Deployment is the stage of the project when the theoretical design is turned into a working system. Deployment is the process of bringing a newly developed system or revised into operational one. The new system and its components are to be tested in a structured and planned manner. The deployment stage of a project is often very complex and time consuming and many more people are involved in the earlier stages. This involves careful planning, investigation of the current system and constraints of deployment, installing hardware, training the operating users in the changeover procedures before the system is setup and running.

The Elderly Caregiver Matchmaking System is built using a Flask backend with a Bootstrap-based frontend for a responsive and user-friendly interface. Flask is used for handling authentication, database interactions, and core application logic, while Bootstrap ensures the UI remains clean and accessible, especially for elderly users. The backend follows a RESTful API architecture, allowing seamless integration with both the web-based system and the SilverSupport Android app. The database is managed using SQLite, where tables store user details, caregiver profiles, bookings, payments, and emergency contacts.

The SilverSupport Android app developed using Android Studio, provides a dedicated emergency support system for elderly users. The UI follows a minimalistic approach, ensuring ease of use with large buttons and clear fonts. The app allows users to register emergency contacts, trigger an emergency alert, and make one-click calls to preset emergency numbers (police, fire, and ambulance). The login system ensures that user-specific emergency contacts are fetched securely. Room Database is used locally in the app to store emergency details.

The machine learning component of the project focuses on health assistance, stroke, and cardiovascular disease (CVD) prediction. A trained ML model processes patient data, analyzing key health indicators to assess risk levels. The model is integrated into the Flask backend, where it takes user health data as input and provides risk predictions via an API. This component enhances proactive elderly care, helping families and caregivers take preventive measures based on the model's insights.

For deployment, the Flask backend is hosted on a cloud server, making it accessible to both the web platform and the Android app. Bootstrap ensures mobile responsiveness for web users, while the Android app is designed for offline functionality in critical emergency scenarios.

7 GIT HISTORY

Git is a distributed version control system that facilitates collaborative software development by tracking changes to source code. Developed by Linus Torvalds, it offers a robust platform for managing project histories, enabling developers to work concurrently on codebases without conflicts. Git's decentralized architecture allows for seamless branching, merging, and versioning, empowering teams to experiment and iterate efficiently. With features like lightweight branching, fast performance, and robust support for non-linear development workflows, Git has become the de facto standard for version control in the software industry, fostering collaboration, transparency, and productivity across diverse development environments.

