The Egyptian E-Learning University

Faculty of Computers and Information Technology

Recommendation system for diabetic patient to recommend lifestyle

Supervised By: Dr. Amira Idris

Teaching Assistant: Eng. Raghda Mohamed

Team Member:

Student Name	ID
Momen Abdalla Mohamed Saleh	2000557
Marwa Hamdy Abd El Maqsood	2001641
Ahmed Mohamed Fathy Oraby	2001685
Hussien Mostafa Lotfy	2001091
Mohamed Sayed Ismail	2001856
Mahmoud Tantawy Mahmoud	2001993
Ahmed Nabil Mohamed Fathy	2000474
Abdelrhman Gamal Ishaq	2002015

ABSTRACT

As the prevalence of diabetes continues to rise globally, the effective management of this chronic condition becomes paramount. Lifestyle modifications, encompassing diet, exercise, sleep, and stress management, play a crucial role in controlling diabetes and improving overall well-being. This graduation project introduces a novel Recommendation System designed to assist diabetic patients in adopting personalized and sustainable lifestyle changes. The proposed system leverages machine learning algorithms to analyze patient data, including medical history, lifestyle factors, and outcomes. Through a thoughtful combination of collaborative filtering and content-based filtering, the model generates tailored recommendations aimed at optimizing the patient's lifestyle for better diabetes management. The interface, developed with user experience in mind, presents clear and actionable suggestions to empower patients in their self-care journey. Ethical considerations are integral to the project, ensuring data privacy and adherence to healthcare regulations. The system's performance is evaluated rigorously, considering both quantitative metrics and qualitative user feedback. Additionally, the project emphasizes transparency through comprehensive documentation, facilitating seamless integration into healthcare practices. This recommendation system not only contributes to advancing personalized healthcare but also holds the potential to positively impact the lives of diabetic patients by fostering sustainable and health-promoting lifestyle changes. The project's findings and methodologies are presented in detail, providing a valuable resource for researchers, healthcare professionals, and individuals seeking innovative solutions in diabetes management.

Acknowledgment

First and foremost, our team would like to gratefully thank **Prof. Amira Idris** for her supporting leadership and flexibility throughout our project journey along the whole year. He is very insightful in directing research topics and also patient to listen to our opinions. Through the project work, he always helped us adjusting our direction towards the most useful, fruitful and innovative destination. We also express our benefit gratitude to our Computers and Information Technology Department as well as all the staff members throughout our four-year-long journey of education in The Egyptian E-Learning University

University. We are students of bachelor's degree in **Faculty of Computers and Information Technology** also thankful to our classmates and others who helped us in direct or indirect way in solving problems that faced us either in making our project more efficient or through our learning journey in general.

Table of Contents

Contents

Table of figure :

1 Chapter One

1. Introduction

2. Preamble

The project aims to develop a Recommendation System tailored for diabetic patients, providing personalized lifestyle suggestions to enhance their overall well-being and effectively manage their condition. In light of the escalating global prevalence of diabetes, there exists a pressing need for innovative solutions that empower individuals in making sustainable and health-promoting lifestyle choices. The recommendation system employs advanced machine learning algorithms, combining collaborative filtering and content-based filtering techniques.

By analyzing diverse patient data, encompassing medical history, lifestyle factors, and outcomes, the model generates individualized recommendations.

These recommendations span key lifestyle domains, including diet, exercise, sleep, and stress management. The significance of the project lies in its potential to revolutionize diabetes management by offering a proactive and personalized approach to lifestyle modification. As patients face unique challenges and preferences, a one-size-fits-all approach is often inadequate. This system aims to bridge this gap by delivering tailored recommendations, promoting patient engagement, and ultimately improving health outcomes. The project's scope is defined by its focus on lifestyle factors directly impacting diabetes management.

The software and hardware requirements encompass state-of-the-art machine learning frameworks and tools to ensure the efficacy and efficiency of the recommendation system. However, the project acknowledges certain limitations, such as potential challenges in data availability and the need for ongoing user feedback for continuous improvement. Anticipated project outputs include a fully functional recommendation system, a user-friendly interface, and a comprehensive report documenting the methodologies, results, and ethical considerations. The project schedule outlines key milestones, ensuring a systematic and timely execution of the various project phases. This project aspires to contribute to the broader field of personalized healthcare and lifestyle management, emphasizing transparency, ethical practices, and a user-centered approach. The report outline provides a roadmap for presenting the project's findings and insights in a structured and cohesive manner.

3. Problem Background

The background of the problem is shaped by the escalating prevalence of diabetes, a chronic condition with significant health implications worldwide. Diabetes, characterized by elevated blood glucose levels, poses a considerable challenge to global healthcare systems, affecting individuals across diverse demographics and geographical regions.

Epidemiological Trends:

Diabetes has reached epidemic proportions, with a rising global prevalence. Factors such as sedentary lifestyles, unhealthy dietary habits, and genetic predispositions contribute to the increasing incidence of diabetes.

Lifestyle Challenges:

Managing diabetes effectively often requires substantial lifestyle modifications, including changes in diet, physical activity, sleep patterns, and stress management. However, individuals face challenges in adopting and sustaining these changes due to various factors such as lack of awareness, motivation, and personalized guidance.

One-Size-Fits-All Approach:

Conventional approaches to diabetes management often rely on generalized guidelines, overlooking the diverse and individualized needs of patients. A one-size-fits-all strategy may not effectively address the unique circumstances and preferences of each individual, leading to suboptimal health outcomes.

Information Overload:

Diabetic patients are bombarded with vast amounts of health information from diverse sources, making it challenging to discern personalized and actionable advice. This information overload can lead to confusion and hinder the adoption of beneficial lifestyle changes.

Healthcare System Strain:

The increasing prevalence of diabetes exerts significant pressure on healthcare systems. Effective management of diabetes not only improves individual health but also contributes to the overall sustainability and efficiency of healthcare delivery.

In light of these challenges, there is a critical need for innovative solutions that leverage technology to provide personalized and accessible recommendations for lifestyle modification. The project aims to address these issues by developing a recommendation system that takes into account individual patient data, preferences, and unique health requirements. This personalized approach is expected to enhance patient engagement, improve adherence to lifestyle changes, and ultimately contribute to better diabetes management outcomes

Figure 1 Public Health

3.1. Problem Statement

Diabetes is a pervasive chronic condition requiring individuals to make crucial lifestyle changes, encompassing diet, exercise, sleep, and stress management. However, diabetic patients often encounter challenges in implementing and sustaining these changes, leading to suboptimal health outcomes. The existing gap lies in the absence of a personalized recommendation system that considers individual health profiles, preferences, and motivational factors to guide diabetic patients in making and maintaining effective lifestyle modifications.

Generic Lifestyle Guidelines:

Current diabetes management primarily relies on generic lifestyle guidelines that do not adequately consider the diverse needs and preferences of individual patients. As a result, the one-size-fits-all approach may not be effective in addressing the unique circumstances of each diabetic individual.

Limited Personalization in Recommendations:

Diabetic patients lack access to personalized lifestyle recommendations that align with their specific health conditions, preferences, and daily routines. A tailored approach is crucial for encouraging adherence to lifestyle modifications and optimizing health outcomes.

Insufficient Motivational Support:

Sustaining lifestyle changes requires ongoing motivation and support.

Many diabetic patients face challenges in staying motivated due to the absence of personalized feedback, encouragement, and tangible incentives tailored to their individual progress and goals.

Data Overload and Information Ambiguity:

Patients are often overwhelmed with information from various sources, making it challenging to discern relevant advice for their unique situations. An effective recommendation system must filter through this information overload to provide clear and actionable guidance.

Underutilization of Technology:

While technology has the potential to revolutionize healthcare, there is a gap in leveraging advanced technologies, such as machine learning, to analyze individual patient data comprehensively. Integrating these technologies can enhance the precision and personalization of lifestyle recommendations.

The problem at hand is the absence of a sophisticated recommendation system that utilizes cutting-edge technologies to provide diabetic patients with personalized and actionable lifestyle recommendations. This project aims to address this gap by developing an intelligent recommendation system that analyzes individual health data, preferences, and motivational factors to empower diabetic patients in making informed and sustainable lifestyle choices.

3.2. Significance of the Project

The proposed recommendation system for lifestyle modification in diabetic patients holds significant importance in several key areas, contributing to advancements in healthcare, patient well-being, and the broader societal impact. The project's significance is outlined as follows.

Personalized Healthcare:

The project addresses the critical need for personalized healthcare solutions in diabetes management. By tailoring lifestyle recommendations to individual patient profiles, the system goes beyond generic guidelines, offering a more precise and effective approach to treatment.

Improved Health Outcomes:

Personalized lifestyle recommendations are expected to lead to improved health outcomes for diabetic patients. By considering individual health conditions, preferences, and motivational factors, the system aims to enhance patient adherence to recommended lifestyle changes, ultimately resulting in better disease management.

Empowerment of Patients:

The project empowers diabetic patients by providing them with actionable and personalized guidance. This empowerment is crucial for fostering a sense of control and autonomy in managing their health, thereby promoting a proactive and engaged approach to lifestyle modifications.

Reduction in Healthcare Costs:

Effective diabetes management can contribute to the reduction of healthcare costs associated with complications and long-term treatment. By promoting preventive measures through lifestyle changes, the project aligns with the broader goal of creating a more sustainable and cost-effective healthcare system.

Utilization of Advanced Technologies:

The incorporation of machine learning algorithms showcases the potential of advanced technologies in healthcare. The project serves as an exemplar of how technology can be harnessed to analyze complex health data and provide meaningful insights, setting a precedent for future advancements in the field.

Patient-Centric Approach:

The patient-centric nature of the recommendation system fosters a more holistic and compassionate approach to healthcare. By considering individual preferences and motivations, the project aligns with the shift towards patient-centered care, emphasizing the importance of the patient's experience and well-being.

Research Contribution:

The project contributes to the academic and research landscape by exploring innovative solutions in the intersection of healthcare and technology. It provides a platform for further studies and advancements in personalized medicine and recommendation systems for chronic disease management.

Community Health Impact:

The potential scalability of the recommendation system can extend its impact to a broader community of diabetic patients. As individuals adopt healthier lifestyles, the cumulative effect can lead to positive changes in community health and well-being.

In conclusion, the significance of the project lies in its potential to revolutionize diabetes management through personalized, technology-driven solutions. By addressing the unique challenges faced by diabetic patients, the project contributes to a paradigm shift in healthcare towards more individualized, effective, and patient-friendly approaches.

3.3. Project Aim and Objectives

Aim:

The aim of this project is to design, develop, and implement a robust recommendation system for lifestyle modification tailored to the unique needs of diabetic patients, with a focus on improving overall health outcomes and enhancing patient well-being.

Objectives:

Literature Review:

Conduct an extensive review of existing literature on diabetes management, personalized healthcare, and recommendation systems.

Identify key methodologies, technologies, and gaps in the current approaches to lifestyle recommendations for diabetic patients.

Data Collection and Preprocessing:

Collect diverse datasets, including patient health records, lifestyle data, and relevant medical information.

Clean and preprocess the collected data, addressing issues such as missing values, outliers, and data format standardization.

Feature Identification and Engineering:

Identify crucial features relevant to lifestyle modification in diabetes management.

Explore and implement feature engineering techniques to enhance the effectiveness of the recommendation system.

Model Development:

Choose and implement suitable machine learning algorithms for the recommendation system, considering collaborative filtering and content-based filtering approaches.

Train and fine-tune the model using the preprocessed data to optimize its performance.

User Interface Design:

Design an intuitive and user-friendly interface for diabetic patients to interact with the recommendation system.

Ensure the interface accommodates individual preferences, provides clear recommendations, and promotes user engagement.

Integration of Recommendation Model:

Integrate the trained machine learning model seamlessly into the recommendation system.

Establish effective communication between the model and the user interface to deliver real-time, personalized recommendations.

Ethical Considerations:

Address ethical concerns related to data privacy, informed consent, and confidentiality of patient information.

Implement security measures to ensure the responsible handling of sensitive healthcare data.

Evaluation and Validation:

Evaluate the recommendation system's performance using relevant metrics, including accuracy, precision, and user satisfaction.

Validate recommendations through comparison with expert opinions or established guidelines.

Solicit feedback from users and healthcare professionals to assess the system's usability and effectiveness.

Documentation and Reporting:

Document the entire project, including methodologies, implementation details, and outcomes.

Prepare a comprehensive report outlining the project's findings, challenges, and contributions.

Dissemination and Knowledge Sharing:

Present the project's findings through academic channels, conferences, or publications.

Share knowledge and insights gained from the project with the broader healthcare and research communities.

Iterative Improvement:

Gather feedback from users and stakeholders for continuous improvement of the recommendation system.

Iterate on the system based on feedback, technological advancements, and emerging best practices in healthcare.

By accomplishing these objectives, the project aims to provide a valuable and innovative solution that enhances the quality of diabetes management through personalized lifestyle recommendations.

3.4. Project Scope

The scope of this project is defined by its focus on developing a recommendation system for lifestyle modification tailored specifically for diabetic patients. The project encompasses various aspects related to personalized healthcare and technological integration. The scope is outlined as follows.

Diabetic Patient Population:

The recommendation system targets individuals diagnosed with diabetes, covering a range of diabetic types and severity levels.

It considers the diverse needs and preferences of diabetic patients across different demographic groups.

Lifestyle Factors:

The project addresses lifestyle modification in the context of diet, exercise, sleep, and stress management, which are crucial components in diabetes management.

Recommendations are personalized based on individual patient data, health conditions, and lifestyle preferences.

Data Sources:

The project utilizes diverse datasets, including electronic health records, lifestyle data, and relevant medical information.

Data sources may include historical patient data, health monitoring devices, and self-reported lifestyle information.

Machine Learning Techniques:

The project employs machine learning algorithms, including collaborative filtering and content-based filtering, to analyze patient data and generate personalized recommendations.

The focus is on developing a model that adapts to individual patient profiles and continuously improves with user feedback.

User Interface:

The user interface is designed to be user-friendly and accessible for diabetic patients.

It provides a platform for patients to interact with the recommendation system, view personalized suggestions, and track their progress.

Ethical Considerations:

The project addresses ethical considerations related to patient privacy, data security, and informed consent.

It ensures compliance with healthcare regulations and standards, prioritizing the responsible handling of sensitive health information.

Integration and Deployment:

The recommendation system is integrated into a deployable solution, potentially within a controlled environment, such as a healthcare facility or a pilot program.

The project considers the scalability and adaptability of the system for broader deployment in healthcare settings.

Evaluation Metrics:

The project evaluates the recommendation system's performance using metrics such as accuracy, precision, and user satisfaction.

The evaluation considers the effectiveness of the system in providing personalized and actionable lifestyle recommendations.

3.5. Project Software and Hardware Requirements

Software Tools:

Python:

Used as the primary programming language for the implementation of the recommendation system.

NumPy:

Leveraged for numerical operations and efficient handling of arrays and matrices, which can be crucial for data manipulation and machine learning model development.

Streamlet:

Employed for building interactive web-based user interfaces. Streamlit is known for its simplicity and ease of use, making it an excellent choice for creating data-driven applications.

Machine Learning Framework:

K-Nearest Neighbors (KNN):

Implemented for developing the recommendation system. KNN is a straightforward and effective algorithm for collaborative filtering, which can be suitable for personalized recommendations based on user similarities.

Development Environment:

Integrated Development Environment (IDE):

Popular IDEs like PyCharm, Visual Studio Code, or Jupyter Notebook can be used for Python development, depending on your team's preferences.

Collaboration and Project Management Tools:

Git:

Utilized for version control and collaborative development.

Communication Tools:

Platforms like Slack, Microsoft Teams, or other communication tools to facilitate collaboration among team members.

3.6. Project Limitations

While the recommendation system for lifestyle modification in diabetic patients aims to provide valuable insights and personalized guidance, it is important to acknowledge certain limitations that may impact the scope and applicability of the project. These limitations include:

Data Availability and Quality:

The effectiveness of the recommendation system heavily relies on the availability and quality of data. Incomplete or inaccurate data may lead to suboptimal recommendations and affect the overall performance of the system.

Generalization to Diverse Populations:

The project's development may be focused on specific demographic groups or data sources, potentially limiting the generalization of recommendations to a broader and more diverse population of diabetic patients.

User Engagement and Adherence:

The success of lifestyle recommendations depends on user engagement and adherence. The project may not account for varying levels of motivation, socioeconomic factors, or other personal barriers that can impact a user's ability to adhere to suggested lifestyle changes.

Machine Learning Model Constraints:

The chosen machine learning model, such as the K-Nearest Neighbors (KNN) algorithm, has inherent limitations. It may struggle with scalability, especially with large datasets, and may not capture complex patterns in the data as effectively as more sophisticated models.

Dynamic Health Conditions:

Health conditions of diabetic patients can be dynamic and may change over time. The recommendation system may not dynamically adapt to changing health statuses, leading to recommendations that become less relevant over time.

Interactions with Healthcare Professionals:

The project may not replace the need for direct interactions with healthcare professionals. It should be considered as a supplementary tool rather than a substitute for personalized medical advice.

Privacy Concerns:

The project must adhere to privacy regulations, but inherent risks related to data breaches or unauthorized access may still pose concerns. Ensuring robust security measures is crucial to mitigate these risks.

Limited Coverage of Lifestyle Factors:

The scope of lifestyle factors considered may be limited to certain domains (diet, exercise, sleep, and stress). Other influential factors, such as social determinants of health, may not be fully incorporated.

Technology and Access Barriers:

Patients with limited access to technology or low digital literacy may face challenges in utilizing the recommendation system effectively. This may introduce disparities in access and utilization.

Continuous Model Improvement:

The project may lack continuous improvement mechanisms. Machine learning models require ongoing refinement based on user feedback and evolving health guidelines, which may not be fully addressed in the project timeline.

3.7. Project Expected Output

The culmination of the recommendation system for lifestyle modification in diabetic patients is anticipated to result in a comprehensive and functional system that delivers

personalized recommendations and facilitates a positive impact on diabetes management.

The expected outputs include:

User-Friendly Interface:

A user-friendly web-based interface developed using Streamlit that allows diabetic patients to interact with the recommendation system seamlessly.

Personalized Lifestyle Recommendations:

Tailored lifestyle recommendations for individual diabetic patients, encompassing dietary suggestions, exercise routines, sleep guidelines, and stress management strategies.

Real-Time Feedback:

Implementation of real-time feedback mechanisms, enabling users to receive instant insights into their lifestyle choices and their impact on diabetes management.

Integration of K-Nearest Neighbors (KNN) Model:

Successful integration and deployment of the KNN algorithm-based machine learning model for collaborative filtering in generating personalized recommendations.

Data Privacy Measures:

Implementation of robust data privacy measures to ensure the confidentiality and security of patient data, complying with relevant healthcare regulations and standards.

Documentation and User Guide:

Comprehensive documentation outlining the methodologies, algorithms, and implementation details of the recommendation system.

A user guide providing instructions on how to use the system, interpret recommendations, and navigate the user interface.

Evaluation Metrics and Reports:

Evaluation metrics assessing the performance of the recommendation system, including accuracy, precision, and user satisfaction.

Reports summarizing the findings of the system's evaluation, highlighting strengths, areas for improvement, and user feedback.

Ethical Considerations Addressed:

A clear articulation of ethical considerations related to data privacy, informed consent, and responsible use of healthcare information.

Measures to mitigate ethical risks and ensure the ethical deployment of the recommendation system.

Iterative Improvements and Future Recommendations:

Documentation outlining potential areas for future improvements and iterations, considering user feedback, technological advancements, and evolving healthcare guidelines.

Presentation Materials:

Materials for project presentations, including slides, visual aids, and demonstrations showcasing the functionality and benefits of the recommendation system.

Deployment Plan:

A deployment plan outlining the steps for deploying the recommendation system in a controlled environment or pilot program, ensuring scalability and adaptability.

Knowledge Transfer:

Transfer of knowledge to relevant stakeholders, including healthcare professionals and system administrators, to facilitate effective use and maintenance of the recommendation system. By achieving these expected outputs, the project aims to deliver a tangible and valuable solution that contributes to personalized diabetes management and enhances the well-being of diabetic patients. Regular feedback, continuous improvement, and adaptability to emerging healthcare trends will be crucial for the sustained success of the recommendation system.

3.8. Project Schedule

Creating a detailed project schedule is crucial for effective planning, execution, and timely delivery. The schedule below outlines key milestones and tasks for the recommendation system project. Keep in mind that the timeline may vary based on the project's complexity, team size, and unforeseen challenges.

3.9. Phase 1: Project Initiation (Weeks 1-2)

Define Project Scope and Objectives:

Review and finalize project scope, objectives, and deliverables.

Literature Review:

Conduct an extensive literature review on diabetes management, personalized healthcare, and recommendation systems.

Setup Development Environment:

Set up the development environment, including selecting and installing necessary tools and frameworks.

3.10. Phase 2: Data Collection and Preprocessing (Weeks 3-4)

Collect Datasets:

Identify and collect relevant datasets, ensuring diversity and representativeness.

Data Preprocessing:

Clean and preprocess collected data, handling missing values, outliers, and standardizing formats.

3.11. Phase 3: Feature Engineering and Model Development (Weeks 5-8)

Identify Key Features:

Identify and select key features for lifestyle recommendations.

Feature Engineering: Apply feature engineering techniques to enhance the effectiveness of the recommendation system.

Model Selection and Development:

Choose the K-Nearest Neighbors (KNN) algorithm and develop the machine learning model.

3.12. Phase 4: User Interface Design and Integration (Weeks 9-12)

Design User Interface:

Design an intuitive and user-friendly interface using Streamlit.

Integration of Model:

Integrate the KNN model with the user interface for real-time recommendations.

3.13. Phase 5: Evaluation and Optimization (Weeks 13-16)

Evaluation Metrics:

Implement and analyze evaluation metrics, including accuracy and precision.

Optimization:

Optimize the model and user interface based on initial evaluation results.

Phase 6: Documentation and Reporting (Weeks 17-18)

Project Documentation:

Document methodologies, algorithms, and implementation details.

User Guide:

Develop a comprehensive user guide for diabetic patients.

4. Chapter Two: Related Existing Systems

4.1. Introduction

The purpose of this chapter was to conduct theoretical and empirical literature done on diabetes Type 2 and identify existing technological solutions for a food recommendation to diabetes Type 2 patients. It also illustrated a detailed description of the used machine learning models, algorithms, architecture, system, and the conceptual framework with the operationalized variables. Finally, it illustrated the existing research gaps in the literature that the study hoped to resolve.

4.2. Diabetes management

4.2.1. Global perspective of diabetes

National Institute of diabetes and digestive and kidney disease (NIDDK) defines Diabetes mellitus also known as diabetes as a chronic disease that occurs when blood glucose is too high. This occurs when the pancreas does not produce enough insulin or when the body cannot effectively utilize the insulin produced.

Diabetes Type 1: causes the body to produce very little or no insulin; hence a diabetes patient is required to administer insulin on daily basis to maintain the glycaemia to the appropriate levels. Diabetes Type 1is mostly seen in children and adolescent.

Diabetes Type 2 occurs when the body cannot properly utilize the produced insulin. Diabetes Type 2 is mostly seen in adults, but lately, the numbers are increasing in children and adolescents due to the increase of obesity cases, decrease in physical activity and unhealthy eating habits. The third one is Gestational diabetes which is hyperglycemia with glycaemia values seen to be above normal occurring during pregnancy, this can occur to a woman who has never been diagnosed with diabetes before. KJ Hunt et al., (2008) gestational diabetes affects close to 4% of all pregnant women in the world's population. According to IDF (2019), Type 2 diabetes is the most common type of diabetes covering 90% of the world's diabetes population. WHO, (2017) reported that the number of diabetes patients rose from 108 million in 1980 to 422 million in 2014, with an estimate of 1.6 million deaths in 2015 caused by diabetes. In 2016 diabetes was the direct cause of 1.6 million deaths. Between 2000 and 2016, there was a 5% increment in premature mortality from diabetes. According to IDF (2019), a report projected that by 2030 there will be 578.4 million, and by 2045, 700.2 million adults aged between 20-79 years will be diabetes patients. This is enough proof that diabetes is a major public health problem that calls for immediate action to help prevent and manage it. Fig 2.1 illustrates the worldwide diabetes estimates and projections according to IDF.

At a glance	2019	2030	2045
Total world population	7.7 billion	8.6 billion	9.5 billion
Adult population (20–79 years)	5.0 billion	5.7 billion	6.4 billion
Diabetes (20–79 years)			
Global Prevalence	9.3%	10.2%	10.9%
Number of people with diabetes	463.0 million	578.4 million	700.2 million
Number of deaths due to diabetes	4.2 million	-	-
Total health expenditures for diabetes[1]	USD 760.3 billion	USD 824.7 billion	USD 845.0 billion
Hyperglycaemia in pregnancy (20–49 years)			
Proportion of live births affected	15.8%	14.0%[2]	13.3%[2]
Number of live births affected	20.4 million	18.3 million	18.0 million
Impaired glucose tolerance (20–79 years)			
Global prevalence	7.5%	8.0%	8.6%
Number of people with impaired glucose tolerance	373.9 million	453.8 million	548.4 million
Type 1 diabetes (0–19 years)			
Number of children and adolescents with type 1 diabetes	1,110,100	-	-
Number of newly diagnosed cases each year	128,900	-	-

Figure 2.1 Worldwide diabetes estimates and projections according to IDF 2019

Figure 2.1Worldwide diabetes

	2019		2030		2045	
	Number of people with diabetes (millions)	Prevalence (%)	Number of people with diabetes (millions)	Prevalence (%)	Number of people with diabetes (millions)	Prevalence (%)
Men	240.1	9.6	296.7	10.4	357.7	11.1
Women	222.9	9.0	281.8	10.0	342.5	10.8

Figure 2.2 The total number diabetes men and women worldwide IDF(2019)

Figure 3.1 the total number of diabetes men and women

4.2.2. Diabetes management

Management of diabetes differs depending on the type and severity of diabetes. For type 1 diabetes patients, insulin becomes part of an individual life, this is because it is a life-long treatment. It involves a wide variety of doses that include multiple dose injections or insulin pumps. Diabetes Management also incorporates monitoring glycemic control and to ensure this is achieved optimally one is required to self-monitor their glucose levels multiple times to modify insulin intake, diet, or physical activity as required. According to Povey& Carter (2007), consistency in diabetes self-management of diabetes is associated with the fulfilment of health outcomes in terms of good blood glucose control and fewer complications linked to diabetes, improved quality of life and a decline in diabetes-related death risks. Self-management refers to day to day activities an individual must undertake to reduce the impact of a disease on their health and wellbeing and as a result prevent further complications NM Clark et al., (1991 as cited in Mary D. Adu et al., 2019). Diabetes self-management will involve constant involvement in recommended behavioral activities such as healthy eating habits, physical activity, and daily blood glucose monitoring, which are all very crucial to successfully manage diabetes. According to ADA (2018) recommendations, it recognizes the integral role of nutritional

therapy in overall diabetes management and has commended that each diabetes patient is actively engaged in self-management, education, and treatment planning with their health caregiver. Institute of Medicine (IOM) 1991 as cited in ADA (2013), published a report that showed evidence demonstrating that medical nutrition therapy (MNT) can improve clinical outcomes while cutting down the cost of general Medicare for managing diabetes. Every diabetic patient has nutritional needs, and it is ideal for them to be referred to a registered dietitian or a credentialed nutrition professional for nutrition therapy at or soon after diagnosis

4.3. Existing System (Models)

4.3.1. Transtheoretical model

It is a model associated with the likelihood of change in individual behavior and provides approaches to guide the individual. (Prochaska et al., 1970 as cited in Alison Hammond 2010). With diabetes management, a patient is encouraged to adopt new behaviors such as healthy eating habits and physical activity. A Diabetes type 2 patient attempting to adopt the new behavior must move through five stages, precontemplation, contemplation, preparation, action, and maintenance. In precontemplation stage information about the problem is shared illustrating the advantages of following a healthy lifestyle. In the Contemplation stage, the patient has begun to think about change, they start evaluating the benefits and barriers of change. In Preparation the stage there is clear goal setting which must be SMART. In the Action stage, the patient makes specific lifestyle behavior modifications. For example, always eating a balanced diet. Finally, in

the maintenance stage, the person has adopted the new lifestyle for more than 6 months and are doing everything possible, not relapse. For example, they have managed to control their glycaemia to the recommended levels. A transtheoretical model is effective in changing the nutritional behavior in patients, Hashemzadeh M et al., (2019).DM type 2 is greatly influenced by the patients' lifestyle therefore the theory is a guideline to help them understand the process of change when it comes to adopting new behavior. It focuses on enhancing motivation to the patient to eat a healthy diet, adopt physical activity and monitor their blood glucose levels. After the patient is motivated to change behavior, they can maintain the new habit of living a healthy lifestyle

4.3.2. Factors affecting nutritional dietary for diabetes Type 2 patients

A study conducted by Shamsi N et al., (2013) illustrates that improving dietary practice can help manage glycemic control and is likely to reduce glycosylated hemoglobin (HbA1c) by 1% to 2%. Nutritional therapy which is a crucial part of selfmanagement of diabetes for diabetes Type 2 patients which should entail a detailed evaluation of their eating patterns, type, and the size of foods and the quantity of beverages consumed, the number of times they eat including meal and snack distribution throughout the day, macronutrient, and micronutrient intake. Not limited to weight history, body mass index (BMI) and target weight, glycaemia targets, knowledge on foods GI and GL and lastly review of the results of self-monitoring of blood glucose. According to Najla Shamsi et al. (2013), Diabetes patients find it challenging to remain consistent in adhering to their diabetic diet program. This is mostly influenced by food access/availability of healthful foods, tradition, cultural food systems, health beliefs, knowledge of foods that promote health and prevent disease, and economics/resources to buy health-promoting foods.

A study by Shamsi N et., al (2013) reported that some of the factors that make the diabetes patients not to stick to their diet regime were, it takes effort to plan what to eat during every meal and to see it through. Edward Byers (2016) most diabetes type 2 patients are lack self-control in matters related to food and diet participants in the study said it was difficult for them to resist some of the foods that they enjoyed eating with restrictions before they were diagnosed with diabetes. Confusion and forgetfulness. Patients stated that sometimes it got confusing on what type of food to eat and the amount they should consume. Some patients said they forgot to check their blood glucose levels due to their daily schedules. Patients see diabetes as an inconvenience that interferes with their day-to-day social life. Y.M Demilew et al., (2019) dietary practice of type 2 diabetes patients is poor due to lack of nutrition education on diabetes diet management to the patients and families.

