A

MINI PROJECT

REPORT

ON

A LIFE STYLE DISEASE PREDICTION USING DECISION TREE

A project report submitted to the
Jawaharlal Nehru Technological University
In partial fulfillment for the award

Bachelor of Technology
in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

G. KALYANI	**– 21RJ1A0596**
K. GOWTHAM KUMAR	**– 21RJ1A05C4**
K. SAI PRASAD	**– 21RJ1A05C8**
G. SHASHANK	**– 21RJ1A0599**

Under the esteemed guidance of
Mr. R. MANTHRU NAIK
Assistant Professor

MALLA REDDY INSTITUTE OF TECHNOLOGY

(Affiliated to JNTU, Hyderabad | Approved by AICTE, New Delhi) Accredited by NBA, Certificated by ISO 9001:2015

Maisammaguda, Dhullapally, Via: Kompally, Hyderabad – 500100 2021 - 2025

MALLA REDDY INSTITUTE OF TECHNOLOGY

(Sponsored by Malla Reddy Educational Society)

Accredited by NBA, Certificated by ISO 9001:2015

Approved by AICTE & Affiliated to JNTU,

Hyderabad

Maisammaguda, Dhullapally Post, (Via: Kompally), Secunderabad - 500100.

CERTIFICATE

This is to certify that mini project work entitled **"LIFE STYLE DISEASE PREDICTION USING DECISION TREE"** is a bonafide work carried by **G. KALYANI (21RJ1A0596), K. GOWTHAM KUMAR (21RJ1A05C4), K.SAI PRASAD(21RJ1A05C8), G. SHASHANK (21RJ1A0599)** of **COMPUTER SCIENCE AND ENGINEERING DEPARTMENT** in **MALLA REDDY INSTITUTE OF TECHNOLOGY** and submitted to **JNT UNIVERSITY, Hyderabad** in the partial fulfillment of the requirement for the award of **BACHELOR OF TECHNOLOGY**.

Project Guide	Project Coordinator	Head of Department
Mr.R.Manthru Naik	Mr. U.Sudhakar Reddy	Mrs.K.Bhavani
Assistant Professor	Assistant Professor	Associate Professor

External Examiner

DECLARATION

We hereby declare that the project entitles **"LIFE STYLE DISEASE PREDICTION USING DECISION TREE"** submitted to **Malla Reddy Institute of Technology**, affiliated to Jawaharlal Nehru Technological University Hyderabad (**JNTUH**) for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** is a result of work done by us.

It is further declared that the project report or any part thereof has not been previously submitted to any University or Institute for the award of degree.

G. KALYANI – 21RJ1A0596
K. GOWTHAM KUMAR – 21RJ1A05C4
K. SAI PRASAD – 21RJ1A05C8
G. SHASHANK – 21RJ1A0599

ACKNOWLEDGEMENT

We are happy to express our deep sense of gratitude to the **Principal** of the college **Dr. K. Srinivasa Rao, Professor, Malla Reddy Institute of Technology** for having provided us the adequate facilities to pursue our project.

We would like to thank **Mrs.K.Bhavani, Associate Professor and Head, Department of Computer Science and Engineering, Malla Reddy Institute of Technology,** for having provided the freedom to use all the facilities available in the department, especially the laboratories and library.

We also grateful to our project coordinator, **Mr. U.Sudhakar Reddy**, **Assistant Professor**, **Department of Computer Science and Engineering**, **Malla Reddy Institute of Technology**, for extending his support and assisting us throughout our project work.

We very grateful to our project guide, Mr. **R .Manthru naik**, **Assistant Professor, Department of Computer science and engineering**, **Malla Reddy Institute of Technology**, for his extensive patience and guidance throughout our project work.

We sincerely thank all the **Teaching and Non-teaching staff** of the department of **Computer Science and Engineering**, also like to thank our **Classmates** for their timely suggestions, healthy criticism and motivation during the course of our project work. We thank our **parents** who were the backbone behind our deeds.

Finally, we express our immense gratitude with pleasure to all **individuals** who have either directly or indirectly contributed to our need at right time for the development and success of our project work.

.

ABSTRACT

A Proposed Model for Lifestyle Disease Prediction Using Decision Tree Classifier

Lifestyle diseases are illnesses associated with the way individuals or groups live, often influenced by unhealthy habits such as poor diets, excessive energy intake, and physical inactivity. These diseases pose a significant challenge globally, yet the vast amount of healthcare data collected often remains underutilized. Mining this data for valuable insights could enable more effective decision-making in disease prevention and management.

This study focuses on understanding the principles of support vector machines (SVM) and applying them to predict lifestyle diseases an individual may be at risk of developing. By analyzing patterns in lifestyle-related factors, SVM can identify potential health threats before they become severe. This predictive capability can guide early interventions and personalized healthcare strategies, enhancing the quality of care and prevention measures.

To further innovate, we propose a cost-effective machine learning model as an alternative to deoxyribonucleic acid (DNA) testing. This model simulates an individual's lifestyle data to detect genetic and non-genetic disorders caused by unhealthy behaviors. It offers a low-cost, intelligent solution that forms the foundation for diagnostic tests and preventive measures, addressing the growing need for accessible and efficient healthcare tools

Keywords—Deoxyribonucleic acid testing, healthcare industries, lifestyle diseases, decision tree classifier

INDEX

LIST OF FIGURES

LIST OF SCREEN SHOTS

1. INTRODUCTION

A report prepared by the World Health Organization and World Economic Forum says that India will incur an accumulated loss of $236.6 billion by 2015 because of morbid lifestyles as well as imperfect diet [1]. Lifestyle and diet are the two main factors that are considered to influence receptiveness to various diseases. Diseases are mainly caused by a combination of transformation, lifestyle selections, and surroundings. In addition, identifying health risks in an individual's family is one of the most crucial things an individual can do to help his/her practitioner understand and diagnose hereditarily linked syndromes like cancer, diabetes, and mental illness. Diseases that are associated with the way a person or group of people live are known as lifestyle diseases. They include atherosclerosis; heart disease and stroke; obesity and type II diabetes; and smoking and alcohol-related diseases. This study aims to understand support vector machine (SVM) and use it to predict lifestyle diseases that an individual might be susceptible to. The need for public awareness is not stressed enough, but lifestyle diseases are easy to prevent. Simply modifying an individual's lifestyle to reduce and eliminate risks can be interesting. Deoxyribonucleic acid (DNA) and genetic testing are creating a new expanse of personalized medicine. However, on an average, DNA testing may incur ₹ 10,000 to 20,000 [2], which is expensive. Though there are many receding diseases and tests, they are erratically tested because they are costly, and factual tests have not been developed yet. Our lifestyles are imperative in increasing or decreasing risks of various diseases. According to some research conducted in the discipline of epigenetics determines that an individual's lifestyle selections can modify his/her well-being at genetic level. This study discusses about a model that can predict the probabilities of an individual obtaining a lifestyle disease. Lifestyle diseases depend on factors like heaviness, workout, and food likings and thus have a strong association with the abovementioned factors. In our simulated model, an actor will input his/her details like fatness, sleeping habits and will discover the likelihood of suffering from lifestyle diseases.

The remainder of this manuscript is organized as follows. Section 2 provides a brief summary about related work in machine learning (ML) domain. Section 3 focuses on ML and SVM (linear and multiclass) algorithm. Section 4 explains the proposed system (block diagram and working) for lifestyle disease prediction. Section 5 presents simulation results for the proposed system.
Section 6 concludes the study with future scope.

2. LITERATURAL SURVEY

TITLE: Occupational lifestyle diseases: An emerging issue AUTHOR: Sharma, M. and Majumdar, P.K., 2009.

Lifestyle diseases characterize those diseases whose occurrence is primarily based on the daily habits of people and are a result of an inappropriate relationship of people with their environment. The main factors contributing to lifestyle diseases include bad food habits, physical inactivity, wrong body posture, and disturbed biological clock. A report, jointly prepared by the World Health Organization (WHO) and the World Economic Forum, says India will incur an accumulated loss of $236.6 billion by 2015 on account of unhealthy lifestyles and faulty diet. According to the report, 60% of all deaths worldwide in 2005 (35 million) resulted from noncommunicable diseases and accounted for 44% of premature deaths. What's worse, around 80% of these deaths will occur in low and middle-income countries like India which are also crippled by an ever increasing burden of infectious diseases, poor maternal and perinatal conditions and nutritional deficiencies. According to a survey conducted by the Associated Chamber of Commerce and Industry (ASSOC-HAM), 68% of working women in the age bracket of 21-52 years were found to be afflicted with lifestyle ailments such as obesity, depression, chronic backache, diabetes and hypertension. The study 'Preventive Healthcare and Corporate Female Workforce' also said that long hours and working under strict deadlines cause up to 75% of working women to suffer from depression or general anxiety disorder, compared to women with lesser levels of psychological demand at work. The study cited scientific evidence that healthy diet and adequate physical activity - at least 30 minutes of moderate activity at least five days a week - helped prevent NCDs. In India, 10% of adults suffer from hypertension while the country is home to 25-30 million diabetics. Three out of every 1,000 people suffer a stroke. The number of deaths due to heart attack is projected to increase from 1.2 million to 2 million in 2010. The diet [or lifestyle] of different populations might partly determine their rates of cancer, and the basis for this hypothesis was strengthened by results of studies showing that people who migrate from one country to another generally acquire the cancer rates of the new host country, suggesting that environmental [or lifestyle factors] rather than genetic factors are the key determinants of the international variation in cancer rates. Some of the common diseases encountered because of occupational lifestyle are Alzheimer's disease, arteriosclerosis, cancer, chronic liver disease/cirrhosis, chronic obstructive pulmonary disease (COPD), diabetes,

hypertension, heart disease, nephritis/CRF, and stroke. Occupational lifestyle diseases include those caused by the factors present in the vicinity like heat, sound, dust, fumes, smoke, cold, and other pollutants. These factors are responsible for allergy, respiratory and hearing problems, and heat or cold shock. So, A healthy lifestyle must be adopted to combat these diseases with a proper balanced diet, physical activity and by giving due respect to biological clock. Kids spending too much time slouched in front of the TV or PCs, should be encourage to find a physical sport or activity they enjoy. Fun exercises should be encouraged into family outings. A pizza-and-video evening should be replaced for a hike and picnic. Kids who do participate in sport, especially at a high competitive level, can find the pressure to succeed very stressful. To decrease the ailments caused by occupational postures, one should avoid long sitting hours and should take frequent breaks for stretching or for other works involving physical movements.

TITLE: **Effect of changes on body weight and lifestyle in nonalcoholic fatty liver disease**

Author: Suzuki, A., Lindor, K., St Saver, J., Lymp, J., Mendes, F., Muto, A., Okada, T. and Angulo, P., 2005.

The effects of lifestyle modifications in nonalcoholic fatty liver disease (NAFLD) are incompletely defined. We aimed at determining the association of changes in body weight and lifestyle with changes in serum ALT levels. We analyzed annual health checkup data from 1546 employees. Of 469 subjects with elevated ALT, we selected 348 male subjects by excluding those who had other causes of liver disease. They were followed for one year to assess the association of change in lifestyle with change in serum ALT. The 136 subjects who had ALT normalization were followed for two years to assess the association between lifestyle management and persistently normal ALT.

TITLE: Prediction system for heart disease using Naïve

Bayes. AUTHOR: Pattekari, S.A. and Parveen, A., 2012.

Data Mining refers to using a variety of techniques to identify suggest of information or decision making knowledge in thedatabase and extracting these in a way that they can put to use in areas such as decision support, predictions, forecasting and estimation. The healthcare industry collects huge amounts of healthcare data which, unfortunately, are not “mined” to discover hidden

informationfor effective decision making. Discovering relations that connect variables in a database is the subject of data mining. This research has developed a Decision Support in Heart Disease Prediction System (DSHDPS) using data mining modeling technique, namely, Naïve Bayes. Using medical profiles such as age, sex, blood pressure and blood sugar it can predict the likelihood of patients getting a heart disease. It is implemented as web based questionnaire application. It can serve a training tool to train nurses and medical students to diagnose patients with heart disease

TITLE: **Prediction of diabetes based on personal lifestyle indicators**

AUTHOR: Anand, A. and Shakti, D., 2015.

Diabetes Mellitus or Diabetes has been portrayed as worse than Cancer and HIV (Human Immunodeficiency Virus). It develops when there are high blood sugar levels over a prolonged period. Recently, it has been quoted as a risk factor for developing Alzheimer, and a leading cause for blindness & kidney failure. Prevention of the disease is a hot topic for research in the healthcare community. Many techniques have been discovered to find the causes of diabetes and cure it. This research paper is a discussion on establishing a relationship between diabetes risk likely to be developed from a person's daily lifestyle activities such as his/her eating habits, sleeping habits, physical activity along with other indicators like BMI (Body Mass Index), waist circumference etc. Initially, a Chi-Squared Test of Independence was performed followed by application of the CART (Classification and Regression Trees) machine learning algorithm on the data and finally using Cross- Validation, the bias in the results was removed.

TITLE: Study of machine learning algorithms for special disease prediction using principal of component analysis

AUTHOR: Kanchan, B.D. and Kishor, M.M., 2016.

The worldwide study on causes of death due to heart disease/syndrome has been observed that it is the major cause of death. If recent trends are allowed to continue, 23.6 million people will die from heart disease in coming 2030. The healthcare industry collects large amounts of heart disease data which unfortunately are not “mined” to discover hidden information for effective decision making. In this paper, study of PCA has been done which finds the minimum number of attributes required to enhance the precision of various supervised machine learning algorithms. The purpose of this research is to study supervised machine learning algorithms to predict heart disease. Data mining has number of important techniques like categorization, preprocessing. Diabetic is a

life threatening disease which prevent in several urbanized as well as emergent countries like India. The data categorization is diabetic patients datasets which is developed by collecting data from hospital repository consists of 1865 instances with dissimilar attributes. The examples in the dataset are two categories of blood tests, urine tests. In this research paper we discuss a variety of algorithm approaches of data mining that have been utilized for diabetic disease prediction. Data mining is a well known practice used by health organizations for classification of diseases such as diabetes and cancer in bioinformatics research.

TITLE: **Graph theoretical metrics and machine learning for diagnosis of Parkinson's disease using rs-fMRI.**

AUTHOR: Kazeminejad, A., Golbabaei, S. and Soltanian-Zadeh, H., 2017.

In this study, we investigated the suitability of graph theoretical analysis for automatic diagnosis of Parkinson's disease. Resting state fMRI data from 18 healthy controls and 19 patients were used in the study. After data preprocessing and identifying 90 regions of interest using the AAL atlas, average time series of each region was obtained. Next, a brain network graph was constructed using the regions as nodes and the Pearson correlation between their average time series as edge weights. A percentage of edges with the highest magnitude were kept and the rest were omitted from the graph using a thresholding method ranging from 10% to 30% with 2% increments. Global graph theoretical metrics for integration (Characteristic path length and Efficiency), segregation (Clustering Coefficient and Transitivity) and small-worldness were extracted for each subject and their between group differences were subjected to statistical analysis. Local metrics, including integration, segregation, centrality (betweenness, z-score, and participation coefficient) and nodal degree, were also extracted for each subject and used as features to train a support vector machine classifier. We have shown a statistically significant increase in characteristic path length as well as a decrease in segregation metrics and efficiency in Parkinson's patients. A floating forward automatic feature selection method was used to select the 5 best features from all extracted metrics to classify patients. Our classifier was able to achieve a diagnosis accuracy of ~95% when subjected to a leave-one-out cross-validation test. These features belonged to cuneus (right hemisphere), precuneus (left), superior (right) and middle (both)frontal gyri which were all previously reported to undergo alterations in Parkinson's disease. This investigation confirmed that

global brain network alterations are associated with Parkinson's patients' symptoms and showed the potency of using graph theoretical metrics

TITLE: One against one" or "one against all": Which one is better for handwriting recognition with SVMs

AUTHOR: Milgram, J., Cheriet, M. and Sabourin, R., 2006.