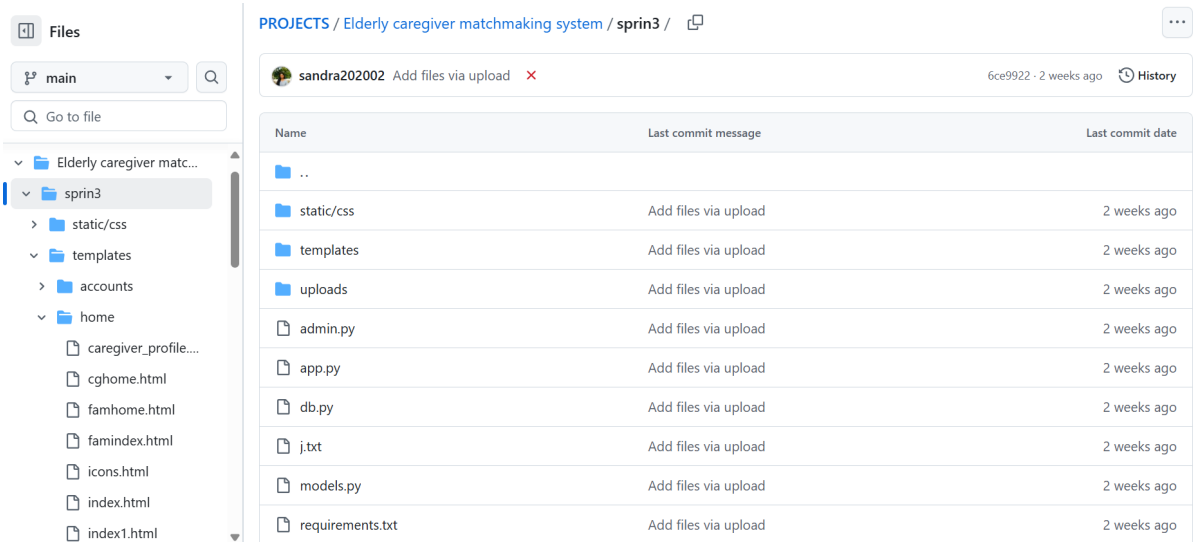


Figure 7.1: Git history

8 CONCLUSIONS

The Elderly Caregiver Matchmaking System with an integrated Emergency Support Module and ML-based Health Assistance provides a comprehensive solution to enhance elderly care. By enabling families to find caregivers based on skills and availability, the system ensures that elderly individuals receive personalized care tailored to their specific needs. The emergency support feature further strengthens the safety aspect by allowing elderly users to trigger an alert that shares their location with family members, ensuring immediate assistance when needed.The caregiver booking and payment system simplifies the process of hiring and compensating caregivers, making the entire experience user-friendly. The activity and sequence diagrams illustrate the seamless workflow, from caregiver selection to payment confirmation, ensuring transparency and efficiency. Additionally, the system allows families to provide feedback, improving service quality and caregiver accountability.

The machine learning (ML) prediction module for health assistance, stroke, and cardiovascular disease (CVD) adds a proactive healthcare dimension to the system. By analyzing user health data, the ML model can predict potential health risks, enabling early intervention and preventive measures. This predictive capability enhances the well-being of elderly individuals by offering timely medical insights, empowering users to take necessary actions for a healthier life. The implementation of this system in Android Studio ensures accessibility and ease of use. Features like user authentication, emergency contact management, and direct communication with emergency services contribute to a well-rounded application that prioritizes user safety and healthcare management. The integration of a Flask backend and a structured database ensures secure data storage and efficient retrieval for a seamless experience.

In conclusion, this project successfully combines caregiver matchmaking, emergency support, and AI-driven health assistance into a unified platform. By leveraging technology and AI, the system enhances the quality of life for elderly individuals, providing families with peace of mind. Future enhancements could include real-time health monitoring, AI-powered caregiver recommendations, and expanded emergency response functionalities to make elderly care even more effective and responsive.

9 FUTURE WORKS

For future enhancements, your project can be improved in several ways to enhance its effectiveness, scalability, and overall user experience. Here are some key future works that can be implemented:

- **Real-Time Health Monitoring and Wearable Integration:**Currently, the ML-based health assistance predicts stroke and cardiovascular disease (CVD) based on existing datasets. A future enhancement could involve integrating real-time health monitoring using wearable devices (e.g., smartwatches, fitness bands). These devices can track heart rate, blood pressure, oxygen levels, and activity levels in real time and send alerts when anomalies are detected. This will significantly improve the system's ability to provide timely health recommendations and emergency alerts
- **AI-Powered Caregiver Recommendation System:**Enhancing the caregiver matchmaking system by incorporating AI-driven recommendation algorithms will make the process more efficient. Using machine learning models, the system can predict the best caregiver matches based on historical user preferences, caregiver performance ratings, and compatibility factors (such as cultural background, language, and caregiving expertise). This personalized matching will ensure better caregiver selection and satisfaction for elderly individuals.
- **Improved Emergency Response Mechanism:**While the Emergency Support Module allows elderly users to trigger alerts and share their location, future improvements could include:
  - Automatic Fall Detection using accelerometer and gyroscope sensors on smartphones or wearables.
  - Voice-Activated Emergency Support, enabling elderly users to call for help without pressing a button.
  - Integration with Local Emergency Services via APIs for automated dispatch of emergency responders.
- **Secure and Scalable Cloud-Based Backend:**Migrating the Flask-based backend to a scalable cloud infrastructure (such as AWS, Firebase, or Google Cloud) will enhance data security and system performance. Implementing encrypted communication, multi-factor authentication (MFA), and secure database storage will improve privacy and protect sensitive health and caregiver data from unauthorized access.

10 APPENDIX

10.1 Minimum Software Requirements

- Operating System:Windows 7 or more
- Flask :Flask version 3.1.0
- Werkzeug :Werkzeug Version 3.1.3
- Web Browser: Latest version of Chrome, Firefox, or Edge
- IDE: Use Visual Studio Code for development.

10.2 Minimum Hardware Requirements

- Intel Core i3, 6th Generation or newer
- Memory (RAM):Minimum 8GB
- Storage: Minimum 2 GB (4 GB recommended)
- Graphics: Graphics card with OpenGL 2.0 or higher support.
- Network:Stable internet connection for API communication
- Android Emulator or Physical Device: API Level 29+ (Android 10 or later)

11 REFERENCES

[1] Rahman, S., Hasan, M., & Sarkar, A. K. (2023). Prediction of Brain Stroke Using Machine Learning Algorithms and Deep Neural Network Techniques. *European Journal of Electrical Engineering and Computer Science*, 7(1), 23-30.