4.3.3. Machine learning techniques used in diabetes management

Machine learning is a data analytics technique that will teach a computer to do that which comes naturally to humans and animals. Machine learning techniques are classified into four categories, supervised learning, and unsupervised learning, semi supervised and reinforcement learning. Chen et.al (2019) machine learning techniques applied in the health informatics sector for instance healthcare and telemedicine, is playing a big role to the users who are clinicians, patients and their caregivers for better decision making in less time and very affordable

Decision trees

Decision trees is a supervised learning method, which is used for solving classification problems. S. Sneha & Gangil, T (2019).

The goal of the method is to predict the class value of the target variable. The decision tree will help to segregate the data set and build the decision model to predict the unknown class labels. A decision tree can be constructed to both binary and continuous variables. Decision tree optimally finds the root node based upon the highest entropy value. This gives the decision tree the advantage of choosing the most consistent hypothesis among the training datasets. An input to the decision tree is a dataset, consisting of several attributes and instances values and output will be the decision model. A study done by Ramezankhani A (2014) used decision tree analysis, using routine demographic, clinical, anthropometric and laboratory measurements to create a simple tool to predict individuals with low risk for type 2 diabetes

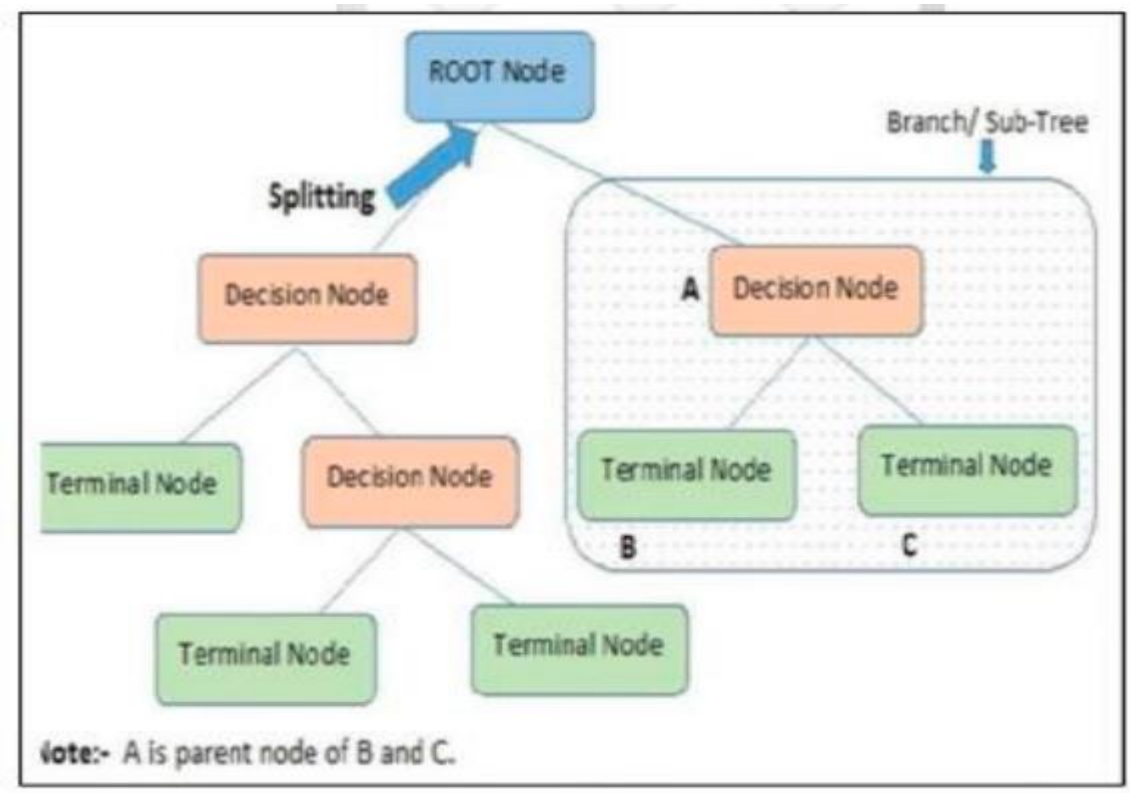


Figure 4.1 Decision trees

4.3.4. K nearest neighbor (KNN)

is a classification technique which classifies the new sample based on similarity measure or distance measure. The measure includes 3 distance measures Euclidean distance, Manhattan, Minkowski. Steps used in KNN are training phase of the algorithm consists of only storing the feature sample and class label of the training sample. Classification phase: the user must define a "k" value for the classification of the undefined sample for the k number of the class labels, so the unlabeled sample can be classified into the defined class based on the feature similarity. Majority of voting classification occurs for unlabeled class. The value of the k can be selected by various techniques like heuristic technique

4.4. Related works

Today with the use of machine learning techniques healthcare systems that were quite complex to evaluate, with emerging techniques it has become easier than before. M Mohri, (2012) supervised learning uses labelled datasets to train algorithms that classify data or used in predicting outcomes accurately. From a given corpus, you have you divide it into the training and the test datasets that contain both the input and the desired output to train the algorithm learn of any new cases that are fed into the predictive model and are required to give the correct output. Unsupervised learning uses unlabeled data leaving it to discover the hidden patterns. The global spread of diabetes led to the urgent need for automated tools and services that will support diabetes patients with carbohydrate (CHO) counting to control their glycaemia. The ease of availability for smartphones with enhanced capabilities alongside the latest advances allowed for the development of image analysis applications used for automatic assessment of food taken by a patient

4.4.1. GOCARB system a smartphone application

Anthimopoulos M et.al (2015) developed GoCARB system a smartphone application designed to support diabetes Type 1 patient used computer vision to estimate the amount of carbohydrates in meals. The method used to develop GOCARB system was computer vison which was used to train the model interpret and understand the visual images. The patients take 2 pictures using their smartphone the plate is detected and the foods segmented, the food volumes are calculated, and the CHO content is estimated. The limitation of computer vison is more information was needed from the patient when it comes to complex foods that are mixed together or covered in sauces.

4.4.2. • Diet-Organizer system

web-based recommender system called DIETOS (DIET-Organizer System) to improve the quality of life of individuals affected by chronic diet-related diseases. DIETOS system could recommend foods with personalized nutritional related suggestions based on the user health profile. DIETOS was an expert system that did not use statistical or data mining algorithms to classify users, because the system implicitly used guidelines furnished by the medical specialist. This system used health profiling which was based on the answers the patients gave to recommend food to the patient. One limitation of the system was the recommendations were not based on the patient's dietary needs at the moment. Another limitation was DIETOS did not advise users on diets compatible with their health status

4.4.3. Diet-right a smart food recommender system

F Rehman, et.al (2017), proposed a model that recommended various foods to people based on their pathological test reports.

The method used to develop the system was Root Mean Square Error (RMSE) to select globally best solution. The normal ranges of the indicators are given in the test reports this way, the patient can identify the abnormalities after comparing with the normal ranges. The system gathers the data of normal ranges for the tests that were conducted. The system is trained on various types of age groups and their respective ranges of parameters allowing the system to suggest diets as per the needs of the patient. The ranges of the same component may differ based on gender, age groups, and fasting or no fasting. One limitation of the system was it did not breakdown for different timings of the day, such as breakfast, lunch, and dinner considering the amount of nutrition in different food items at different timings and daily needs of the patients

4.5. Overall Problems of Existing Systems

1. Diabetes Phenotype in Old Age: In addition to the traditional diabetes-related vascular and neuropathic complications, physical and mental disabilities are only now emerging as important categories of complications in people with diabetes that affect older people disproportionately. Diabetes is directly associated with accelerated loss of muscle strength and muscle quality, increasing the risk of sarcopenia Additionally, diabetes-related complications such as renal impairment and diabetes-associated comorbidities such as hypertension increase the likelihood of frailty The combination of sarcopenia and frailty, often complicated by various types of neuropathies, mediate the pathway to physical disability and lower-limb dysfunction

2. vascular disease: Managing vascular disease risk includes treatment of risk factors such hyperglycemia, hypertension, and dyslipidemia.

3. Hypertension: A target systolic blood pressure (SBP) of 140 mmHg is reasonable in older people with diabetes because it is associated with a reduction of cardiovascular risk compared with SBP >140 mmHg. Several meta-analyses have concluded that lower SBP of <130 mmHg is not associated with better cardiovascular outcomes because

cardiovascular benefits appear to reach a plateau after attaining an SBP of 140 mmHg.

More intensive SBP reduction to <130 mmHg may be beneficial in patients with high risk of stroke, but this is likely to be associated with a significant increase in serious adverse events

4. Reliability and Accuracy of Data: Inaccurate or incomplete data from various sources can impact the reliability of recommendations. Calibration issues with monitoring devices or outdated information in health records can lead to inaccurate suggestions.

Addressing these challenges requires a strategic approach:

• Interoperability Solutions: Developing standardized data formats and APIs to facilitate seamless data exchange among systems.

• Enhanced Security Measures: Implementing robust encryption, access controls, and regular audits to safeguard patient data.

• Personalization Algorithms: Advancing recommendation algorithms to incorporate

machine learning for more personalized and adaptive suggestions.

• User-Centric Design: Focusing on creating intuitive interfaces and patient-centered experiences to improve engagement and adherence

4.6. Overall Solution Approach

There are several existing systems and approaches used to classify diabetic patients and recommend lifestyle changes. Some common methods include:

Clinical Guidelines: Healthcare providers often follow established clinical guidelines, such as those provided by organizations like the American Diabetes Association or the World Health Organization, to classify diabetic patients based on factors like blood sugar levels, A1C levels, and other health indicators.

Risk Assessment Tools: Various risk assessment tools are used to evaluate a patient's risk of developing diabetes or complications. These tools often consider factors like age, weight, family history, blood pressure, and lifestyle habits.

Machine Learning Models: Advanced technologies like machine learning are increasingly used to develop predictive models that classify diabetic patients based on a multitude of parameters, including medical history, genetic factors, and lifestyle habits. These models can offer personalized recommendations for lifestyle changes.

Mobile Apps and Wearable Devices: There are numerous mobile applications and wearable devices available that track various health metrics like physical activity, diet, sleep patterns, and glucose levels. These tools can help users monitor their lifestyle and receive personalized recommendations based on their data.

Telemedicine Platforms: Telemedicine services often integrate tools for monitoring and classifying diabetic patients remotely. They enable healthcare professionals to assess patients' conditions and recommend lifestyle changes through virtual consultations.

The choice of system or method often depends on factors such as the healthcare provider's preferences, the patient's needs, available technology, and the level of personalization desired for lifestyle recommendations

4.7. Summary

Existing systems for predicting diabetes employ diverse technologies, including machine learning models like logistic regression and neural networks, clinical decision support systems, risk assessment tools, and genetic risk prediction. Mobile apps and wearables, along with data mining techniques, contribute to real-time data collection. Remote patient monitoring and analysis of genetic makeup are integral. The effectiveness depends on specific characteristics and ethical considerations. These systems aim to enhance diabetes risk prediction, enabling proactive healthcare measures.

5. Chapter Three: System Requirements Engineering and Planning

5.1. Introduction

System Requirements Engineering and Planning refer to the processes and activities involved in determining, documenting, and managing the requirements for a system. These processes are critical in the early stages of system development, helping to ensure that the resulting system meets the needs and expectations of its stakeholders. Here's an overview of these concepts:

5.1.1. System Requirements Engineering:

Definition:

System Requirements Engineering involves gathering, analyzing, documenting, and managing the requirements for a system. Requirements encompass functional, nonfunctional, and sometimes, design constraints.

2- Key Activities:

5.1.2. Requirements Elicitation:

Engaging with stakeholders to identify and understand their needs and expectations regarding the system.

5.1.3. Requirements Analysis:

Examining and organizing gathered information to define clear, concise, and unambiguous requirements.

5.2. Requirements Specification:

Documenting requirements in a systematic and structured manner. This may include use cases, user stories, functional and non-functional requirements.

3- Key Concepts:

3.1- Functional Requirements:

Describing what the system must do, usually in terms of features or capabilities.

3.2- Non-functional Requirements:

Describing qualities or constraints on the system, such as performance, security, usability, and reliability.

3.3- User Stories:

Describing system behavior from an end-user perspective.

3.4 – Use Cases:

Describing interactions between the system and its users or other systems.

4- Tools and Techniques:

4.1- Interviews: Direct discussions with stakeholders.

4.2- Surveys/Questionnaires:

Gathering information from a broader audience.

4.3- Prototyping:

Creating a simplified version of the system to gather feedback.

4.4- Observations:

Watching users interact with existing systems or prototypes.

5- Challenges:

5.1- Changing Requirements:

Dealing with evolving needs and expectations.

5.2- Communication Issues:

Ensuring effective communication between stakeholders.

5.3- Scope Creep:

Managing changes in project scope without appropriate analysis.

6- System Requirements Planning:

6.1- Definition:

System Requirements Planning involves developing a strategy for gathering, analyzing, documenting, and managing requirements throughout the entire system development lifecycle.

6.2- Key Activities:

6.2.1- Stakeholder Identification:

Identifying and categorizing individuals or groups with a vested interest in the system.

6.2.2- Communication Planning:

Developing a strategy for effective communication among stakeholders.

6.2.3- Risk Assessment:

Identifying potential risks related to requirements and developing mitigation strategies.

6.2.4- Resource Planning:

Allocating resources, including personnel, tools, and time, for requirements engineering activities.

6.3- Key Concepts:

6.3.1- Traceability Matrix:

Documenting the relationships between requirements, ensuring that each requirement is linked back to its source and forward to the relevant design and test artifacts.

6.3.2- Change Management:

Establishing a process for handling changes to requirements.

6.3.3- Requirements Repository:

A centralized location for storing and managing requirements documents and related artifacts.

6.4- Tools and Techniques:

6.4.1- Project Management Software:

Tools for planning, scheduling, and tracking requirements-related activities.

6.4.2 – Collaboration Platforms:

Facilitating communication and collaboration among team members and stakeholders.

6.4.3 -Version Control Systems:

Managing changes to requirements documents.

6.5. Challenges:

6.5.1- Unclear Objectives:

Ensuring a clear understanding of project objectives and aligning requirements activities with those objectives.

6.5.2 – Resource Constraints:

Managing limitations in time, personnel, and tools.

6.5.3 – Managing Stakeholder Expectations:

Ensuring that stakeholder expectations are realistic and achievable.

In summary, System Requirements Engineering and Planning are crucial aspects of system development, ensuring that the resulting system meets the needs of stakeholders, is well-documented, and can be effectively managed throughout its lifecycle.

5.3. Feasibility

What Is a Feasibility Study?

A project feasibility study is an analysis used to determine the practicality and likeliness of success for a project or new business idea. A feasibility study can look at several variables, including cost, profitability forecasts, market analysis findings, technical requirements, government regulations, and scheduling or timeline constraints. Businesses use feasibility studies to decide if they should develop a new product or expand into an untapped market; they begin with a preliminary analysis before conducting a full feasibility report.

The Benefits of a Feasibility Study.

A feasibility study has several benefits because it requires the project managers and participants to consider a project's potential obstacles before diving in, allowing them to make informed decisions about the best approach. In addition, a feasibility study helps the team recognize early on if a project requires too much time or too many resources for its projected benefits.

Key Elements of a Feasibility Study:

4. An executive summary: This report begins by providing a comprehensive overview of the diabetes condition, offering a quick insight into the scope of the problem and its impact on public health. The report addresses current studies on the prevalence of diabetes and its effects on individuals and society. It highlights the latest technologies and advancements in treatment and management, with a focus on innovations in medications and preventive measures

The summary includes an analysis of the challenges facing diabetes patients, such as difficulties in accessing healthcare and the cost of treatment. The report also sheds light on the urgent need to raise awareness about the causes of diabetes and promote a healthy lifestyle

In conclusion, the team presents effective recommendations to enhance the management and prevention of diabetes, with specific guidance for making a "go/no-go decision."

2. Details of the proposed project: The proposed project aims to establish a comprehensive diabetes management initiative to address the increasing prevalence of diabetes in our community. Focusing on prevention, education, and improved patient care, the project seeks to enhance awareness, provide accessible healthcare services, and implement innovative strategies for diabetes management.

3. Economic considerations: Economic considerations for diabetes encompass a wide range of factors that impact both individuals and society at large. Here are key economic aspects related to diabetes.

4. Legal feasibility: Legal feasibility for diabetes involves assessing the legal aspects related to the management, treatment, and support of individuals with diabetes. Here are key considerations:

5. Organizational considerations: Organizational considerations for diabetes involve assessing and addressing various factors within healthcare organizations and other relevant entities to ensure effective prevention, management, and support for individuals with diabetes. Here are key organizational considerations:

6. Schedule considerations: Schedule considerations for diabetes encompass various

aspects related to the timing, frequency, and coordination of activities aimed at prevention, management, and support for individuals with diabetes. Here are key schedule considerations:

7. Technology considerations: Technology considerations for diabetes involve leveraging innovative solutions to enhance prevention, management, monitoring, and support for individuals

with diabetes. Here are key technology considerations in the context of diabetes:

5.4. Requirements Elicitation Techniques

What Does System Requirements Mean

System requirements are the configuration that a system must have in order for a hardware or software application to run smoothly and efficiently. Failure to meet these requirements can result in installation problems or performance problems. The former may prevent a device or application from getting installed, whereas the latter may cause a product to malfunction or perform below expectation or even to hang or crash.

5.5. Targeted Users

The targeted users for a predictive diabetic system can encompass a diverse range of individuals and professionals involved in the management, prevention, and care of diabetes.

Here are the key user groups for such a system:

The WHO targets are:

Target 1: people with diabetes are diagnosed.

Target 2: people diagnosed have good control of glycaemia.

Target 3: people diagnosed have good control of blood pressure.

Target 4: people with diabetes over 40 years receive statins.

Target 5: people with T1D have access to affordable insulin treatment and blood glucose selfmonitoring.

5.6. Functional Requirements Definition

A recommendation system for diabetic patients aiming to recommend lifestyle changes typically involves the use of algorithms and data analysis to provide personalized advice on diet, exercise, and other aspects of daily living. The goal is to help individuals manage their diabetes more effectively and improve their overall well-being. Here's a basic outline of the key components and functions that such a recommendation system might include.

5.7. Functional Requirements Specification

A Functional Requirements Specification (FRS) for a predictive diabetic system further details the specific functionalities and behaviors of the system.

Below is an expanded set of functional requirements for such a system:

1- User Authentication and Authorization:

The system should have a secure login process for healthcare providers and patients with role-based access control.

The system must be able to predict diabetics.

The results must be classified if “predict “or “no”.

2- Data Analysis and Prediction:

This information is:

Patient: -

• Login

Patients can log in to the system if they sign in the system, if they don’t sign up must do it

• Signup

The patient can register their data to the system

• Logout The patient can search for the nearest doctor to him and book an appointment with him.

The patient can make an appointment with the available doctors.

• Book appointments

Doctor: -

• Login

A doctor can log in to the website.

• Signup

A doctor can register their data on the website.

• Logout

A doctor can log out from his account to log in with another account.

• Profile of the doctor

The website creates profiles for doctors available in the application.

• Organizing appointments with the doctor

The website knows the available appointments with the doctor to show them to

the patients.

• Updating patient profile

The system should employ machine learning algorithms to analyze the collected data and

predict the likelihood of diabetes based on medical history.

3- Prediction Display:

The predictions should be presented in a clear and understandable format for both healthcare providers and patients.

4- Alerts and Notifications:

The system should notify healthcare providers or patients about the risk of diabetes based on the predictions.

5- User Profile Creation:

Collect and store relevant information about the diabetic patient, such as age, gender, weight, height, medical history, blood sugar levels, and lifestyle preferences.

6- Data Collection:

Gather data from various sources, including wearable devices, health apps, electronic health records, and self-reported information from the user.

7- Data Preprocessing:

Clean and preprocess the collected data to handle missing values, outliers, and inconsistencies.

8- Feature Extraction:

Identify key features related to lifestyle, such as dietary habits, physical activity, sleep patterns, stress levels, and medication adherence.

9- Personalized Recommendation Algorithm:

Develop an algorithm that takes into account the user's profile and extracted features to generate personalized recommendations. This may involve machine learning techniques, collaborative filtering, content-based filtering, or a hybrid approach.

10- Dietary Recommendations:

Provide personalized dietary advice based on the user's preferences, nutritional needs, and restrictions. This could include meal plans, recipe suggestions, and nutritional guidelines.

11- Physical Activity Recommendations:

Recommend appropriate exercise routines and physical activities based on the user's fitness level, preferences, and health conditions.

12- Behavioral Recommendations:

Suggest behavioral changes to improve overall well-being, including stress management techniques, sleep hygiene, and mindfulness practices.

13- Real-time Monitoring:

Implement real-time monitoring capabilities to track the user's progress and adjust recommendations accordingly. This may involve continuous feedback from wearable devices or regular check-ins with the user.

14- Education and Engagement:

Provide educational content to help users understand the importance of lifestyle choices in managing diabetes. Encourage user engagement through alerts, reminders, and motivational messages.

15- Privacy and Security:

Implement robust security measures to ensure the confidentiality and privacy of the user's health data. Comply with relevant healthcare regulations and standards.

16- Medication Adherence Monitoring:

Integrate features to monitor and encourage adherence to prescribed medications. Remind users to take medications on time and provide educational content about the importance of medication compliance.

17- Blood Sugar Trend Analysis:

Analyze trends in blood sugar levels over time and provide insights into the impact of lifestyle choices on these trends. Offer recommendations to help users maintain optimal blood glucose levels.

18- Meal Logging and Analysis:

Allow users to log their daily meals and snacks. Analyze nutritional content and provide feedback on meal choices, suggesting alternatives that better align with dietary goals.

19- Physical Activity Tracking:

Integrate with fitness trackers or smartphone apps to monitor physical activity. Provide feedback on exercise routines and suggest adjustments to meet personalized fitness goals.

20- Stress Management Techniques:

Recommend stress management techniques, such as deep breathing exercises, meditation, or mindfulness activities. Integrate relaxation techniques into the user's daily routine.

21- Sleep Quality Assessment:

Monitor sleep patterns and assess sleep quality. Provide recommendations for improving sleep hygiene, such as establishing a consistent sleep schedule and creating a conducive sleep environment.

22- Social Support Integration:

Facilitate connections with support networks, such as friends, family, or online communities. Encourage social engagement as it can positively influence lifestyle changes.

23- Goal Setting and Tracking:

Enable users to set realistic and achievable health goals. Track progress and celebrate milestones to motivate continued adherence to lifestyle recommendations.

24- Emergency Response Integration:

Implement features for emergency situations, such as providing emergency contact information, integrating with emergency services, and offering guidance on managing diabetes-related emergencies.

25- Personalized Feedback and Education:

Continuously provide personalized feedback on the user's progress and offer educational content related to diabetes management. Tailor information based on the user's specific needs and preferences.

26- User Preferences Customization:

Allow users to customize their preferences and priorities within the recommendation system. This could include preferred communication channels, types of reminders, and the level of detail in feedback.

27- Behavior Change Support:

Incorporate behavior change techniques, such as motivational interviewing, to support users in adopting and maintaining healthier lifestyle choices. Provide positive reinforcement and encouragement.

28- Integration with Electronic Health Records (EHR):

Ensure seamless integration with the user's electronic health records to provide a comprehensive view of their health history and facilitate communication with healthcare professionals.

29- Continuous Learning and Improvement:

Implement mechanisms for continuous learning and improvement of the recommendation algorithms. Incorporate user feedback, adapt to changing health conditions, and stay current with advancements in diabetes management research.

30- Gamification Elements:

Introduce gamification elements to make the experience engaging and enjoyable. Reward users for achieving goals, completing challenges, and consistently following the recommended lifestyle changes.

It's important to note that the effectiveness of a recommendation system depends on the quality and accuracy of the data, the sophistication of the algorithms, and the user's willingness to follow the provided recommendations. Additionally, collaboration with healthcare professionals may enhance the system's reliability and impact on the user's health. By incorporating these functions, a recommendation system can offer a comprehensive and personalized approach to supporting diabetic patients in making and sustaining positive lifestyle changes

5.8. Non-Functional Requirements

Non-functional requirements for a predictive diabetic system define the characteristics and qualities that describe how the system should perform, rather than specifying specific behaviors. These requirements are essential for ensuring the system's overall performance, usability, security, and compliance.

Here are some non-functional requirements for a predictive diabetic system:

Privacy and data security: the system should ensure the privacy and security of any personal data it processes, such as images of individuals' faces.

- Scalability: the system should be able to handle large amounts of data and be easily expandable to handle more data or users in the future.

- Interoperability: the system should be able to integrate with other systems and technologies, such as cameras or alarm systems.

- Low power consumption: the system should be designed to use minimal power when operating.

- Usability: the system should be user-friendly and easy to use for the end-users.

- Cost-effective: the system should be cost-effective to develop, deploy, and maintain.

Real-time performance

5.9. Summary

Chapter Summary: "Predict Diabetic System Requirements Engineering and Planning"

This chapter delves into the foundational aspects of designing and implementing a predictive diabetic system. Beginning with an insightful introduction (3.1), it sets the stage for a comprehensive exploration of essential phases in system development.

The feasibility study (3.2) is a pivotal step, evaluating the viability of the predictive diabetic solution. This includes technical, financial, legal, and operational considerations to ensure a robust foundation for subsequent phases.

Requirements elicitation techniques (3.3) are then discussed, offering insights into methodologies for gathering critical information from stakeholders. From stakeholder interviews to observational approaches, these techniques ensure a holistic understanding of user needs.

Identifying the targeted users (3.4) is crucial for tailoring the system to diverse stakeholders. Patients, healthcare professionals, researchers, and more are considered, emphasizing a user centric approach.

The subsequent sections focus on the functional and non-functional aspects of the system. Functional requirements definition (3.5) delineates specific features, such as patient profiling, predictive modeling, and personalized recommendations, crucial for accurate diabetes predictions.

6. Chapter six: System Design

6.1. Introduction

System design is a critical phase in the software development life cycle that focuses on translating requirements gathered during the analysis phase into a detailed blueprint for the construction of a system. It involves creating a comprehensive plan that outlines the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. The goal of system design is to produce a solution that not only meets the functional and performance requirements but is also scalable, maintainable, and adaptable to future changes.

6.2. Context Diagram

A context diagram, also known as a system context diagram or level 0 DFD (Data Flow Diagram), is a visual representation that provides a high-level view of a system or process and its interactions with external entities. It is commonly used in system analysis and design to depict the boundaries of a system and the flow of data between the system and its external entities.

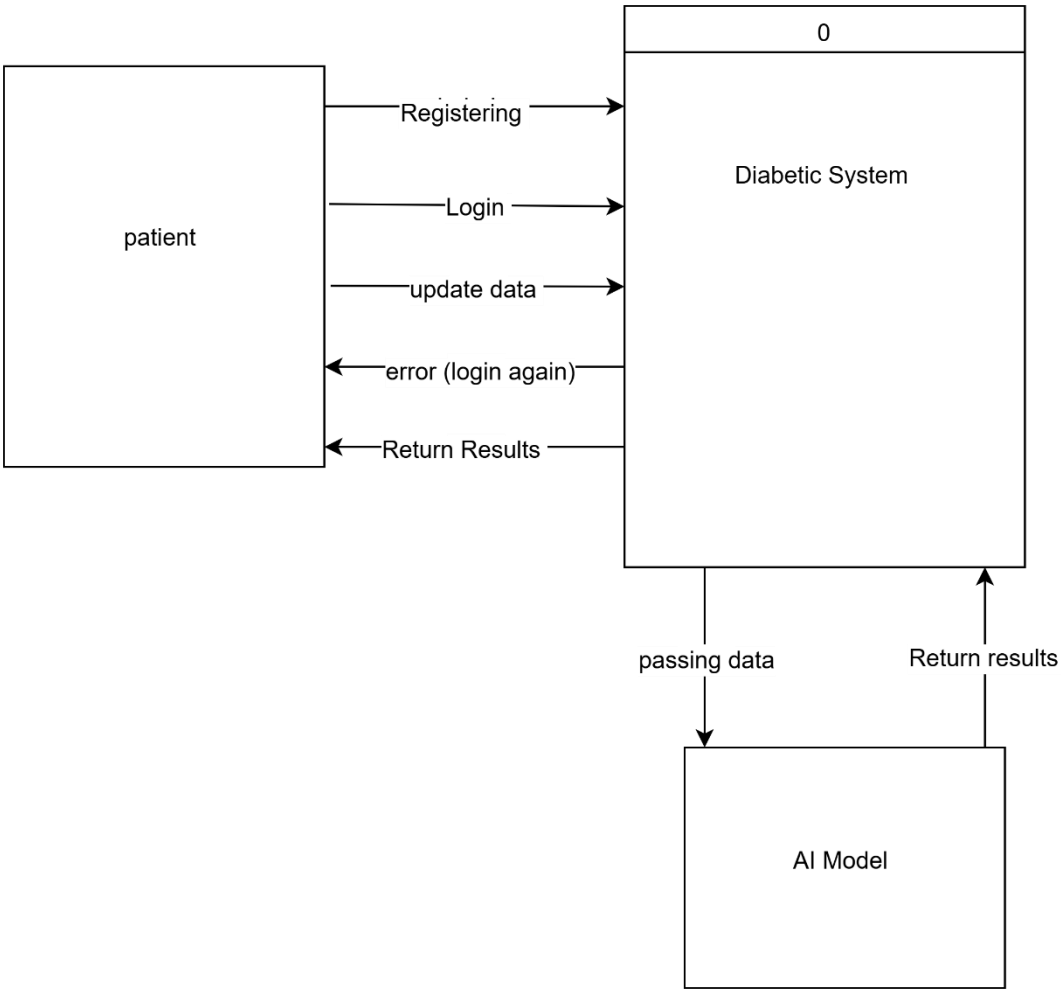


Figure 5.1 context diagram

6.3. Data Flow Diagram (DFD)

A Data Flow Diagram (DFD) is a graphical representation that illustrates how data moves within a system, showing the flow of information between processes, data stores, external entities, and data destinations. DFDs are commonly used in system analysis and design to model the

processes and data involved in a system and to depict the interactions between different components.