The "one against one" and the "one against all" are the two most popular strategies for multi-class SVM; however, according to the literature review, it seems impossible to conclude which one is better for handwriting recognition. Thus, we compared these two classical strategies on two different handwritten character recognition problems. Several post-processing methods for estimating posterior probability were also evaluated and the results were compared with the ones obtained using MLP. Finally, the "one against all" strategy appears significantly more accurate for digit recognition, while the difference between the two strategies is much less obvious with uppercase letters. Besides, the "one against one" strategy is substantially faster to train and seems preferable for problems with a very large number of classes. To conclude, SVMs allow significantly better estimation of probabilities than MLP, which is promising from the point of view of their incorporation into handwriting recognition systems.

TITLE: PRMT: Predicting Risk Factor of Obesity among MiddleAged People Using Data Mining Techniques. Procedia Computer Science

AUTHOR: Hossain, R., Mahmud, S.H., Hossin, M.A., Noori, S.R.H. and Jahan, H., 2018.

Obesity is an anatomical condition characterized by an extreme growth of body fat. The obesity rate is increasing gradually; from prior research, obesity is the serious health disease in the globe. This study collected 259 data from specified urban and rural areas regarding different risk factor of our daily activities. The purpose of the study is to simulate the risk factor by using statistical tools (SPSS), which helpsto predict the major risk factor of obesity by testing the class level attribute according to cross- sectional study with other attributes. By analyzing the P-value ($p<0.05$), the outcome of this process Age (0.002), Height (0.002), Weight 0.000), Healthy lifestyle (0.000), Marital status (0.001), BMI (0.000), Economic (0.028), Sleep per day (0.011) has a significant relationship with our obesity class. This study proposed a risk mining technique (PRMT)that foretells a model to analyze the risk factor of obesity class using different data mining

classifiers, using WEKA to estimate the accuracy and error measurement. The outcome of this process Naïve Bayes is the best classifier for the 10-fold cross- validation study. The proposed model collaborates to predict human factor who want to control and mitigate this major cardiovascular disease.

TITLE: Disease prediction by using machine learning.

AUTHOR: Sayali Ambekar and Dr.Rashmi Phalnikar, 2018

Disease Prediction using Machine Learning is the system that is used to predict the diseases from the symptoms which are given by the patients or any user. The system processes the symptoms provided by the user as input and gives the output as the probability of the disease. Naïve Bayes classifier is used in the prediction of the disease which is a supervised machine learning algorithm. The probability of the disease is calculated by the Naïve Bayes algorithm. With an increase in biomedical and healthcare data, accurate analysis of medical data benefits early disease detection and patient care. By using linear regression and decision tree we are predicting diseases like Diabetes, Malaria, Jaundice, Dengue, and Tuberculosis.

3. SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Identifying health risks in an individual's family is one of the most crucial things an individual can do to help his/her practitioner understand and diagnose hereditarily linked syndromes like cancer, diabetes, and mental illness. Diseases that are associated with the way a person or group of people live are known as lifestyle diseases. They include atherosclerosis; heart disease and stroke; obesity and type II diabetes; and smoking and alcohol-related diseases Deoxyribonucleic acid (DNA) and genetic testing are creating a new expanse of personalized medicine. However, on an average, DNA testing may incur
₹ 10,000 to 20,000, which is expensive. Though there are many receding diseases and tests, they are erratically tested because they are costly, and factual tests have not been developed yet.

Identifying health risks within a family is a proactive approach to better healthcare and disease prevention. By mapping out an individual's family medical history, healthcare practitioners gain critical insights into hereditary conditions that can be passed down through generations. Syndromes such as cancer, diabetes, and mental illnesses often have genetic components. Understanding these familial patterns helps medical professionals make early diagnoses, recommend lifestyle changes, and initiate targeted treatment plans.

In addition to hereditary diseases, lifestyle diseases are increasingly becoming a global concern. These are conditions that arise from unhealthy lifestyle choices or habits, such as poor diet, lack of exercise, and substance abuse. Examples include atherosclerosis, heart disease, stroke, obesity, and type II diabetes. Lifestyle-related conditions often develop over time and are preventable with lifestyle modifications and regular medical check-ups. Additionally, smoking and excessive alcohol consumption are major contributors to diseases like lung cancer, liver cirrhosis, and cardiovascular disorders.

A groundbreaking advancement in modern medicine is the use of deoxyribonucleic acid (DNA) and genetic testing. These technologies have revolutionized personalized medicine by allowing for precise risk assessment, early disease detection, and customized treatment plans tailored to an individual's genetic profile. For example, genetic testing can identify mutations

associated with breast or ovarian cancer (e.g., BRCA1 and BRCA2 genes) or predict susceptibility to diseases like Alzheimer's or cardiovascular conditions.

However, despite its promise, genetic testing comes with significant financial barriers. The average cost of such tests in India ranges from ₹10,000 to ₹20,000, making it inaccessible to many. The high expense arises from the sophisticated technology and expertise required for analysis. Additionally, while tests for some genetic conditions are well-developed, others remain in their infancy due to gaps in research and development. This results in erratic testing practices, as people often opt for such diagnostics only when deemed absolutely necessary.

To address these challenges, efforts are needed to make genetic testing more affordable and widely available. Subsidized healthcare programs, public awareness campaigns, and investments in genomic research could lower costs and expand the scope of testing. Moreover, advancements in biotechnology may eventually lead to the development of more accurate, cost- effective diagnostic tools, paving the way for a healthier future where genetic and lifestyle diseases are more effectively managed

DISADVANTAGES

1. Manual analysis of large amount data becomes critical
2. Unable find the disease more accurately
3. More expensive to find out diseases

3.2 PROPOSED SYSTEM

This study aims to understand Decision Tree Classifier and use it to predict lifestyle diseases that an individual might be susceptible to. The need for public awareness is not stressed enough, but lifestyle diseases are easy to prevent. Simply modifying an individual’s lifestyle to reduce and eliminate risks can be interesting.

This study focuses on utilizing the **Decision Tree Classifier**, a machine learning algorithm, to predict lifestyle diseases that an individual may be at risk for. The goal is to leverage data-driven approaches to identify patterns and relationships between an individual's health data and their susceptibility to lifestyle-related conditions. Decision Tree Classifiers are particularly suitable for

this purpose due to their intuitive structure, ease of interpretation, and effectiveness in handling both categorical and numerical data.

Decision Tree Classifier for Lifestyle Disease Prediction

A Decision Tree Classifier works by breaking down complex decision-making processes into simpler, sequential decisions represented in a tree-like structure. For predicting lifestyle diseases, it can analyze various risk factors such as:

- **Dietary habits** (e.g., high sugar or fat consumption)
- **Physical activity levels**
- **Body Mass Index (BMI)**
- **Smoking and alcohol consumption**
- **Family medical history**
- **Stress levels and sleep patterns**

Using this data, the Decision Tree Classifier can identify key risk factors and provide a prediction on the likelihood of developing conditions such as obesity, diabetes, heart disease, or hypertension.

For example:

1. If a person has a sedentary lifestyle, the tree might branch to assess their BMI.
2. If the BMI is high, it could further evaluate dietary patterns and family history.
3. Based on this branching logic, the model can predict whether the individual is at high risk for a lifestyle disease.

Importance of Public Awareness

Despite the significant impact of lifestyle diseases on public health, awareness and prevention efforts often fall short. Many people are unaware of how small but consistent changes in their daily routines can drastically reduce the risk of these conditions. Educating the public about the causes, risks, and preventative measures is critical.

Preventing Lifestyle Diseases

Lifestyle diseases are unique in that they are often preventable through proactive measures. By identifying high-risk individuals and promoting healthier habits, the burden of these diseases can be significantly reduced. Some effective interventions include:

1. **Dietary Changes**: Reducing processed foods, sugary drinks, and high-fat diets while increasing fruits, vegetables, and whole grains.
2. **Regular Exercise**: Encouraging at least 150 minutes of moderate physical activity per week.
3. **Stress Management**: Practices like yoga, meditation, and time management can help mitigate the effects of chronic stress.
4. **Quitting Smoking and Alcohol**: Eliminating or significantly reducing tobacco and alcohol consumption can lower risks for cardiovascular and liver-related diseases.
5. **Regular Health Checkups**: Routine screenings can detect early warning signs of diseases like hypertension or prediabetes.

Role of the Study

This study aims to combine the predictive power of machine learning with actionable public health strategies. By making disease prediction tools accessible, individuals can better understand their risk levels and take informed steps toward prevention. Furthermore, such studies can serve as a basis for larger public health initiatives, focusing on awareness campaigns and community interventions that promote healthier lifestyles.

In summary, the Decision Tree Classifier is not just a technical tool but a means to empower individuals and communities in addressing the growing epidemic of lifestyle diseases. Through early prediction, targeted interventions, and widespread awareness, this study underscores the importance of proactive health management in improving quality of life and reducing the strain on healthcare systems.

Advantages

1. Data analysis is more accurate using Machine Learning
2. Using DTC can predict the accurate Disease

4. SYSTEM STUDY

4.1 FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Feasibility analysis is a critical process conducted during the initial stages of a project to determine whether the proposed system is viable, practical, and beneficial for the organization. It ensures that the project is not only achievable within the given resources but also aligned with the company's objectives without becoming a financial or operational burden. This analysis involves assessing the requirements, identifying potential challenges, and evaluating the overall impact of the system on existing workflows, infrastructure, and stakeholders. By examining these factors, feasibility analysis helps in identifying the practicality of the project, estimating costs, and outlining a broad implementation strategy, ensuring that the organization commits to a solution that is both sustainable and value-driven.

Three key considerations involved in the feasibility analysis are,

- **ECONOMICAL FEASIBILITY**
- **TECHNICAL FEASIBILITY**
- **SOCIAL FEASIBILITY**

4.1.1 ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

This study evaluates the economic feasibility of implementing the proposed system and its potential impact on the organization's financial structure. Given the constraints on the budget allocated for research and development, every expenditure must be justified to ensure cost-

effectiveness. By carefully selecting the tools and technologies, the system has been designed to remain within the financial limits, making it a practical and sustainable solution.

To achieve this, the majority of the technologies used in the system are open-source or freely available, significantly reducing the upfront investment required. The use of these freely accessible tools not only ensures budget compliance but also supports innovation and adaptability during the development process. Only essential customized components and software, tailored to meet specific organizational needs, were purchased, optimizing resource allocation without compromising functionality or quality.

This cost-conscious approach ensures that the system delivers value without straining the organization's financial resources. By demonstrating that advanced technological solutions can be developed within a constrained budget, this study sets a benchmark for economically viable innovation. It underscores the importance of leveraging open-source technologies and strategic investments to maximize returns, making such systems accessible and scalable for organizations of all sizes.

4.1.2 TECHNICAL FEASIBILITY

This study focuses on assessing the technical feasibility of the proposed system, ensuring that it aligns with the available technical resources and infrastructure. For a system to be practical and widely adoptable, it must avoid imposing excessive demands on the existing resources, as this could result in significant challenges for clients, such as high operational costs or complex implementation procedures. Therefore, the developed system has been designed to require minimal technical adjustments, ensuring a seamless integration process.

One of the key considerations in this feasibility study is to keep the system's technical requirements modest. This approach not only reduces the strain on the client's infrastructure but also ensures compatibility with diverse environments, making it more adaptable and user- friendly. By limiting the need for extensive hardware or software upgrades, the system minimizes implementationhurdles and ensures that organizations can adopt it with ease, even if they have limited technological capacity.

Additionally, designing a system with low technical demands enhances its scalability and accessibility. Organizations with varying levels of technical expertise and infrastructure can implement the system without substantial investments in new resources. This thoughtful design ensures that the system remains both effective and efficient, bridging the gap between technical innovation and practical usability while promoting broader adoption across different client bases

4.1.3 SOCIAL FEASIBILITY

This study emphasizes evaluating the user acceptance of the proposed system, recognizing that its success hinges on how well users adapt to and embrace it. An integral part of this process is ensuring that users feel comfortable and confident in interacting with the system. Rather than perceiving it as a threat or a challenge, users should view it as a valuable and essential tool that enhances their productivity and effectiveness. Building this perception requires thoughtful planning and implementation of user training and support mechanisms.

A structured training program forms the cornerstone of fostering user acceptance. By employing clear, user-friendly methods to educate individuals about the system's features and benefits, the study aims to demystify its operations. Hands-on sessions, comprehensive guides, and interactive tutorials are crucial in making users feel at ease. Personalized training approaches can further enhance familiarity, ensuring that users can navigate the system with ease and efficiency.

Another critical factor in ensuring acceptance is encouraging user feedback and constructive criticism. By actively involving users in the improvement process, their confidence in the system is strengthened. This inclusive approach not only addresses user concerns but also instills a sense of ownership and trust, as their input directly contributes to refining the system. Ultimately, fostering open communication and prioritizing user-centric development will ensure the system becomes a well- integrated and indispensable part of their workflow.

5.SYSTEM SPECIFICATION

HARDWARE REQUIREMENTS

- **System** : Pentium IV 2.4 GHz
- **Hard Disk** : 40 GB
- **Monitor** : 14’ Colour Monitor
- **Mouse** : Mouse
- **Ram** : 512 Mb

SOFTWARE REQUIREMENTS

- **Operating system** : Windows 7 & above
- **Coding Language** : Python
- **Designing** : Html, css, javascript
- **Data Base** : MySQL
- **Framework** : Django

6. SYSTEM DESIGN

6.1 System Architecture

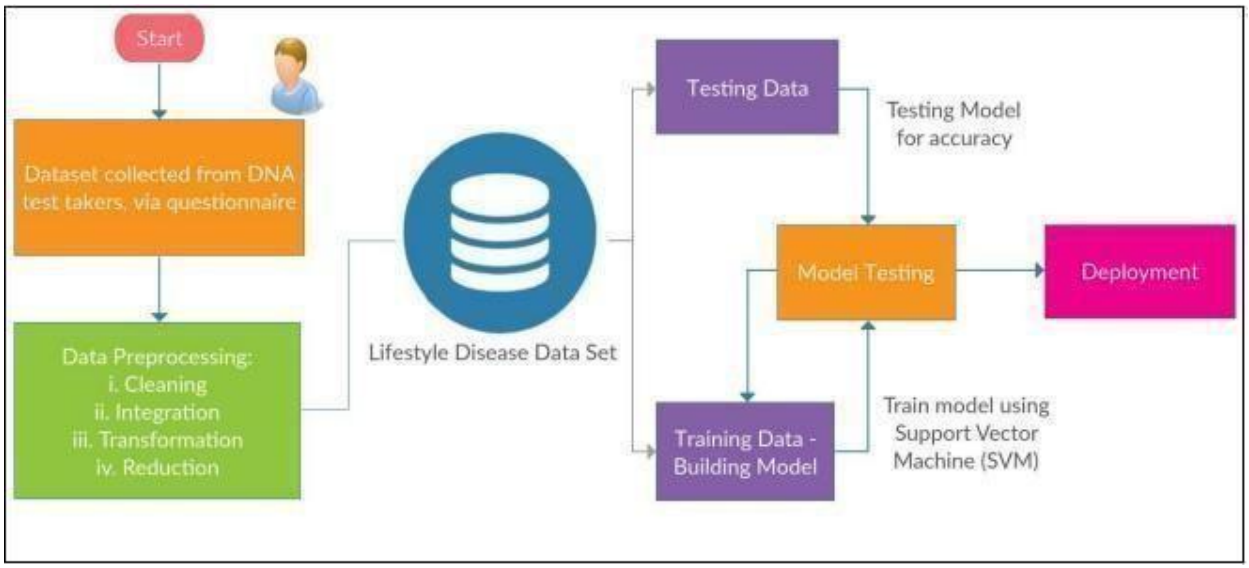


Fig 6.1: System Architecture

The process of building and deploying a machine learning model for analyzing lifestyle diseases begins with data collection. This involves gathering responses from DNA test-takers through questionnaires. These responses serve as raw data and may include information about genetic markers, health history, and lifestyle habits such as diet, exercise, and smoking. Collecting this data is crucial as it forms the foundation for detecting patterns and correlations linked to lifestyle diseases like diabetes, hypertension, and cardiovascular disorders.