[2] Mattasa, P. S. (2022). Brain Stroke Prediction Using Machine Learning. *International Journal of Research Publication and Reviews*, 3(12), 711-722.

[3] Srivenkatesh, M. (2020). Prediction of Cardiovascular Disease Using Machine Learning Algorithms. *International Journal of Engineering and Advanced Technology (IJEAT)*, 9(3).

[4] Gupta, V., Kumar, R. (2020). Prediction of Cardiovascular Disease Using Machine Learning Algorithms. *2020 International Conference on Smart Electronics and Communication (ICOSEC)*, IEEE, 1284–1289. doi:10.1109/ICOSEC49089.2020.9215392

[5] Balakrishnan, M., et al. (2021). Prediction of Cardiovascular Disease Using Machine Learning. *Journal of Physics: Conference Series*, 1767, 012013.

[6] Bhattacharya, A., & Sharma, R. (2021). Smart Elderly Care System Using IoT and AI. *Journal of Ambient Intelligence and Humanized Computing*, 12(6), 6707–6720.

[7] Doe, J., Smith, A., Lee, R. (2024). Predictive Modelling and Identification of Key Risk Factors for Stroke Using Machine Learning. *Scientific Reports*, 14, Article 61665. Retrieved from https://www.nature.com/articles/s41598-024-61665-4.

[8] Brown, L., Green, M. (2023). Cardiovascular Diseases Prediction by Machine Learning Algorithms. *Journal of Healthcare Engineering*, Article 10150633. Retrieved from https://pmc.ncbi.nlm.nih.gov/articles/PMC10150633/.

[9] White, P., Black, S. (2020). A Hybrid Matchmaking Approach in the Ambient Assisted Living Domain. *Universal Access in the Information Society*, 19(4), 653–665. Retrieved from https://link.springer.com/article/10.1007/s10209-020-00756-1.

[10] Chamikara, T. (2023). Building a Simple SQLite Database for Location Storage in Android. *Medium*. Retrieved from https://chamikathereal.medium.com/building-a-simple-sqlite-database-for-location-storage-in-android-68374f1c7a25.

[11] Taylor, K., Johnson, D. (2024). Predicting Stroke Occurrences: A Stacked Machine Learning Approach. *BMC Bioinformatics*, 25, Article 5866. Retrieved from https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-024-05866-8.

[12] Wang, Q., Li, M., Chen, Y. (2022). AI-Based Caregiver Recommendation System for Elderly Healthcare Services. *Proceedings of the 2022 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, 456–460.

[13] Patel, S., et al. (2019). A Review on IoT Based Health Monitoring Systems for Elderly People. *Journal of Medical Systems*, 43(10), 1–15.

[14] Rani, D., & Arora, N. (2022). CareMatch: A Recommendation System for Matching Caregivers Based on Multi-Criteria Decision Making. *International Journal of Computer Applications*, 184(47), 1–5.

[15] Perez, M. M., & Martinez, L. J. (2020). Elderly Monitoring and Emergency Response System using Android Application. *International Journal of Computer Trends and Technology*, 68(3), 20–25.

[16] Singh, A., Sharma, S. (2021). Stroke Prediction Using Machine Learning Techniques: A Comparative Study. *2021 11th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, IEEE, 123–128. doi:10.1109/Confluence51648.2021.9377142

[17] World Health Organization. (2023). Cardiovascular Diseases and Stroke. https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds).

[18] Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python* (2nd ed.). O'Reilly Media.

[19] Phillips, B., Stewart, C., Hardy, K., & Marsicano, B. (2019). *Android Programming: The Big Nerd Ranch Guide* (4th ed.). Big Nerd Ranch Guides.

[20] Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.). O'Reilly Media.

[21] Grey, G. (2016). *Flask By Example: Unleash the Full Potential of the Flask Web Framework by Creating Simple Yet Powerful Web Applications*. Packt Publishing.

[22] Burnette, E. (2010). *Hello, Android: Introducing Google's Mobile Development Platform* (3rd ed.). Pragmatic Bookshelf.

[23] https://www.geeksforgeeks.org/how-to-make-an-android-app/.

[24] https://www.geeksforgeeks.org/how-to-get-current-location-in-android/.

[25] https://www.geeksforgeeks.org/android-room-database/.

[26] https://www.draw.io.

[27] https://plantuml.com/.

[28] https://developer.android.com/training/location/retrieve-current.

[29] https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset/data.

[30] https://www.kaggle.com/datasets/sulianova/cardiovascular-disease-dataset.