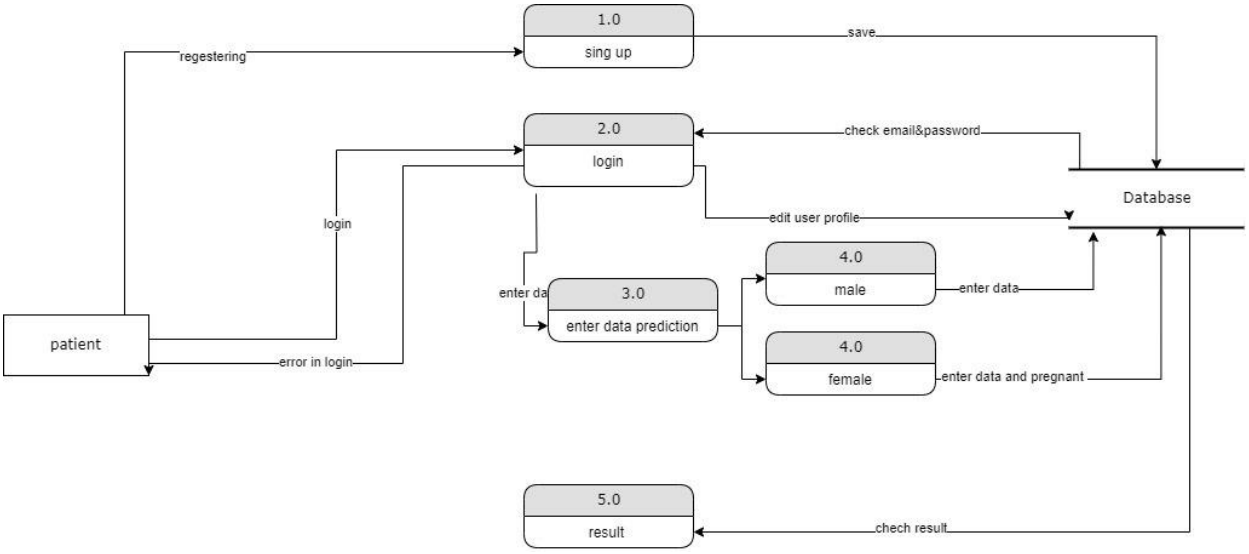


Figure 6.1 Data Flow Diagram (DFD)

6.4. Entity Relationship Diagram (ERD)

An Entity-Relationship Diagram (ERD) is a visual representation of the data model that describes the relationships between different entities within a system. ERDs are commonly used in database design and software engineering to model the structure of a database and illustrate how different entities interact with each other.

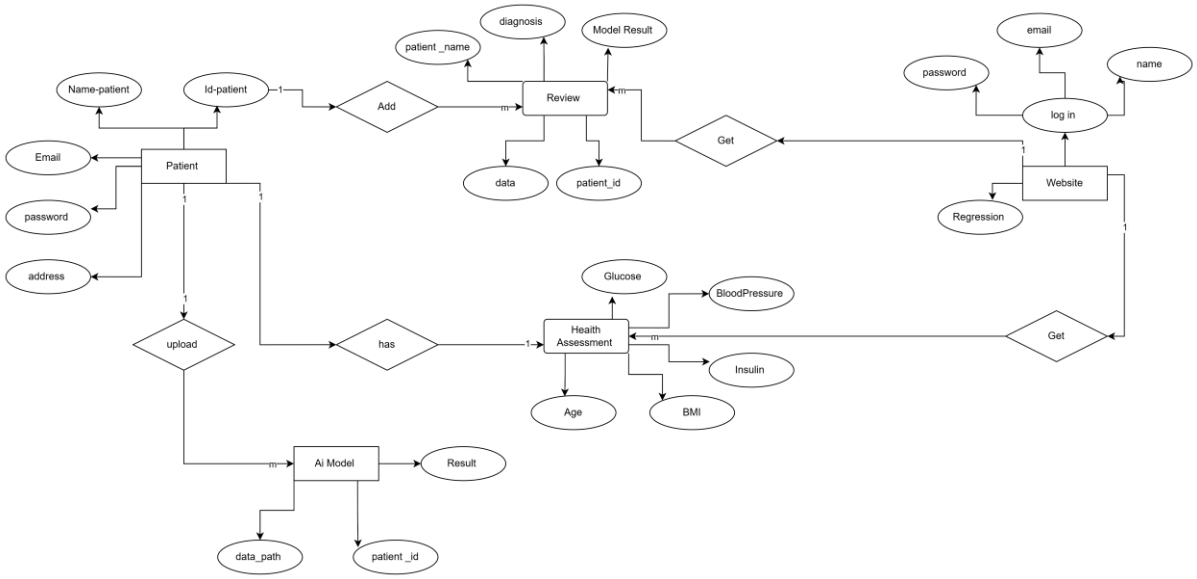


Figure 7.1Entity-Relationship Diagram (ERD)

6.5. UML Use Case Diagram

A Unified Modeling Language (UML) Class Diagram is a type of diagram within the UML that provides a visual representation of the structure and relationships of classes within a system. It is widely used **in** software engineering and design to model the static aspects of a system, particularly the classes, attributes, and methods that make up the system's object-oriented structure.

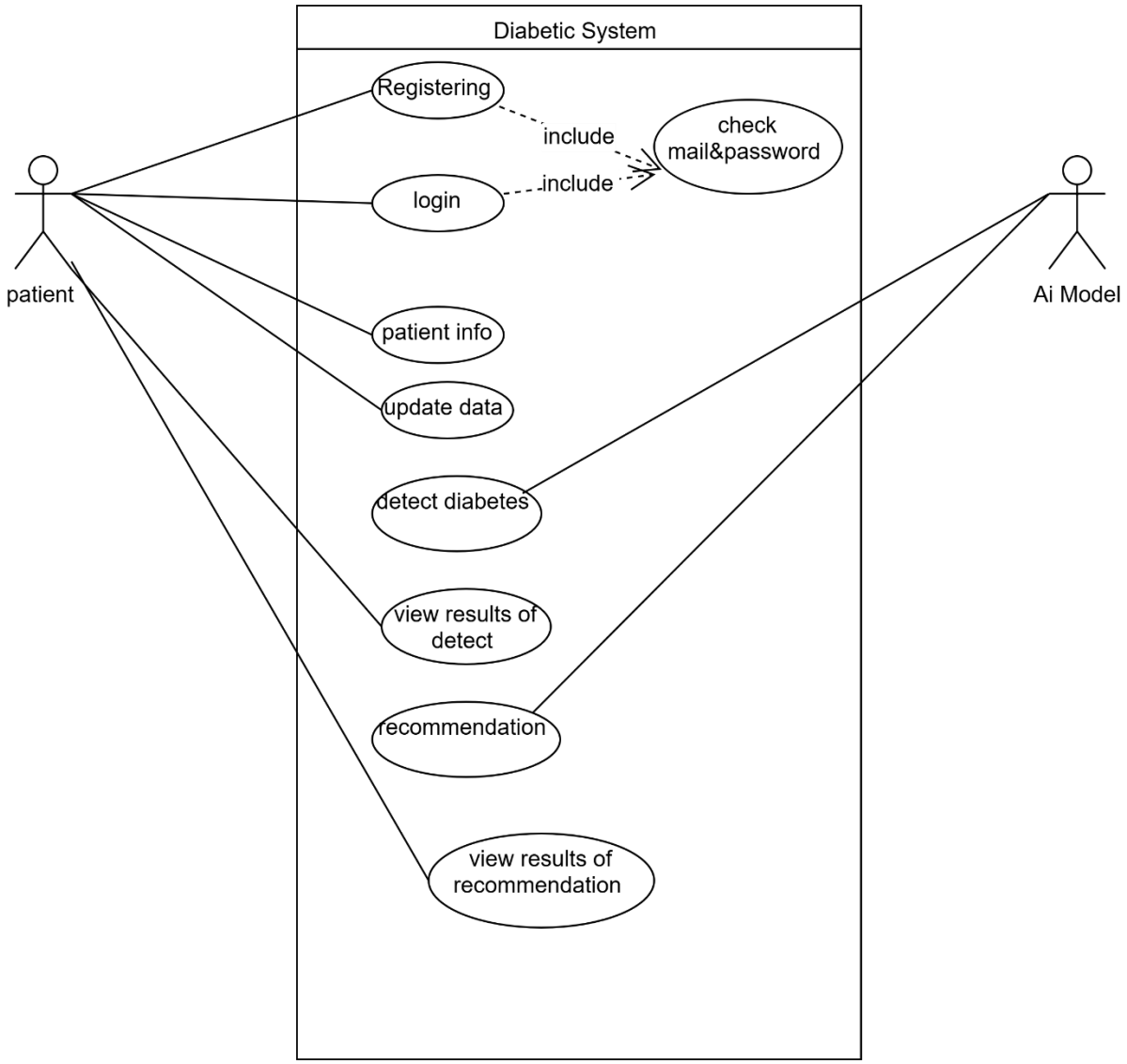


Figure 8.1(UML) Class Diagram

6.6. UML Activity Diagram

A UML (Unified Modeling Language) Activity Diagram is a graphical representation that illustrates the flow of activities within a system or business process. It is commonly used in software engineering to model workflows and the sequential and parallel activities of a system.

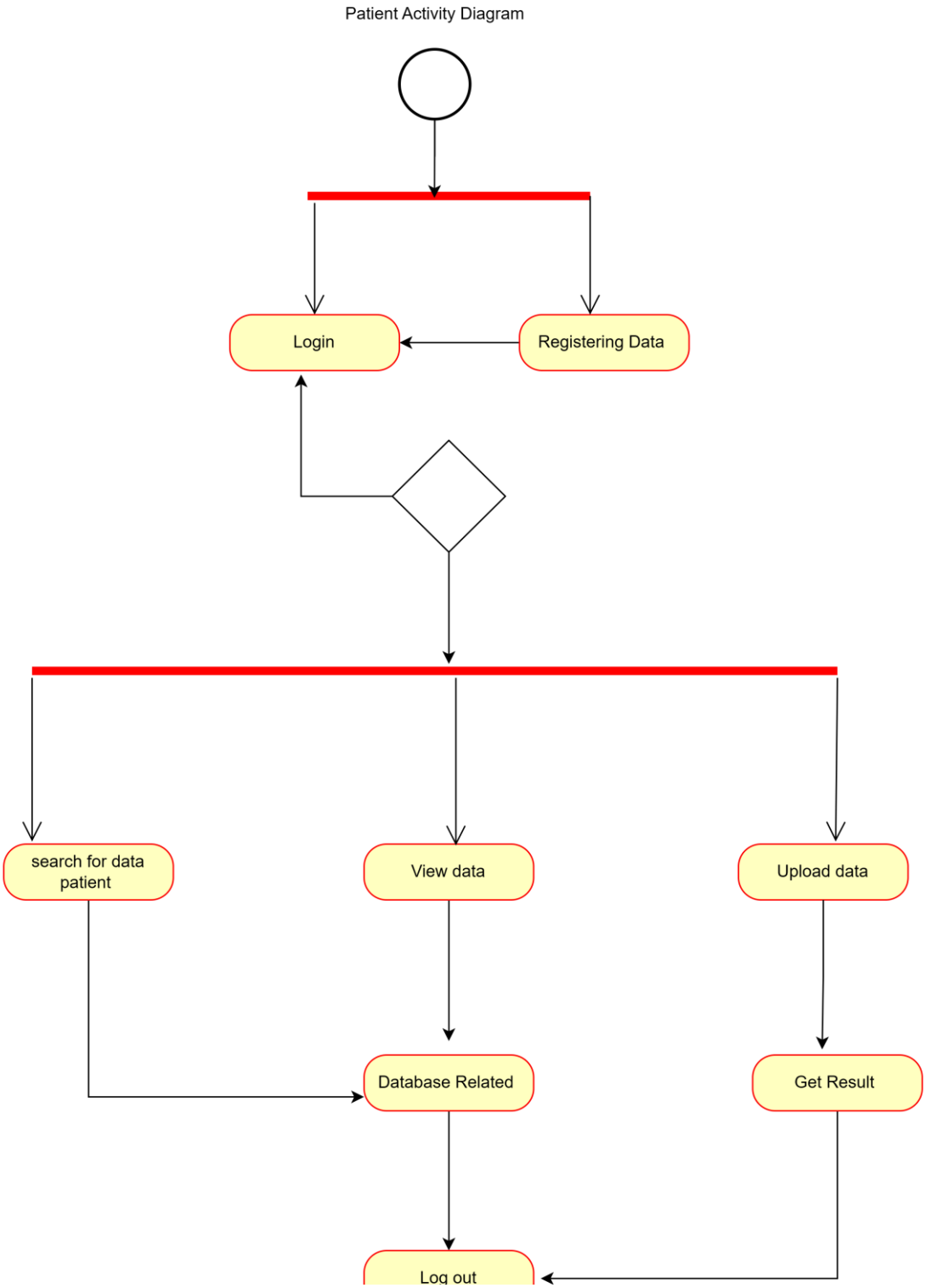


Figure 9.1Activity Diagram

6.7. UML Sequence Diagram

A Sequence Diagram is a type of Unified Modeling Language (UML) diagram that illustrates the interactions between objects or components within a system over time. It shows the flow of messages and the sequence of events that occur as various entities collaborate to achieve a specific functionality. Sequence diagrams are particularly useful in visualizing the dynamic aspects of a system, focusing on the order of interactions between objects.

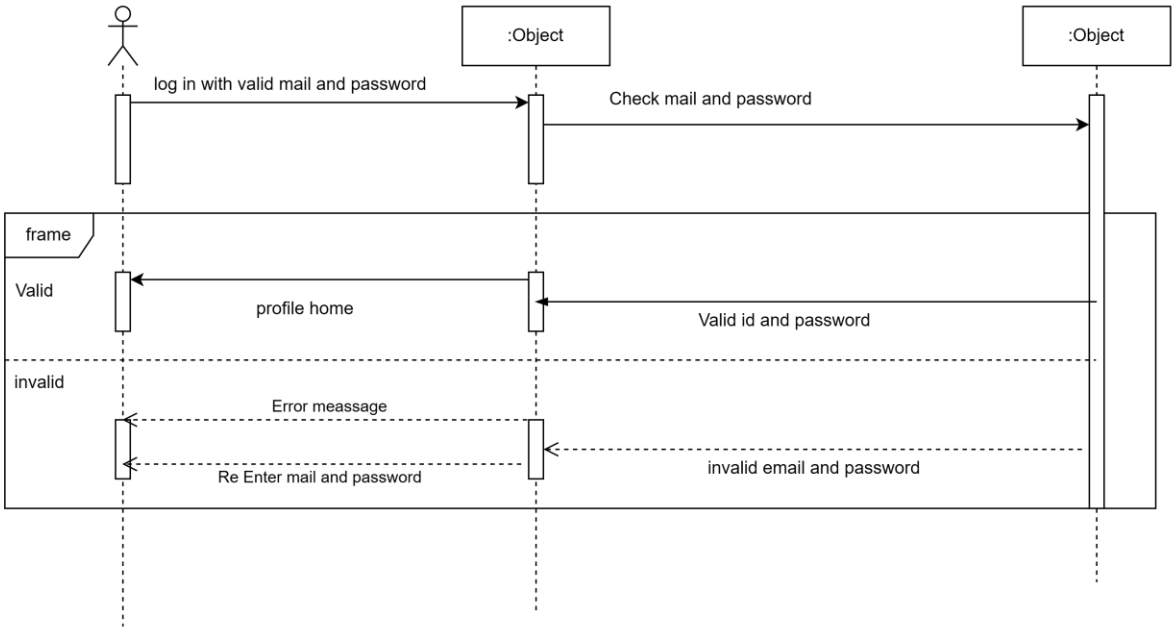


Figure 10.1Sequence Diagram

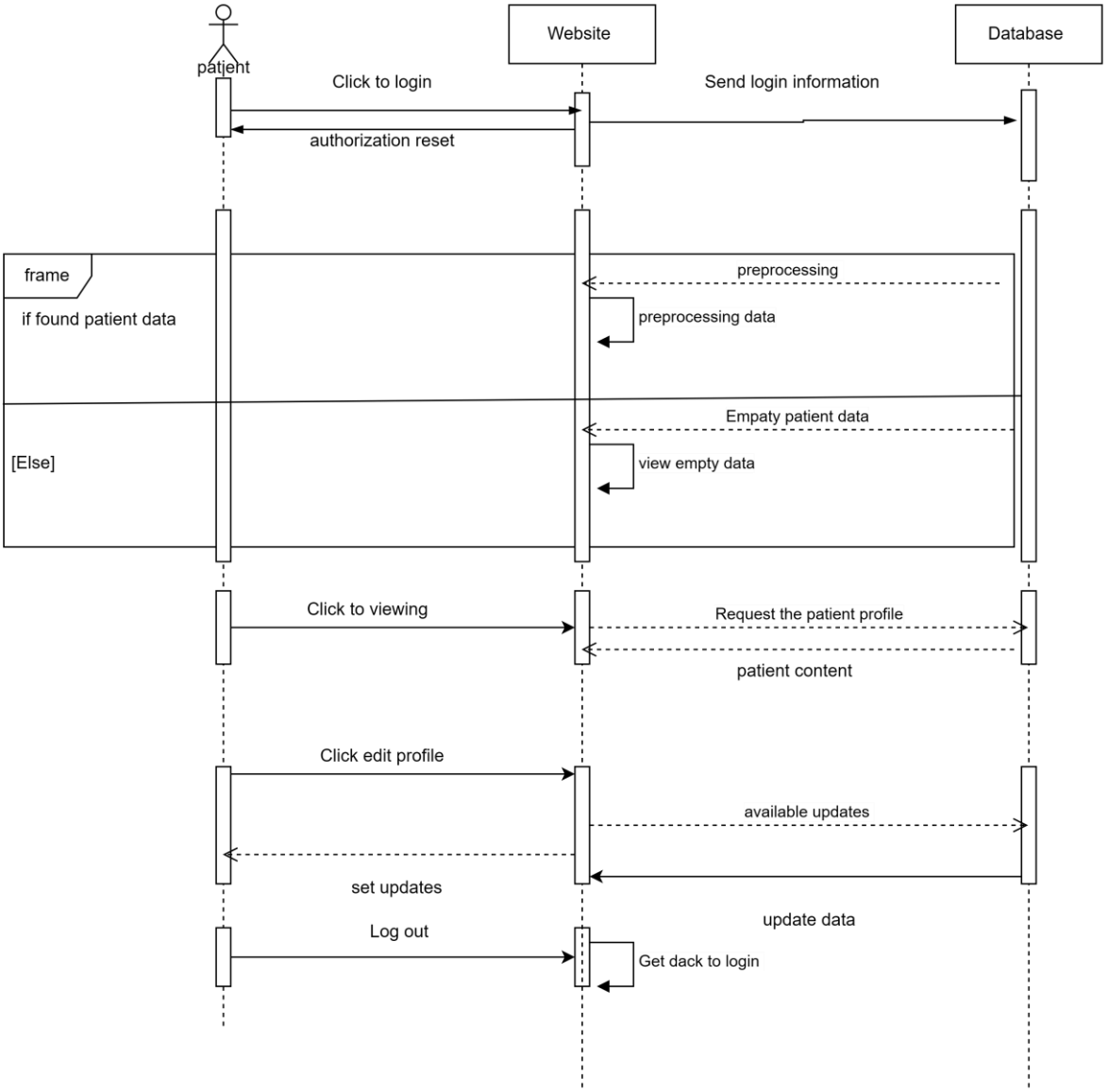


Figure 11.1Sequence Diagram

6.8. UML Class Diagram

A Unified Modeling Language (UML) Class Diagram is a type of diagram within the UML that provides a visual representation of the structure and relationships of classes within a system. It is widely used in software engineering and design to model the static aspects of a system, particularly the classes, attributes, and methods that make up the system's object-oriented structure.

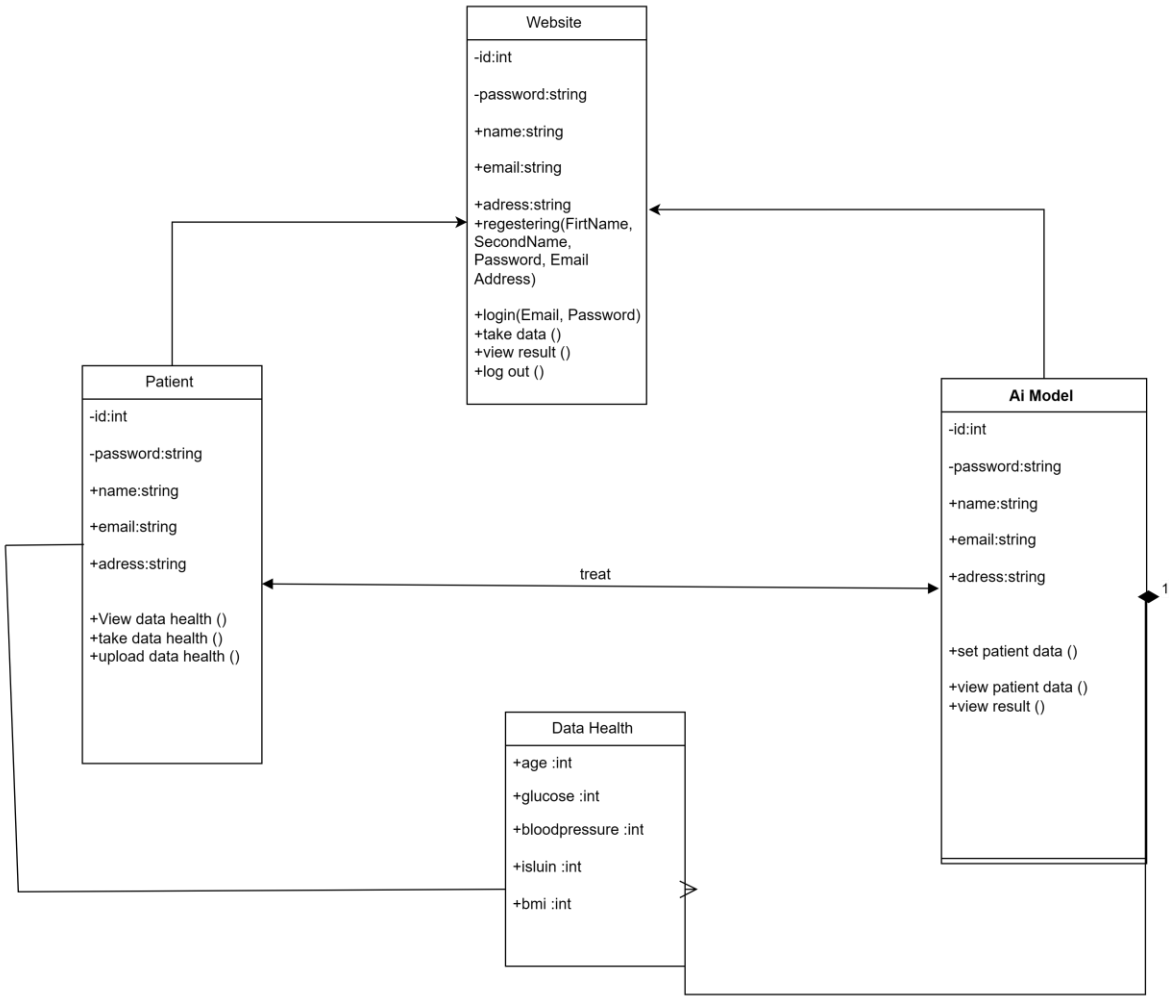


Figure 12.1Class Diagram

6.9. Summary

summary, the system design involves both structured and object-oriented approaches. The structured design components, including the Context Diagram, DFD, and ERD, focus on data flow and relationships. On the other hand, the object-oriented design components, such as UML Use Case Diagram, UML Activity Diagram, UML Sequence Diagram, and UML Class **Diagram**.

7. Chapter Five: System Implementation

7.1. Tools

7.1.1. Python

7.1.2. Why do we use Python?

- Reasons for using the Python language in Machine Learning.
- It has a huge number of libraries and frameworks: The Python language comes with many libraries and frameworks that make coding easy. This also saves a significant amount of time.
- The most popular libraries are NumPy, which is used for scientific calculations; SciPy for more advanced computations; and scikit, for learning data mining and data analysis.
- These libraries work alongside powerful frameworks like TensorFlow, CNTK, and Apache Spark. These libraries and frameworks are essential when it comes to machine and deep learning projects.
- Simplicity: Python code is concise and readable even to new developers, which is beneficial to machine and deep learning projects. Due to its simple syntax, the development of applications with Python is fast when compared to many programming languages. Furthermore, it allows the developer to test algorithms without implementing them.
- Readable code is also vital for collaborative coding. Many individuals can work together on a complex project.
- One can easily find a Python developer for the team, as Python is a familiar platform. Therefore, a new developer can quickly get acquainted with Python's concepts and work on the project instantly.
- The massive online support: Python is an open-source programming language and enjoys excellent support from many resources and quality documentation worldwide. It also has a large and active community of developers who provide their assistance at any stage of development.
- Most scientists have adopted Python for Machine Learning and Deep Learning projects, which means most of the brightest minds worldwide, can be found in Python communities.
- Fast development: Python has a syntax that is easy to understand and friendly. Furthermore, the numerous frameworks and libraries boost software development. By using out-of-box solutions, a lot can be done with a few lines of code. Python is good for developing prototypes, which boosts productivity.
- Flexible integrations: Python projects can be integrated with other systems coded in different programming languages. This means that it is much easier to blend it with other AI projects written in other languages.
- Also, since it is extensible and portable, Python can be used to perform cross languages tasks. The adaptability of Python makes it easy for data scientists and developers to train machine learning models.
- Fast code tests: Python provides a lot of code review and test tools. Developers can quickly check the correctness and quality of the code.
- AI projects tend to be time-consuming, so a well-structured environment for testing and checking for bugs is needed. Python is the ideal language since it supports these features.

- Visualization tools: Python comes with a wide variety of libraries. Some of these frameworks offer good visualization tools. In AI, Machine learning, and Deep learning, it is important to present data in a human-readable format. Therefore, Python is a perfect choice for implementing this feature.

7.1.2.1 NumPy

NumPy stands for Numerical Python**,** is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more. In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

Its benefits:

- Higher speed: NumPy uses algorithms written in C that complete in nanoseconds rather than seconds.
- Fewer loops: NumPy helps you to reduce loops and keep from getting tangled up in iteration indices.
- Clearer code: Without loops, your code will look more like the equations you're trying to calculate.
- Better quality: There are thousands of contributors working to keep NumPy fast, friendly, and bug free.

7.1.2.2. Pandas

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open-source data analysis/manipulation tool available in any language. It is already well on its way toward this goal.

The two primary data structures of pandas, Series (1-dimensional) and Data Frame (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering

Here are just a few of the things that pandas does well:

- Easy handling of missing data (represented as NaN, NA, or NaT) in floating point as well as non-floating-point data.
- Size mutability: columns can be inserted and deleted from Data Frame and higher dimensional objects.
- Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let Series, Data Frame, etc. automatically align the data for you in computations.
- Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data.
- Make it easy to convert ragged, differently-indexed data in other Python and NumPy data structures into Data Frame objects
- Intelligent label-based slicing, fancy indexing, and sub setting of large data sets.
- Intuitive merging and joining data sets.
- Flexible reshaping and pivoting of data sets.
- Hierarchical labeling of axes (possible to have multiple labels per tick).
- Robust IO tools for loading data from flat files (CSV and delimited), Excel files, databases, and saving/loading data from the ultrafast HDF5 format.
- Time series-specific functionality: date range generation and frequency conversion, moving window statistics, date shifting and lagging.

7.1.2.3. Matplotlip

Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable open-source alternative to MATLAB. Developers can also use matplotlib's APIs (Application Programming Interfaces) to embed plots in GUI applications.

A Python matplotlib script is structured so that a few lines of code are all that is required in most instances to generate a visual data plot.

matplotlib scripting layer overlays two APIs:

- pyplot API is a hierarchy of Python code objects topped by *matplotlib. pyplot*
- An OO (Object-Oriented) API collection of objects that can be assembled with greater flexibility than pyplot. This API provides direct access to Matplotlib's backend layers.

Fig 4.1 Matplotlib graphs

7.1.2.4. Seaborn

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures.

Seaborn helps you explore and understand your data. Its plotting functions operate on data frames and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

Benefits of Data Visualization:

- Graphs make it easier to explain your data to non-technical people.
- Visually attractive graphs can make presentations and reports much more appealing to the reader.

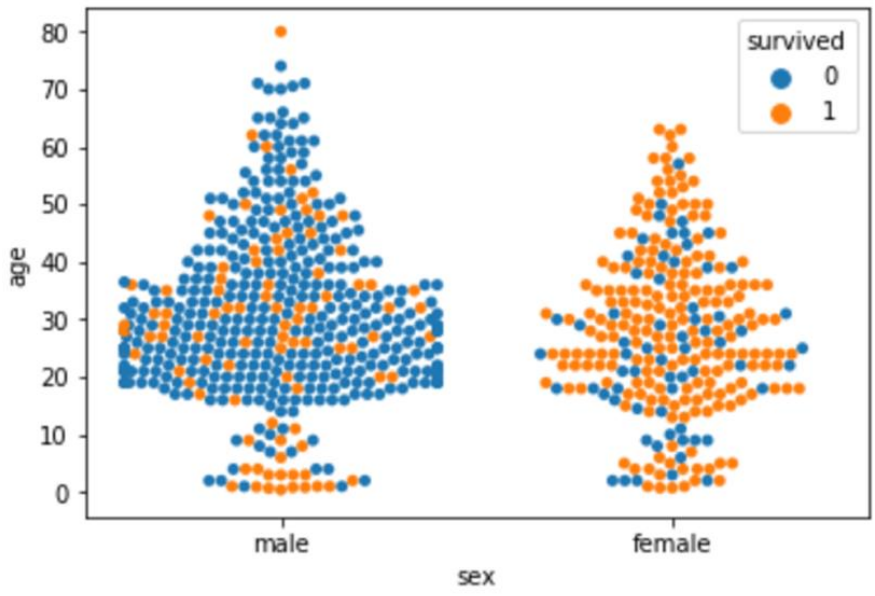


Fig 4.2 data visualization

7.1.2.5. Scikit-learn

Scikit-learn is an open-source data analysis library, and the gold standard for Machine Learning (ML) in the Python ecosystem. Key concepts and features include**:**

- Algorithmic decision-making methods, including:

1. Classification
2. Regression
3. Clustering
   - Algorithms that support predictive analysis ranging from simple linear regression to neural network pattern recognition

train-test split

procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model.

It is a fast and easy procedure to perform, the results of which allow you to compare the performance of machine learning algorithms for your predictive modeling problem. Although simple to use and interpret, there are times when the procedure should not be used, such as when you have a small dataset and situations where additional configuration is required, such as when it is used for classification and the dataset is not balanced.

7.1.3. Streamlet Framework

7.1.3.1. What is Streamlet?

Streamlet is a free and open-source framework to rapidly build and share beautiful machine learning and data science web apps. It is a Python-based library specifically designed for machine learning engineers. Data scientists or machine learning engineers are not web developers and they're not interested in spending months learning to use these frameworks to build web apps. Instead, they want a tool that is easier to learn and to use, as long as it can display data and collect needed parameters for modeling. Streamlet allows you to create a stunning-looking application with only a few lines of code.

7.1.3.2. Why should data scientists use Streamlet?

The best thing about Streamlet is that you don't even need to know the basics of web development to get started or to create your first web application. So, if you're somebody who's into data science and you want to deploy your models easily, quickly, and with only a few lines of code, Streamlet is a good fit.

One of the important aspects of making an application successful is to deliver it with an effective and intuitive user interface. Many of the modern data-heavy apps face the challenge of building an effective user interface quickly, without taking complicated steps. Streamlet is a promising open-source Python library, which enables developers to build attractive user interfaces.