Once collected, the raw dataset undergoes preprocessing to ensure its quality, relevance, and usability for machine learning tasks. Preprocessing includes multiple steps. First, **cleaning** is performed to remove errors, inconsistencies, or missing values. Next, **integration** combines data from various sources, such as combining genetic data with lifestyle information.

Transformation follows, converting raw data into a suitable format, such as encoding categorical variables (e.g., "smoker" or "non-smoker") into numerical forms. Finally, **reduction** is applied to minimize the dataset size without losing critical information, improving computational

efficiency while maintaining analytical power. After preprocessing, the refined and high-quality data forms the "Lifestyle Disease Dataset," which focuses on traits and indicators relevant to predicting diseases associated with lifestyle and genetics.

The dataset is then split into two parts: training and testing datasets. The majority is allocated for training, where the machine learning model—a **Decision Tree** in this case—is developed. A Decision Tree is an interpretable and efficient algorithm that makes predictions by segmenting data into branches based on decision rules, such as "If BMI > 25, classify as high risk for heart disease." The model iteratively learns these rules by analyzing patterns in the training data, building a structure that effectively separates different classes of outcomes. Decision Trees are particularly valuable in healthcare applications as they are easy to interpret, enabling healthcare professionals to understand the reasoning behind predictions.

The trained Decision Tree is then validated using the testing dataset to evaluate its performance. Key metrics, such as **accuracy**, **precision**, **recall**, and the **F1-score**, are calculated to determine how well the model predicts outcomes, such as identifying individuals at risk for lifestyle diseases. If the model demonstrates high accuracy and reliability, it is considered ready for deployment.

Deployment involves integrating the validated Decision Tree model into real-world applications. This step enables practical uses such as personalized health assessments, where individuals receive tailored advice based on their genetic and lifestyle data. It can also aid in early disease detection, allowing healthcare providers to intervene before conditions worsen. Moreover, the model could be used for large-scale health risk assessments, offering insights to public health organizations for planning prevention strategies. By combining genetic and lifestyle information, this system represents a powerful tool for advancing precision medicine and promoting healthier lifestyles.

In summary, the process of building a Decision Tree model for analyzing lifestyle diseases transforms raw data into actionable insights. From careful data collection and rigorous preprocessing to model training, validation, and deployment, each step ensures the model's accuracy and effectiveness in predicting and managing lifestyle-related health risks. This approach exemplifies the potential of machine learning in revolutionizing healthcare through data-driven decision-making.

6.2 Uml diagrams

UML (Unified Modeling Language) diagrams are visual tools used in software engineering to represent the structure and behavior of a system. They are broadly divided into **structural diagrams** and **behavioral diagrams**. Structural diagrams, like **class diagrams**, show the static architecture of a system, including its classes, attributes, methods, and relationships. **Object diagrams** depict a specific instance of the system, while **component diagrams** focus on physical components such as executables and libraries. **Deployment diagrams** represent how software components are deployed on hardware, and **package diagrams** help organize related classes or components into modules.

Behavioral diagrams, on the other hand, focus on dynamic interactions and workflows within the system. **Use case diagrams** highlight user interactions with the system, showing actors and their associated functionalities. **Activity diagrams** map workflows, including decision points and sequential or parallel processes. **Sequence diagrams** capture the order of interactions between objects over time, while **collaboration diagrams** emphasize object relationships and message flows. **State diagrams** describe how objects transition between states in response to events, and **timing diagrams** focus on the temporal aspects of system behavior.

These diagrams are essential for visualizing, documenting, and communicating system designs. They ensure that stakeholders, developers, and analysts can collaboratively understand and refine system requirements and implementation.

6.2.1 Use case diagram

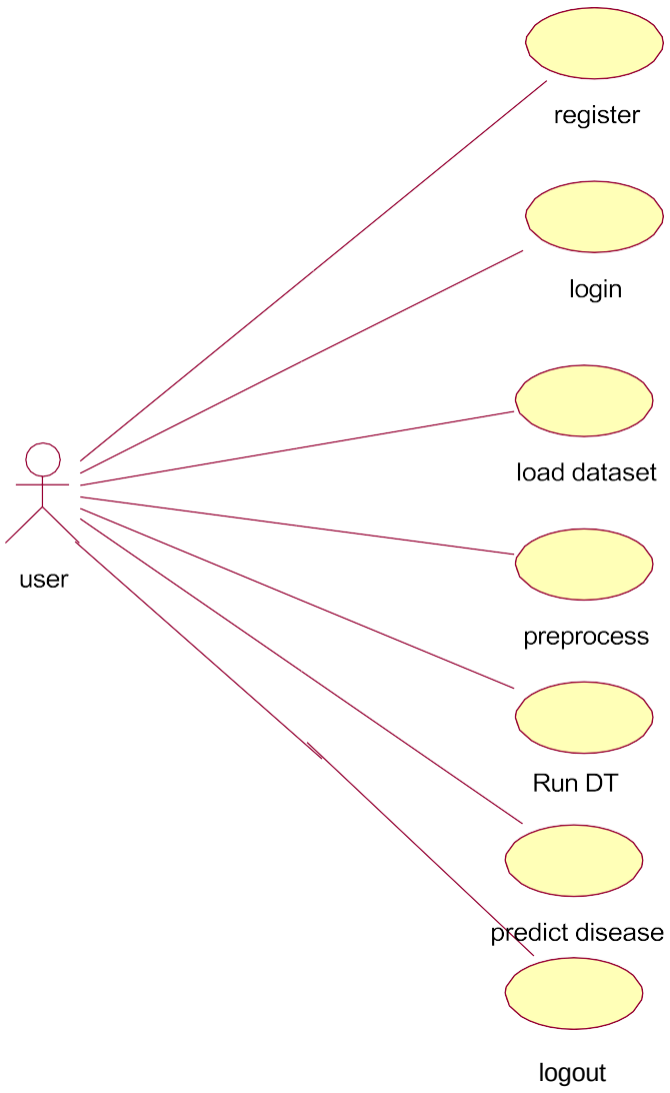


Fig 6.2.1 use case diagram

A Use Case Diagram is a graphical tool that represents how users interact with a system by depicting the various actions (use cases) they can perform and the system's responses. It helps to visualize user requirements and the system's functionality at a high level.

In the described example:

- **User**: This actor could be a doctor or a patient, representing the primary end-users who interact with the system.
- **Use Cases**: These are shown as ellipses, each representing a specific action or function.

The included use cases are:

- **Register**: Allows the creation of new user accounts.
- **Login**: Facilitates logging into the system using valid credentials.
- **Load Dataset**: Enables the upload of datasets containing patient information and medical records.
- **Preprocess**: Involves cleaning and preparing the uploaded dataset for further analysis.
- **Run Decision Tree Algorithm**: (Previously Run SVM, as per the updated design) Uses the decision tree algorithm to train a predictive model.
- **Predict Disease**: Applies the trained model to predict diseases for new patients.
- **Logout**: Lets the user safely exit the system.

- **Relationships**: The lines connecting the actor to the use cases represent the association or interaction between the user and the system. These are typically of the "uses" type, indicating that the user utilizes the system to execute these actions.

6.2.2 Class diagram

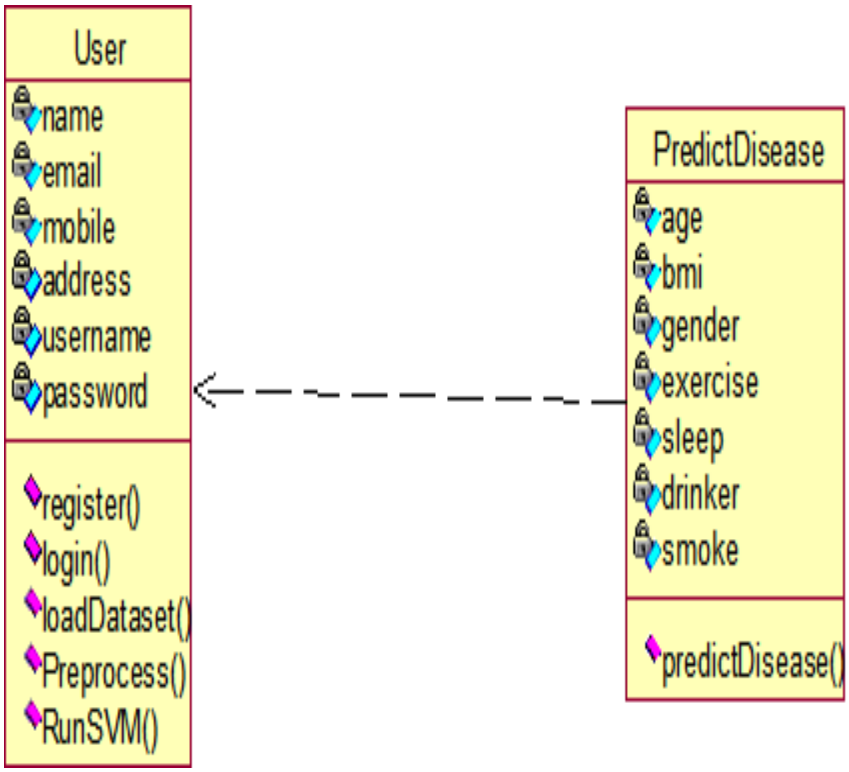


Fig 6.2.2 class diagram

This is a **Class Diagram**, which is used to represent the structure of a system by detailing its classes, their attributes, methods, and the relationships between them. The diagram contains two primary classes: **User** and **PredictDisease**.

The **User class** defines several attributes such as name, email, mobile, address, username, and password, which are associated with a user of the system. It also includes several methods: register(), which allows a user to create an account; login(), which authenticates the user for system access; loadDataset(), which enables the user to upload medical datasets; preprocess(),

which prepares thedataset for further analysis; and runSVM(), which runs a Support Vector Machine (SVM) algorithm to train a predictive model. If necessary, this method could be updated to **runDecisionTree()** to reflect the use of a decision tree algorithm instead of SVM.

The **PredictDisease class** represents the functionality for disease prediction, which depends on several attributes such as age, bmi, gender, exercise, sleep, drinker, and smoke. These attributes are used as input for the predictDisease() method, which predicts a disease for a new patient based on these factors.

The relationship between the **User class** and the **PredictDisease class** is shown by the dashed line, indicating an **association**. This suggests that the user interacts with the PredictDisease functionality by providing the necessary input data (like age, BMI, exercise habits, etc.) to predict the disease. The diagram illustrates how the system is structured to separate user operations from the prediction functionality, ensuring clear and modular design.

6.2.3 Sequence diagram

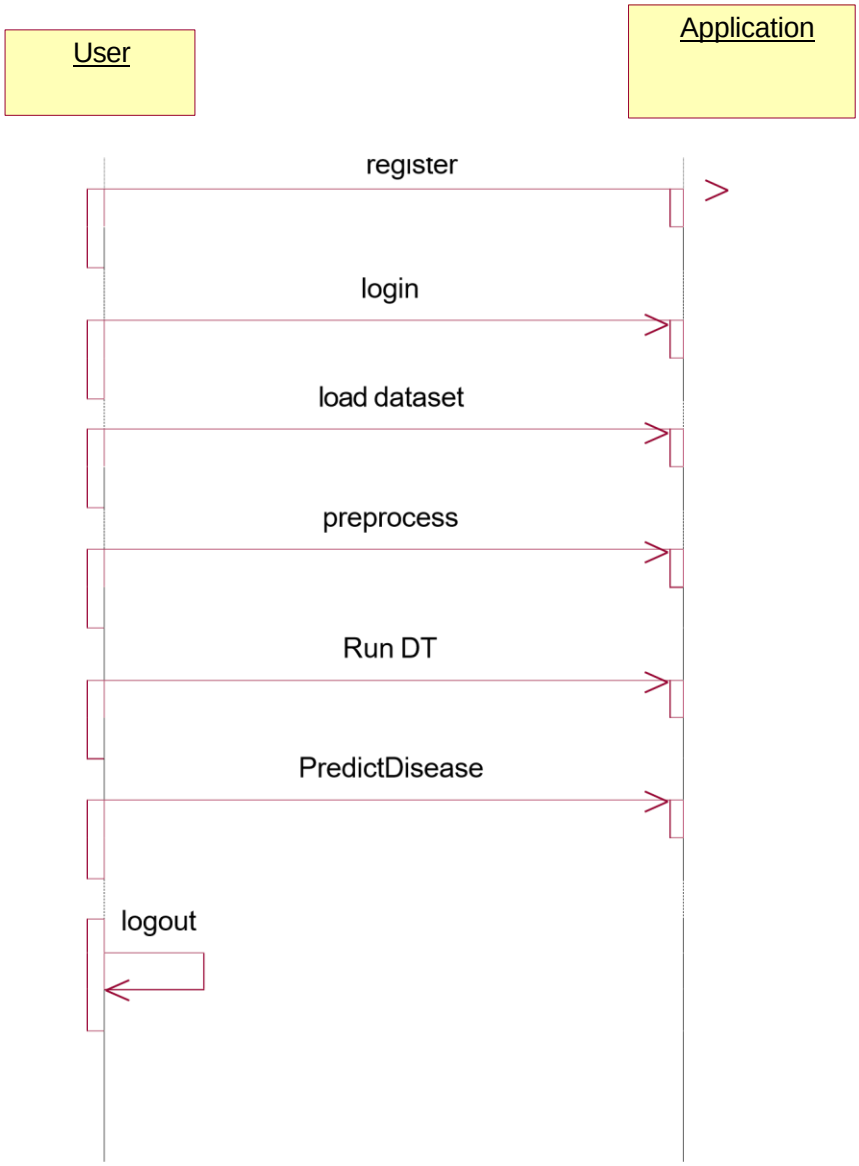


Fig 6.2.3 Sequence diagram

A **Use Case Diagram** is a visual representation that illustrates how users interact with a system, highlighting the actions they can perform and the system's responses. In this specific diagram, the **User** represents the end-user of the system, who could either be a doctor or a patient. The **Use Cases**, represented as numbered items, depict the various actions or functions available to

the user. These include **Register**, where users can create a new account; **Login**, which allows users to access the system with valid credentials; **Load Dataset**, which involves uploading patient information and medical records; and **Preprocess**, a step where the uploaded dataset is prepared for analysis.

The **Run Decision Tree Algorithm** use case replaces the previous **Run SVM** step. In this phase, the system executes the decision tree algorithm on the preprocessed data to train a predictive model. This trained model is subsequently utilized in the **Predict Disease** use case to forecast diseases for new patients based on their input data. Finally, the **Logout** use case ensures that users can securely exit the system.

The relationships between the user and the use cases are represented by lines, indicating the actions the user performs with the system. These relationships are typically of the "uses" type, signifying that the user utilizes the system to complete these tasks. This diagram offers a clear, high-level understanding of system functionality and user interactions, aiding in both analysis and design.

6.2.4 Collaboration diagram

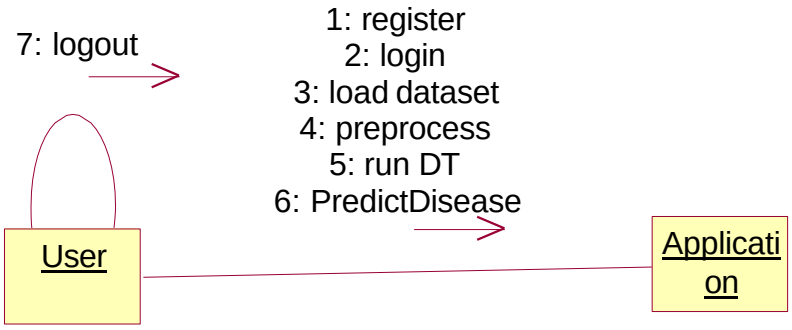


Fig 6.2.4 Collaboration diagram

A **Use Case Diagram** visually represents the interaction between users and a system, showing the various actions (use cases) users can perform and the system's corresponding responses. In this diagram, the **User** represents the end-user of the system, who could either be a doctor or a patient. The user interacts with the system to perform specific tasks, which are represented as **use cases** (shown as ellipses). The primary use cases include actions like **Register**,

where users create a new account; **Login**, for accessing the system with valid credentials; **Load Dataset**, which involves uploading datasets containing patient information and medical records; and **Preprocess**, where the uploaded dataset is prepared for analysis.

In this updated system, the **Run Decision Tree Algorithm** use case replaces the previous **Run SVM** step. Here, the decision tree algorithm is executed on the preprocessed data to train a predictive model. The trained model is then used in the **Predict Disease** use case to forecast the disease for new patients based on their input. Finally, the **Logout** use case allows users to safely exit the system.

The **relationships** between the user and use cases are depicted by lines, indicating the actions the user can perform through the system. These are typically of the "uses" type, signifying that the user leverages the system to carry out these functions. This diagram provides a high-level overview of system functionality and user interaction, making it a critical tool for understanding and designing the system's behavior

6.2.5 Activity diagram

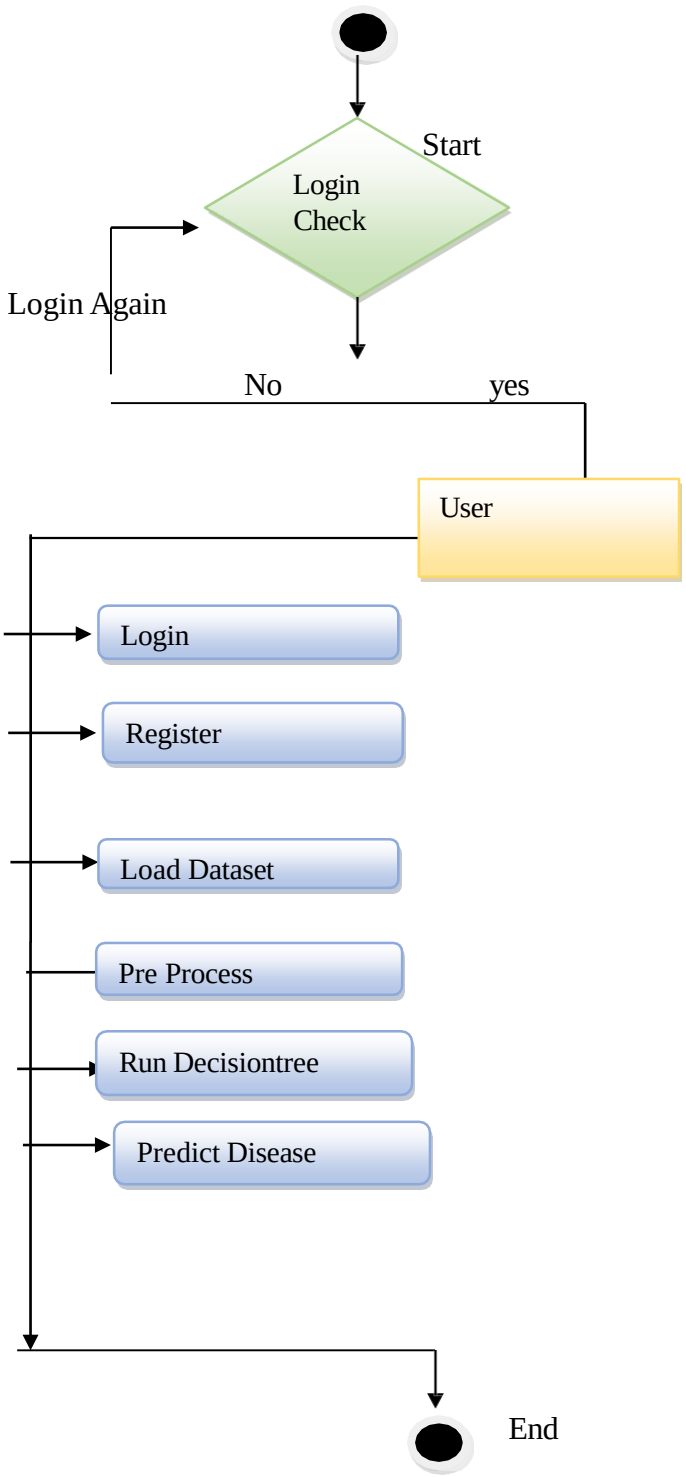


Fig 6.2.5 Activity diagram

Updated Activities

1. **Start**:
    - The process begins.
2. **Login / Register**:
    - **Decision Node**: The user chooses whether to log in (existing user) or register (new user).
3. **Load Dataset**: The system loads the required dataset.
4. **Preprocess Data**:
    - The dataset is preprocessed, including cleaning, feature engineering, and preparation for analysis.
5. **Run Decision Tree**:
    - A **Decision Tree** model is executed using the preprocessed dataset. Decision Trees split data recursively based on feature values to predict the outcome.
6. **Predict Disease**: The Decision Tree predicts the disease based on the input features.
7. **Display Results**:
    - The prediction or diagnosis is displayed to the user.
8. **End**:
    - The process ends.

7. IMPLEMENTATION

7.1 MODULES

1. **Data Collection Module**: Collects patient data, including medical history, lifestyle habits, and demographics.

2. **Data Preprocessing Module:** Cleans, transforms, and prepares data for analysis.

3. **Model Development Module:** Develops and trains decision tree models using patient data.

4. **Prediction Module**: Uses trained models to predict disease risk and provide personalized recommendations.

5. **User Interface Module**: Provides a user-friendly interface for patients, doctors, and administrators.

6. **Feedback and Evaluation Module:** Evaluates model performance, collects feedback, and updates models.
7. **Integration Module:** Integrates with electronic health records (EHRs), wearable devices, and other health systems.

7.2 SAMPLE CODE

VIEWS.PY

```
from django.shortcuts import render
import pymysql import pandas as
pd from sklearn.model_selection
import train_test_split from sklearn
import decisontree
import os
# Create your views
here.
def index(request):
  return
render(request,'user/index.html')
def userlogin(request):
  return
render(request,'user/Users.html')
def register(request):
  return
render(request,'user/Register.html')
def userhome(request):
 return render(request,'user/UserHome.html')
def RegAction(request):
```

```
name=request.POST['name']
email=request.POST['email']
mobile=request.POST['mobile']
address=request.POST['address']
username=request.POST['userna
me']
password=request.POST['passwo
rd']
con=pymysql.connect(host="localhost",user="root",password="root",database="lifestyle")
cur=con.cursor()
i=cur.execute("insert into user
values(null,'"+name+"','"+email+"','"+mobile+"','"+address+"','"+username+"','"+password+"')")
con.com
mit() if i>0:
  context={'data':'Registration
Successful...!!'}
  return
render(request,'user/Register.html',context)
else: context={'data','Registration Failed...!!'}
return
render(request,'user/Register.html',context)
def LogAction(request):
```

```
username=request.POST.get('username')

password=request.POST.get('password')

con=pymysql.connect(host="localhost",user="root",password="root",database="lifestyle")

cur=con.cursor() cur.execute("select * from user where username='"+username+"'and

password='"+password+"'") data=cur.fetchone() if data is not None:

    request.session['user']=usernam

    e request.session['userid']=data[0]

  return

  render(request,'user/UserHome.html'

  ) else:

    context={'data':'Login Failed ......!!'}

    return

render(request,'user/Users.html',context)

global data

def ViewData(request):
 global data

  BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(
  file_))) data=pd.read_csv(BASE_DIR+"\\dataset\\dataset.csv")

  context={'data':data}

  return

render(request,"user/ViewData.html",context

) global X global y global
```

```
X_train,X_test,y_train,y_test

def

Preprocess(request):global  global  y

global X_train,X_test,y_train,y_test

  X=data[['Age','Bmi','Drinking','Excercise','Gender','Sleep','Smoking']]

 y=data[['output']]

 X_train,X_test,y_train,y_test=train_test_split(X,y,test_size

    =0.1) context={'message':'Preprocess Has Do        !!!'}

 return

render(request,"user/Preprocess.html",context

) global model def RunDecisionTree(request):

global     model     model=decisiontree.SVC()

model.fit(X_train,y_train)

acc=model.score(X_test,y_test)*              100

context={'score':acc}                        return

render(request,"user/RunDecisionTree.html",

context) def Predict(request):

 return

render(request,'user/Predict.html')

def PredAction(request):

age=request.POST.get('age')

bmi=request.POST.get('bmi')
```

```
gender=request.POST.get('gender')

drinker=request.POST.get('drinker'

)

exercise=request.POST.get('exercis

e

')
  sleep=request.POST.get('sleep')

  smoker=request.POST.get('smok

  er')

  pred=model.predict([[age,bmi,gender,drinker,exercise,sleep,smoke

  r]]) print("predicted value: "+str(pred)) if pred[0]==1:

  context={'pred':'Depression'} return

  render(request,'user/UserHome.html',contex

  t) elif pred[0]==2:

  context={'pred':'Diabetes'} return

  render(request,'user/UserHome.html',contex

  t) else:

    context={'pred':'Hypertension'}

    return render(request,'user/UserHome.html',context)
```

8. SYSTEM TEST

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

TYPES OF TESTS

8.1 Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Unit testing is a crucial phase in the software development lifecycle that focuses on validating the functionality of individual components or units of an application. It involves designing test cases to ensure the internal program logic operates correctly, and that given inputs produce the expected outputs. By verifying all decision branches and internal code flows, unit testing ensures the robustness and accuracy of the software at a granular level.

This form of testing is structural and invasive, relying on an in-depth understanding of the program’s construction. Conducted after the completion of an individual unit and before integration with other components, it isolates specific business processes, application modules, or system configurations for evaluation. Unit tests are designed to examine the fundamental building blocks of the software, providing a foundation for identifying and addressing issues early in the development process.

The primary goal of unit testing is to ensure that every unique path of a business process or logic behaves as per documented specifications. Each test case contains clearly defined inputs and expected results, enabling developers to detect inconsistencies or deviations efficiently. By guaranteeing that each component functions as intended, unit testing lays the groundwork for smoother integration, reducing the risk of defects and enhancing overall system reliability.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

8.2 Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is

correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Integration testing plays a crucial role in the software development lifecycle by ensuring that individual components of an application, which have already passed unit testing, work together seamlessly as a complete system. Unlike unit testing, which focuses on verifying the functionality of individual units in isolation, integration testing is designed to test the interactions between those units once they are combined. The primary goal is to identify and resolve

issues that may not have beenevident during unit testing, particularly those arising from the integration of different modules or systems.

Integration tests are often event-driven, meaning they focus on simulating real-world user interactions or system events to observe how the components function when they are linked together. These tests typically assess high-level outcomes, such as the display and functionality of screens, fields, or user interfaces. For example, integration testing might involve verifying that data entered in one part of the application flows correctly to another component, ensuring that the integrated system functions as expected from a user’s perspective.

One of the key purposes of integration testing is to confirm that the combination of components behaves correctly and consistently. Even if individual components pass unit testing successfully, their interactions when integrated could reveal unforeseen problems. These problems can arise from mismatched data formats, incompatible interfaces, incorrect handling of shared resources, or issues with communication between different parts of the system. Integration testing helps to uncover these issues by verifying that the components interact as intended, ensuring that data flows seamlessly, and that the system functions as a whole.

Unlike unit testing, which focuses on ensuring each unit's internal logic is correct, integration testing is more concerned with how well the components work together when integrated. It aims to identify problems such as miscommunications between modules, errors in data sharing, and mismatches in expected outputs when the system undergoes more complex workflows. It can also detect issues related to the timing of events or the sequencing of operations, which may not be apparent when individual components are tested separately.

In integration testing, the scope can vary. It can be conducted incrementally, by integrating components progressively and testing each interaction as it occurs, or as a "big bang" approach, where all components are integrated at once and the entire system is tested. The former is generally more controlled and allows for easier detection of issues early on, while the latter may expose more widespread integration problems at once.

In summary, integration testing is essential for ensuring that all components of a software system work harmoniously when combined. By focusing on the interactions between modules, it identifies issues that might not be visible during unit testing and ensures that the overall system is

reliable, consistent, and functional. Effective integration testing leads to a more robust and stable software product, as it confirms that the system operates as intended when all its individual pieces come together

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

8.3 Functional testing

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals. Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input:identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

8.4 System Testing

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

System testing is a comprehensive phase of the software testing process that evaluates the entire integrated software system to ensure it meets the specified requirements and functions as intended. Unlike unit testing and integration testing, which focus on individual components and their interactions, system testing examines the complete software solution in its final, integrated form. It aims to verify that all components, modules, and subsystems work together cohesively, ensuring that the system meets both functional and non-functional requirements as outlined in the system specifications.

One of the primary goals of system testing is to ensure that the system behaves as expected under various conditions, producing predictable and known results. It tests the system as a whole, verifying that all requirements are met and that the software performs correctly across different scenarios, environments, and configurations. System testing typically encompasses a wide range of test cases, including functional testing (validating that the system performs the required tasks), performance testing (evaluating the system's speed and scalability), security testing (ensuring the system's robustness against vulnerabilities), and usability testing (assessing the system's userfriendliness).

An example of system testing is the configuration-oriented system integration test, which focuses on ensuring that the system works correctly with specific hardware or software configurations. This type of test checks whether the software is compatible with different system environments, configurations, or external systems it might need to interact with, such as databases, networks, or third- party services. It ensures that the system can adapt to various setups without introducing errors or inconsistencies, thus confirming that the system is flexible and robust enough for real-world deployment.

System testing is typically driven by the process descriptions and workflows defined in the system’s specifications. These descriptions outline the processes that the software must support, including the sequence of actions, data flow, and user interactions. During system
testing, these process flows are executed to ensure that each step of the process links correctly with the next, and that all integration points are functioning as expected. The integration points are critical touchpoints where different subsystems or external systems interact, and testing these points ensures that data, control, and communication flow smoothly throughout the system.

Emphasizing pre-driven process links means that system testing focuses on validating that the predefined process workflows are followed correctly. For example, if the system is designed to handle a particular business process, such as order processing, system testing ensures that the entire order process—from the initial input through to payment and shipping—is executed without errors. It ensures that data entered at one point in the process propagates correctly to the next step and that each subsystem involved in the process (e.g., inventory management, billing, and shipping) functions as expected in the integrated system.

In summary, system testing is a crucial step in the software development process that ensures the fully integrated system meets all requirements and functions as a cohesive, reliable solution. By testing the software in its entirety, system testing validates that the configuration is correct and produces the desired results across different scenarios. It also ensures that all processes and integration points work as expected, confirming that the software is ready for deployment and meets both user and technical specifications

8.5 White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

White Box Testing, also known as structural testing or clear-box testing, is a software testing method where the tester has full knowledge of the internal workings, structure, and logic of the software. Unlike black-box testing, which focuses solely on the system's inputs and outputs without regard to its internal operations, white-box testing involves examining the code, algorithms, and design decisions that form the core of the application. The tester's knowledge of the internal code is essential for identifying logical errors, security vulnerabilities, or areas of improvement that might not be easily detectable through external observation alone.

The primary objective of white-box testing is to verify that the internal functions of the software operate as expected. This involves testing individual components (such as functions, methods, or classes) at the code level to ensure that the system performs the intended operations correctly. The tester may create test cases based on knowledge of the code's structure, decision points, loops, and data flow. For example, they might test the code to ensure that all paths within

the program are executed, check for boundary conditions, or validate the handling of specific input values.

White-box testing is particularly valuable for detecting issues that may not be apparent through other testing methods. For instance, it can uncover logical errors, such as incorrect if-else conditions, flaws in data processing, or missing edge case handling. It is also used to evaluate the performance and efficiency of algorithms, ensuring that the code runs optimally and without unnecessary resource consumption. Additionally, it can help identify security vulnerabilities, such as insecure code or potential points of failure where malicious users might exploit the system.

One of the core principles of white-box testing is code coverage, which measures the extent to which the codebase is tested. Testers aim to achieve high code coverage by creating test cases that exercise every possible execution path, condition, loop, or statement in the software. This ensures that all parts of the code are thoroughly tested, and potential defects are detected early in the development process. Common techniques used in white-box testing to achieve high code coverage include statement coverage, branch coverage, path coverage, and condition coverage, each of which focuses on different aspects of the code's logic.