Streamlet is the easiest way especially for people with no front-end knowledge to put their code into a web application:

- No front-end (html, CSS, js) experience or knowledge is required.
- You don’t need to spend months to create a web app.
- It is compatible with the majority of Python libraries (e.g. pandas, matplotlib, seaborn, platy, Kera's, PyTorch).
- Less code is needed to create amazing web apps.
- Data caching simplifies and speeds up computation pipelines.

8 Chapter Six: System Testing and Installation

9. Implementation

9.1. Diabetes Prediction (Model Building)

9.1.1. Random Forest Classifier Algorithm

9.2. Step1: Importing all necessary libraries

```
import NumPy as np
import pandas as pd
import matplotlib. pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn. impute import Simple Imputer
from sklearn. ensemble import RandomForestClassifier
```

9.3. Step2: Reading the dataset

Download the dataset from kaggle and keep it ready. Then we will read our dataset which in the format of .csv. We will be using the read_csv() function from the pandas library.

```
[ ]  d1 = pd.read_csv('/content/diabetes.csv')
     d2 = pd.read_csv('/content/diabetes_data.csv')
     data = pd.concat([d1,d2])
```

9.4. Step3: Analyzing our dataset

Let us see what is inside our dataset. We either use the info() function to get more information on the different attributes present in our dataset. We can also use the head() function to print the first few rows of the dataset to analyze on what we are dealing with. We can also use the describe() function to get some statistical properties on our data.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2768 entries, 0 to 1999
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               2768 non-null   int64  
 1   Glucose                   2768 non-null   int64  
 2   BloodPressure             2768 non-null   int64  
 3   SkinThickness             2768 non-null   int64  
 4   Insulin                   2768 non-null   int64  
 5   BMI                       2768 non-null   float64
 6   DiabetesPedigreeFunction  2768 non-null   float64
 7   Age                       2768 non-null   int64  
 8   Outcome                   2768 non-null   int64  
dtypes: float64(2), int64(7)
memory usage: 216.2 KB
```

```
data.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
data.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	2768.000000	2768.000000	2768.000000	2768.000000	2768.000000	2768.000000	2768.000000	2768.000000	2768.000000
mean	3.742775	121.102601	69.134393	20.824422	80.127890	32.137392	0.471193	33.132225	0.343931
std	3.323801	32.036508	19.231438	16.059596	112.301933	8.076127	0.325669	11.777230	0.475104
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.244000	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	37.000000	32.200000	0.375000	29.000000	0.000000
75%	6.000000	141.000000	80.000000	32.000000	130.000000	36.625000	0.624000	40.000000	1.000000
max	17.000000	199.000000	122.000000	110.000000	846.000000	80.600000	2.420000	81.000000	1.000000

9.5. Step4: Data Cleaning

In our context, data cleaning is basically the process of detecting or correcting corrupt or inaccurate records from the dataset.

First step in data cleaning is to check for consistent column name

```
[ ]  data.columns

    Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
           'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
          dtype='object')
```

```
[ ]  data.dtypes

    Pregnancies                 int64
    Glucose                     int64
    BloodPressure               int64
    SkinThickness               int64
    Insulin                     int64
    BMI                       float64
    DiabetesPedigreeFunction  float64
    Age                         int64
    Outcome                     int64
    dtype: object
```

Now let us check for any missing data in any of the columns of the dataset. For instance, we might want to look at the total number of missing values for each feature. We use the isnull() function to point out a particular value missing in that column. We use the sum() function to give us the count of the missing values in that column.

```
[ ]  print(data.isnull().sum())
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

We can see that there are no null values in any of the columns. Which means that the dataset is p erfect for us to build the machine learning model.

9.6. Step5: Feature Selection

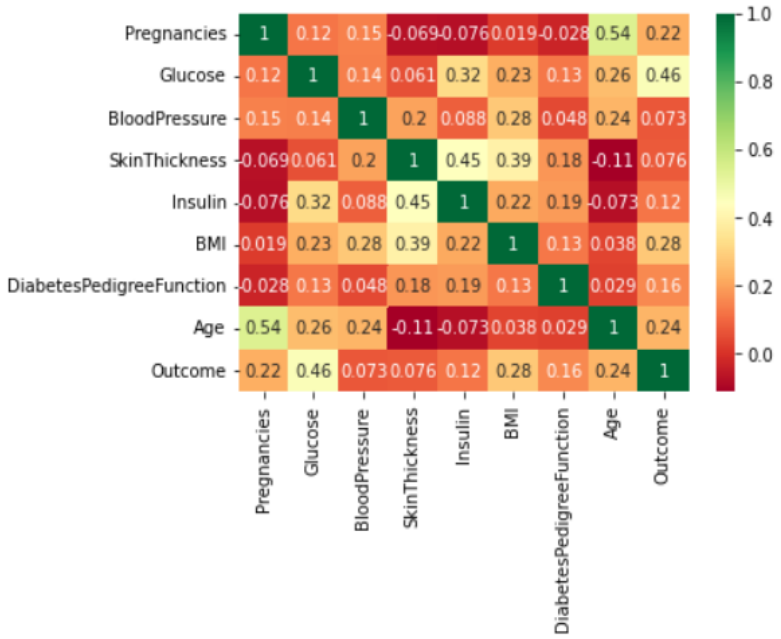
It is the most important part of feature engineering in machine learning. Here we are basically using co-relation. This is basically done in order to find out if all the features are positively co-related or negatively co-related to each other. The library used for this is seaborn, which we will import for feature selection.

```
[ ]  #get correlations of each features in dataset

     corrmat = data.corr()

     top_corr_features = corrmat.index

     plt.figure(figsize=(30,30))
```

```
     <Figure size 2160x2160 with 0 Axes>
     <Figure size 2160x2160 with 0 Axes>
```

We use the corr() to get the correlations of each feature present in the dataset. We then plot a heatmapwhich will help us understand the correlations in a better way.

```
#plot heat map

g=sns.heatmap(data[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```




```
data.corr()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.122839	0.147253	-0.068673	-0.075734	0.018761	-0.027731	0.540805	0.223796
Glucose	0.122839	1.000000	0.142095	0.061023	0.323445	0.225308	0.127195	0.256958	0.460644
BloodPressure	0.147253	0.142095	1.000000	0.201167	0.087823	0.281560	0.048471	0.238684	0.072900
SkinThickness	-0.068673	0.061023	0.201167	1.000000	0.445345	0.393494	0.179830	-0.111895	0.075603
Insulin	-0.075734	0.323445	0.087823	0.445345	1.000000	0.215926	0.190500	-0.073458	0.123646
BMI	0.018761	0.225308	0.281560	0.393494	0.215926	1.000000	0.129766	0.038175	0.280928
DiabetesPedigreeFunction	-0.027731	0.127195	0.048471	0.179830	0.190500	0.129766	1.000000	0.028544	0.160664
Age	0.540805	0.256958	0.238684	-0.111895	-0.073458	0.038175	0.028544	1.000000	0.237050
Outcome	0.223796	0.460644	0.072900	0.075603	0.123646	0.280928	0.160664	0.237050	1.000000

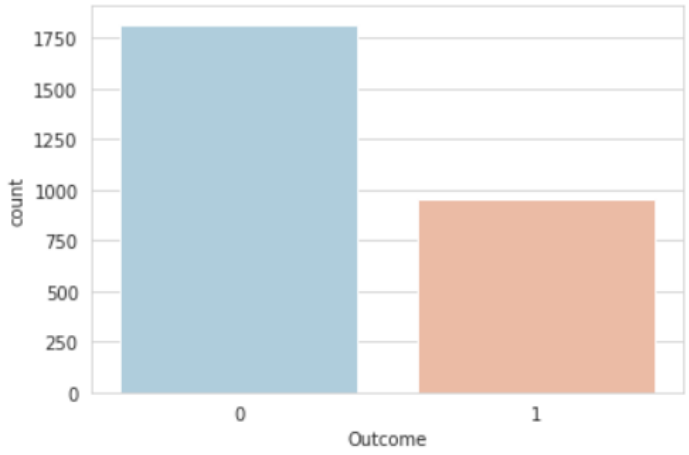
9.7. Step6: Inspecting for a balanced dataset

It is always a good practice to check whether a dataset is balanced or not. Balanced dataset is where the target classes are of approximately equal size.

Then we can plot a counterplot which will help us find the count of the target output present.

```
[ ]  sns.set_style('whitegrid')

     sns.countplot(x='Outcome', data=data, palette='RdBu_r')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f64188b1550>
```



Here we can see that the 0 count is 500 and the 1 count remains to be 268. The ratio between the 1 and 0 count is approximately 1:2 which will still work to the algorithm we will apply later on. This is not exactly an imbalanced data set since we have sufficient amount of data for 1 and 0.

9.8. Step7: Data Preprocessing

9.8.1. Creating independent features

```
[ ]  feature_columns= ['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']
     predicted_class = ['Outcome']
```

9.8.2. Splitting Data into training and test data

Next we will split the existing data into training and test data. We do this to train our machine learning model using the training data and test it's accuracy on the test data. We will be using train_test_split inbuilt function from scikit-learn for splitting the data array into two subsets. We use 80% of the split data as training data and 20% of it as test data.

```
[ ]  X = data[feature_columns].values
     Y = data[predicted_class].values
```

```
[ ]  X_train, X_test, Y_train, Y_test= train_test_split(X,Y,test_size=0.20, random_state=0)
```

If we carefully observe the data set, there are features with zero as their value. So, in this scenario, the data is basically not captured since they are supposed to be having some value. In that case let us find out how many missing values are present with respect to all the features.

```
print("Number of rows missing Glucose:{0}".format(len(data.loc[data['Glucose']==0])))
print("Number of rows missing BloodPressure:{0}".format(len(data.loc[data['BloodPressure']==0])))
print("Number of rows missing Insulin:{0}".format(len(data.loc[data['Insulin']==0])))
print("Number of rows missing BMI:{0}".format(len(data.loc[data['BMI']==0])))
print("Number of rows missing DiabetesPedigreeFunction:{0}".format(len(data.loc[data['DiabetesPedigreeFunction']==0])))
print("Number of rows missing Age:{0}".format(len(data.loc[data['Age']==0])))
print("Number of rows missing SkinThickness:{0}".format(len(data.loc[data['SkinThickness']==0])))
```

```
Number of rows missing Glucose:18
Number of rows missing BloodPressure:125
Number of rows missing Insulin:1330
Number of rows missing BMI:39
Number of rows missing DiabetesPedigreeFunction:0
Number of rows missing Age:0
Number of rows missing SkinThickness:800
```

9.8.3. Using Imputation function

Next, we have to apply an imputation function and the imputation function here works on the strategy of mean. In Imputer, wherever there are missing values as zero, we will apply the strategy of mean with respect to that particular feature to try and replace it.

```
[ ]  fill_values = SimpleImputer(missing_values = 0, strategy = 'mean')
```

Next, we are going to fit and transform our X_train and X_test.

```
[ ]  X_train = fill_values.fit_transform(X_train)
     X_test = fill_values.fit_transform(X_test)
```

9.9. Step8: Building and Training a Machine Learning Model

Next, we are going to build a machine learning model which is going to make our predictions after we train our model with the training data and test its accuracy with the test data.

We will be using Random Forest Classifier.

Random Forest Classifier creates a set of decision trees from randomly selected subset of training set. It then aggregates the votes from different decision trees to decide the final class of the test object. First let us import it from sklearn.ensemble and then we will assign it to a variable called 'model'.

```
[ ]  model=RandomForestClassifier(n_estimators=100, criterion='entropy', random_state=0)
```

Next, we will train this model using our training dataset. Training the model is very important as it will determine the outcome of its prediction. We use the fit () function to fit in both X_train and Y_train

```
model.fit(X_train,Y_train)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  """Entry point for launching an IPython kernel.
RandomForestClassifier(criterion='entropy', random_state=0)
```

9.10. Step9: Testing the accuracy of the Machine Learning Model

Since our model has been trained using the training data, we will now test the accuracy of the model on the test data. But firs let us fit our test data to our model.

```
model.fit(X_test,Y_test)
model.score(X_test,Y_test)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  """Entry point for launching an IPython kernel.
1.0
```

```
[ ]  y_test_predicted = model.predict(X_test)
     y_train_predicted = model.predict(X_train)

[ ]  from sklearn.metrics import accuracy_score

[ ]  accuracy_score(Y_test,y_test_predicted)

     0.987012987012987

[ ]  accuracy_score(Y_train,y_train_predicted)

     0.8581752484191508

[ ]
```

9.11. Step10: Making Predictions

```
[ ]  x_demo=[[1,111,65,25,25,0.350,35,1.3]]
     if model.predict(x_demo)==0:
         print("Person is likely to NOT have diabetes")
     else:
         print("Person is likely to have diabetes")
```

```
Person is likely to NOT have diabetes
```

```
[ ]  x_demo=[[5,126,70,160,35,0.242,0.000,66]]
     if model.predict(x_demo)==0:
         print("Person is likely to NOT have diabetes")
     else:
         print("Person is likely to have diabetes")
```

```
Person is likely to NOT have diabetes
```

```
[ ]  x_demo=[[1,122,90,51,220,49.7,66,0.325]]
     if model.predict(x_demo)==0:
         print("Person is likely to NOT have diabetes")
     else:
         print("Person is likely to have diabetes")
```

```
Person is likely to have diabetes
```

10. Predictive System

```
[ ]  x_demo=[[input('pregnancies'),input('Glucose'),input('BloodPressure'),input('SkinThickness'),
              input('Insulin'),input('BMI'),input('Age'),input('DiabetesPedigreeFunction')]]
     if model.predict(x_demo)==0:
         print("Person is likely to NOT have diabetes")
     else:

         print("Person is likely to have diabetes")
```

```
pregnancies1
Glucose128
BloodPressure88
SkinThickness39
Insulin110
BMI36.5
Age37
DiabetesPedigreeFunction1.057
Person is likely to NOT have diabetes
```

11. Using another dataset

```
[ ]  data2 = pd.read_csv("diabetes_data.csv")
```

```
[ ]  data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               2000 non-null   int64
 1   Glucose                   2000 non-null   int64
 2   BloodPressure             2000 non-null   int64
 3   SkinThickness             2000 non-null   int64
 4   Insulin                   2000 non-null   int64
 5   BMI                       2000 non-null   float64
 6   DiabetesPedigreeFunction  2000 non-null   float64
 7   Age                       2000 non-null   int64
 8   Outcome                   2000 non-null   int64
dtypes: float64(2), int64(7)
memory usage: 140.8 KB
```

```
[ ]  data2.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	2	138	62	35	0	33.6	0.127	47	1
1	0	84	82	31	125	38.2	0.233	23	0
2	0	145	0	0	0	44.2	0.630	31	1
3	0	135	68	42	250	42.3	0.365	24	1
4	1	139	62	41	480	40.7	0.536	21	0

```
[ ]  data2.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000
mean	3.703500	121.182500	69.145500	20.935000	80.254000	32.193000	0.470930	33.090500	0.342000
std	3.306063	32.068636	19.188315	16.103243	111.180534	8.149901	0.323553	11.786423	0.474498
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	63.500000	0.000000	0.000000	27.375000	0.244000	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	40.000000	32.300000	0.376000	29.000000	0.000000
75%	6.000000	141.000000	80.000000	32.000000	130.000000	36.800000	0.624000	40.000000	1.000000
max	17.000000	199.000000	122.000000	110.000000	744.000000	80.600000	2.420000	81.000000	1.000000

```
[ ] data2.columns
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

```
[ ] data2.dtypes
```

```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

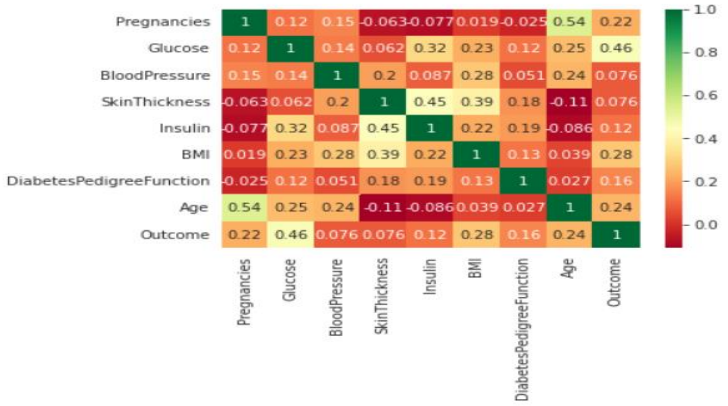
```
print(data2.isnull().sum())
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
[ ] #get correlations of each features in dataset
    corrmat = data2.corr()
    top_corr_features = corrmat.index
    plt.figure(figsize=(20,20))
```

```
<Figure size 1440x1440 with 0 Axes>
<Figure size 1440x1440 with 0 Axes>
```

```
#plot heat map
g=sns.heatmap(data2[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```

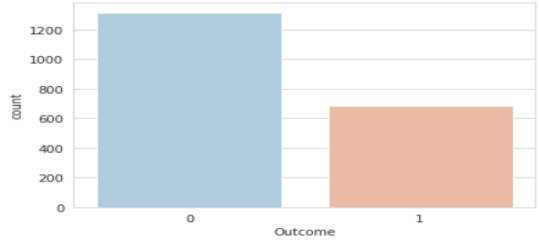



```
data2.corr()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.120405	0.149672	-0.063375	-0.076600	0.019475	-0.025453	0.539457	0.224437
Glucose	0.120405	1.000000	0.138044	0.062368	0.320371	0.226864	0.123243	0.254496	0.458421
BloodPressure	0.149672	0.138044	1.000000	0.198800	0.087384	0.281545	0.051331	0.238375	0.075958

```
[ ] sns.set_style('whitegrid')
    sns.countplot(x='Outcome', data=data2, palette='RdBu_r')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbcdb7d8110>
```



```
[ ] feature2_columns= ['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']
    predicted2_class = ['Outcome']

[ ] X2 = data2[feature2_columns].values
    Y2 = data2[predicted2_class].values

[ ] X2_train, X2_test, Y2_train, Y2_test= train_test_split(X2,Y2,test_size=0.20, random_state=1)
```

```
print("Number of rows missing Glucose:{0}".format(len(data2.loc[data2['Glucose']==0])))
print("Number of rows missing BloodPressure:{0}".format(len(data2.loc[data2['BloodPressure']==0])))
print("Number of rows missing Insulin:{0}".format(len(data2.loc[data2['Insulin']==0])))
print("Number of rows missing BMI:{0}".format(len(data2.loc[data2['BMI']==0])))
print("Number of rows missing DiabetesPedigreeFunction:{0}".format(len(data2.loc[data2['DiabetesPedigreeFunction']==0])))
print("Number of rows missing Age:{0}".format(len(data2.loc[data2['Age']==0])))
print("Number of rows missing SkinThickness:{0}".format(len(data2.loc[data2['SkinThickness']==0])))
```

```
Number of rows missing Glucose:13
Number of rows missing BloodPressure:90
Number of rows missing Insulin:956
Number of rows missing BMI:28
Number of rows missing DiabetesPedigreeFunction:0
Number of rows missing Age:0
Number of rows missing SkinThickness:573
```

```
[ ] fill_values2 = SimpleImputer(missing_values = 0, strategy = 'mean' )

[ ] X2_train = fill_values2.fit_transform(X2_train)
    X2_test = fill_values2.fit_transform(X2_test)
```

```
model2=RandomForestClassifier(n_estimators=10, criterion='entropy', random_state=1)
model2.fit(X2_train,Y2_train)
model2.fit(X2_test,Y2_test)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  """
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=1)
```

```
[ ]  model2.score(X2_test,Y2_test)

     0.9925

[ ]  y2_train_predicted = model2.predict(X2_train)

[ ]  accuracy_score(Y2_train,y2_train_predicted)

     0.820625
```

```
model2.predict(X_train)

array([1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1])
```

Predictive System2

```
[ ]  x_demo=[[input('pregnancies'),input('Glucose'),input('BloodPressure'),input('SkinThickness'),
             input('Insulin'),input('BMI'),input('Age'),input('DiabetesPedigreeFunction')]]
     if model2.predict(x_demo)==0:
         print("Person is likely to NOT have diabetes")
     else:

         print("Person is likely to have diabetes")
```

```
pregnancies4
Glucose125
BloodPressure70
SkinThickness18
Insulin122
BMI28.9
Age45
DiabetesPedigreeFunction1.144
Person is likely to NOT have diabetes
```

11.1. SVM (Support Vector Machine) Algorithm

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import svm
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import Normalizer
from keras.layers import Activation, Dense, Dropout, BatchNormalization, Input
from keras.models import Model
from tensorflow.keras.optimizers import Adam
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
%matplotlib inline
plt.style.use('fivethirtyeight')
```

```
db = pd.read_csv('/content/diabetes_binary_health_indicators_BRFSS2015.csv')
db.head()
```

	Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	..
0	0.0	1.0	1.0	1.0	40.0	1.0	0.0	0.0	0.0	0.0	.
1	0.0	0.0	0.0	0.0	25.0	1.0	0.0	0.0	1.0	0.0	.
2	0.0	1.0	1.0	1.0	28.0	0.0	0.0	0.0	0.0	1.0	.
3	0.0	1.0	0.0	1.0	27.0	0.0	0.0	0.0	1.0	1.0	.
4	0.0	1.0	1.0	1.0	24.0	0.0	0.0	0.0	1.0	1.0	.

5 rows × 22 columns

	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age	Education	Income
.	1.0	0.0	5.0	18.0	15.0	1.0	0.0	9.0	4.0	3.0
.	0.0	1.0	3.0	0.0	0.0	0.0	0.0	7.0	6.0	1.0
.	1.0	1.0	5.0	30.0	30.0	1.0	0.0	9.0	4.0	8.0
.	1.0	0.0	2.0	0.0	0.0	0.0	0.0	11.0	3.0	6.0
.	1.0	0.0	2.0	3.0	0.0	0.0	0.0	11.0	5.0	4.0

```
db.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 253680 entries, 0 to 253679
Data columns (total 22 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   Diabetes_binary       253680 non-null  float64
 1   HighBP                253680 non-null  float64
 2   HighChol              253680 non-null  float64
 3   CholCheck             253680 non-null  float64
 4   BMI                   253680 non-null  float64
 5   Smoker                253680 non-null  float64
 6   Stroke                253680 non-null  float64
 7   HeartDiseaseorAttack  253680 non-null  float64
 8   PhysActivity          253680 non-null  float64
 9   Fruits                253680 non-null  float64
 10  Veggies               253680 non-null  float64
 11  HvyAlcoholConsump     253680 non-null  float64
 12  AnyHealthcare         253680 non-null  float64
 13  NoDocbcCost           253680 non-null  float64
 14  GenHlth               253680 non-null  float64
 15  MentHlth              253680 non-null  float64
 16  PhysHlth              253680 non-null  float64
 17  DiffWalk              253680 non-null  float64
 18  Sex                   253680 non-null  float64
 19  Age                   253680 non-null  float64
 20  Education             253680 non-null  float64
 21  Income                253680 non-null  float64
dtypes: float64(22)
memory usage: 42.6 MB
```

```
db.shape
```

```
(253680, 22)
```

there is no missing data.

```
db.describe()
```

	Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	...
count	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	...
mean	0.139333	0.429001	0.424121	0.962670	28.382364	0.443169	0.040571	0.094186	0.756544	0.634256	...
std	0.346294	0.494934	0.494210	0.189571	6.608694	0.496761	0.197294	0.292087	0.429169	0.481639	...
min	0.000000	0.000000	0.000000	0.000000	12.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
25%	0.000000	0.000000	0.000000	1.000000	24.000000	0.000000	0.000000	0.000000	1.000000	0.000000	...
50%	0.000000	0.000000	0.000000	1.000000	27.000000	0.000000	0.000000	0.000000	1.000000	1.000000	...
75%	0.000000	1.000000	1.000000	1.000000	31.000000	1.000000	0.000000	0.000000	1.000000	1.000000	...
max	1.000000	1.000000	1.000000	1.000000	98.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...

...	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age	Education	Income
...	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000
...	0.951053	0.084177	2.511392	3.184772	4.242081	0.168224	0.440342	8.032119	5.050434	6.053875
...	0.215759	0.277654	1.068477	7.412847	8.717951	0.374066	0.496429	3.054220	0.985774	2.071148
...	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	1.000000
...	1.000000	0.000000	2.000000	0.000000	0.000000	0.000000	0.000000	6.000000	4.000000	5.000000
...	1.000000	0.000000	2.000000	0.000000	0.000000	0.000000	0.000000	8.000000	5.000000	7.000000
...	1.000000	0.000000	3.000000	2.000000	3.000000	0.000000	1.000000	10.000000	6.000000	8.000000
...	1.000000	1.000000	5.000000	30.000000	30.000000	1.000000	1.000000	13.000000	6.000000	8.000000

```
db.corr()
```

	Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits
Diabetes_binary	1.000000	0.263129	0.200276	0.064761	0.216843	0.060789	0.105816	0.177282	-0.118133	-0.040779
HighBP	0.263129	1.000000	0.298199	0.098508	0.213748	0.096991	0.129575	0.209361	-0.125267	-0.040555
HighChol	0.200276	0.298199	1.000000	0.085642	0.106722	0.091299	0.092620	0.180765	-0.078046	-0.040859
CholCheck	0.064761	0.098508	0.085642	1.000000	0.034495	-0.009929	0.024158	0.044206	0.004190	0.023849
BMI	0.216843	0.213748	0.106722	0.034495	1.000000	0.013804	0.020153	0.052904	-0.147294	-0.087518
Smoker	0.060789	0.096991	0.091299	-0.009929	0.013804	1.000000	0.061173	0.114441	-0.087401	-0.077666
Stroke	0.105816	0.129575	0.092620	0.024158	0.020153	0.061173	1.000000	0.203002	-0.069151	-0.013389
HeartDiseaseorAttack	0.177282	0.209361	0.180765	0.044206	0.052904	0.114441	0.203002	1.000000	-0.087299	-0.019790
PhysActivity	-0.118133	-0.125267	-0.078046	0.004190	-0.147294	-0.087401	-0.069151	-0.087299	1.000000	0.142756
Fruits	-0.040779	-0.040555	-0.040859	0.023849	-0.087518	-0.077666	-0.013389	-0.019790	0.142756	1.000000
Veggies	-0.056584	-0.061266	-0.039874	0.006121	-0.062275	-0.030678	-0.041124	-0.039167	0.153150	0.254342

..	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age	Education	Income
...	0.016255	0.031433	0.293569	0.069315	0.171337	0.218344	0.031430	0.177442	-0.124456	-0.163919
...	0.038425	0.017358	0.300530	0.056456	0.161212	0.223618	0.052207	0.344452	-0.141358	-0.171235
...	0.042230	0.013310	0.208426	0.062069	0.121751	0.144672	0.031205	0.272318	-0.070802	-0.085459
...	0.117626	-0.058255	0.046589	-0.008366	0.031775	0.040585	-0.022115	0.090321	0.001510	0.014259
...	-0.018471	0.058206	0.239185	0.085310	0.121141	0.197078	0.042950	-0.036618	-0.103932	-0.100069
...	-0.023251	0.048946	0.163143	0.092196	0.116460	0.122463	0.093662	0.120641	-0.161955	-0.123937
...	0.008776	0.034804	0.177942	0.070172	0.148944	0.176567	0.002978	0.126974	-0.076009	-0.128599
...	0.018734	0.031000	0.258383	0.064621	0.181698	0.212709	0.086096	0.221618	-0.099600	-0.141011
...	0.035505	-0.061638	-0.266186	-0.125587	-0.219230	-0.253174	0.032482	-0.092511	0.199658	0.198539
...	0.031544	-0.044243	-0.103854	-0.068217	-0.044633	-0.048352	-0.091175	0.064547	0.110187	0.079929
...	0.029584	-0.032232	-0.123066	-0.058884	-0.064290	-0.080506	-0.064765	-0.009771	0.154329	0.151087

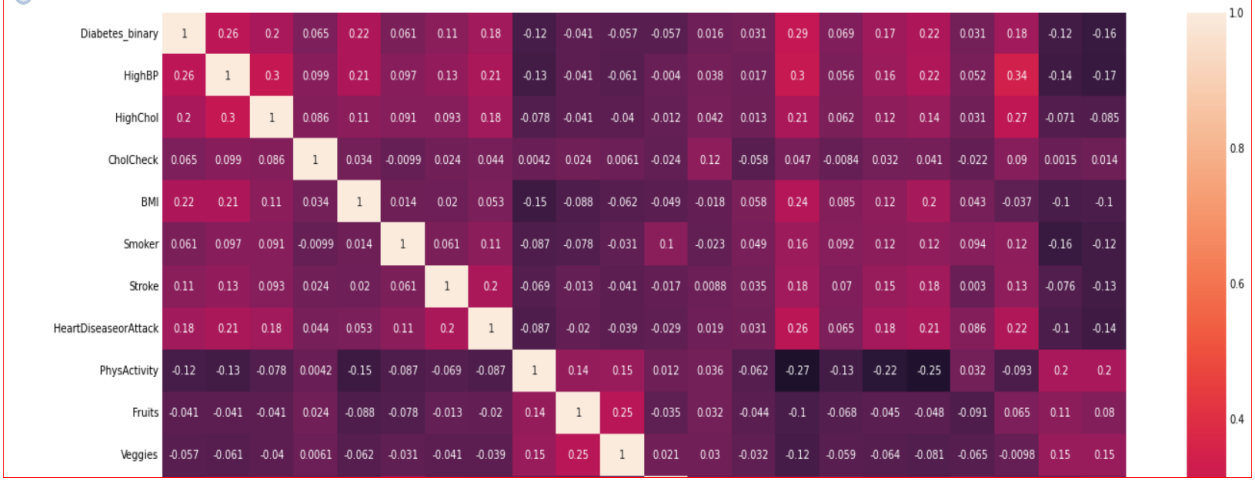
HvyAlcoholConsump	-0.057056	-0.003972	-0.011543	-0.023730	-0.048736	0.101619	-0.016950	-0.028991	0.012392	-0.035288	...
AnyHealthcare	0.016255	0.038425	0.042230	0.117626	-0.018471	-0.023251	0.008776	0.018734	0.035505	0.031544	...
NoDocbcCost	0.031433	0.017358	0.013310	-0.058255	0.058206	0.048946	0.034804	0.031000	-0.061638	-0.044243	...
GenHlth	0.293569	0.300530	0.208426	0.046589	0.239185	0.163143	0.177942	0.258383	-0.266186	-0.103854	...
MentHlth	0.069315	0.056456	0.062069	-0.008366	0.085310	0.092196	0.070172	0.064621	-0.125587	-0.068217	...
PhysHlth	0.171337	0.161212	0.121751	0.031775	0.121141	0.116460	0.148944	0.181698	-0.219230	-0.044633	...
DiffWalk	0.218344	0.223618	0.144672	0.040585	0.197078	0.122463	0.176567	0.212709	-0.253174	-0.048352	...
Sex	0.031430	0.052207	0.031205	-0.022115	0.042950	0.093662	0.002978	0.086096	0.032482	-0.091175	...
Age	0.177442	0.344452	0.272318	0.090321	-0.036618	0.120641	0.126974	0.221618	-0.092511	0.064547	...
Education	-0.124456	-0.141358	-0.070802	0.001510	-0.103932	-0.161955	-0.076009	-0.099600	0.199658	0.110187	...
Income	-0.163919	-0.171235	-0.085459	0.014259	-0.100069	-0.123937	-0.128599	-0.141011	0.198539	0.079929	...