White-box testing is often conducted by developers or testers who are familiar with the system's internal structure and design. Since it requires access to the source code, it is typically performed during the earlier phases of development, such as unit testing or integration testing. This makes it an effective method for catching issues early before the software is integrated or released to end users.

However, white-box testing has its limitations. It requires deep knowledge of the codebase, and its effectiveness is heavily reliant on the tester's understanding of the software's internal logic. It may also be time-consuming, especially for large and complex systems. Moreover, while it can detect errors in code structure and logic, it may not fully account for issues related to the software's interaction with the external environment, such as user experience,
system configuration, or performance under real- world conditions.In summary, white-box testing is a valuable technique that allows testers to examine and validate the internal components of a software application. By having insight into the software's code, structure, and logic, white-box testing ensures that the system functions as intended, performs efficiently, and is free from critical

flaws. It is particularly useful for detecting hidden defects that cannot be uncovered through blackbox testing and plays a crucial role in ensuring the quality and reliability of the software.

8.6 Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

Black Box Testing is a software testing method where the tester focuses entirely on the functionality of the application, without any knowledge of its internal structure, code, or logic. In this approach, the software is treated as a "black box," meaning the tester does not have visibility into the inner workings of the system. The primary objective of black-box testing is to verify whether the software behaves as expected based on the specified requirements and user needs, by examining the system’s outputs in response to various inputs.

Unlike white-box testing, where the tester has access to the source code and internal design of the application, black-box testing is based purely on the software’s external behavior. The tester is unaware of how the system processes data, and instead focuses on testing the system’s functionality from an end-user perspective. For example, in a web application, black- box testing would involve checking whether the user can successfully input data into a form, whether the correct output is displayed after submitting, and whether any errors are appropriately handled.

Black-box tests are typically derived from definitive source documents, such as the software’s specification, requirements document, or user stories. These documents outline the expected behavior of the system, which serves as the basis for test case creation. The goal is to ensure that the software meets the functional requirements and performs the intended tasks under various conditions. This approach is especially useful for validating that the system provides the correct outputs in response to a variety of inputs, such as user actions, system events, or external data sources.

In black-box testing, the tester does not need to understand the inner workings of the system or the code behind it. Instead, the focus is on providing different inputs and checking if the software produces the correct, expected outputs. This can involve testing individual features, such as login functionality, or performing end-to-end tests that simulate real-world usage scenarios. For example, the tester might input various data into a form to see if the system correctly validates the input and processes it according to the specified rules, such as ensuring that email addresses follow a proper format or that numeric values are within a specified range.

One of the main benefits of black-box testing is that it can be performed by individuals who may not have any programming knowledge, as it does not require insight into the system's internal code. This makes black-box testing a great approach for user acceptance testing (UAT) or quality assurance (QA), where the goal is to ensure the system works correctly from the user's perspective. Additionally, because the tester is focused on the system's behavior rather than its internal structure, black-box testing helps to validate whether the software meets the needs of the end-user and is easy to use.

However, black-box testing does have some limitations. Since the tester has no knowledge of the internal workings of the system, there is a risk of missing potential issues related to the code's logic, structure, or performance. For example, it may not identify bugs that are caused by improper handling of certain data types, security vulnerabilities, or inefficient algorithms. Additionally, black- box testing may not cover all edge cases or paths through the application, especially if the test cases are not comprehensive enough.

Overall, black-box testing is an essential method for validating the functionality, usability, and correctness of a software application. It ensures that the software delivers the expected results based on user inputs and meets the requirements outlined in the design documentation. By focusing on external behavior and outcomes, black-box testing complements other testing techniques like white-box testing and integration testing, providing a well-rounded approach to software validation and quality assurance.

8.7 Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

User Acceptance Testing (UAT) is a critical phase in any project, serving as the final checkpoint to ensure the system meets functional and business requirements. It involves end users, stakeholders, or representatives of the target audience rigorously testing the system in real- world scenarios. The primary purpose of UAT is to validate that the system aligns with the requirements outlined during the initial phases of the project, confirming its readiness for deployment. It ensures the product is functional, user-friendly, and integrates seamlessly with business processes.

End-user participation is crucial during UAT as it allows for realistic testing scenarios, usability feedback, and validation of workflows. By engaging users, organizations can uncover gaps or discrepancies that may have been overlooked in earlier testing phases. UAT also builds user confidence in the system, ensuring it meets their needs and expectations. The testing process typically involves preparing detailed test cases, executing them in a production-like environment, documenting outcomes, resolving any identified issues, and obtaining formal sign-off from users.

The benefits of UAT include reduced risk of deployment failures, increased user satisfaction, and cost savings by addressing defects before production. However, challenges such as time constraints, scope creep, and lack of user training can arise.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

9. HARDWARE AND SOFTWARE SPECIFICATIONS

The project aimed to improve the usability and user experience (UX) of certain applications by analyzing their design and making key changes to ensure they were more user- friendly. One of the most important aspects of this process was enhancing the navigation flow between screens. Effective navigation is essential for any application, as it determines how easily users can move from one part of the application to another. By ensuring that navigation was intuitive and wellordered, users could access features and information without confusion or frustration. The goal was to streamline the process so that users could perform tasks quickly and efficiently, without having to navigate through unnecessary steps.

Reducing the amount of typing required by the user was another critical aspect of the project. Excessive typing can be cumbersome and can slow down the user's interaction with the application, especially on mobile devices or for users with limited typing skills. To address this, the design focused on minimizing text input by incorporating features such as auto-fill, drop- down menus, and selection options where applicable. By using these design patterns, the application could offer a more seamless and faster experience, allowing users to complete tasks with fewer manual inputs. This not only improved usability but also reduced the likelihood of user errors, further enhancing the application's overall effectiveness.

In addition to improving navigation and reducing typing, another important consideration was making the application more accessible to a wider range of users. This was achieved by choosing to develop a browser-based version of the application that would be compatible with most modern web browsers. By ensuring compatibility across popular browsers, such as Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge, the application could be accessed from different devices and platforms without requiring users to install additional software. This cross- browser compatibility not only expanded the reach of the application but also ensured that users could access it regardless of their preferred browser or operating system, making the application more versatile and user-friendly in a broader context.

In summary, the project focused on improving the design and usability of the application by refining navigation, minimizing typing requirements, and ensuring accessibility through browser compatibility. By addressing these key areas, the application became easier to use, more efficient, and more widely accessible, ultimately providing a better experience for users. These changes were

aimed at enhancing the application's overall functionality while also making it more intuitive and accessible to users across various platforms and devices

REQUIREMENT SPECIFICATION

Functional Requirements

- Graphical User interface with the User. **Software**

Requirements

For developing the application the following are the Software Requirements:

- Python
- Django

Operating Systems supported

- Windows 7
- Windows XP
- Windows 8

Technologies and Languages used to Develop

- Python

Debugger and Emulator

- Any Browser (Particularly Chrome)

Hardware Requirements

For developing the application the following are the Hardware Requirements:

- Processor: Pentium IV or higher
- RAM: 256 MB
- Space on Hard Disk: minimum 512MB

10. SOFTWARE ENVIRONMENT

10.1 PYTHON

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. As an interpreted language, Python has a design philosophy that emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords). It also has a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java. Python provides constructs that enable clear programming on both small and large scales. Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is opensource software and follows a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object- oriented, imperative, functional, and procedural, and has a large and comprehensive standard library

Python is a versatile and widely-used programming language known for its simplicity, readability, and ease of use. It is a **general-purpose language**, meaning it can be applied to a wide variety of applications, including web development, data analysis, artificial intelligence, machine learning, scientific computing, and more. Python's design and features make it particularly appealing to both beginners and experienced developers.

Key Features of Python

1. **Interpreted Language**

   Python is an interpreted language, meaning code written in Python is executed line by line, rather than being compiled into machine language before execution. This allows for greater flexibility, ease of debugging, and quick prototyping of programs.

2. **Interactive and High-Level**

   Python provides an **interactive shell** where developers can write and test code snippets directly. Being a high-level language, Python abstracts many of the complex details of the machine, allowing developers to focus on programming logic rather than hardware-specific instructions.

3. **Object-Oriented and Multi-Paradigm Support**

   Python fully supports **object-oriented programming (OOP)** principles, enabling developers to model real-world entities using classes and objects. It also supports other paradigms, including imperative (step-by-step instructions), functional (functions as first-class citizens), and procedural programming, giving flexibility to developers.

4. **Emphasis on Readability**

   One of Python's standout features is its focus on **code readability**. The language enforces clean, readable code through its use of whitespace indentation to define code blocks, rather than relying on curly braces ({}) or keywords. This makes Python code more intuitive and easier to maintain.

5. **Concise Syntax**

   Python's syntax allows developers to express concepts in fewer lines of code compared to languages like C++ or Java. For example, a simple task like reading a file or looping through a list can be achieved with minimal code in Python.

6. **Dynamic Typing and Automatic Memory Management**

   Python features a **dynamic type system**, meaning variables are not explicitly declared with a type, and their type is determined at runtime. Additionally, Python manages memory automatically through a built-in garbage collector, freeing developers from manually handling memory allocation and deallocation.

7. **Cross-Platform Availability**

   Python interpreters are available for most operating systems, including Windows, macOS, Linux, and more. This ensures that Python programs are highly portable and can run on different platforms without modification.

8. **Extensive Standard Library**

   Python boasts a **comprehensive standard library** that provides pre-built modules and functions for various tasks, including file I/O, string manipulation, regular expressions, networking, and data serialization. This reduces the need for writing code from scratch and accelerates development.

CPython and Open-Source Development

The **reference implementation of Python** is called **CPython**, written in C. CPython is open- source software, meaning its source code is freely available, and it is maintained by the global Python community. This implementation is managed by the **Python Software Foundation (PSF)**, a non-profit organization responsible for overseeing Python's development and promoting its use. The open- source nature of Python fosters innovation, as developers worldwide can contribute to its development, create libraries, or build variants of the language tailored to specific use cases.

Python's Flexibility in Application Development

Python's ability to support both **small-scale scripts** and **large-scale applications** makes it incredibly flexible. Developers can use Python for tasks ranging from automating mundane tasks to building complex systems like web servers, data analysis pipelines, or AI-powered applications.

Conclusion

Python's unique combination of simplicity, versatility, and powerful features has made it one of the most popular programming languages in the world. Its widespread adoption across industries, thriving community, and extensive ecosystem of libraries and frameworks continue to drive its growth and relevance in modern software development. Whether for beginners learning to code or experts building cutting-edge applications, Python remains a go-to language for many developers.

10.2 DJANGO

Django is a high-level Python web framework designed to promote rapid development and clean, pragmatic design. Developed by experienced developers, it handles much of the complexity involved in web development, allowing you to focus on building your application without the need to reinvent the wheel. It is free and open-source.

Django's main objective is to simplify the creation of complex, database-driven websites. It emphasizes reusability and "pluggability" of components, rapid development, and follows the principle of "Don't Repeat Yourself" (DRY). Python is used throughout the framework, including for settings files and data models.

.

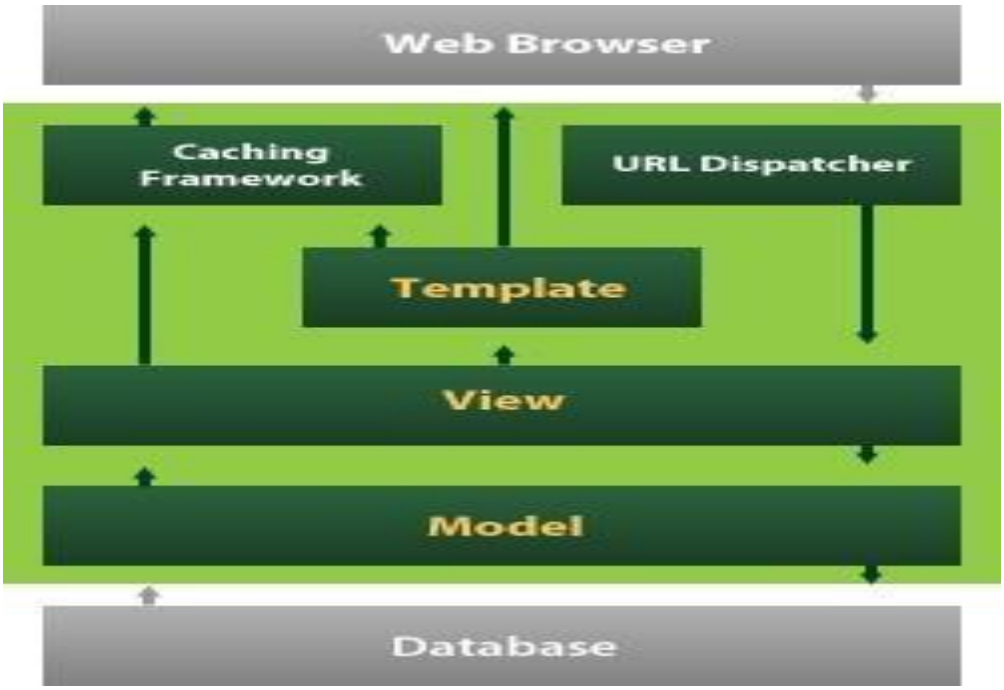


Fig 10.2 Django framework

Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models.

Django provides a built-in, optional administrative interface that is a powerful tool for managing and interacting with the data in your web application. This interface allows administrators or users with the right permissions to perform common database operations, such as create, read, update, and delete (CRUD), directly from the web interface without needing to write custom code for these actions.

What sets Django's admin interface apart is its ability to be generated dynamically using introspection. Introspection refers to Django's ability to examine the database models defined in your application and automatically generate a user-friendly admin interface to interact with these models. The admin interface is constructed based on the structure of your models, and it provides a simple yet highly functional interface to view, edit, add, or delete data records stored in the database. This makes the admin interface an essential tool for administrators or content managers

who need to manage the data without needing to interact directly with the underlying

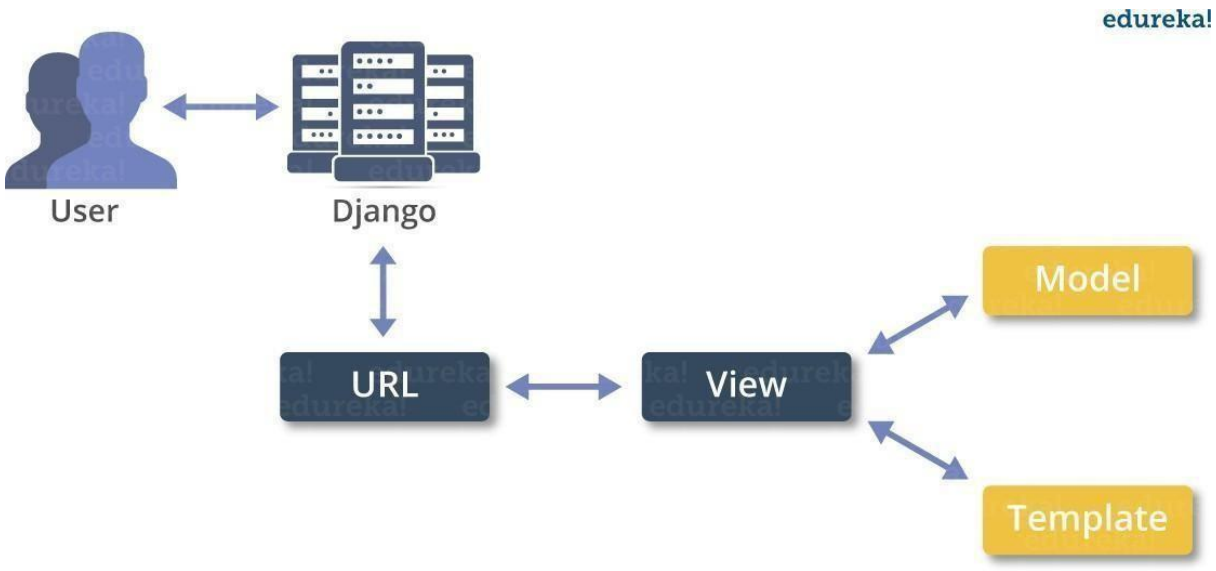
database or code.