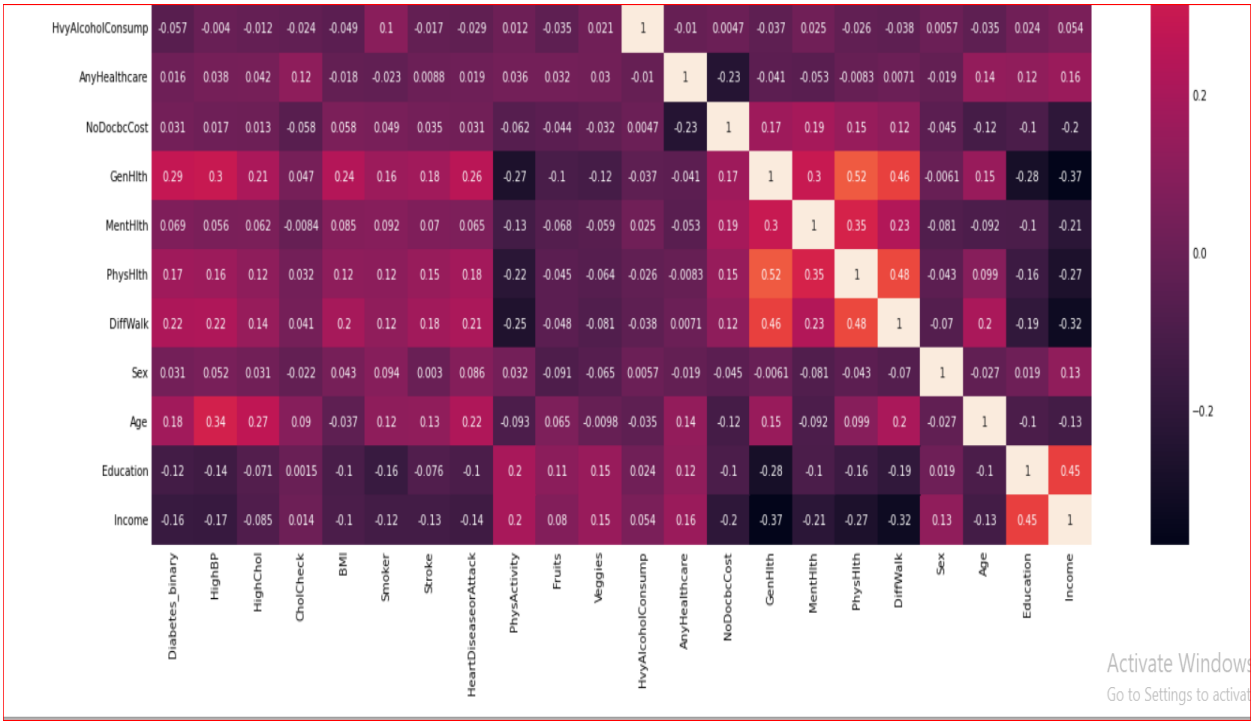
22 rows × 22 columns

-0.028991	0.012392	-0.035288	...	-0.010488	0.004684	-0.036724	0.024716	-0.026415	-0.037668	0.005740	-0.034578	0.023997	0.053619
0.018734	0.035505	0.031544	...	1.000000	-0.232532	-0.040817	-0.052707	-0.008276	0.007074	-0.019405	0.138046	0.122514	0.157999
0.031000	-0.061638	-0.044243	...	-0.232532	1.000000	0.166397	0.192107	0.148998	0.118447	-0.044931	-0.119777	-0.100701	-0.203182
0.258383	-0.266186	-0.103854	...	-0.040817	0.166397	1.000000	0.301674	0.524364	0.456920	-0.006091	0.152450	-0.284912	-0.370014
0.064621	-0.125587	-0.068217	...	-0.052707	0.192107	0.301674	1.000000	0.353619	0.233688	-0.080705	-0.092068	-0.101830	-0.209806
0.181698	-0.219230	-0.044633	...	-0.008276	0.148998	0.524364	0.353619	1.000000	0.478417	-0.043137	0.099130	-0.155093	-0.266799
0.212709	-0.253174	-0.048352	...	0.007074	0.118447	0.456920	0.233688	0.478417	1.000000	-0.070299	0.204450	-0.192642	-0.320124
0.086096	0.032482	-0.091175	...	-0.019405	-0.044931	-0.006091	-0.080705	-0.043137	-0.070299	1.000000	-0.027340	0.019480	0.127141
0.221618	-0.092511	0.064547	...	0.138046	-0.119777	0.152450	-0.092068	0.099130	0.204450	-0.027340	1.000000	-0.101901	-0.127775
-0.099600	0.199658	0.110187	...	0.122514	-0.100701	-0.284912	-0.101830	-0.155093	-0.192642	0.019480	-0.101901	1.000000	0.449106
-0.141011	0.198539	0.079929	...	0.157999	-0.203182	-0.370014	-0.209806	-0.266799	-0.320124	0.127141	-0.127775	0.449106	1.000000

```
f, ax = plt.subplots(1, figsize=(20,15))
sns.heatmap(db.corr(), annot=True, ax=ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f674ec93350>
```

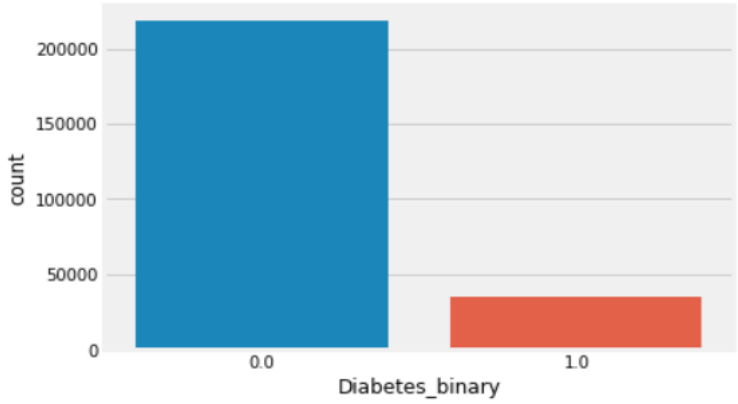




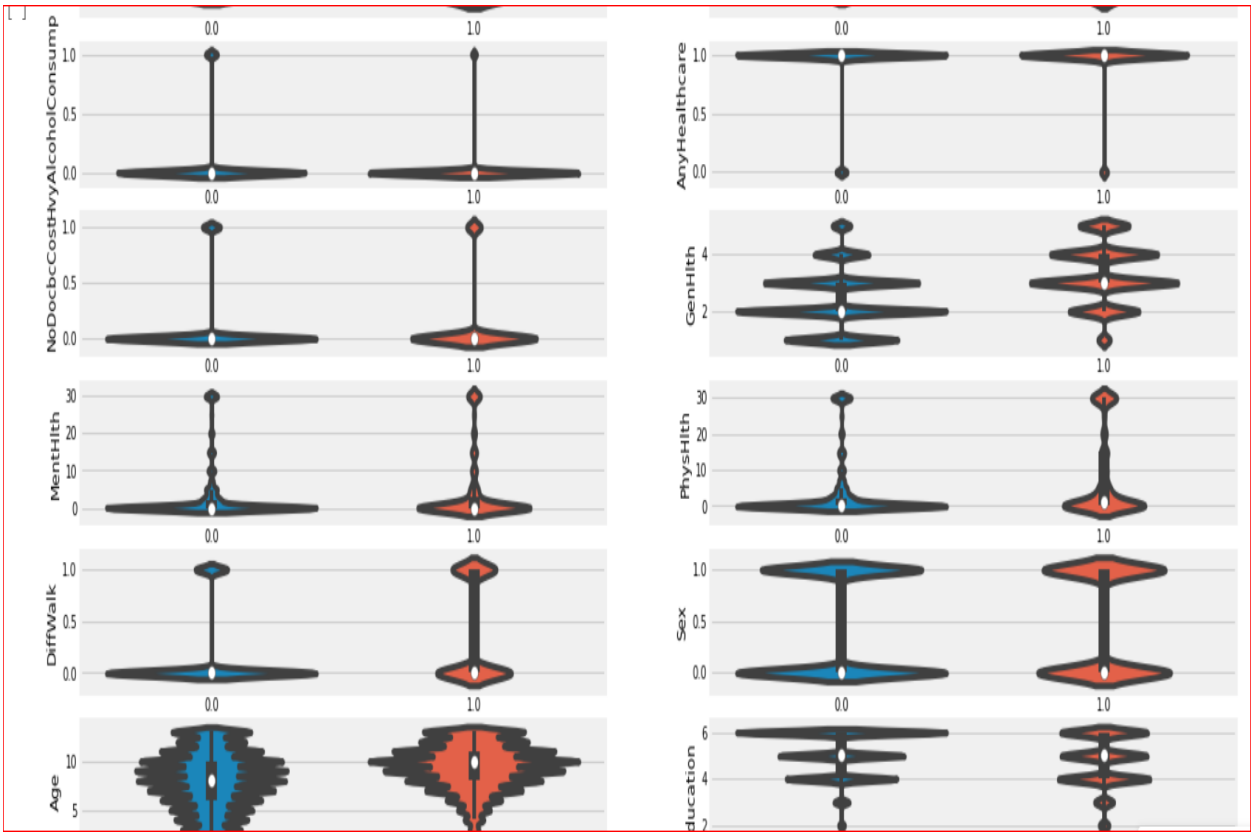
GenHlth, HighBp, BMI, DiffWalk, HighChol, HeartDiseaseorAttack, Age, PhysHlth are in order affect the diabetes out.

```
sns.countplot(x=db.Diabetes_binary)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f674bf1af90>
```



```
f, axes = plt.subplots(11,2, figsize=(20,20))
sns.violinplot(x=db.Diabetes_binary ,y=db.HighBP, ax=axes[0,0])
sns.violinplot(x=db.Diabetes_binary ,y=db.HighChol, ax=axes[0,1])
sns.violinplot(x=db.Diabetes_binary ,y=db.CholCheck, ax=axes[1,0])
sns.violinplot(x=db.Diabetes_binary,y=db.Smoker, ax=axes[1,1])
sns.violinplot(x=db.Diabetes_binary ,y=db.Stroke, ax=axes[2,0])
sns.violinplot(x=db.Diabetes_binary ,y=db.BMI, ax=axes[2,1])
sns.violinplot(x=db.Diabetes_binary ,y=db.HeartDiseaseorAttack, ax=axes[3,0])
sns.violinplot(x=db.Diabetes_binary ,y=db.PhysActivity, ax=axes[3,1])
sns.violinplot(x=db.Diabetes_binary ,y=db.Fruits, ax=axes[4,0])
sns.violinplot(x=db.Diabetes_binary ,y=db.Veggies, ax=axes[4,1])
sns.violinplot(x=db.Diabetes_binary ,y=db.HvyAlcoholConsump, ax=axes[5,0])
sns.violinplot(x=db.Diabetes_binary ,y=db.AnyHealthcare, ax=axes[5,1])
sns.violinplot(x=db.Diabetes_binary ,y=db.NoDocbcCost, ax=axes[6,0])
sns.violinplot(x=db.Diabetes_binary ,y=db.GenHlth, ax=axes[6,1])
sns.violinplot(x=db.Diabetes_binary ,y=db.MentHlth, ax=axes[7,0])
sns.violinplot(x=db.Diabetes_binary ,y=db.PhysHlth, ax=axes[7,1])
sns.violinplot(x=db.Diabetes_binary ,y=db.DiffWalk, ax=axes[8,0])
sns.violinplot(x=db.Diabetes_binary ,y=db.Sex, ax=axes[8,1])
sns.violinplot(x=db.Diabetes_binary ,y=db.Age, ax=axes[9,0])
sns.violinplot(x=db.Diabetes_binary ,y=db.Education, ax=axes[9,1])
sns.violinplot(x=db.Diabetes_binary ,y=db.Income, ax=axes[10,0])
```



```
db.columns
```

```
Index(['Diabetes_binary', 'HighBP', 'HighChol', 'CholCheck', 'BMI', 'Smoker',
       'Stroke', 'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',
       'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth',
       'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education',
       'Income'],
      dtype='object')
```

```
column_names = db.columns
column_names = column_names.drop('Diabetes_binary')
for name in column_names:
    print('{}\n'.format(name))
    print(db.groupby(['Diabetes_binary'])[name].mean())
    print('/'*50)
    print()
```

```
HighBP

Diabetes_binary
0.0    0.378096
1.0    0.754250
Name: HighBP, dtype: float64
//////////////////////////////////////////////////

HighChol

Diabetes_binary
0.0    0.385875
1.0    0.671031
Name: HighChol, dtype: float64
//////////////////////////////////////////////////

CholCheck

Diabetes_binary
0.0    0.958444
1.0    0.993165
Name: CholCheck, dtype: float64
//////////////////////////////////////////////////

BMI

Diabetes_binary
0.0    27.831891
1.0    31.982966
Name: BMI, dtype: float64
//////////////////////////////////////////////////
```

In the dataset, people diagnosed with diabetes had higher values for (Age, DiffWalk, PhysHlth, MentHlth, GenHlth, NoDocbcCost, AnyHealthcare, HeartDiseaseorAttack, Stroke, BMI, Smoker, HighBp, HighChol, CholCheck).

```
f, axes = plt.subplots(11,2, figsize=(40,40))
sns.distplot(db.HighBP, ax=axes[0,0])
sns.distplot(db.HighChol, ax=axes[0,1])
sns.distplot(db.CholCheck, ax=axes[1,0])
sns.distplot(db.BMI, ax=axes[1,1])
sns.distplot(db.Smoker, ax=axes[2,0])
sns.distplot(db.Stroke, ax=axes[2,1])
sns.distplot(db.HeartDiseaseorAttack, ax=axes[3,0])
sns.distplot(db.AnyHealthcare, ax=axes[3,1])
sns.distplot(db.NoDocbcCost, ax=axes[4,0])
sns.distplot(db.GenHlth, ax=axes[4,1])
sns.distplot(db.MentHlth, ax=axes[5,0])
sns.distplot(db.PhysHlth, ax=axes[5,1])
sns.distplot(db.DiffWalk, ax=axes[6,0])
sns.distplot(db.Sex, ax=axes[6,1])
sns.distplot(db.Age, ax=axes[7,0])
sns.distplot(db.PhysActivity, ax=axes[7,1])
sns.distplot(db.Fruits, ax=axes[8,0])
sns.distplot(db.Veggies, ax=axes[8,1])
sns.distplot(db.HvyAlcoholConsump, ax=axes[9,0])
sns.distplot(db.Education, ax=axes[9,1])
sns.distplot(db.Income, ax=axes[10,0])
```

```
db.HighBP.replace(0, db.HighBP.median(), inplace=True)
db.HighChol.replace(0, db.HighChol.median(), inplace=True)
db.CholCheck.replace(0, db.CholCheck.median(), inplace=True)
db.BMI.replace(0, db.BMI.median(), inplace=True)
db.Smoker.replace(0, db.Smoker.median(), inplace=True)
db.Stroke.replace(0, db.Stroke.median(), inplace=True)
db.HeartDiseaseorAttack.replace(0, db.HeartDiseaseorAttack.median(), inplace=True)
db.AnyHealthcare.replace(0, db.AnyHealthcare.median(), inplace=True)
db.NoDocbcCost.replace(0, db.NoDocbcCost.median(), inplace=True)
db.GenHlth.replace(0, db.GenHlth.median(), inplace=True)
db.MentHlth.replace(0, db.MentHlth.median(), inplace=True)
db.PhysHlth.replace(0, db.PhysHlth.median(), inplace=True)
db.DiffWalk.replace(0, db.DiffWalk.median(), inplace=True)
db.Sex.replace(0, db.Sex.median(), inplace=True)
db.Age.replace(0, db.Age.median(), inplace=True)
db.PhysActivity.replace(0, db.PhysActivity.median(), inplace=True)
db.Fruits.replace(0, db.Fruits.median(), inplace=True)
db.Veggies.replace(0, db.Veggies.median(), inplace=True)
db.HvyAlcoholConsump.replace(0, db.HvyAlcoholConsump.median(), inplace=True)
db.Education.replace(0, db.Education.median(), inplace=True)
db.Income.replace(0, db.Income.median(), inplace=True)
```

```
f, axes = plt.subplots(11,2, figsize=(40,40))
sns.distplot(db.HighBP, ax=axes[0,0])
sns.distplot(db.HighChol, ax=axes[0,1])
sns.distplot(db.CholCheck, ax=axes[1,0])
sns.distplot(db.BMI, ax=axes[1,1])
sns.distplot(db.Smoker, ax=axes[2,0])
sns.distplot(db.Stroke, ax=axes[2,1])
sns.distplot(db.HeartDiseaseorAttack, ax=axes[3,0])
sns.distplot(db.AnyHealthcare, ax=axes[3,1])
sns.distplot(db.NoDocbcCost, ax=axes[4,0])
sns.distplot(db.GenHlth, ax=axes[4,1])
sns.distplot(db.MentHlth, ax=axes[5,0])
sns.distplot(db.PhysHlth, ax=axes[5,1])
sns.distplot(db.DiffWalk, ax=axes[6,0])
sns.distplot(db.Sex, ax=axes[6,1])
sns.distplot(db.Age, ax=axes[7,0])
sns.distplot(db.PhysActivity, ax=axes[7,1])
sns.distplot(db.Fruits, ax=axes[8,0])
sns.distplot(db.Veggies, ax=axes[8,1])
sns.distplot(db.HvyAlcoholConsump, ax=axes[9,0])
sns.distplot(db.Education, ax=axes[9,1])
sns.distplot(db.Income, ax=axes[10,0])
```

11.2. Neural Network Model

```
X = db.drop('Diabetes_binary', axis =1).values
y = db.Diabetes_binary.values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2)
nl = Normalizer()
```

```
nl.fit(X_train)
X_train = nl.transform(X_train)
X_dev, X_test, y_dev, y_test = train_test_split(X_test, y_test, test_size=0.5, random_state=2)
X_dev = nl.transform(X_dev)
X_test = nl.transform(X_test)
```

```
[ ]  def nn():
        inputs = Input(name='inputs', shape=[X_train.shape[1],])
        layer = Dense(512, name='FC1')(inputs)
        layer = BatchNormalization(name='BC1')(layer)
        layer = Activation('relu', name='Activation1')(layer)
        layer = Dropout(0.3, name='Dropout1')(layer)
        layer = Dense(256, name='FC4')(layer)
        layer = BatchNormalization(name='BC4')(layer)
        layer = Activation('relu', name='Activation4')(layer)
        layer = Dropout(0.3, name='Dropout4')(layer)
        layer = Dense(128, name='FC6')(layer)
        layer = BatchNormalization(name='BC6')(layer)
        layer = Activation('relu', name='Activation6')(layer)
        layer = Dropout(0.3, name='Dropout6')(layer)
        layer = Dense(64, name='FC8')(layer)
        layer = BatchNormalization(name='BC8')(layer)
        layer = Dropout(0.3, name='Dropout8')(layer)
        layer = Dense(1, name='OutLayer')(layer)
        layer = Activation('sigmoid', name='sigmoid')(layer)
        model = Model(inputs=inputs, outputs=layer)
        return model
```

```
model = nn()
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inputs (InputLayer)         [(None, 21)]              0

 FC1 (Dense)                 (None, 512)               11264

 BC1 (BatchNormalization)    (None, 512)               2048

 Activation1 (Activation)    (None, 512)               0

 Dropout1 (Dropout)          (None, 512)               0

 FC4 (Dense)                 (None, 256)               131328

 BC4 (BatchNormalization)    (None, 256)               1024

 Activation4 (Activation)    (None, 256)               0

 Dropout4 (Dropout)          (None, 256)               0

 FC6 (Dense)                 (None, 128)               32896

 BC6 (BatchNormalization)    (None, 128)               512

 Activation6 (Activation)    (None, 128)               0

 Dropout6 (Dropout)          (None, 128)               0

 FC8 (Dense)                 (None, 64)                8256

 BC8 (BatchNormalization)    (None, 64)                256

 Dropout8 (Dropout)          (None, 64)                0

 OutLayer (Dense)            (None, 1)                 65

 sigmoid (Activation)        (None, 1)                 0

=================================================================
Total params: 187,649
Trainable params: 185,729
Non-trainable params: 1,920
_________________________________________________________________
```

```
model.compile(loss='binary_crossentropy', optimizer=Adam(), metrics=['accuracy'])
```

```
[ ] reduce_lr = ReduceLROnPlateau()
    early_stopping = EarlyStopping(patience=20, min_delta=0.0001)
```

```
[ ] model.fit(x=X_train, y=y_train, epochs=500, validation_data=(X_dev, y_dev), callbacks=[reduce_lr, early_stopping], verbose=0)
```

```
<keras.callbacks.History at 0x7fba1224f150>
```

```
x_lst = [X_train, X_dev, X_test]
y_lst = [y_train, y_dev, y_test]
for i,(x,y) in enumerate(zip(x_lst, y_lst)):
    y_pred = model.predict(x)
    y_pred = np.around(y_pred)
    y_pred = np.asarray(y_pred)
    if i == 0:
        print('Training set:')
        print('\tAccuracy:{:0.3f}\n\tClassification Report\n{}'.format(accuracy_score(y, y_pred),
                                                                       classification_report(y, y_pred)))
    elif i == 1:
        print('Dev set:')
        print('\tAccuracy:{:0.3f}\n\tClassification Report\n{}'.format(accuracy_score(y, y_pred),
                                                                       classification_report(y, y_pred)))
    else:
        print('Test set:')
        print('\tAccuracy:{:0.3f}\n\tClassification Report\n{}'.format(accuracy_score(y, y_pred),
                                                                       classification_report(y, y_pred)))
```

```
Training set:
        Accuracy:0.868
        Classification Report
              precision    recall  f1-score   support

           0       0.88      0.99      0.93    152917
           1       0.61      0.14      0.23     24659

    accuracy                           0.87    177576
   macro avg       0.74      0.56      0.58    177576
weighted avg       0.84      0.87      0.83    177576

Dev set:
        Accuracy:0.866
        Classification Report
              precision    recall  f1-score   support

           0       0.87      0.99      0.93     32696
           1       0.61      0.14      0.23      5356

    accuracy                           0.87     38052
   macro avg       0.74      0.56      0.58     38052
weighted avg       0.84      0.87      0.83     38052

Test set:
        Accuracy:0.865
        Classification Report
              precision    recall  f1-score   support

           0       0.87      0.98      0.93     32721
           1       0.58      0.14      0.22      5331

    accuracy                           0.87     38052
   macro avg       0.73      0.56      0.57     38052
weighted avg       0.83      0.87      0.83     38052
```

11.2.1. Recommendation System

11.2.1.1. Data Preprocessing

First, we found a dataset, on Kaggle, scrapped from a restaurant website which has some information provided for each record like: id, contributor_id, time it was submitted and some details about ingredients and steps of each meal.

```
In [16]: url = 'E:\\graduation_project\\New folder\\RAW_recipes.csv'
         df_large = pd.read_csv(url)
         df_large.head(3)
```

Out[16]:

	name	id	minutes	contributor_id	submitted	tags	nutrition	n_steps	steps	description	ingredients	n_ingredients
0	arriba baked winter squash mexican style	137739	55	47892	2005-09-16	['60-minutes-or-less', 'time-to-make', 'course...	[51.5, 0.0, 13.0, 0.0, 2.0, 0.0, 4.0]	11	['make a choice and proceed with recipe', 'dep...	autumn is my favorite time of year to cook! th...	['winter squash', 'mexican seasoning', 'mixed ...	7
1	a bit different breakfast pizza	31490	30	26278	2002-06-17	['30-minutes-or-less', 'time-to-make', 'course...	[173.4, 18.0, 0.0, 17.0, 22.0, 35.0, 1.0]	9	['preheat oven to 425 degrees f', 'press dough...	this recipe calls for the crust to be prebaked...	['prepared pizza crust', 'sausage patty', 'egg...	6
2	all in the kitchen chili	112140	130	196586	2005-02-25	['time-to-make', 'course', 'preparation', 'mai...	[269.8, 22.0, 32.0, 48.0, 39.0, 27.0, 5.0]	6	['brown ground beef in large pot', 'add choppe...	this modified version of 'mom's' chili was a h...	['ground beef', 'yellow onions', 'diced tomato...	13

We used this data because of the provided details about the ingredients. As we'll see, this can be used to generate a nutrient-column and a category-column.

Finally, this dataset was somehow converted to:

```
In [4]: user_meal = pd.read_csv('E:\\graduation_project\\diet rec system\\nut_df.csv')
        user_meal = user_meal[user_meal.columns[2:]]
```

```
In [5]: user_meal.head(3)
```

Out[5]:

	Meal_Id	user_id	name	category	rating	nutrient	n_ingredients	ingredients	n_steps	steps	Disease	diet
0	1	user_1883	arriba baked winter squash mexican style	baked	1	Fat	7	['winter squash', 'mexican seasoning', 'mixed ...	11	['make a choice and proceed with recipe', 'dep...	diabeties obesity	low_carb_diet ketogenic_diet
1	2	user_3850	a bit different breakfast pizza	pizza	1	Fat Carbohydrates	6	['prepared pizza crust', 'sausage patty', 'egg...	9	['preheat oven to 425 degrees f', 'press dough...	pregnancy diabeties obesity	NaN
2	3	user_4340	alouette potatoes	egg	6	Carbohydrates	11	['spreadable cheese with garlic and herbs', 'n...	11	['place potatoes in a large pot of lightly sal...	pregnancy obesity	low_fat_diet

This is the pre-last form we have developed.

Meal_Id: id given to a certain meal.

user_id: id given to a certain user.

nutrient: the dominant nutrients in a certain meal or the dominant nutrients in a certain patient's diet. It can take multiple values of: Fat, Carbohydrates, Fiber and Protien.

n_ingredients: how many ingredients are in a certain meal.

ingredients: ingredients in a certain meal.

n_steps: how many steps are required to cook a certain meal.

steps: steps required to cook a certain meal

Disease: diseases whom a certain meal is suitable for or diseases a certain patient has. It can take multiple values of: pregnancy, anemia, diabeties and obesity.

diet: diets whom a certain meal is suitable for or diseases a certain patient has. It can take multiple values of: low_carb_diet, low_fat_diet, high_protien_diet and ketogenic_diet.

Note: Disease and diet columns can take a NaN value if the meal is not classified as suitable for any of the mentioned diseases / diets or the user does not have any of the mentioned diseases / diets.

Summary of this process:

1. Generate a category column from words in ingredients, name and description columns.
2. Generate a nutrient column from words in ingredients, name and description columns.
3. Generate a Disease column from nutrient column.
4. Generate a diet column from nutrient column.

From this new dataset we can extract three forms:

- Profiles – dataset: contains users' data only.

```
In [8]: profiles = pd.read_csv('E:\\graduation_project\\diet rec system\\user_profiles.csv') # profiles of all users
```

```
In [9]: profiles.head()
```

Out[9]:

	Unnamed: 0	Unnamed: 0.1	user_id	nutrient	Disease	diet
0	0	0	user_1	Fat	NaN	low_carb_diet ketogenic_diet
1	1	1	user_10	Fat	pregnancy anemia diabeties obesity	low_carb_diet ketogenic_diet
2	2	2	user_100	Fiber Fat Carbohydrates	diabeties	NaN
3	3	3	user_1000	Fat Carbohydrates	NaN	NaN
4	4	4	user_10000	Fat	anemia diabeties	low_carb_diet ketogenic_diet

Here, nutrient, Disease and diet are specified a certain user. In other words, nutrient, Disease and diet here represent the user_features for a certain user.

- Recent – activity – dataset: contains data about how multiple users interact with multiple meals.

```
In [6]: recent_activity = pd.read_csv('E:\\graduation_project\\diet rec system\\recent_activity.csv') # recent activities of current user
        recent_activity.head()
```

Out[6]:

	user_id	Meal_Id	rated	liked	searched	tried	Timestamp
0	user_1883	1	1	0	0	0	2022-02-01 00:00:00
1	user_6586	5	0	0	1	1	2022-02-01 00:30:34
2	user_7743	8	0	0	1	1	2022-02-01 01:01:08
3	user_1778	14	0	1	0	1	2022-02-01 01:31:42
4	user_2179	16	0	0	1	0	2022-02-01 01:31:42

Here, the above dataset illustrates types of interactions may occur between users and meals.

- Meals – dataset: contains meals data only.

```
In [27]: meals.head()
```

Out[27]:

	Meal_Id	name	category	rating	nutrient	n_ingredients	ingredients	n_steps	steps	Disease	diet
0	1	arriba baked winter squash mexican style	baked	1	Fat	7	['winter squash', 'mexican seasoning', 'mixed ...	11	['make a choice and proceed with recipe', 'dep...	diabeties obesity	low_carb_diet ketogenic_diet
1	2	a bit different breakfast pizza	pizza	1	Fat Carbohydrates	6	['prepared pizza crust', 'sausage patty', 'egg...	9	['preheat oven to 425 degrees f', 'press dough...	pregnancy diabeties obesity	NaN
2	3	alouette potatoes	egg	6	Carbohydrates	11	['spreadable cheese with garlic and herbs', 'n...	11	['place potatoes in a large pot of lightly sal...	pregnancy obesity	low_fat_diet
3	4	amish tomato ketchup for canning	tomato	5	Carbohydrates	8	['tomato juice', 'apple cider vinegar', 'sugar...	5	['mix all ingredients& boil for 2 1 / 2 hours ...	pregnancy	low_fat_diet
4	5	apple a day milk shake	milk	7	Fat Carbohydrates	4	['milk', 'vanilla ice cream', 'frozen apple ju...	4	['combine ingredients in blender', 'cover and ...	pregnancy anemia diabeties obesity	NaN

11.2.1.2. Content-based Recommender

```
    def content_based(self, user_features):

        feature_df = self.get_features(self.df)

        #initialize model with k=40 neighbors
        model = NearestNeighbors(n_neighbors=40,algorithm='ball_tree')

        # fit model with features
        model.fit(feature_df)

        # Empty dataframe to contain results
        df_results = pd.DataFrame(columns=list(self.df.columns))
        #
        total_features = list(feature_df.columns)
        #print(total_features)
        #print(user_features)
        d = dict()

        for i in total_features:
            d[i]= 0

        for i in list(user_features):
            d[i] = 1

        final_input = list(d.values())
        #print(d)


        # getting distance and indices for k nearest neighbor
        distnaces , indices = model.kneighbors([final_input])

        for i in list(indices):
            df_results = df_results.append(self.df.iloc[i])

# convert zeros to ones for sample_input elements
        df_results = df_results.filter(['name', 'nutrient', 'Disease', 'diet', 'ingredients', 'steps'])
        df_results = df_results.drop_duplicates(subset=['name'])
        df_results = df_results.reset_index(drop=True)
        return df_results
```

Using KNN, content_based function takes a list of user_featues as an input and returns a dataframe of recommended meals which have similar features to the user's.