To configure this administrative interface, developers use **admin models**, which are Python classes that define how the data should be displayed and managed in the admin interface. By registering models with the Django admin site, developers can customize the layout, search functionality, filtering options, and display of individual fields. For example, you can configure the admin interface to display data in tables, offer options for sorting and filtering records, and even create custom forms for data input or editing.

In addition to the basic CRUD functionality, Django's admin interface supports more advanced features, such as user permissions and access control. You can specify who has access to which models, which actions they can perform, and even limit the visibility of certain data fields depending on the user's role. This ensures that the right people have the right level of access to the system's data and helps maintain the security and integrity of the application.

Django's admin interface is a great example of how the framework promotes rapid development and reduces the time required to implement essential features. With minimal configuration, developers can quickly generate a fully functional, customizable admin panel that can handle most of the administrative tasks needed for a web application. This "out-of-the-box" feature significantly speeds up development, especially when creating data-driven websites or applications that require frequent data management

10.3 MVT Architectural Pattern in Django



10.3 MVT Architectural Pattern in Django

Django follows the Model-View-Template (MVT) architectural pattern, which is a variation of the Model-View-Controller (MVC) pattern. [1] This pattern promotes a clear separation of concerns, making your web applications more organized and maintainable.

Model The Model layer handles the data and logic of your application. It defines the structure of your data, such as tables and fields, and provides methods for interacting with the database. Django's Object-Relational Mapper (ORM) simplifies database interactions through Python code.

View The View layer processes user requests and generates responses. It interacts with the Model layer to retrieve or modify data and then decides which template to render. Views are typically Python functions that handle incoming requests, process data, and return HTTP responses.

Template The Template layer is responsible for the presentation layer. It defines the HTML structure of your web pages, incorporating dynamic content from the View layer. Django's templating language allows you to create reusable templates with placeholders for dynamic data.

How it Works

1. **User Request:** A user interacts with your web application by making a request, such as clicking a link or submitting a form.
2. **URL Mapping:** Django's URL dispatcher matches the incoming request to a specific URL pattern, which is then routed to the appropriate View.
3. **View Processing:** The View handles the request, potentially fetching data from the Model layer.
4. **Template Rendering:** The View selects a suitable Template and passes the necessary data to it. The Template renders the HTML output, incorporating the dynamic data.
5. **Response to User:** The generated HTML response is sent back to the user's browser, displaying the requested content.

Benefits of the MVT Pattern

- **Clear Separation of Concerns:** Each layer has a distinct responsibility, making code more organized and easier to maintain.
- **Reusability:** Templates can be reused across multiple Views, promoting consistency and reducing code duplication.
- **Rapid Development:** Django's built-in tools and conventions streamline the development process.
- **Security:** Django provides robust security features like CSRF protection, SQL injection prevention, and XSS protection.

By understanding the MVT pattern, you can effectively structure and develop Django web applications that are efficient, scalable, and maintainable.

11. INPUT AND OUTPUT DESIGN

11.1 INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

input design plays a crucial role in system development as it establishes the connection between the information system and the user. It involves developing the specifications, procedures, and interfaces needed to collect, prepare, and input data into the system. The effectiveness of the input design significantly impacts the accuracy, efficiency, and usability of the overall system, as it determines how transaction data is prepared and entered for processing.

Objectives of Input Design

The primary objective of input design is to ensure that data is entered into the system accurately, securely, and efficiently while minimizing the complexity of the process for the user.
This involves:

- **Minimizing Input Errors**: By designing intuitive and error-resistant input mechanisms, the system can reduce the chances of invalid or incorrect data being entered.
- **Optimizing the Input Process**: Reducing redundant steps and unnecessary delays ensures that data input is streamlined.
- **Maintaining Security and Privacy**: Input methods should safeguard sensitive data while ensuring only authorized users can access or modify the information.
- **Enhancing Usability**: The input interface must be simple and user-friendly, guiding users through the process effortlessly.

Steps in Input Design

1. **Data Specification**

   The first step in input design is to determine what data needs to be collected for the system to function effectively. This involves identifying the:

   - **Type of data**: Numerical, textual, or a combination.
   - **Source of data**: Whether it comes from user input, documents, or external systems.

2. **Data Arrangement and Coding**

   Input data must be arranged in a structured and standardized format to facilitate processing. Coding ensures consistency and enables the system to interpret the data correctly. Examples include:

   - Using predefined formats like dates (DD/MM/YYYY) or IDs (e.g., alphanumeric codes). ○ Employing drop-down lists, radio buttons, or checkboxes to limit user input to valid options.

3. **Dialog Design**

   The input design must include a clear and interactive dialog to guide users. This ensures that users understand what is required at each step. Good dialog design involves:

   - Providing instructions or tooltips to assist users. ○ Designing clear labels, prompts, and error messages.
   - Ensuring the navigation between input fields is intuitive and logical.

4. **Input Validation**

   Validation mechanisms are crucial to maintaining data integrity. They ensure that only valid, complete, and accurate data is entered. Common input validation methods include:

   - **Range checks**: Ensuring numeric inputs fall within a specified range.
   - **Format checks**: Verifying that input adheres to a specific format, like email addresses.
   - **Consistency checks**: Cross-referencing input data with existing records to avoid duplication or conflicts.

5. **Error Handling**

   When errors occur during input, the system must provide clear guidance on how to correct them. This involves:

- Displaying descriptive error messages that indicate the issue and suggest a resolution.
- Offering users the ability to edit their input without restarting the process.
- Logging errors for system administrators to analyze and address recurring issues.

Key Considerations in Input Design

To achieve a successful input design, the following aspects are emphasized:

1. **Volume of Input**: The design should limit the amount of input required from users, asking only for essential data to reduce their workload and minimize errors.
2. **Error Control**: Input methods should be equipped with features like auto-completion, default values, and validation checks to prevent incorrect data entry.
3. **Process Simplicity**: The input process should avoid unnecessary steps, providing a straightforward and seamless experience for users.
4. **Security and Privacy**: Input mechanisms must safeguard user data through encryption, access controls, and secure transmission protocols to protect sensitive information.
5. **Ease of Use**: The design should cater to users of all skill levels, ensuring that the interface is intuitive and accommodating.

Practical Implementation

For example, consider an online banking system where users must input data to transfer funds. The input design for this system would:

- Include clearly labeled fields for account numbers, transfer amounts, and recipient details.
- Use dropdown menus to select bank branches, reducing manual typing errors.
- Validate inputs to ensure the account number format is correct and the transfer amount is within the account's balance.
- Provide real-time error messages if incorrect information is entered, such as invalid account numbers.
- Offer privacy by masking sensitive data like account numbers and using secure connections for data transmission

- What data should be given as input?
- How the data should be arranged or coded?
- The dialog to guide the operating personnel in providing input.
- Methods for preparing input validations and steps to follow when error occur.

OBJECTIVES

Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.

It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.

When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in

maize of instant. Thus the objective of input design is to create an input layout that is easy to follow

11.2 OUTPUT DESIGN

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.

.Select methods for presenting information.

Create document, report, or other formats that contain information produced by the system.

The output form of an information system should accomplish one or more of the following objectives.

- Convey information about past activities, current status or projections of the Future.
- Signal important events, opportunities, problems, or warnings.
- Trigger an action.
- Confirm an action.

Output design is a critical phase in system development, as it defines how information is presented to users and other systems. The quality of the output directly impacts user satisfaction and the effectiveness of the system in achieving its goals. An effective output not only meets user requirements but also presents information in a clear, concise, and actionable format, making it an essential tool for decision-making and system communication.

Importance of Output Design

The output of any system represents the end result of processing, making it the most direct source of information for users. A well-designed output enhances the user experience, helps in making informed decisions, and establishes a reliable relationship between the system and its users. For this reason, the design of system outputs must be approached with care and strategic planning.

Key Considerations in Output Design

1. **Meeting User Requirements**

   The primary goal of output design is to ensure that the output aligns with the needs of the end user. This involves understanding what information users require, in what format, and for what purpose. Outputs should be tailored to their specific roles, whether for management, operations, or analysis.

2. **Clarity and Accessibility**

   Outputs must present information in a manner that is easy to understand and interpret. Clear labeling, logical organization, and intuitive formats ensure that users can quickly extract the information they need.

3. **Medium of Output**

   Outputs can be displayed in various forms, such as:

   - **On-Screen Display**: For immediate needs, such as dashboards, alerts, or real-time data.
   - **Hard Copy**: For physical records or official documentation, such as reports or invoices.
   - **Electronic Formats**: For distribution via email, cloud sharing, or integration with other systems.

4. **Efficiency and Intelligence**

   Intelligent output design reduces unnecessary complexity and emphasizes key information, helping users make faster and more informed decisions. Features like highlights, summaries, or interactive graphs can significantly improve usability.

5. **Actionability**

   Outputs should be actionable, meaning they not only convey information but also enable users to take appropriate steps based on that information, such as triggering an alert or confirming a process.

Steps in Designing Computer Output

1. **Identify Specific Outputs Needed**

   Begin by identifying the exact type and content of the output required to meet user and system requirements. This includes determining the data fields, calculations, and visualizations needed to fulfill the system's objectives.

2. **Select Presentation Methods**

   Choose the most effective way to present the information. For instance:

   - Use tables for detailed data comparisons.
   - Employ charts or graphs for visual representation of trends or performance metrics.
   - Implement alerts or notifications for time-sensitive information.

3. **Create Formats**

   Design appropriate formats for outputs, such as documents, reports, or interactive screens. Formats should be standardized to ensure consistency across different outputs and user interfaces.

4. **Incorporate Feedback Mechanisms**

Allow for user feedback on outputs to improve their relevance and usability. Iterative testing and revisions help refine the output design.

Objectives of Output Design

A well-designed output form achieves the following objectives:

1. **Convey Past, Present, and Future Information**

   Outputs should provide insight into historical data, the current status, and future projections. For example, a sales report can summarize past performance while forecasting future trends.

2. **Signal Key Events or Warnings**

   Outputs should highlight critical information, such as errors, opportunities, or alerts. For example, an inventory management system might generate a notification when stock levels are low.

3. **Trigger Actions**

   Outputs can serve as prompts for specific actions. For instance, a payroll system's output might signal that salaries need to be disbursed.

4. **Confirm Actions**

   Outputs can validate that a process has been completed successfully, such as generating a receipt after a transaction.

Example Scenario: Bank Statement Output

Consider a bank system output that generates monthly statements for customers. A good output design for this statement might:

- Present a summary of account activity (past transactions).
- Include warnings for low balances or suspicious transactions (signal important events).
- Offer actionable insights, such as suggested savings plans or loan offers (trigger actions).
- Confirm the accuracy of recent deposits and withdrawals (confirm actions).