In this part, the similarity is measured between user_features and meal_featues. So, based, only, on the user profile, 40 recommendations are made and arranged by distance from the data point which is the user_features, in our case.

The output of this function is provided in the figure below.

```
In [133]: inputs = ['Fat', 'diabeties']
          content_based(inputs)
```

get_features()

Out[133]:

	name	nutrient	diet	Disease	ingredients	steps
0	chex caramel corn	Fat Carbohydrates	NaN	diabeties	['corn chex', 'microwave popcorn', 'honey roas...	['in large microwavable bowl , mix cereal , po...
1	chick o stick ice cream	Fat Carbohydrates	NaN	diabeties	['water', 'sugar', 'peanut butter', 'coconut m...	['heat the sugar and water in a small saucepan...
2	chewy philadelphia style soft pretzels	Fat Carbohydrates	NaN	diabeties	['milk', 'brown sugar', 'warm water', 'active ...	['in a saucepan , heat milk just until bubbles...
3	lima bean and corn casserole	Fat Carbohydrates	NaN	diabeties	['frozen lima beans', 'frozen whole kernel cor...	['cook lima beans and corn together , followin...
4	chewy oatmeal chip cookies	Fat Carbohydrates	NaN	diabeties	['unsalted butter', 'brown sugar', 'egg', 'van...	['preheat oven to 350 degrees', 'beat the butt...
5	lemony corn on the cob	Fat Carbohydrates	NaN	diabeties	['butter', 'lemon pepper seasoning', 'fresh co...	['combine the butter and the lemon pepper', 's...
6	lettuce wrapped corn	Fat Carbohydrates	NaN	diabeties	['butter', 'rosemary', 'marjoram', 'corn', 'ro...	['mix butter , rosemary and marjoram', 'spread...
7	like no other zucchini loaf	Fat Carbohydrates	NaN	diabeties	['brown sugar', 'white sugar', 'vegetable oil'...	['in a bowl , beat together first 7 ingredient...
8	chewy apple oatmeal cookies	Fat Carbohydrates	NaN	diabeties	['butter', 'brown sugar', 'sugar', 'eggs', 'va...	['in a large mixing bowl , cream butter and su...
9	chewy oatmeal brownies	Fat Carbohydrates	NaN	diabeties	['butter', 'sugar', 'eggs', 'vanilla extract',...	['combine sugar , eggs and vanilla with a fork...
10	chewy oatmeal peanut butter cookies	Fat Carbohydrates	NaN	diabeties	['flour', 'rolled oats', 'brown sugar', 'caste...	['preheat the oven to 180c', 'cream butter and...

get_featues is a function that returns dummies of all features. We used the built-in function get_dummies() to implement it as shown in the figure below.

In [113]:
```
def get_features(meals):
    # meals = dataframe
        # get dummies of meals

        #pd.read_csv('E:\\graduation_project\\diet rec system\\nut_df.csv') # main dataset
        nutrient_dummies = meals.nutrient.str.get_dummies(sep=' ')
        disease_dummies = meals.Disease.str.get_dummies(sep=' ')
        diet_dummies = meals.diet.str.get_dummies(sep=' ')
        feature_df = pd.concat([nutrient_dummies,disease_dummies,diet_dummies],axis=1)

        return feature_df
```

11.2.1.3. **User-Based Recommender:**

In [114]:
```
get_features(dataset)
```

Out[114]:

	Carbohydrates	Fat	Fiber	Protien	anemia	diabeties	obesity	pregnancy	high_protien_diet	ketogenic_diet	low_carb_diet	low_fat_diet
0	0	1	0	0	0	1	1	0	0	1	1	0
1	1	1	0	0	0	1	1	1	0	0	0	0
2	1	0	0	0	0	0	1	1	0	0	0	1
3	1	0	0	0	0	0	0	1	0	0	0	1
4	1	1	0	0	1	1	1	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
246139	1	0	0	0	1	0	0	1	0	0	0	1
246140	1	1	0	0	0	0	1	1	0	0	0	0
246141	1	0	0	0	1	1	1	1	0	0	0	1
246142	1	1	0	0	0	0	0	0	0	0	0	0
246143	0	1	0	0	1	1	1	1	0	1	1	0

246144 rows × 12 columns

```
def find_neighbors(self,user_features,k):
    # dataframe = profiles
    # features = user features
    features_df = self.get_features(self.profiles)

    #initialize model with k neighbors
    model = NearestNeighbors(n_neighbors=k,algorithm='ball_tree')

    # fit model with dataset features
    model.fit(features_df)


    total_features = features_df.columns
    d = dict()
    for i in total_features:
        d[i]= 0
    for i in user_features:
        d[i] = 1
    final_input = list(d.values())
    #rint(total_features)

    similar_neighbors = pd.DataFrame(columns=list(self.profiles.columns))

    # getting distance and indices for k nearest neighbor
    distnaces , indices = model.kneighbors([final_input])

    for i in list(indices):
        similar_neighbors = similar_neighbors.append(self.profiles.loc[i])

    similar_neighbors = similar_neighbors.reset_index(drop=True)

  #  similar_neighbors = self.k_neighbor([final_input],dataframe,k)
    return similar_neighbors
```

To implement a user – based recommender, we need to find the similar users in the very first step. That's why we implemented a find_neighbors function which returns a dataframe (similar_neighbors) of most 10 similar user out of the whole website users as shown in the figure.

In this part, we used the same function used in content – based recommender, KNN, but the difference here is in the way we measure the similarity. In other words, the difference is between whom we measure the similarity.

Similarity in this function is measured between the user_features and the features of other users which are provided from the profiles dataset we explored before.

Summarizing this point, KNN in the content – based function works on the meals dataset and measures the similarity between user_featues and meals he hasn't ever tried or interacted with. However, KNN in find_neighbors function works on the profiles dataset and takes the similarity between users, only, in consideration.

So, results of such a function is expected to be some profiles' data similar to what we have seen in the profiles dataset. The figure below says the same thing.

Out[149]:

	user_id	nutrient	Disease	diet
0	user_2103	Fat Carbohydrates	diabeties	NaN
1	user_4305	Fat Carbohydrates	diabeties	NaN
2	user_1376	Fat Carbohydrates	diabeties	NaN
3	user_4252	Fat Carbohydrates	diabeties	NaN
4	user_4145	Fat Carbohydrates	diabeties	NaN
5	user_4164	Fat Carbohydrates	diabeties	NaN
6	user_4288	Fat Carbohydrates	diabeties	NaN
7	user_4370	Fat Carbohydrates	diabeties	NaN
8	user_4106	Fat Carbohydrates	diabeties	NaN
9	user_4231	Fat Carbohydrates	diabeties	NaN

Now, find_neighbors() can be used to implement the user – based recommender.

Simply, we filter the very first dataset we explored to get the meals whom similar_neighbors had interacted with and the user haven't, the resulting dataframe of this filtering process is the user – based recommendations made for the user as shown.

```
def user_based(self,user_features,user_id):


    similar_users = self.find_neighbors(user_features,10)
    users = list(similar_users.user_id)

    results = self.recent_activity[self.recent_activity.user_id.isin(users)] #taking acitivies

    results = results[results['user_id'] != user_id] # selecting those which are not reviewed by user

    meals = list(results.Meal_Id.unique())

    results = self.df[self.df.Meal_Id.isin(meals)]

    results = results.filter(['Meal_Id','name','nutrient','ingredients','steps'])

    results = results.drop_duplicates(subset=['name'])
    results = results.reset_index(drop=True)
    return results
```

It should be taken in consideration that the length of the resulting dataframe in unknown. It can be any number.

38	88511	lemony herbed rice	Carbohydrates	diabeties obesity	low_fat_diet	['reduced-sodium chicken broth', 'long grain r...	['in a large saucepan , combine the broth , ri...
39	88499	lemony french bread grilled	Carbohydrates	diabeties obesity	low_fat_diet	['unsalted butter', 'lemon zest', 'garlic powd...	['preheat grill to 350 to 400 degrees f', 'in ...
40	850	3 bean hot dish	Carbohydrates	NaN	NaN	['hamburger', 'onion', 'pork and beans', 'lima...	['mix all of the ingredients and bake for 1 ho...
41	1489	a w coney island hot dog sauce	Carbohydrates	NaN	NaN	['hamburger', 'onions', 'tomato paste', 'tomat...	['brown hamburger and onions in very large ski...
42	1776	adventurous spirit lemon infused rice	Carbohydrates	NaN	NaN	['jasmine rice', 'water', 'salt', 'lemongrass'...	['in sauce pan combine the rice , water , salt...
43	2561	almond hot chocolate	Carbohydrates	NaN	NaN	['almond milk', 'bittersweet chocolate', 'unsw...	['place the chocolate and cocoa in a mixing bo...
44	4041	another pumpkin cheesecake recipe	Fat	NaN	NaN	['butter', 'granulated sugar', 'egg', 'all-pur...	['preheat oven to 400f', 'cream butter and 1 /...

11.2.1.4. Recent-Activity-based Recommender

```
def recent_activity_based(self,user_id):
    recent_df = self.recent_activity[self.recent_activity['user_id'] == user_id]
    meal_ids = list(recent_df.Meal_Id.unique())
    recent_data = self.df[self.df.Meal_Id.isin(meal_ids)][['nutrient','category','Disease','diet']].reset_index(drop=True)

    disease = []
    diet = []
    nut = []
    user_features = []

    for i in range(recent_data.shape[0]):
        for word in recent_data.loc[i,'Disease'].split():
            disease.append(word)

    for i in range(recent_data.shape[0]):
        for word in recent_data.loc[i,'diet'].split():
            diet.append(word)

    for i in range(recent_data.shape[0]):
        for word in recent_data.loc[i,'nutrient'].split():
            nut.append(word)

    nut_counts = dict(Counter(nut))
    mean_nut = np.mean(list(nut_counts.values()))
    for i in nut_counts.items():
        if i[1] > mean_nut:
            user_features.append(i[0])

    dis_counts = dict(Counter(disease))
    mean_dis = np.mean(list(dis_counts.values()))
    for i in dis_counts.items():
        if i[1] > mean_dis:
            user_features.append(i[0])


    diet_counts = dict(Counter(diet))
    mean_diet = np.mean(list(diet_counts.values()))
    for i in diet_counts.items():
        if i[1] > mean_diet:
            user_features.append(i[0])

    similar_neighbors = self.find_neighbors(user_features,10)
    return similar_neighbors.filter(['Meal_Id','name','nutrient','ingredients','steps','rating'])
```

recent_activity_based() function uses the same technique used in user_based() but the main difference is in the definition of the user_features.

In user_based recommender, we have discussed the point of measuring the similarity between the user and other users to find the most similar neighbors used in recommendation process. What we haven't discussed

is that the user_features was referring to the user's profile features which is totally different in this function recrent_activity_based().

In recrent_activity_based() function, multiple looping process calculates the features whom the user has recently interacted with more than his average interaction in this period of time, which is 30 days in this case. Then, these features is considered to be the user_features which are, then, used to find the most similar_neighbors based on.

Then, the code shown in the figure is doing the exact same thing as the remaining part of user_based recommender, except getting the user_features part, does.

```
similar_neighbors = self.find_neighbors(user_features,10)
return similar_neighbors.filter(['Meal_Id','name','nutrient','ingredients','steps','rating'])
```

To summarize this part, KNN is used to implement another user – based recommender but with different user_features.

Recommender

Now, as we have discussed each part of our recommendation system, it's time to get all parts together in one class named **Recommender** which is shown in the figure.

```
In [193]: class Recommender:

              def __init__(self,profiles,recent_activity,meals):
                  self.df = meals
                  self.profiles = profiles
                  self.recent_activity = recent_activity

              def get_features(self,dataframe):
                  # get dummies of dataframe: meals or profiles
                  nutrient_dummies = dataframe.nutrient.str.get_dummies(sep=' ')
                  disease_dummies = dataframe.Disease.str.get_dummies(sep=' ')
                  diet_dummies = dataframe.diet.str.get_dummies(sep=' ')
                  feature_df = pd.concat([nutrient_dummies,disease_dummies,diet_dummies],axis=1)

                  return feature_df
```

```
def content_based(self, user_features):

    feature_df = self.get_features(self.df)

    #initialize model with k=40 neighbors
    model = NearestNeighbors(n_neighbors=40,algorithm='ball_tree')

    # fit model with features
    model.fit(feature_df)

    # Empty dataframe to contain results
    df_results = pd.DataFrame(columns=list(self.df.columns))
    #
    total_features = list(feature_df.columns)
    #print(total_features)
    #print(user_features)
    d = dict()

    for i in total_features:
        d[i]= 0

    for i in list(user_features):
        d[i] = 1

    final_input = list(d.values())
    #print(d)
```

```
        final_input = list(d.values())
        #print(d)


        # getting distance and indices for k nearest neighbor
        distnaces , indices = model.kneighbors([final_input])

        for i in list(indices):
            df_results = df_results.append(self.df.iloc[i])

# convert zeros to ones for sample_input elements
        df_results = df_results.filter(['name', 'nutrient', 'Disease', 'diet', 'ingredients', 'steps'])
        df_results = df_results.drop_duplicates(subset=['name'])
        df_results = df_results.reset_index(drop=True)
        return df_results
```

```
    def find_neighbors(self,user_features,k):
        # dataframe = profiles
        # features = user features
        features_df = self.get_features(self.profiles)

        #initialize model with k neighbors
        model = NearestNeighbors(n_neighbors=k,algorithm='ball_tree')

        # fit model with dataset features
        model.fit(features_df)


        total_features = features_df.columns
        d = dict()
        for i in total_features:
            d[i]= 0
        for i in user_features:
            d[i] = 1
        final_input = list(d.values())
        #rint(total_features)
```

```
    total_features = features_df.columns
    d = dict()
    for i in total_features:
        d[i]= 0
    for i in user_features:
        d[i] = 1
    final_input = list(d.values())
    #rint(total_features)

    similar_neighbors = pd.DataFrame(columns=list(self.profiles.columns))

    # getting distance and indices for k nearest neighbor
    distnaces , indices = model.kneighbors([final_input])

    for i in list(indices):
        similar_neighbors = similar_neighbors.append(self.profiles.loc[i])

    similar_neighbors = similar_neighbors.reset_index(drop=True)

  #  similar_neighbors = self.k_neighbor([final_input],dataframe,k)
    return similar_neighbors
```

```
def user_based(self,user_features,user_id):


    similar_users = self.find_neighbors(user_features,10)
    users = list(similar_users.user_id)

    results = self.recent_activity[self.recent_activity.user_id.isin(users)] #taking acitivies

    results = results[results['user_id'] != user_id] # selecting those which are not reviewed by user

    meals = list(results.Meal_Id.unique())

    results = self.df[self.df.Meal_Id.isin(meals)]

    results = results.filter(['Meal_Id','name','nutrient','ingredients','steps'])

    results = results.drop_duplicates(subset=['name'])
    results = results.reset_index(drop=True)
    return results
```

```
def recent_activity_based(self,user_id):
    recent_df = self.recent_activity[self.recent_activity['user_id'] == user_id]
    meal_ids = list(recent_df.Meal_Id.unique())
    recent_data = self.df[self.df.Meal_Id.isin(meal_ids)][['nutrient','category','Disease','diet']].reset_index(drop=True)

    disease = []
    diet = []
    nut = []
    user_features = []

    for i in range(recent_data.shape[0]):
        for word in recent_data.loc[i,'Disease'].split():
            disease.append(word)

    for i in range(recent_data.shape[0]):
        for word in recent_data.loc[i,'diet'].split():
            diet.append(word)

    for i in range(recent_data.shape[0]):
        for word in recent_data.loc[i,'nutrient'].split():
            nut.append(word)

    nut_counts = dict(Counter(nut))
    mean_nut = np.mean(list(nut_counts.values()))
    for i in nut_counts.items():
        if i[1] > mean_nut:
            user_features.append(i[0])

    dis_counts = dict(Counter(disease))
    mean_dis = np.mean(list(dis_counts.values()))
    for i in dis_counts.items():
        if i[1] > mean_dis:
            user_features.append(i[0])


    diet_counts = dict(Counter(diet))
    mean_diet = np.mean(list(diet_counts.values()))
    for i in diet_counts.items():
        if i[1] > mean_diet:
            user_features.append(i[0])

    similar_neighbors = self.find_neighbors(user_features,10)
    return similar_neighbors.filter(['Meal_Id','name','nutrient','ingredients','steps','rating'])
```

```
def recommend(self,user_id):
    #finding user's profile features by id
    profile = self.profiles[self.profiles['user_id'] == user_id]
    user_features = []
    user_features.extend(profile['nutrient'].values[0].split())
    user_features.extend(profile['Disease'].values[0].split())
    user_features.extend(profile['diet'].values[0].split())

    feature_df = self.get_features(self.df)

    df0 = self.content_based(user_features)
    df1 = self.user_based(user_features,user_id)
    df2 = self.recent_activity_based(user_id)
    #df3 = self.k_neighbor(inputs,feature_df,dataframe,k)

    df = pd.concat([df0,df1,df2])

    df = df.drop_duplicates('ingredients').reset_index(drop=True)
    return df
```

recommend() function is implemented to interact with user and get only the user_id to make personalized recommendations using multiple models and appending their results together in one dataframe (df).

Before returning the results, all duplicates are dropped to avoid repeating recommendations.

Finally, we have successfully built our recommendation system based on three models:

- Content – based
- User – based
- Recent – Activity – based

Now, it's time to test out system!

In [194]:
```
user_id = 'user_71'  # user id of current user

profiles = pd.read_csv('E:\\graduation_project\\diet rec system\\user_profiles.csv') # profiles of all users
recent_activity = pd.read_csv('E:\\graduation_project\\diet rec system\\recent_activity.csv') # recent activities of current user
dataset = pd.read_csv('E:\\graduation_project\\diet rec system\\nut_df.csv') # main dataset


example = Recommender(profiles,recent_activity,dataset)
result = example.recommend(user_id)
result
```

Out[194]:

	name	nutrient	Disease	diet	ingredients	steps	Meal_Id
0	annacia s spice islands hot chocolate lighter	Carbohydrates	diabeties obesity	low_fat_diet	['unsweetened chocolate', 'cocoa', 'skim milk'...	['melt chocolate in 1 cup of skim milk , over ...	NaN
1	baked snapper chinoise	Carbohydrates	diabeties obesity	low_fat_diet	['light soy sauce', 'minced garlic clove', 'gi...	['preheat oven to 450f', 'combine soy sauce , ...	NaN
2	another easy pizza sauce	Carbohydrates	diabeties obesity	low_fat_diet	['whole tomatoes', 'olive oil', 'salt', 'red p...	['crush the tomatoes well', 'do not drain', 's...	NaN
3	baked seasoned beet crisps	Carbohydrates	diabeties obesity	low_fat_diet	['beets', 'balsamic vinegar', 'sugar', 'allspi...	['preheat oven to 375 degrees', 'peel beets an...	NaN
4	lindsey s zucchini squash veggie chili	Carbohydrates	diabeties obesity	low_fat_diet	['olive oil', 'yellow onion', 'garlic cloves',...	['heat oil in a large pot over medium heat', '...	NaN
5	baked rigaboney	Carbohydrates	diabeties obesity	low_fat_diet	['olive oil', 'garlic cloves', 'onion', 'tomat...	['preheat oven to 350 degrees', 'in large sauc...	NaN
6	louisiana garlic bread	Carbohydrates	diabeties obesity	low_fat_diet	['unsalted butter', 'garlic', 'fresh lemon jui...	['mix all ingredients well', 'spread butter mi...	NaN
7	anthony s french bread pizza	Carbohydrates	diabeties	low_fat_diet	['french bread', 'pizza sauce',	['slice the french bread lengthwise in	NaN

11.3. Put all together (Stream lit API)

After finishing the implementation of the two models: Diabetes detection and Recommendation system, we think how to make the two models in one application

We search a lot for how to do that and we find it possible with streamlit.

Now let's see how we do this .

First, we import the libraries we need in our project using command import as shown.

```
import streamlit as st
import pandas as pd
import joblib
import warnings
warnings.filterwarnings('ignore')
from PIL import Image
import Final_Recommender
from Final_Recommender import Recommender
```

Import Streamlit library as st to make our application.

Import pandas as pd: we import pandas as we deal with datasets in table form so we use pandas that is good with table datasets.

Import joblib library to load the two models after saving in jupyter notebook.

Import PIL library to load image into our application.

Import Recommender function from final Recommender class in the recommendation model.

The code of the application consists of 203 line of code which consist of four functions for simplicity we will take about each function and how it works.

```
def home():
    st.title('Homepage')

    image =Image.open('diabetes-report-card-SM.jpg')
    st.image(image,use_column_width=True)
    st.subheader('Type 2 diabetes')


    cont1 =st.container()
    cont1.info("Type 2 diabetes is the most common type of diabetes, accounting for around 90% of all diabetes ca
    cont1.info("""It is generally characterized by insulin resistance, where the body does not fully respond to
                Because insulin cannot work properly, blood glucose levels keep rising, releasing more insulin.

                """)
    cont1.info(""" Type 2 diabetes is most commonly diagnosed in older adults, but is increasingly seen in child
    st.subheader('Risk factors')

    cont2 =st.container()
    cont2.info("Several risk factors have been associated with type 2 diabetes and include:")

    image2 =Image.open('first.png')
```

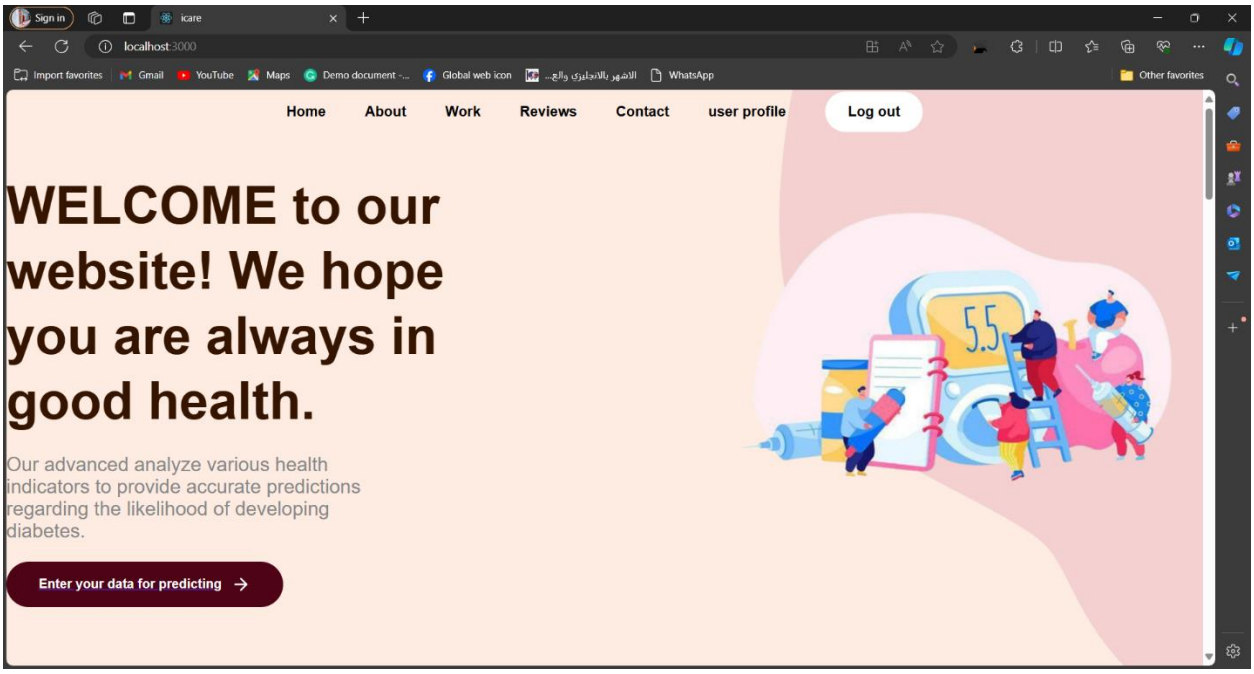
```
            """)

    st.subheader("Symptoms of type 2 diabetes")
    cont3=st.container()
    cont3.info("The symptoms of type 2 diabetes are similar to those of type 1 diabetes and include:")

    image3 =Image.open('sec.png')
    cont3.image(image3,use_column_width=True)

    cont3.info("""

                Excessive thirst and dry mouth.

                Frequent urination.

                Lack of energy, tiredness.

                Slow healing wounds.

                Recurrent infections in the skin.

                Blurred vision.
```

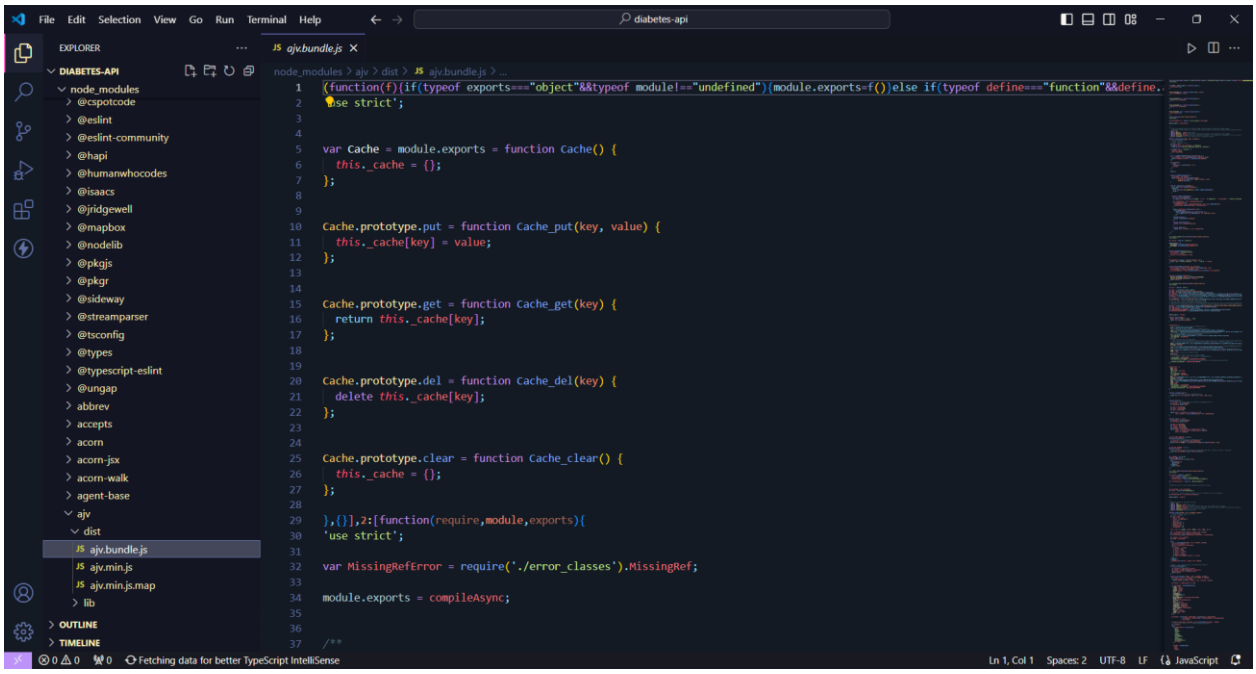
```
st.subheader("Medications for type 2 diabetes")
cont6=st.container()
cont6.info("The most commonly used oral medications for type 2 diabetes include:")
cont6.info("""

          Metformin: reduces insulin resistance and allows the body to use its own insulin more effectively. It is regarded as th

          Sulfonylureas: stimulate the pancreas to increase insulin production. Sulfonylureas include gliclazide, glipizide, glim


          """)


st.success("Are you ready for Daiabetes check?  ... Let's try the app " )
```

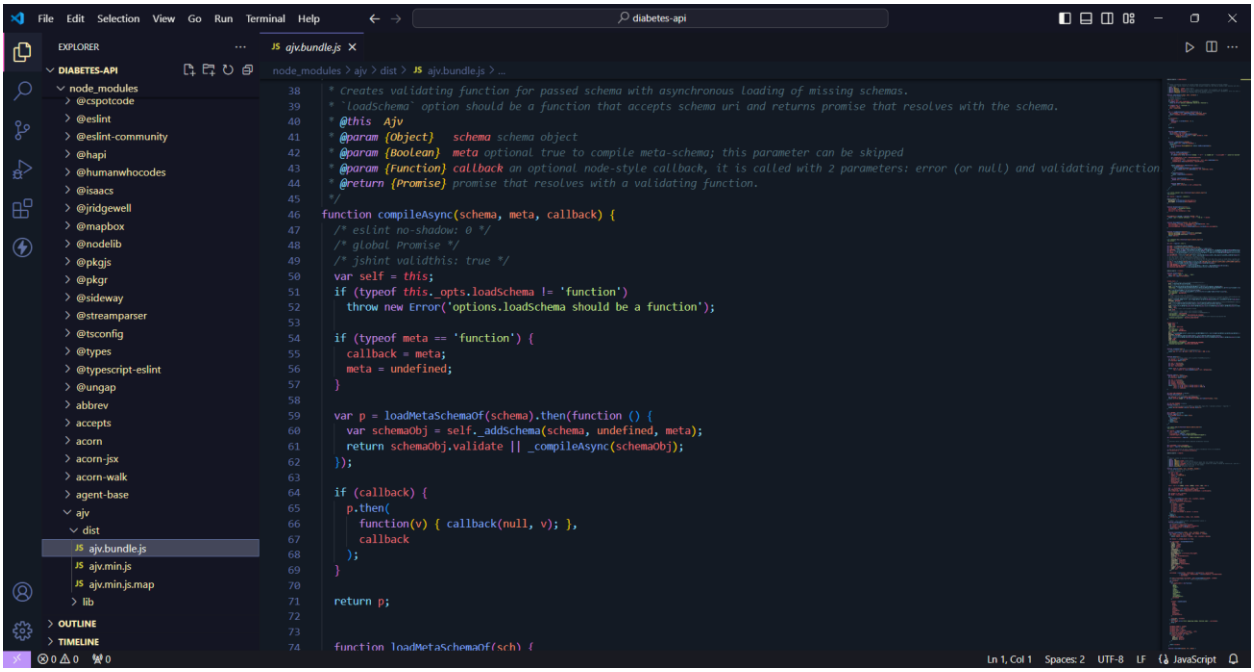
rs, symptoms, prevention management and medications we use six containers to implement those sections with related images for each one.