12. SCREEN SHOTS

Index page

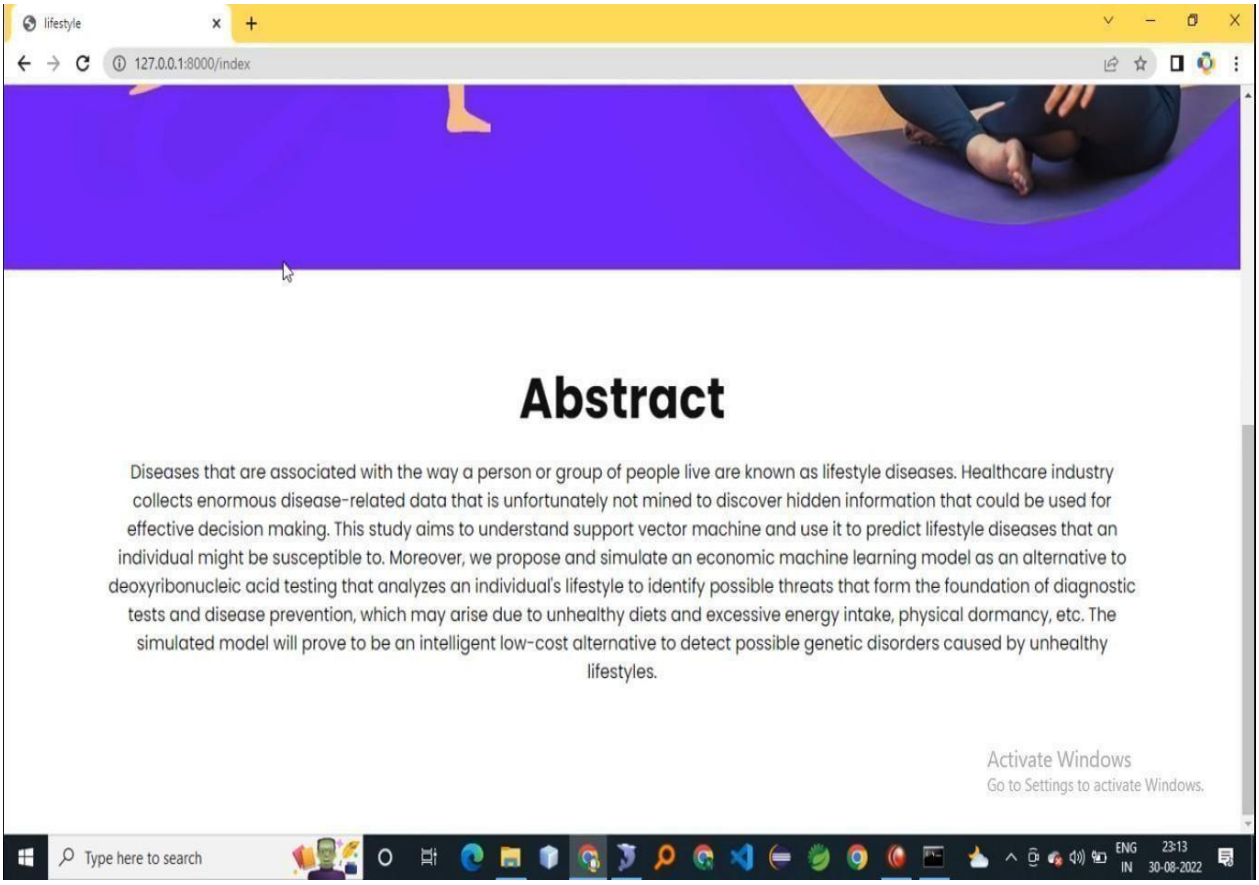


Fig 12.1 Index page

Users login

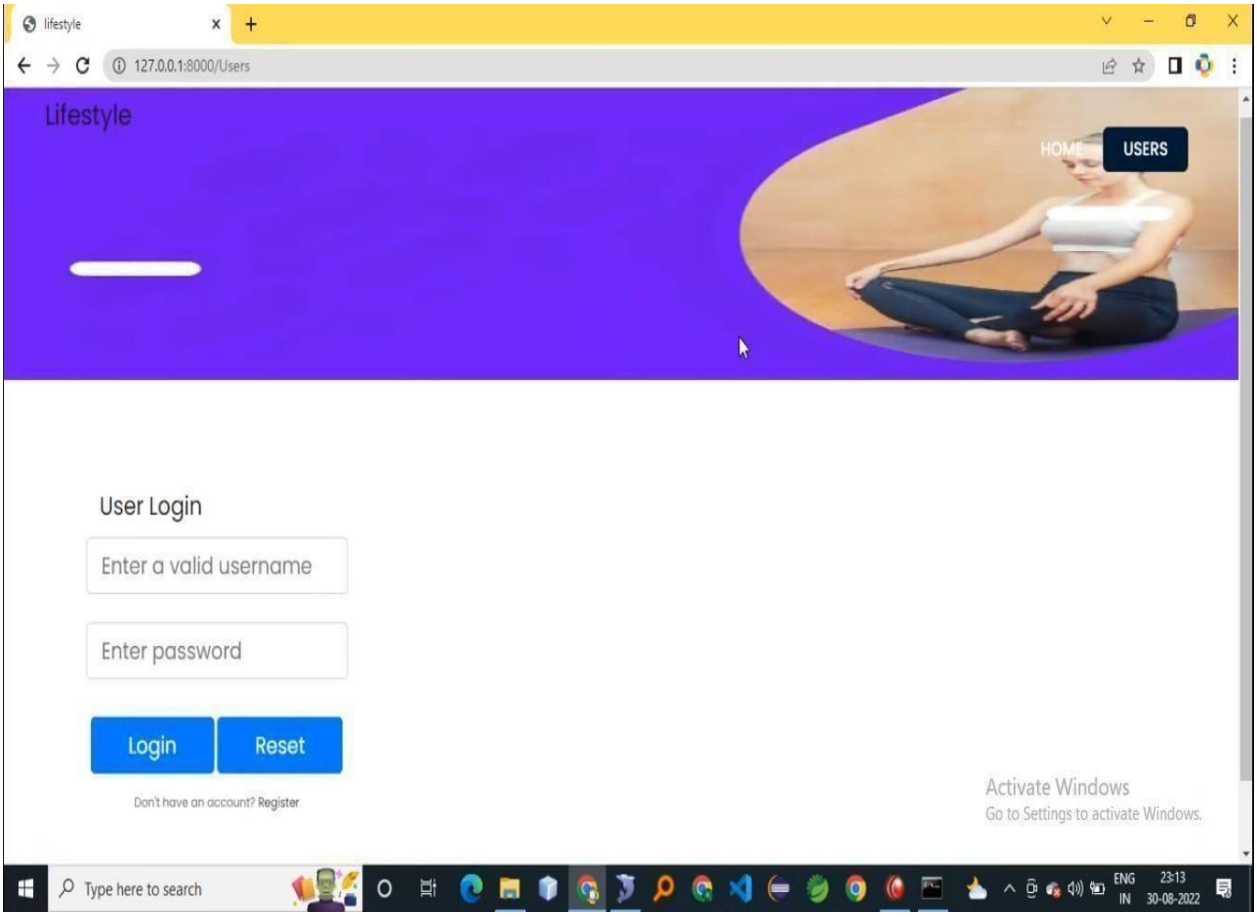


Fig 12.2 Users login

Registration

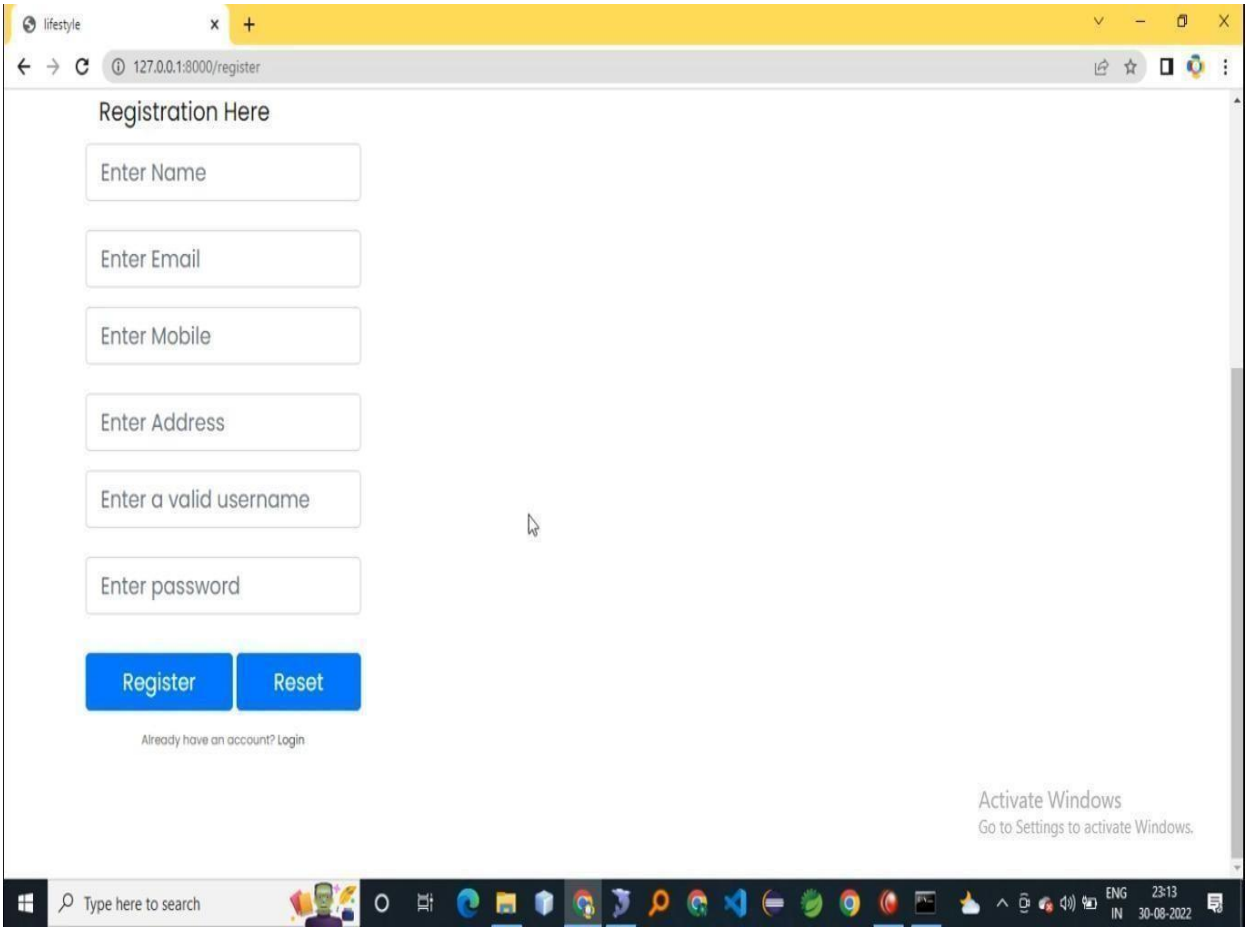


Fig 12.3 Registration

User home screen

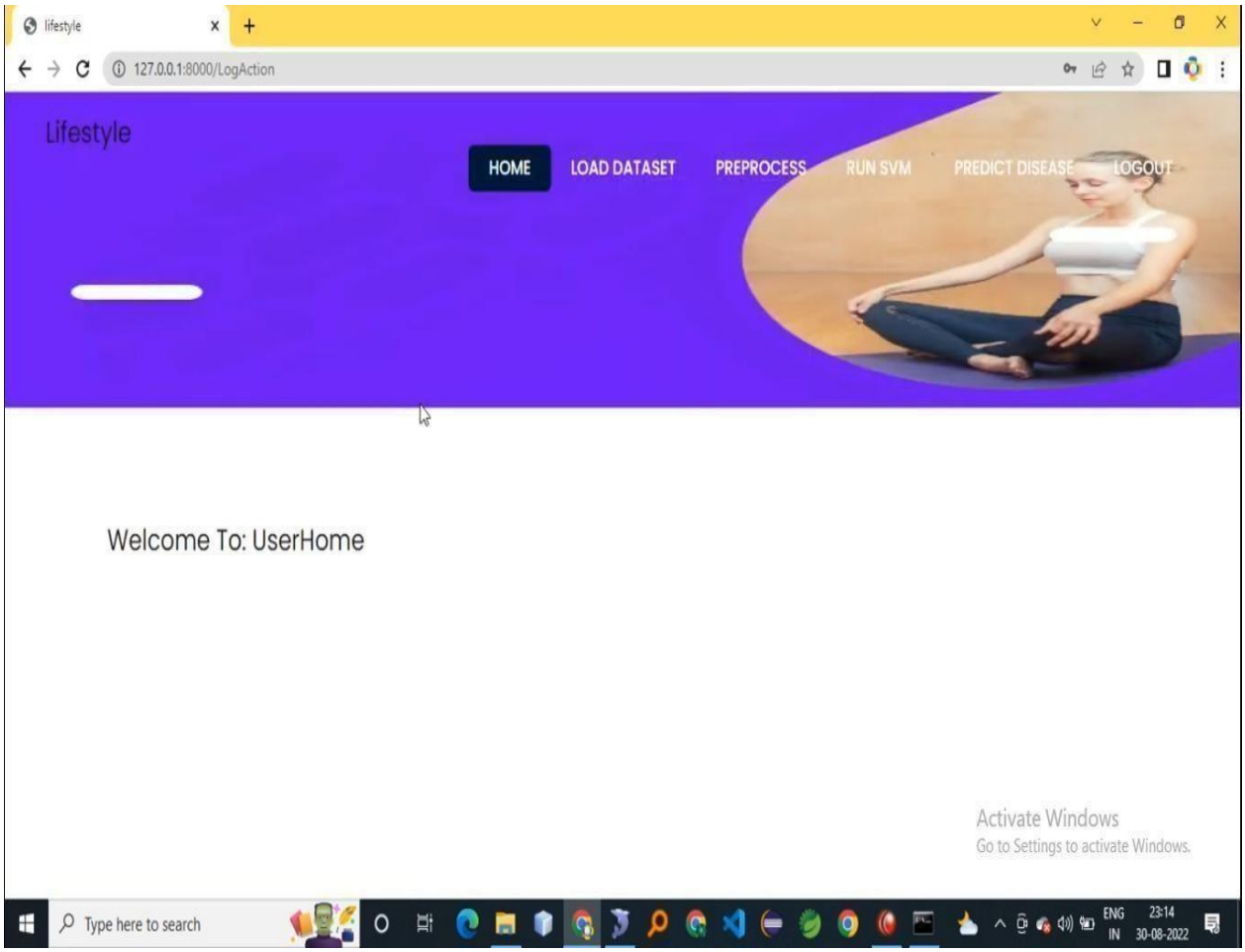


Fig 12.4 User home screen

Load dataset

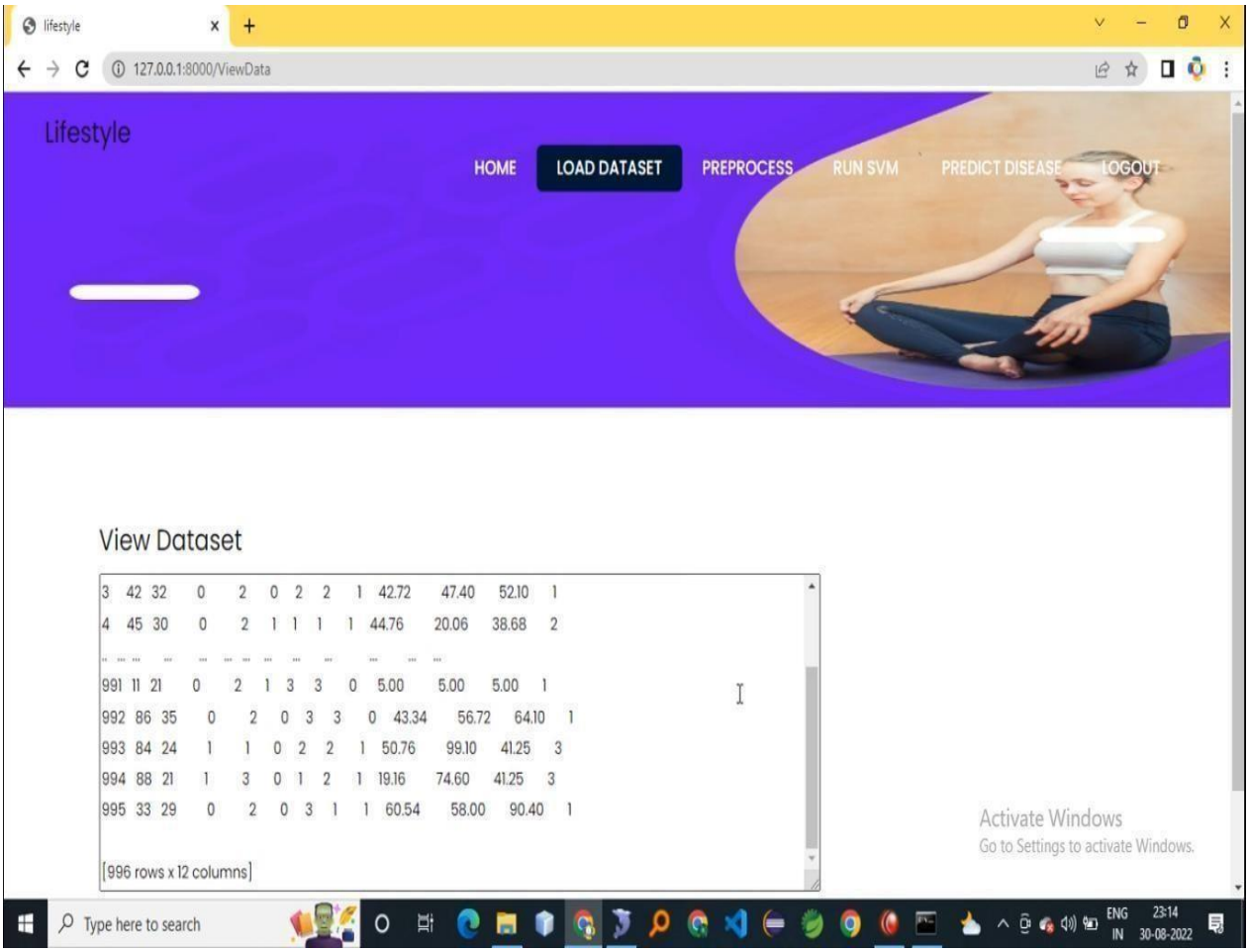


Fig 12.5 Load dataset

Preprocess screen

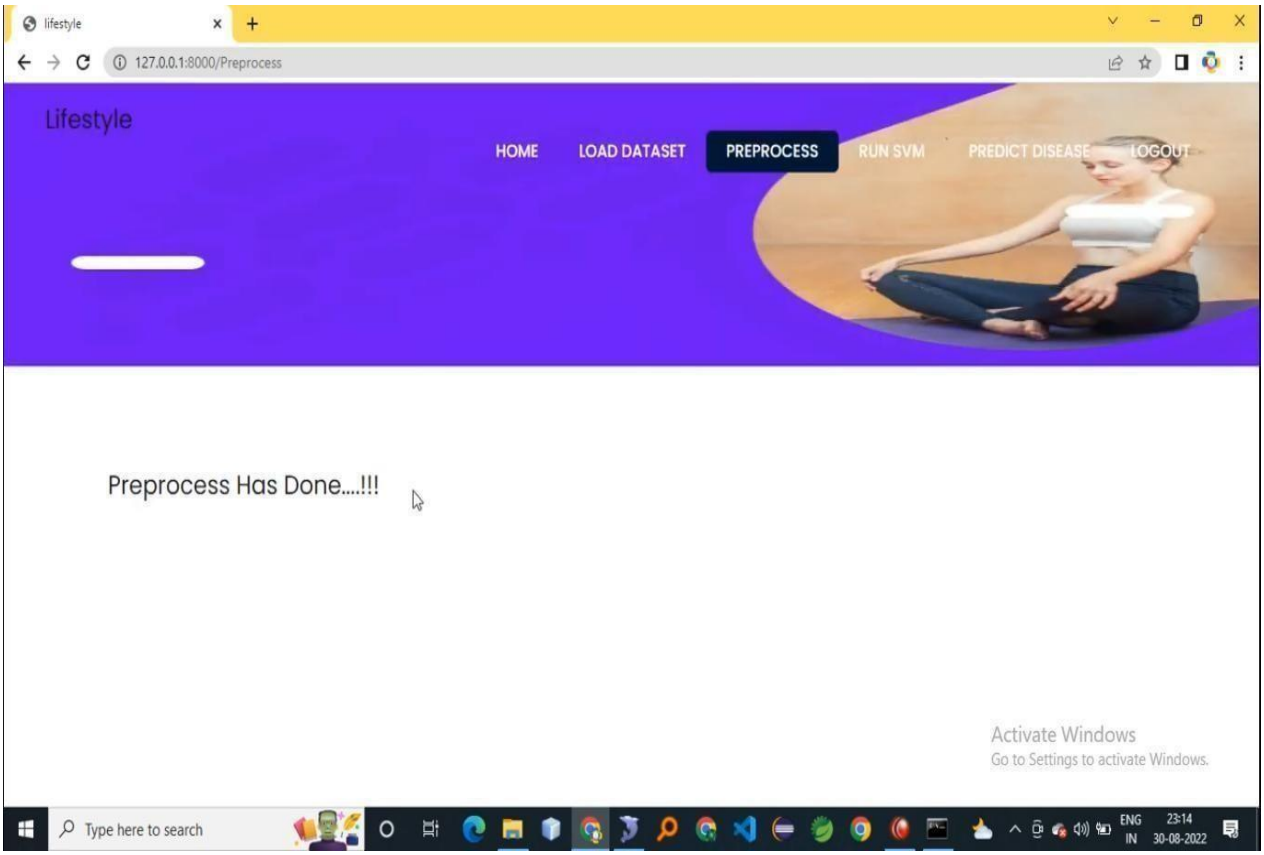


Fig 12.6 Preprocess screen

Run Decision Tree Screen

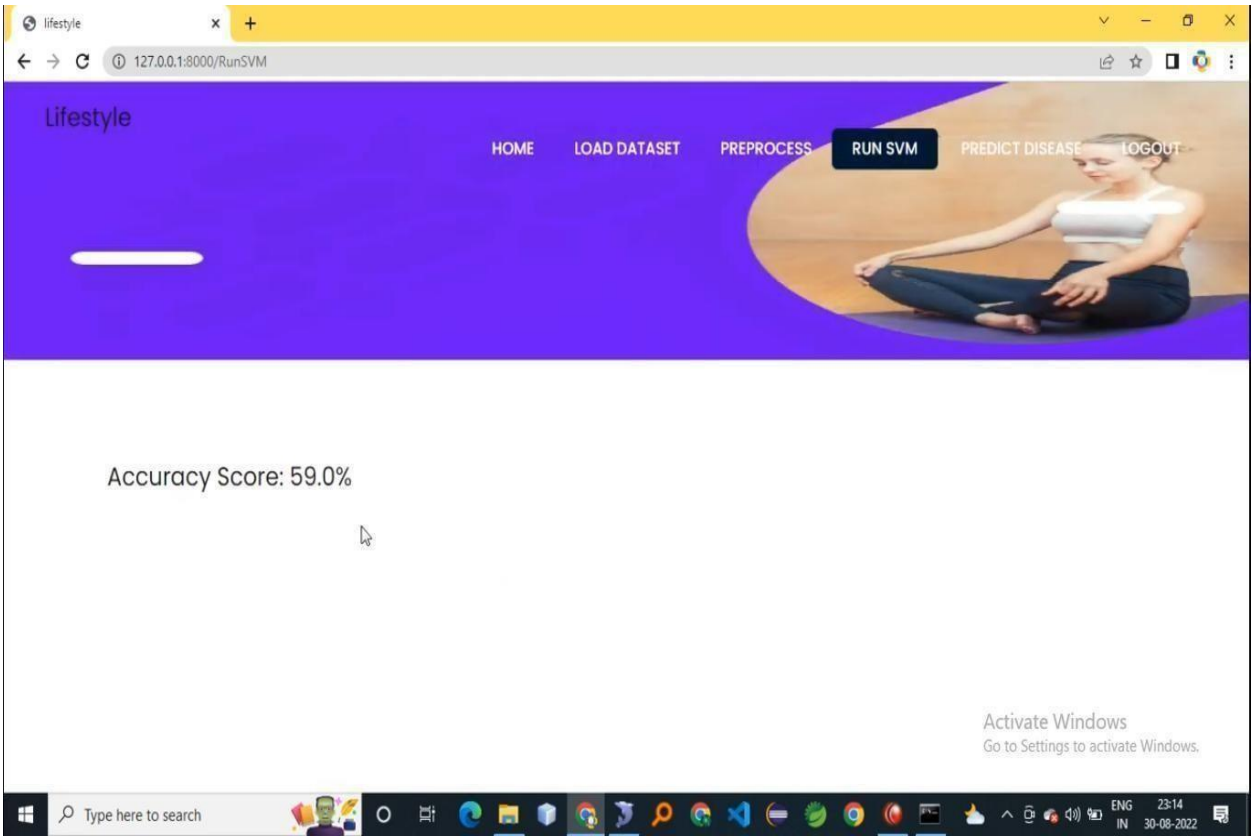


Fig 12.7 Run Decision Tree Screen

Predict disease

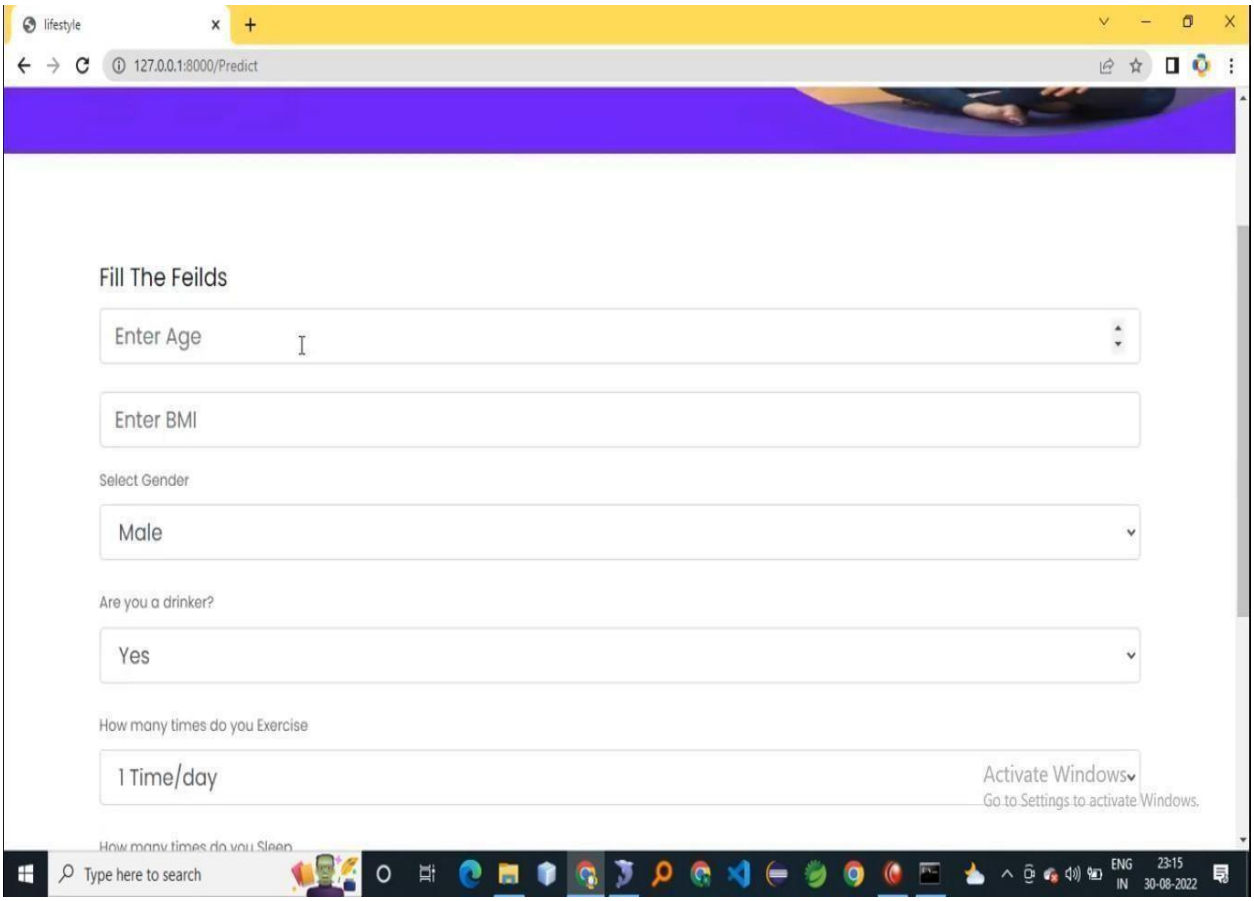


Fig 12.8 Predict disease

Disease predicted screen

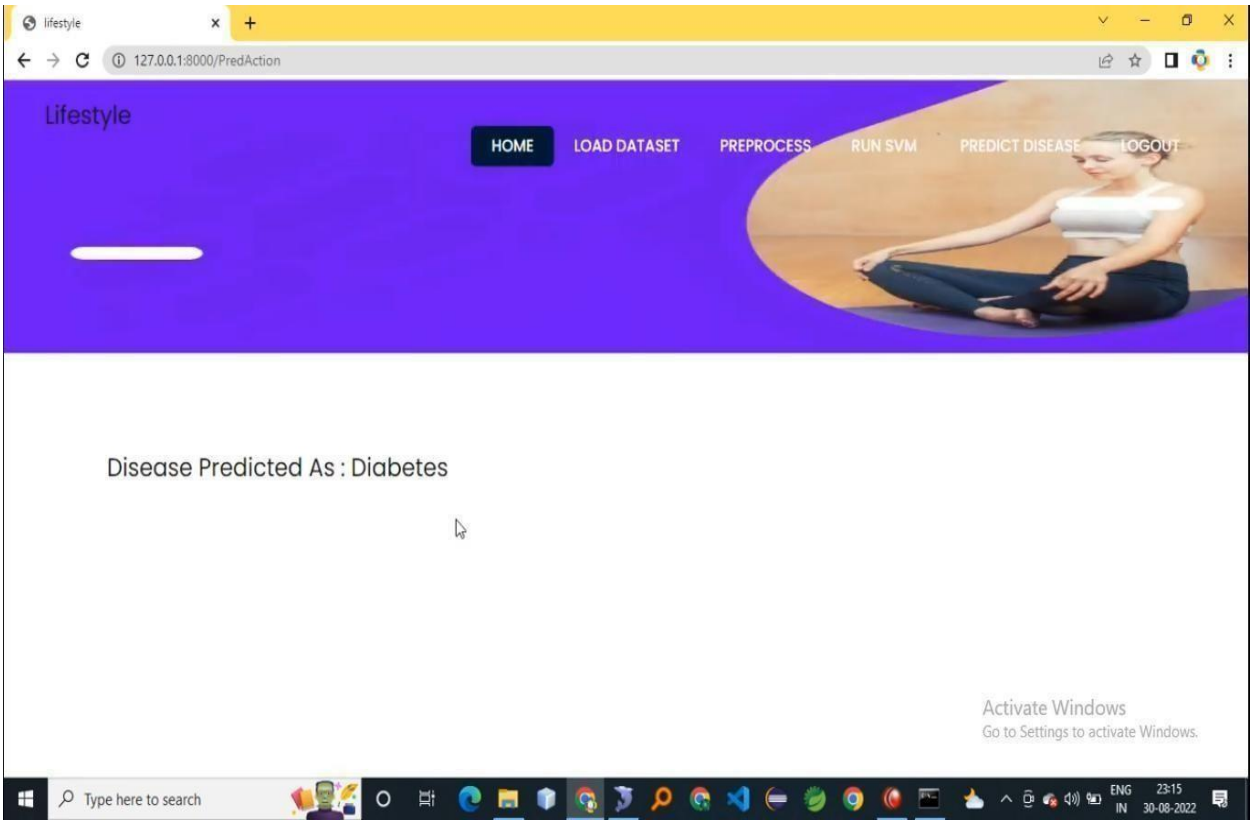


Fig 12.9Diseasepredicted Screen

13. CONCLUSION

ML being an essential CS application is used for predicting results given target input parameters and is being widely used for improving human lifestyle in several ways. Complex disorders—also known as polygenic—are caused by simultaneous effects of more than one gene often in a complex interaction with environment and lifestyle factors, which implies that if a parent has a particular disorder, it does not necessarily mean that a child would develop the same. However, there could be a possibility of high risk of developing the disorder (i.e., genetic susceptibility), and for such a possibility where it cannot be a sure occurrence but risk prevails, the proposed model would provide a detailed report of alterations in an individual's lifestyle such as maintaining a healthy weight, and sugar levels may be able to reduce risk in case of genetic predisposition known that genetic makeup cannot be altered.

Machine learning (ML), a pivotal application of computer science, is increasingly utilized to predict outcomes based on target input parameters, significantly improving human lifestyles in various domains. One of its promising applications lies in healthcare, where ML aids in understanding and addressing complex disorders. These polygenic disorders arise from the combined effects of multiple genes interacting intricately with environmental and lifestyle factors. Unlike single-gene disorders, the presence of such conditions in a parent does not guarantee their occurrence in offspring, though it may indicate a higher genetic susceptibility.

This genetic susceptibility highlights the importance of identifying risks early and taking preventive actions. The proposed model aims to bridge this gap by providing a detailed analysis of an individual's lifestyle patterns and offering actionable recommendations. While genetic makeup cannot be changed, adjustments such as maintaining a healthy weight, regulating sugar levels, and adopting other healthy habits could significantly reduce the risk of developing certain disorders. These insights equip individuals with the tools to manage their health proactively.

14. FUTURE ENHANCEMENT

The model would take into account climatic conditions and pollution levels and rank cities and suburbs with an ideal environment as to the precautionary measures that an individual could take making the model more content specific, accessible, and flexible in terms of customization. The fact that deep learning (DL) is overtaking ML algorithms in terms of accuracy would suggest the possibility of SVM being replaced by DL in the near future.