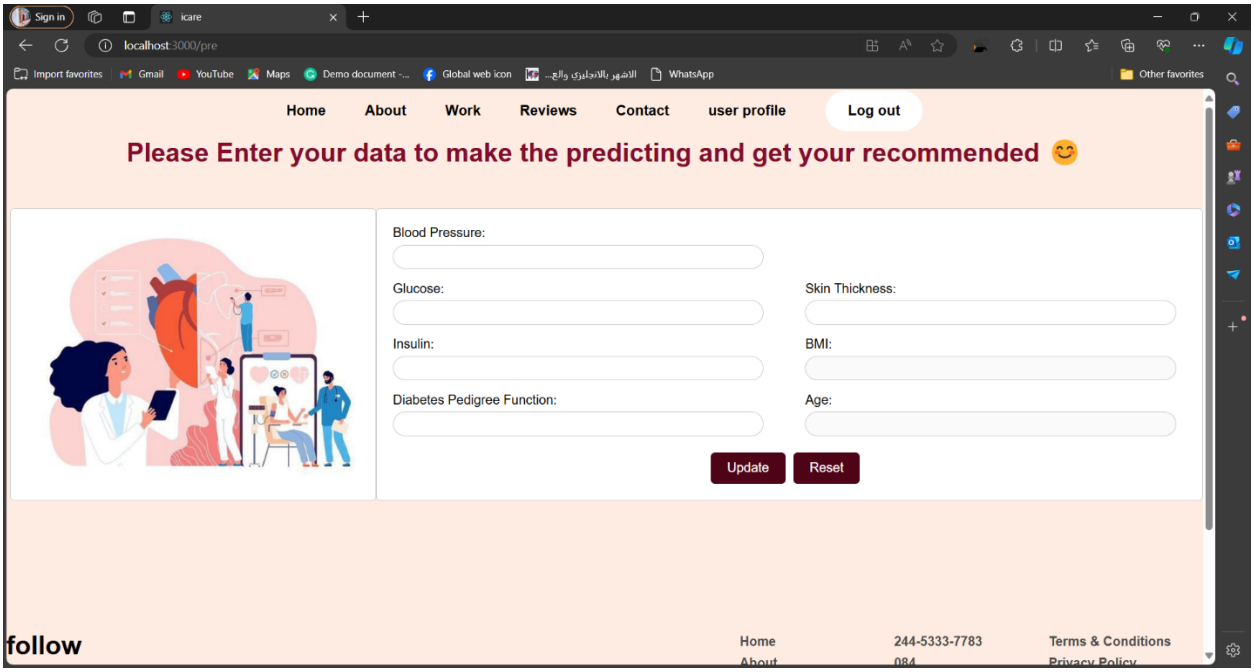
To make user aware with this dangerous illness.

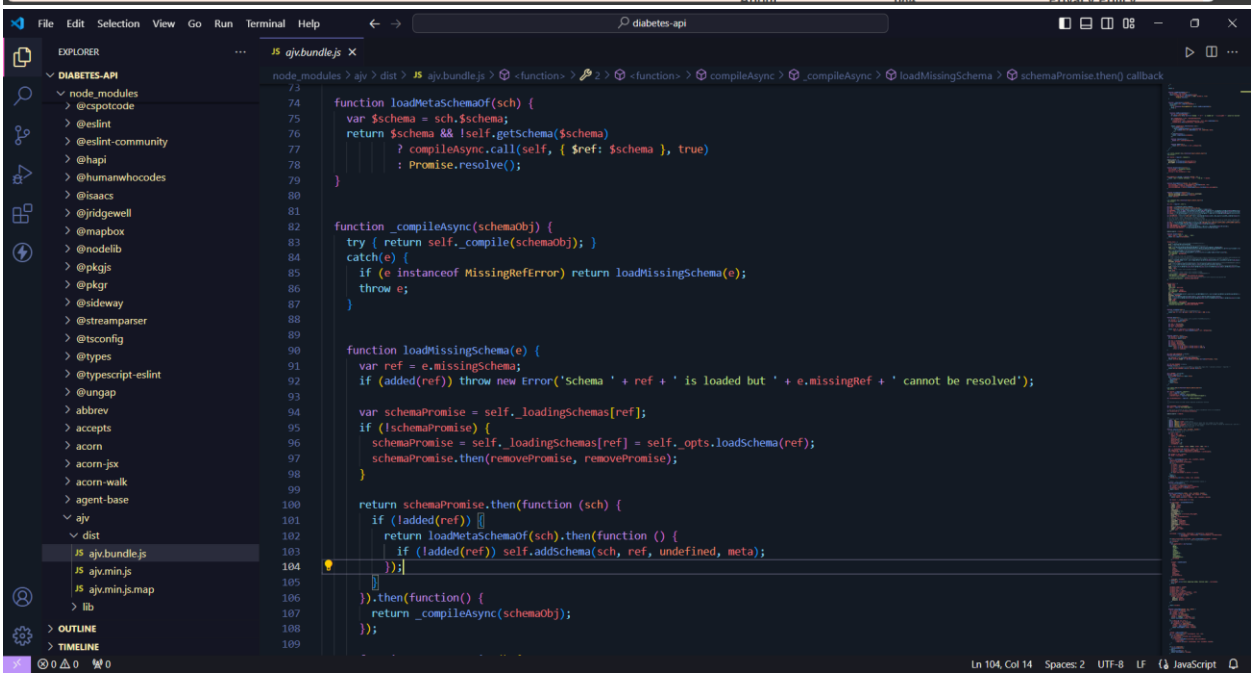
12.System Installation

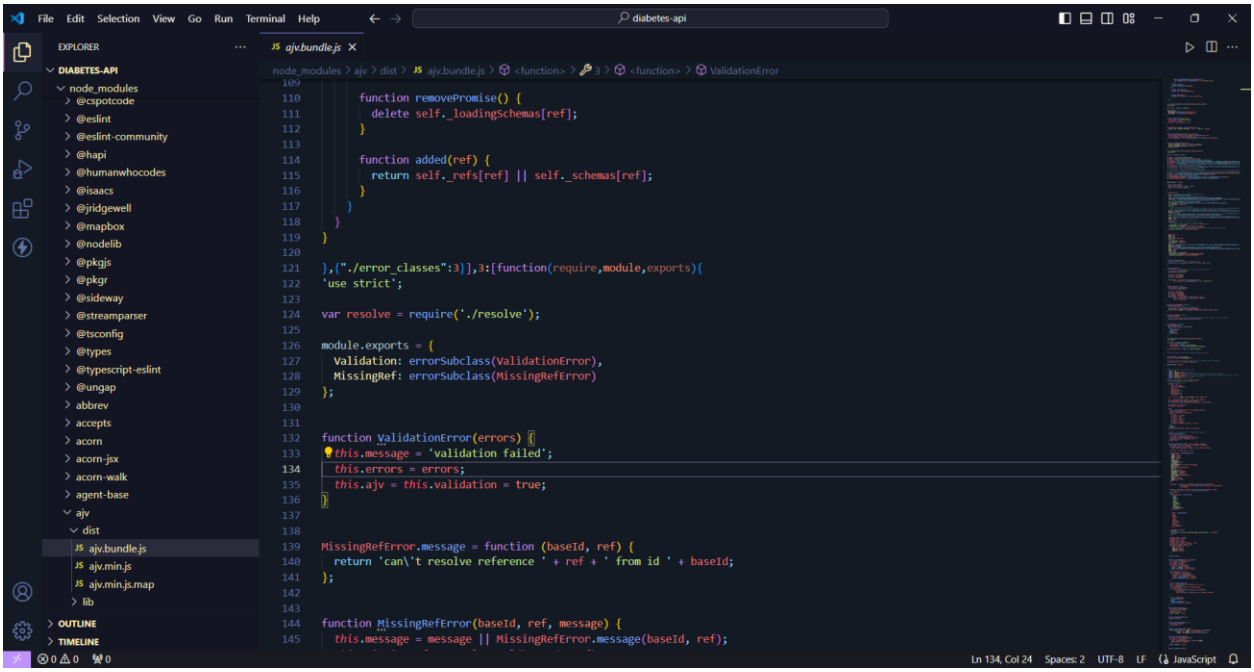


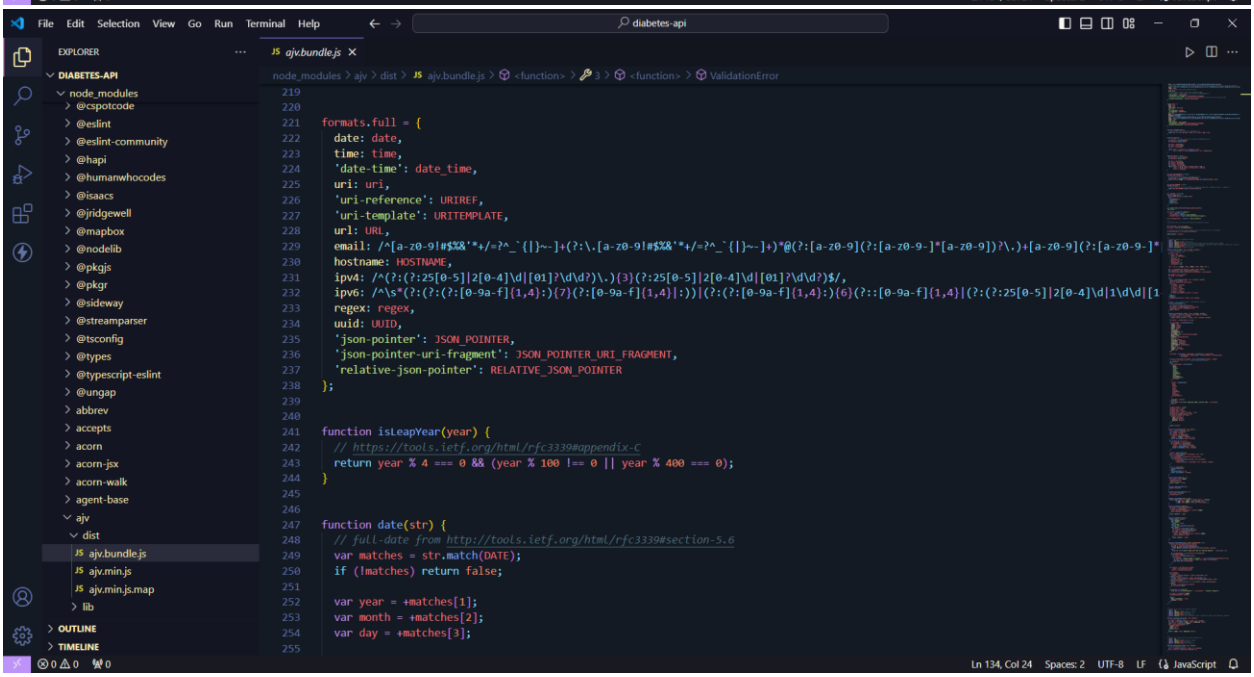


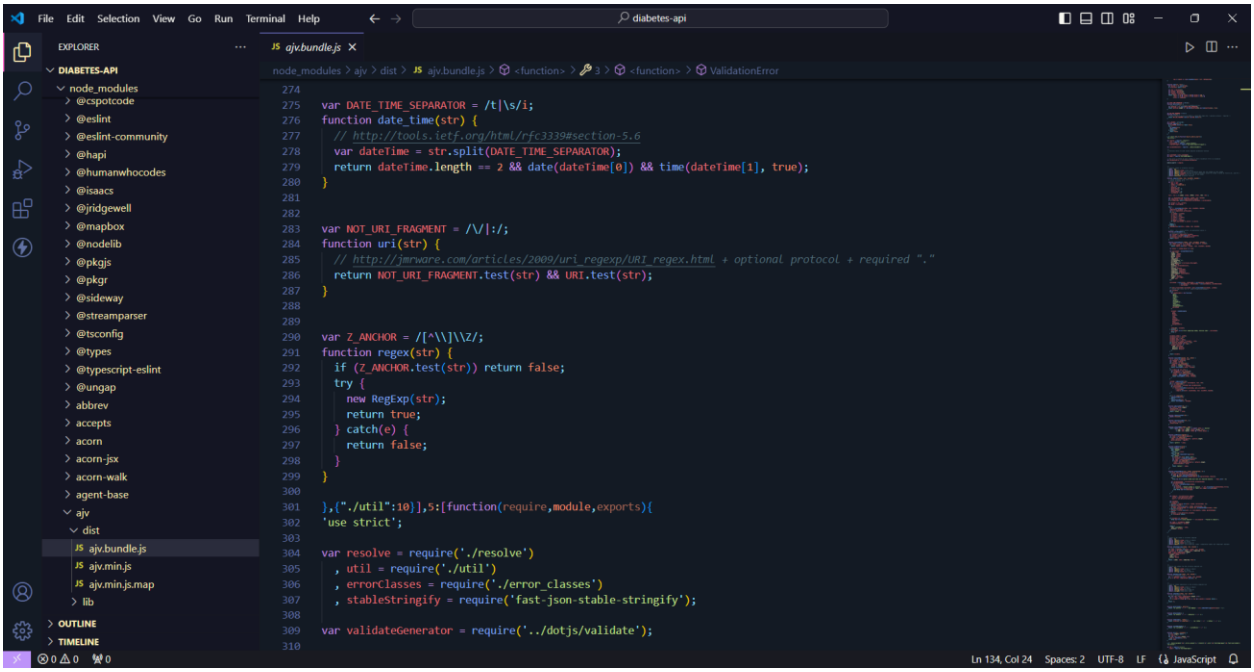

File Edit Selection View Go Run Terminal Help
diabetes-api
EXPLORER
ajv.bundle.js
DIABETES-API
node_modules
@cspotcode
@eslint
@eslint-community
@hapi
@humanwhocodes
@isaacs
@jridgewell
@mapbox
@nodelib
@pkgjs
@pkgr
@sideway
@streamparser
@tsconfig
@types
@typescript-eslint
@ungap
abbrev
accepts
acorn
acorn-jsx
acorn-walk
agent-base
ajv
dist
ajv.bundle.js
ajv.min.js
ajv.min.js.map
lib
OUTLINE
TIMELINE
node_modules > ajv > dist > ajv.bundle.js > ...
* Creates validating function for passed schema with asynchronous loading of missing schemas.
* `loadSchema` option should be a function that accepts schema uri and returns promise that resolves with the schema.
* @this Ajv
* @param {Object} schema schema object
* @param {Boolean} meta optional true to compile meta-schema; this parameter can be skipped
* @param {Function} callback an optional node-style callback, it is called with 2 parameters: error (or null) and validating function
* @return {Promise} promise that resolves with a validating function.
function compileAsync(schema, meta, callback) {
/* eslint no-shadow: 0 */
/* global Promise */
/* jshint validthis: true */
var self = this;
if (typeof this._opts.loadSchema != 'function')
throw new Error('options.loadSchema should be a function');
if (typeof meta == 'function') {
callback = meta;
meta = undefined;
}
var p = loadMetaSchemaOf(schema).then(function () {
var schemaObj = self._addSchema(schema, undefined, meta);
return schemaObj.validate || _compileAsync(schemaObj);
});
if (callback) {
p.then(
function(v) { callback(null, v); },
callback
);
}
return p;
function loadMetaSchemaOf(sch) {
Ln 1, Col 1
Spaces: 2
UTF-8
LF
JavaScript

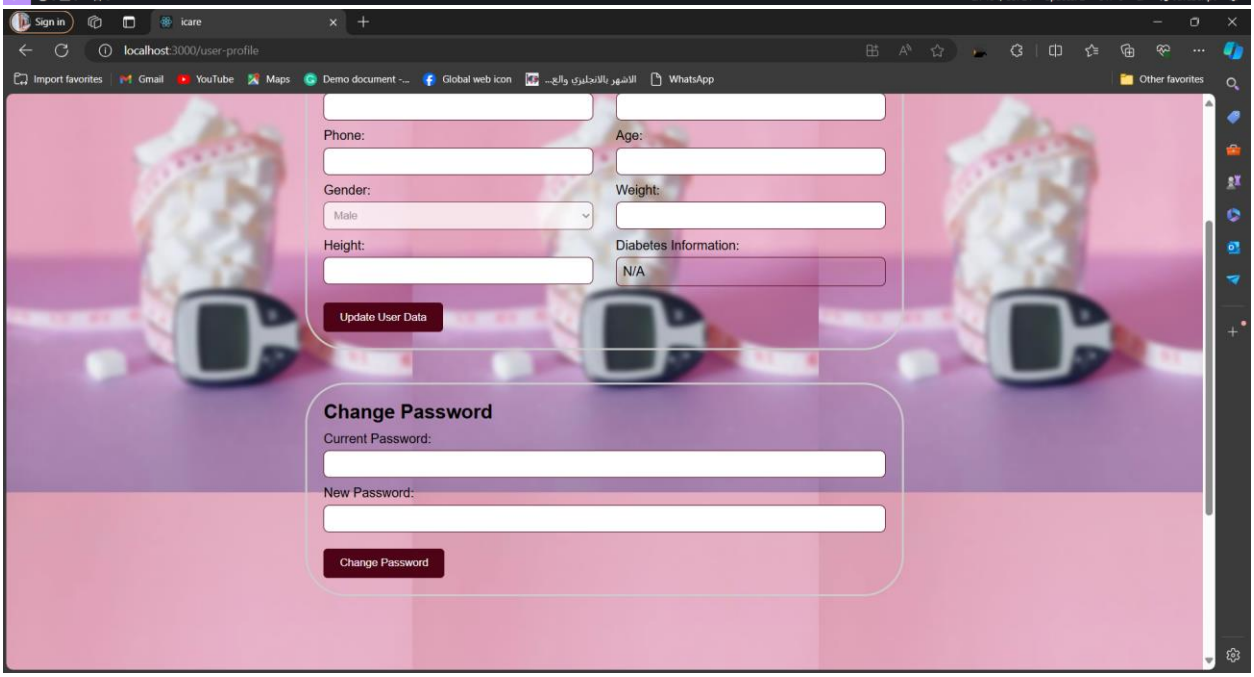

Home
About
Work
Reviews
Contact
user profile
Log out
Please Enter your data to make the predicting and get your recommended
Blood Pressure:
Glucose:
Skin Thickness:
Insulin:
BMI:
Diabetes Pedigree Function:
Age:
Update
Reset
follow
Home
About
244-5333-7783
Terms & Conditions
Privacy Policy

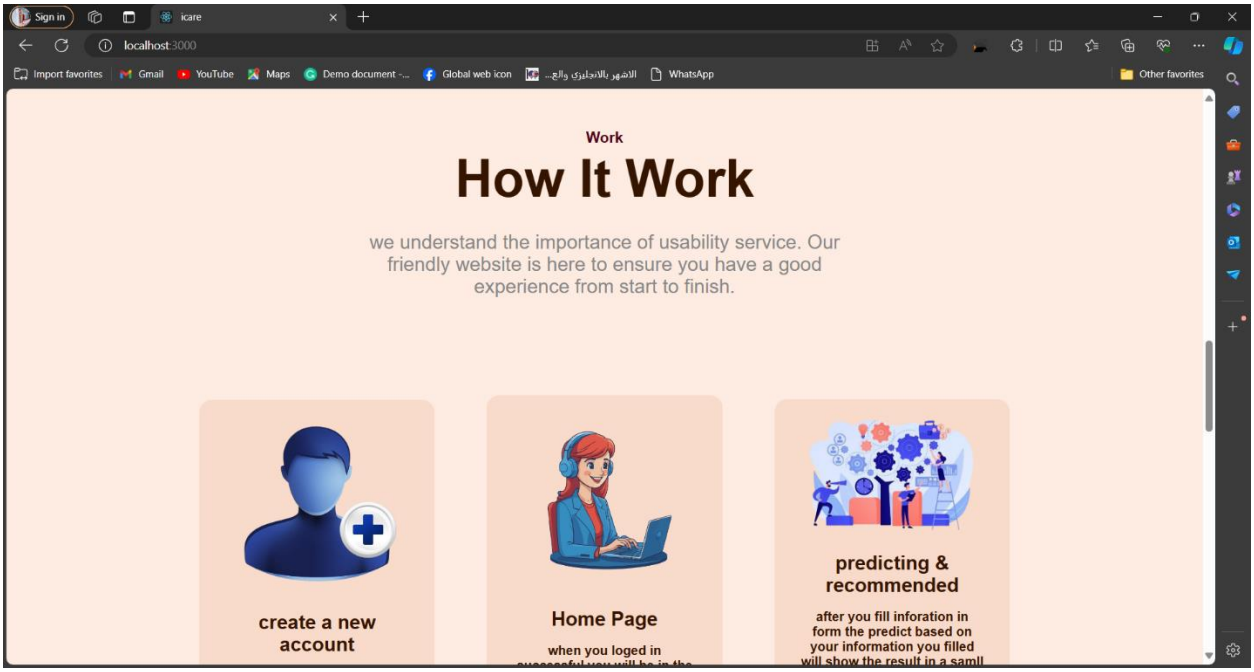

function loadMetaSchemaOf(sch) {
var $schema = sch.$schema;
return $schema && !self.getSchema($schema)
? compileAsync.call(self, { $ref: $schema }, true)
: Promise.resolve();
}
function _compileAsync(schemaObj) {
try { return self._compile(schemaObj); }
catch(e) {
if (e instanceof MissingRefError) return loadMissingSchema(e);
throw e;
}
function loadMissingSchema(e) {
var ref = e.missingSchema;
if (added(ref)) throw new Error('Schema ' + ref + ' is loaded but ' + e.missingRef + ' cannot be resolved');
var schemaPromise = self._loadingSchemas[ref];
if (!schemaPromise) {
schemaPromise = self._loadingSchemas[ref] = self._opts.loadSchema(ref);
schemaPromise.then(removePromise, removePromise);
}
return schemaPromise.then(function (sch) {
if (!added(ref)) {
return loadMetaSchemaOf(sch).then(function () {
if (!added(ref)) self.addSchema(sch, ref, undefined, meta);
});
}
}).then(function() {
return _compileAsync(schemaObj);
});

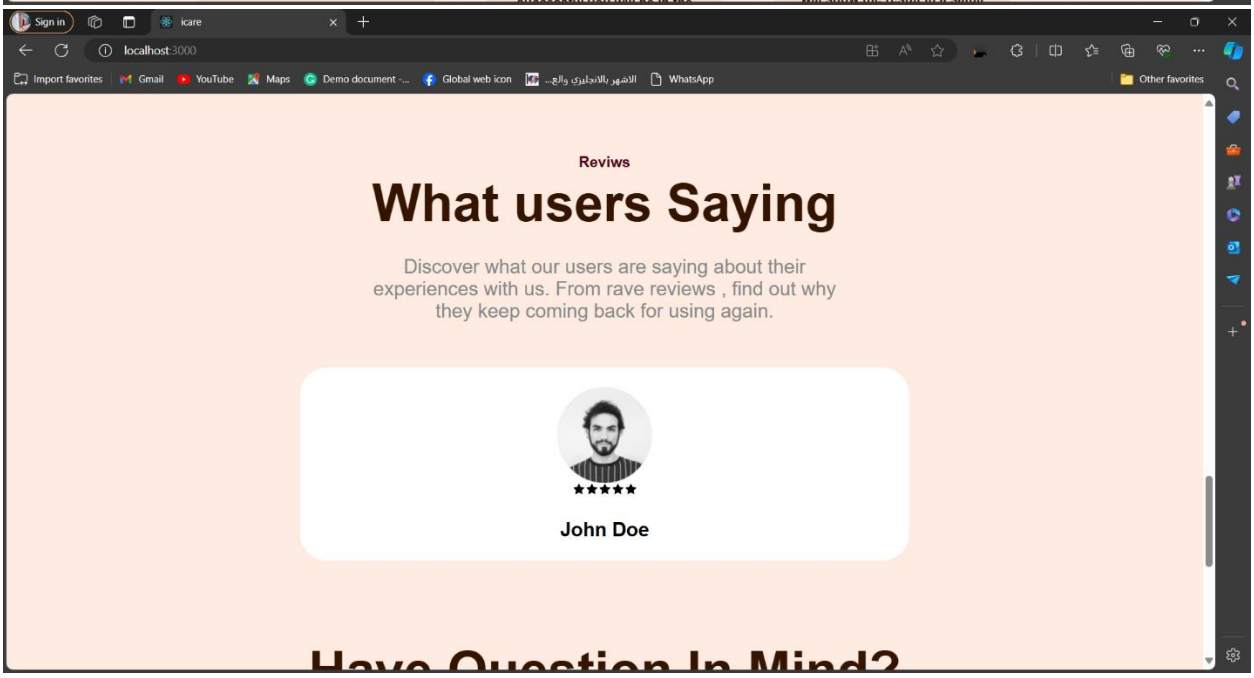

File Edit Selection View Go Run Terminal Help
diabetes-api
EXPLORER
DIABETES-API
node_modules
@cspotcode
@eslint
@eslint-community
@hapi
@humanwhocodes
@isaacs
@jridgewell
@mapbox
@nodelib
@pkgjs
@pkgr
@sideway
@streamparser
@tsconfig
@types
@typescript-eslint
@ungap
abbrev
accepts
acorn
acorn-jsx
acorn-walk
agent-base
ajv
dist
ajv.bundle.js
ajv.min.js
ajv.min.js.map
lib
OUTLINE
TIMELINE
ajv.bundle.js
node_modules > ajv > dist > ajv.bundle.js > <function> > 3 > <function> > ValidationError
function removePromise() {
delete self._loadingSchemas[ref];
}
function added(ref) {
return self._refs[ref] || self._schemas[ref];
}
}
}
},{"./error_classes":3}],3:[function(require,module,exports){
'use strict';
var resolve = require('./resolve');
module.exports = {
Validation: errorSubclass(ValidationError),
MissingRef: errorSubclass(MissingRefError)
};
function ValidationError(errors) {
this.message = 'validation failed';
this.errors = errors;
this.ajv = this.validation = true;
}
MissingRefError.message = function (baseId, ref) {
return 'can\'t resolve reference ' + ref + ' from id ' + baseId;
};
function MissingRefError(baseId, ref, message) {
this.message = message || MissingRefError.message(baseId, ref);
Ln 134, Col 24
Spaces: 2
UTF-8
LF
JavaScript

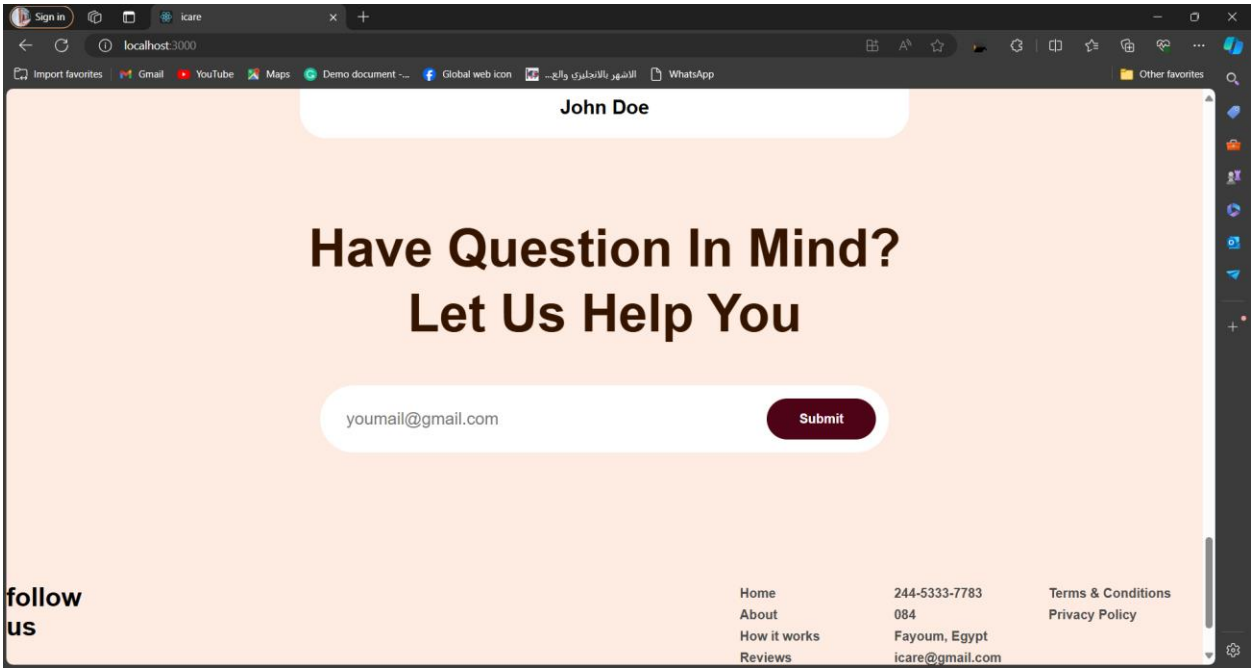

File Edit Selection View Go Run Terminal Help
diabetes-api
EXPLORER
DIABETES-API
node_modules
@cspotcode
@eslint
@eslint-community
@hapi
@humanwhocodes
@isaacs
@jridgewell
@mapbox
@nodelib
@pkgjs
@pkgr
@sideway
@streamparser
@tsconfig
@types
@typescript-eslint
@ungap
abbrev
accepts
acorn
acorn-jsx
acorn-walk
agent-base
ajv
dist
ajv.bundle.js
ajv.min.js
ajv.min.js.map
lib
OUTLINE
TIMELINE
ajv.bundle.js
node_modules > ajv > dist > ajv.bundle.js > <function> > 3 > <function> > ValidationError
formats.full = {
date: date,
time: time,
'date-time': date_time,
uri: uri,
'uri-reference': URIREF,
'uri-template': URITEMPLATE,
url: URL,
email: /^[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*
hostname: HOSTNAME,
ipv4: /^(?:(?:25[0-5]|2[0-4]\d|[01]?\d\d?)\.){3}(?:25[0-5]|2[0-4]\d|[01]?\d\d?)$/,
ipv6: /^\s*(?:(?:(?:[0-9a-f]{1,4}:){7}(?:[0-9a-f]{1,4}|:))|(?:(?:[0-9a-f]{1,4}:){6}(?::[0-9a-f]{1,4}|(?:(?:25[0-5]|2[0-4]\d|1\d\d|[1
regex: regex,
uuid: UUID,
'json-pointer': JSON_POINTER,
'json-pointer-uri-fragment': JSON_POINTER_URI_FRAGMENT,
'relative-json-pointer': RELATIVE_JSON_POINTER
};
function isLeapYear(year) {
// https://tools.ietf.org/html/rfc3339#appendix-C
return year % 4 === 0 && (year % 100 !== 0 || year % 400 === 0);
}
function date(str) {
// full-date from http://tools.ietf.org/html/rfc3339#section-5.6
var matches = str.match(DATE);
if (!matches) return false;
var year = +matches[1];
var month = +matches[2];
var day = +matches[3];
Ln 134, Col 24
Spaces: 2
UTF-8
LF
JavaScript