The proposed model extends its analysis by factoring in climatic conditions and pollution levels, providing a comprehensive ranking of cities and suburbs based on their environmental suitability for healthy living. This feature allows individuals to identify ideal locations and adopt precautionary measures tailored to their specific circumstances. By incorporating these environmental dimensions, the model becomes more content-specific and accessible while offering flexible customization options to cater to diverse user needs.

Incorporating such elements ensures that the model is not only predictive but also practical in promoting healthier lifestyles. Users can leverage its insights to make informed decisions about relocation or adjustments to their daily routines. This adaptability enhances its appeal as a personalized tool for disease prevention and mitigation, fostering greater engagement with preventive healthcare strategies.

The rising prominence of deep learning (DL) in machine learning signifies an exciting shift in predictive accuracy and capability. While support vector machines (SVM) serve as the backbone of this study, DL's superior ability to handle complex datasets suggests it may soon replace traditional ML approaches like SVM. This transition promises to unlock new possibilities for refining the model, delivering even more precise and actionable insights for addressing lifestylediseases.

15. BIBLIOGRAPHY

[1] Sharma, M. and Majumdar, P.K., 2009. Occupational lifestyle diseases: An emerging issue. Indian Journal of Occupational and Environmental medicine, 13(3), pp. 109–112.

[2] DNA Test Cost in India, Available [Online] https://www.dnaforensics.in/dna-test-costin- india/ [Accessed on June 27, 2018].

[3] Suzuki, A., Lindor, K., St Saver, J., Lymp, J., Mendes, F., Muto, A., Okada, T. and Angulo, P., 2005. Effect of changes on body weight and lifestyle in nonalcoholic fatty liver disease. Journal of Hepatology, 43(6), pp. 1060–1066.

[4] Pattekari, S.A. and Parveen, A., 2012. Prediction system for heart disease using Naïve Bayes. International Journal of Advanced Computer and Mathematical Sciences, 3(3), pp. 290–294.

[5] Anand, A. and Shakti, D., 2015. Prediction of diabetes based on personal lifestyle indicators. In Next generation computing technologies (NGCT), 2015 1st international conference on (pp. 673–676). IEEE.

[6] Kanchan, B.D. and Kishor, M.M., 2016. Study of machine learning algorithms for special disease prediction using principal of component analysis. In Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC), 2016 International Conference on (pp. 5– 10). IEEE.

[7] Kazeminejad, A., Golbabaei, S. and Soltanian-Zadeh, H., 2017. Graph theoretical metrics and machine learning for diagnosis of Parkinson's disease using rs-fMRI. In Artificial Intelligence and Signal Processing Conference (AISP), (pp. 134–139). IEEE.

[8] Milgram, J., Cheriet, M. and Sabourin, R., 2006. "One against one" or "one against all": Which one is better for handwriting recognition with SVMs?. Tenth International Workshop on Frontiers in Handwriting Recognition, La Baule (France), Suvisoft, 2006.

[9] Hossain, R., Mahmud, S.H., Hossin, M.A., Noori, S.R.H. and Jahan, H., 2018. PRMT: Predicting Risk Factor of Obesity among MiddleAged People Using Data Mining Techniques. Procedia Computer Science, 132, pp. 1068–1076.