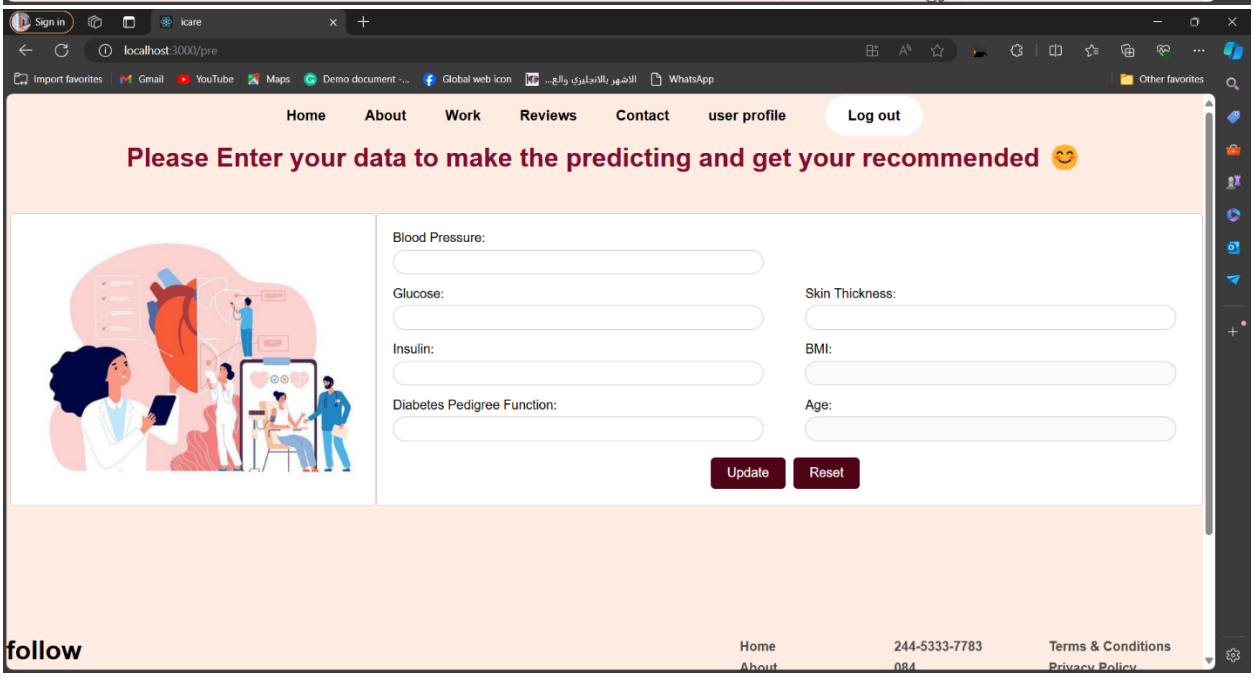

File Edit Selection View Go Run Terminal Help
diabetes-api
EXPLORER
ajv.bundle.js
DIABETES-API
node_modules
@cspotcode
@eslint
@eslint-community
@hapi
@humanwhocodes
@isaacs
@jridgewell
@mapbox
@nodelib
@pkgjs
@pkgr
@sideway
@streamparser
@tsconfig
@types
@typescript-eslint
@ungap
abbrev
accepts
acorn
acorn-jsx
acorn-walk
agent-base
ajv
dist
ajv.bundle.js
ajv.min.js
ajv.min.js.map
lib
OUTLINE
TIMELINE
var DATE_TIME_SEPARATOR = /t|\s/i;
function date_time(str) {
// http://tools.ietf.org/html/rfc3339#section-5.6
var dateTime = str.split(DATE_TIME_SEPARATOR);
return dateTime.length == 2 && date(dateTime[0]) && time(dateTime[1], true);
}
var NOT_URI_FRAGMENT = /\/|:/;
function uri(str) {
// http://jmrware.com/articles/2009/uri_regexp/URI_regex.html + optional protocol + required "."
return NOT_URI_FRAGMENT.test(str) && URI.test(str);
}
var Z_ANCHOR = /[^\\]\\Z/;
function regex(str) {
if (Z_ANCHOR.test(str)) return false;
try {
new RegExp(str);
return true;
} catch(e) {
return false;
}
}
},{"./util":10}],5:[function(require,module,exports){
'use strict';
var resolve = require('./resolve')
, util = require('./util')
, errorClasses = require('./error_classes')
, stableStringify = require('fast-json-stable-stringify');
var validateGenerator = require('../dotjs/validate');
Ln 134, Col 24
Spaces: 2
UTF-8
LF
JavaScript

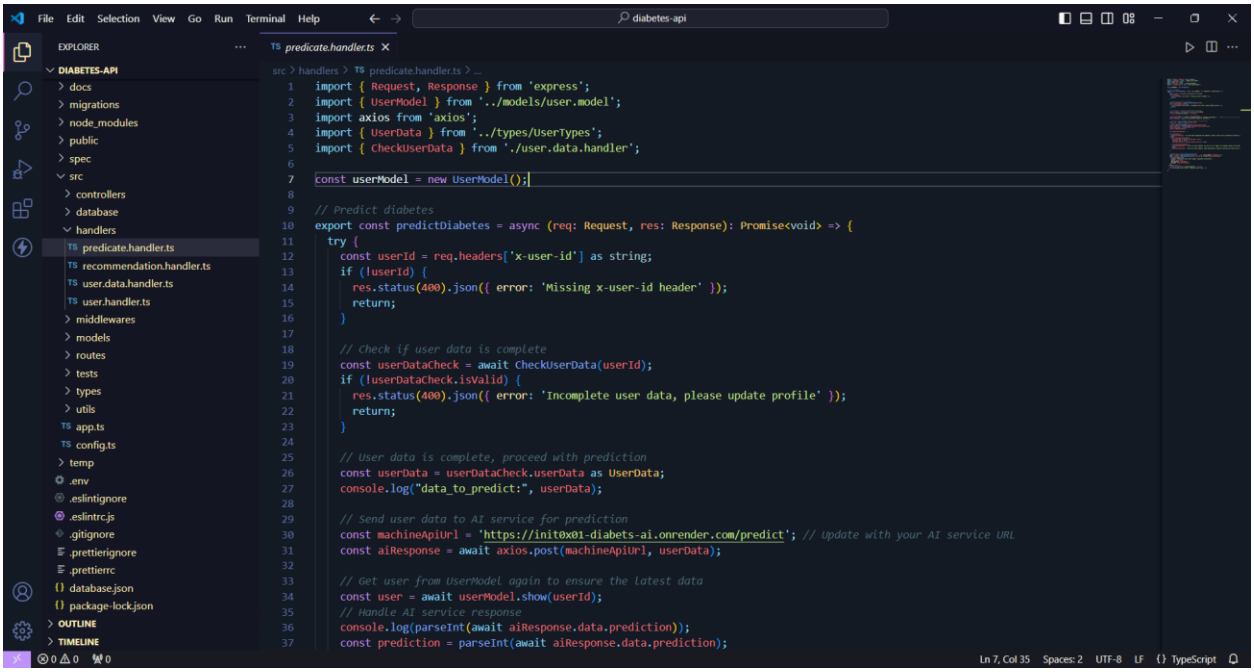

Sign in
icare
localhost:3000/user-profile
Import favorites
Gmail
YouTube
Maps
Demo document -...
Global web icon
WhatsApp
Other favorites
Phone:
Age:
Gender:
Male
Weight:
Height:
Diabetes Information:
N/A
Update User Data
Change Password
Current Password:
New Password:
Change Password

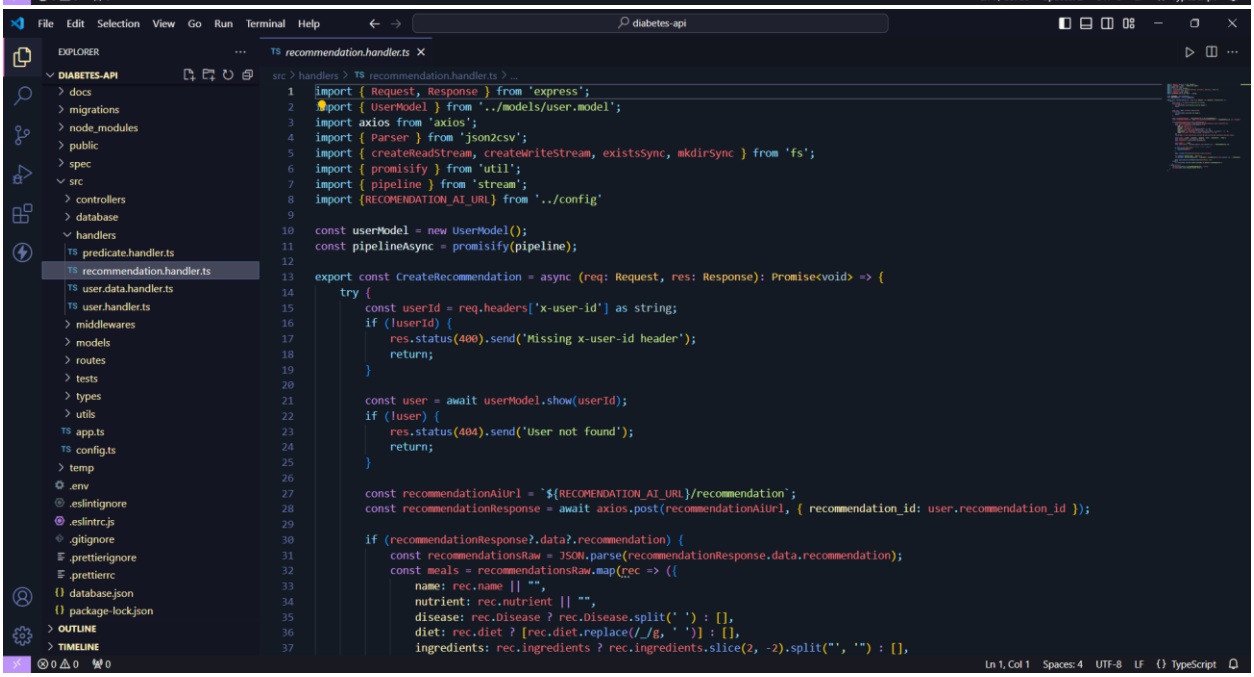

Work
How It Work
we understand the importance of usability service. Our friendly website is here to ensure you have a good experience from start to finish.
create a new account
Home Page
when you loged in
predicting & recommended
after you fill inforation in form the predict based on your information you filled

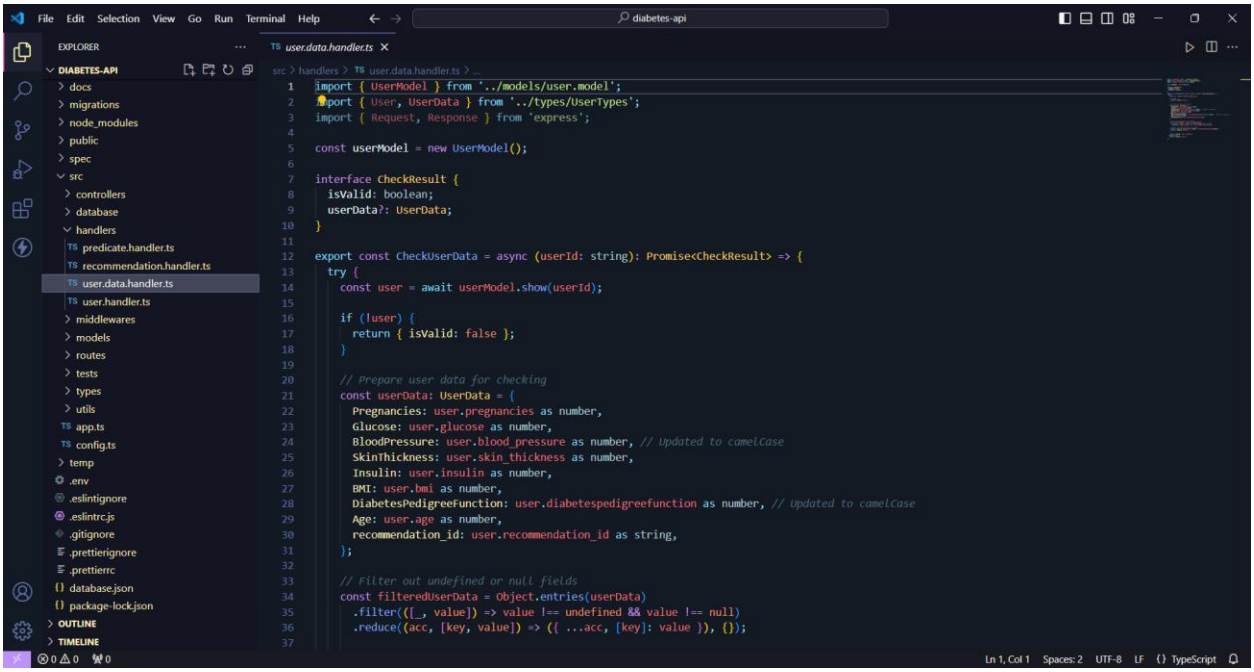

Reviws
What users Saying
Discover what our users are saying about their experiences with us. From rave reviews , find out why they keep coming back for using again.
John Doe

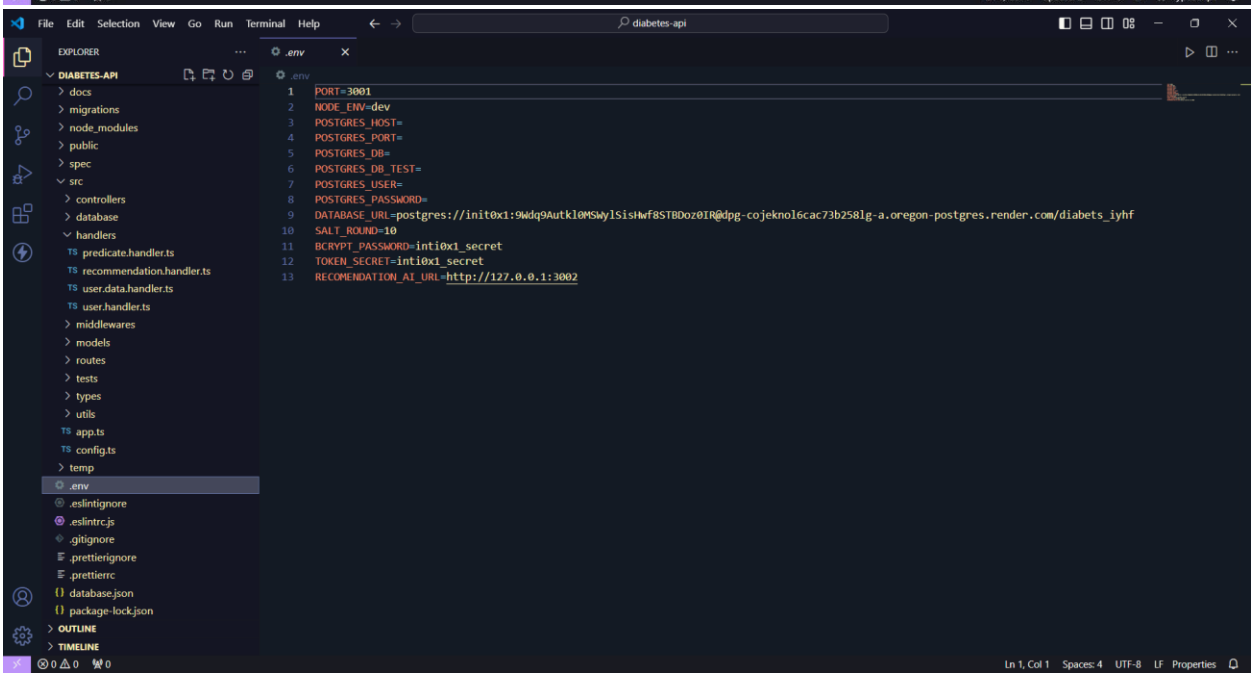

Sign in
icare
localhost:3000
Import favorites
Gmail
YouTube
Maps
Demo document -...
Global web icon
WhatsApp
Other favorites
John Doe
Have Question In Mind?
Let Us Help You
youmail@gmail.com
Submit
follow
us
Home
About
How it works
Reviews
244-5333-7783
084
Fayoum, Egypt
icare@gmail.com
Terms & Conditions
Privacy Policy

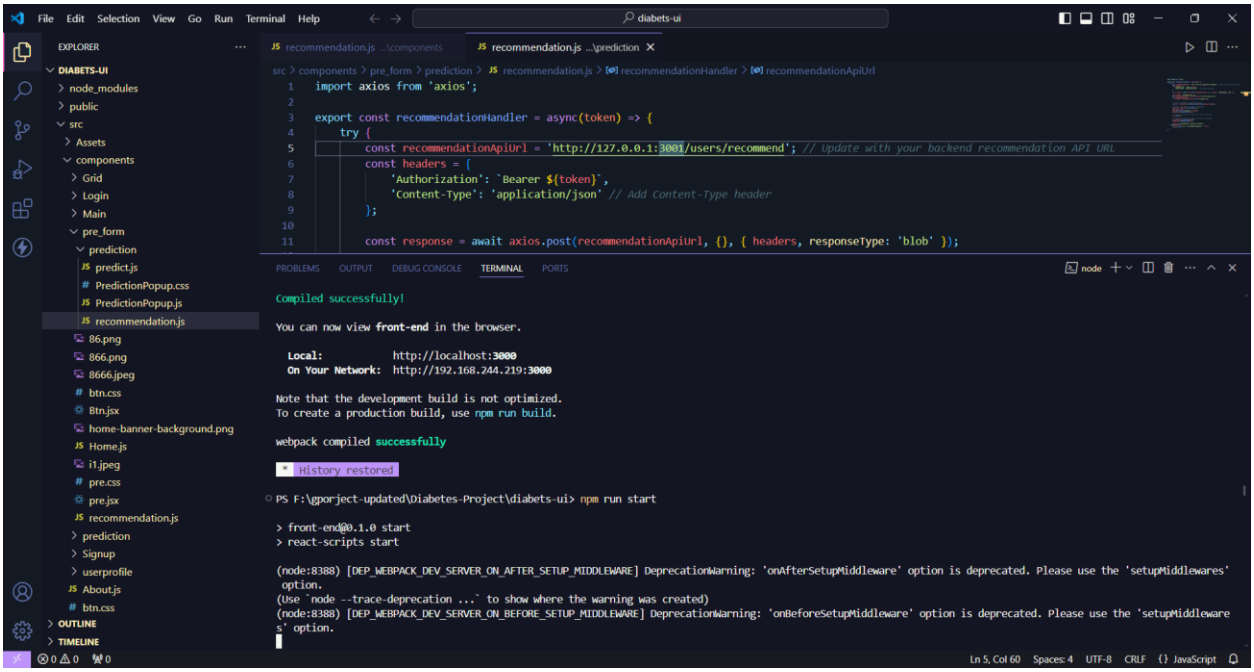

Sign in
icare
localhost:3000/pre
Import favorites
Gmail
YouTube
Maps
Demo document -...
Global web icon
WhatsApp
Other favorites
Home
About
Work
Reviews
Contact
user profile
Log out
Please Enter your data to make the predicting and get your recommended 😊
Blood Pressure:
Glucose:
Skin Thickness:
Insulin:
BMI:
Diabetes Pedigree Function:
Age:
Update
Reset
follow
Home
244-5333-7783
Terms & Conditions


File Edit Selection View Go Run Terminal Help
diabetes-api
EXPLORER
DIABETES-API
predicate.handler.ts
src > handlers > predicate.handler.ts > ...
import { Request, Response } from 'express';
import { UserModel } from '../models/user.model';
import axios from 'axios';
import { UserData } from '../types/UserTypes';
import { CheckUserData } from './user.data.handler';
const userModel = new UserModel();
// Predict diabetes
export const predictDiabetes = async (req: Request, res: Response): Promise<void> => {
try {
const userId = req.headers['x-user-id'] as string;
if (!userId) {
res.status(400).json({ error: 'Missing x-user-id header' });
return;
}
// Check if user data is complete
const userDataCheck = await CheckUserData(userId);
if (!userDataCheck.isValid) {
res.status(400).json({ error: 'Incomplete user data, please update profile' });
return;
}
// User data is complete, proceed with prediction
const userData = userDataCheck.userData as UserData;
console.log("data_to_predict:", userData);
// Send user data to AI service for prediction
const machineApiUrl = 'https://init0x01-diabets-ai.onrender.com/predict'; // Update with your AI service URL
const aiResponse = await axios.post(machineApiUrl, userData);
// Get user from UserModel again to ensure the latest data
const user = await userModel.show(userId);
// Handle AI service response
console.log(parseInt(await aiResponse.data.prediction));
const prediction = parseInt(await aiResponse.data.prediction);
Ln 7, Col 35 Spaces: 2 UTF-8 LF {} TypeScript


File Edit Selection View Go Run Terminal Help
diabetes-api
EXPLORER
DIABETES-API
recommendation.handler.ts
src > handlers > recommendation.handler.ts > ...
import { Request, Response } from 'express';
import { UserModel } from '../models/user.model';
import axios from 'axios';
import { Parser } from 'json2csv';
import { createReadStream, createWriteStream, existsSync, mkdirSync } from 'fs';
import { promisify } from 'util';
import { pipeline } from 'stream';
import {RECOMENDATION_AI_URL} from '../config'
const userModel = new UserModel();
const pipelineAsync = promisify(pipeline);
export const CreateRecommendation = async (req: Request, res: Response): Promise<void> => {
try {
const userId = req.headers['x-user-id'] as string;
if (!userId) {
res.status(400).send('Missing x-user-id header');
return;
}
const user = await userModel.show(userId);
if (!user) {
res.status(404).send('User not found');
return;
}
const recommendationAiUrl = `${RECOMENDATION_AI_URL}/recommendation`;
const recommendationResponse = await axios.post(recommendationAiUrl, { recommendation_id: user.recommendation_id });
if (recommendationResponse?.data?.recommendation) {
const recommendationsRaw = JSON.parse(recommendationResponse.data.recommendation);
const meals = recommendationsRaw.map(rec => ({
name: rec.name || "",
nutrient: rec.nutrient || "",
disease: rec.Disease ? rec.Disease.split(' ') : [],
diet: rec.diet ? [rec.diet.replace(/_/g, ' ')] : [],
ingredients: rec.ingredients ? rec.ingredients.slice(2, -2).split("', '") : [],
Ln 1, Col 1 Spaces: 4 UTF-8 LF {} TypeScript


File Edit Selection View Go Run Terminal Help
diabetes-api
EXPLORER
DIABETES-API
user.data.handler.ts
src > handlers > user.data.handler.ts
import { UserModel } from '../models/user.model';
import { User, UserData } from '../types/UserTypes';
import { Request, Response } from 'express';
const userModel = new UserModel();
interface CheckResult {
isValid: boolean;
userData?: UserData;
}
export const CheckUserData = async (userId: string): Promise<CheckResult> => {
try {
const user = await userModel.show(userId);
if (!user) {
return { isValid: false };
}
// Prepare user data for checking
const userData: UserData = {
Pregnancies: user.pregnancies as number,
Glucose: user.glucose as number,
BloodPressure: user.blood_pressure as number, // Updated to camelCase
SkinThickness: user.skin_thickness as number,
Insulin: user.insulin as number,
BMI: user.bmi as number,
DiabetesPedigreeFunction: user.diabetespedigreefunction as number, // Updated to camelCase
Age: user.age as number,
recommendation_id: user.recommendation_id as string,
};
// Filter out undefined or null fields
const filteredUserData = Object.entries(userData)
.filter(([_, value]) => value !== undefined && value !== null)
.reduce((acc, [key, value]) => ({ ...acc, [key]: value }), {});
docs
migrations
node_modules
public
spec
src
controllers
database
handlers
predicate.handler.ts
recommendation.handler.ts
user.data.handler.ts
user.handler.ts
middlewares
models
routes
tests
types
utils
app.ts
config.ts
temp
.env
.eslintignore
.eslintrc.js
.gitignore
.prettierignore
.prettierrc
database.json
package-lock.json
OUTLINE
TIMELINE
Ln 1, Col 1 Spaces: 2 UTF-8 LF {} TypeScript


File Edit Selection View Go Run Terminal Help
diabetes-api
EXPLORER
DIABETES-API
.env
PORT=3001
NODE_ENV=dev
POSTGRES_HOST=
POSTGRES_PORT=
POSTGRES_DB=
POSTGRES_DB_TEST=
POSTGRES_USER=
POSTGRES_PASSWORD=
DATABASE_URL=postgres://init0x1:9Wdq9Autkl0MSWylSisHwf8STBDoz0IR@dpg-cojeknol6cac73b258lg-a.oregon-postgres.render.com/diabets_iyhf
SALT_ROUND=10
BCRYPT_PASSWORD=inti0x1_secret
TOKEN_SECRET=inti0x1_secret
RECOMENDATION_AI_URL=http://127.0.0.1:3002
docs
migrations
node_modules
public
spec
src
controllers
database
handlers
predicate.handler.ts
recommendation.handler.ts
user.data.handler.ts
user.handler.ts
middlewares
models
routes
tests
types
utils
app.ts
config.ts
temp
.env
.eslintignore
.eslintrc.js
.gitignore
.prettierignore
.prettierrc
database.json
package-lock.json
OUTLINE
TIMELINE
Ln 1, Col 1 Spaces: 4 UTF-8 LF Properties


File Edit Selection View Go Run Terminal Help
diabets-ui
EXPLORER
DIABETS-UI
node_modules
public
src
Assets
components
Grid
Login
Main
pre_form
prediction
predict.js
PredictionPopup.css
PredictionPopup.js
recommendation.js
86.png
866.png
8666.jpeg
btn.css
Btn.jsx
home-banner-background.png
Home.js
i1.jpeg
pre.css
pre.jsx
recommendation.js
prediction
Signup
userprofile
About.js
btn.css
OUTLINE
TIMELINE
recommendation.js ...\components
recommendation.js ...\prediction
src > components > pre_form > prediction > recommendation.js > recommendationHandler > recommendationApiUrl
import axios from 'axios';
export const recommendationHandler = async(token) => {
try {
const recommendationApiUrl = 'http://127.0.0.1:3001/users/recommend'; // Update with your backend recommendation API URL
const headers = {
'Authorization': `Bearer ${token}`,
'Content-Type': 'application/json' // Add Content-Type header
};
const response = await axios.post(recommendationApiUrl, {}, { headers, responseType: 'blob' });
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
node
Compiled successfully!
You can now view front-end in the browser.
Local: http://localhost:3000
On Your Network: http://192.168.244.219:3000
Note that the development build is not optimized.
To create a production build, use npm run build.
webpack compiled successfully
History restored
PS F:\gporject-updated\Diabetes-Project\diabets-ui> npm run start
> front-end@0.1.0 start
> react-scripts start
(node:8388) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:8388) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
Ln 5, Col 60 Spaces: 4 UTF-8 CRLF {} JavaScript

13. Chapter: Project Conclusion and Future Work

13.1. Conclusion and Future Work

Conclusion Many chronic diseases do not have its treatment discovered, yet, many other chronic diseases have existing treatments which have destroying effects or other drug – addiction effects on the body. However, most of these diseases could have been avoided following the suitable dietary instruction. In this project, we have successfully built three different models to help in healthy dietary for prediabetics, diabetics, obese and other diabetes – related patients. If the patient, already, has a previous knowledge of his disease, he can, immediately, start his diet recommendations. Not just for diabetes but in general, If he has no previous knowledge, in this case, a disease detector has been built using machine learning to indicate the probability of having the chronic disease, in case of this project, only Diabetes is considered and more diseases can be added in the future. Then, the patient, also, can start his diet recommendations, after taking the proper tests to make sure, test wise, of his need to such a diet recommendation system.

13.2. Future Work

Make the app available for public users. As early detection is already available, the system is helpful with doctors, but it's urgent for public users, in our age. In order to achieve that, we would dive deeper in four main directions:

• Add more space for more chronic diseases.

• Convert static database to dynamic using cloud storage instead.

• Find approximate ranges for measured parameters like skin thickness and genetic function to suit usage of general users.

• Provide links in the app with outer noninvasive devices for measuring blood sugar level along the day to help recording and monitoring blood sugar levels.

• Apply community features to the application such as groups, pages and other contribution features similar to the highest ranked apps like Facebook, Twitter, etc. in order to trigger the human behavior in the process of following the diet as a social activity. Then, it would be ready for deployment and uploading on Google play store

14.References

[1] " Goldberg, D. Nichols, D., Oki, B. M., and Terry, D. Using collaborative filtering to weave an information tapestry. Common. ACM 35, 12 (Dec.1992), 61—70.

[2] "Alford, B. B., A. C. Blankenship, and R. D. Hagen. The effects of variations in carbohydrate, protein, and fat content of the diet upon weight loss, blood values, and nutrient intake of adult obese women. J. Am. Diet. Assoc. 90:534- 540, 19

[3] "American Diabetes Association. Nutrition recommendations and principles for people with diabetes mellitus. Diabetes Care 17:519-522, 1994.

[4] "Locatelli F, Marcelli D, Conte F et al. For the Registro Lombardo Dialisi e Trapianto. Cardiovascular disease in chronic renal failure: the challenge continues. Nephrol Dial Transplant 2000; 15 [Suppl 5]: S69–S80.

[5] "Lupi GP, Bisegna S, Marcelli D, Grassi C, Locatelli F. Epidemiology and follow-up of early chronic renal failure (CRF) in Lombardy population. Nephrol Dial Transplant 1997; 12: A105.

[6] "Chronic Disease Overview. http://www.cdc.gov/NCCdphp/overview.html

[7] "National Health Expenditure Data, Historical. http://www.cms.hhs.gov/NationalHealthExpendData/02_NationalHealthAccountsHistorical.asp #TopOfPage

[8] https://www.ijert.org/diabetes-prediction-using-machine-learning-techniques

[9] https://www.hindawi.com/journals/jhe/2021/9930985/
its symptoms are: frequent urination, increased thirst, and increased hunger

[10] https://www.javatpoint.com/

[11] https://www.researchgate.net/publication/347091823_Diabetes_Prediction_Using_Machine_Learning

[12] https://towardsdatascience.com/brief-on-recommender-systemsb86a1068a4dd

[13] https://towardsdatascience.com/an-easy-introduction-to-machine-learningrecommender-systems-efc8f7ece829

[14] https://realpython.com/python-ai-neural-network/

[15] https://www.sciencedirect.com/science/article/abs/pii/S2254887420300345