A

Project Report

On

Predict My Health

[Submitted in partial fulfillment of the requirements for the degree of

Master of Computer Applications (MCA)]

Guided By

Anuja Niyogi (Conneqt Business Solutions)

SRM INSTITUTE OF SCIENCE & TECHNOLOGY,

NCR CAMPUS MODINAGAR

(Deemed to be University U/S – 3 of UGC Act. 1956)(F. 9-9/98-U.3)

Year-2025

SUBMITTED TO:-

Dr. Lalit Kishore Arora

Assistant Professor

MCA Department

SUBMITTED BY:-

Kirti Garg

RA2332241030047

MCA Section- A

DECLARATION

I **Kirti Garg**, Registration no. RA2332241030047, Master of Computer Applications(MCA) programme, SRM Institute of Science & Technology (Deemed to be University), NCR Campus, Modinagar-201204, Ghaziabad, batch of 2023-2025 do hereby solemnly declare that this Project work report is an original work of mine and this has not been submitted to any other institute/University towards any other degree/diploma.

Kirti Garg

RA2332241030047

Acknowledgment

I take this opportunity to express my deep sense of gratitude to all those who have contributed significantly by sharing their knowledge and experience in the completion of this project work.

My first word of gratitude is to **Dean Dr. R. P Mahapatra, Dean (S&H) Dr. Navin Ahlawat, HOD-MCA Dr. Vishu V. and Project Coordinator Dr. Rajeev Kumar Sharma** for providing this kind of opportunity and guidance throughout the project.

My heartful thanks to My Project Guide **Dr. Lalit Kishore Arora** without his continuation help the project would not have been materialized in the present form. I am thankful to him for providing me with necessary insights and valuable suggestions helped me at every step. We would like to take this opportunity to extend our warm thoughts to those who helped me in making this project a wonderful experience.

Last but not least; we would also like to thank our family and friends for their support an encouragement.

(Signature)

Kirti Garg

RA2332241030047

Date: _____

BONAFIDE CERTIFICATE

Registration no.: RA2332241030047

Certified to be the bonafide record of work done by **Kirti Garg** of IVth semester / IInd year MCA degree course in SRM INSTITUTE OF SCIENCE & TECHNOLOGY (Faculty of Science & Humanities), Delhi-NCR Campus, Modinagar-201204 for the **Department of Computer Applications,** in Project Work (PCA20P03L) during the academic year **2024-25.**

Dr. Rajeev Kumar Sharma
(Associate Professor)
Project Coordinator

Dr. Lalit Kishore Arora
(Assistant Professor)
Project Guide

Dr. Vishu V.
(Associate Professor)
HOD-MCA

Submitted for end semester project viva held on __/____/___ at SRM INSTITUTE OF SCIENCE & TECHNOLOGY, Delhi-NCR Campus, Modinagar.

Internal Examiner **External Examiner**

SRM Institute of Science & Technology,

NCR Campus, Modinagar

Date: …….

TO WHOM SOEVER IT MAY CONCERN

This is to certify that **Kirti Garg** is a bonafide student of MCA-II Year of this institute for the session 2023-25 and he has prepared **Project Report** (MCA- IVth semester) titled **Predict My Health** for partial fulfillment of **Master of Computer Applications (MCA)** affiliated to SRM Institute of Science & Technology, NCR Campus, Modinagar. She has worked to my supervision and her performance during the project has been satisfactory.

I wish her all the best for her future endeavors

Project Guide Name & Signature:

Dr. Lalit Kishore Arora

(Assistant Professor)

MCA Department

CERTIFICATE

CONNEQT
BUSINESS SOLUTIONS

QUESS
WINNING TOGETHER

TO WHOMSOEVER IT MAY CONCERN

It is to certify that **Kirti Garg** was associated with Conneqt Business Solutions India Pvt. Ltd, in the capacity of an Intern from **6/12/2024** to **20/3/2025**. As part of her contract, she was involved in helping team and client by providing IT help and error-solving support in **Conneqt Business Solutions Private Limited.**

Her performance in the time that she spent with us has been good!

Conneqt Business Solutions Private Limited

Anuja Niyogi
Associate Vice President |Human Resource
Conneqt Business Solutions

TABLE OF CONTENTS

ABSTRACT

In today's world, many people struggle with identifying early signs of diseases like diabetes and heart conditions. According to the World Health Organization (WHO), diabetes affects over 422 million people worldwide, while cardiovascular diseases (CVDs) account for nearly 17.9 million deaths annually making them the leading cause of mortality [1]. A significant challenge in managing these diseases is the lack of early detection, as symptoms often go unnoticed until the condition reaches a critical stage. Barriers such as delayed medical check-ups, expensive diagnostic tests, and low health awareness further contribute to the rising prevalence of these conditions. Early detection can reduce complications by up to 70% through timely lifestyle interventions and medical care. Lack of awareness, delayed check-ups, and expensive medical tests often lead to severe health complications. Conditions like diabetes and heart disease often go unnoticed until they reach a critical stage, leading to serious health complications.

To tackle this issue, we have designed Predict My Health, a user-friendly health prediction platform built using Python and Streamlit. This system allows users to enter basic health details and get an instant risk assessment for diabetes and heart disease. Unlike complex AI-based platforms, our tool is designed to be simple, lightweight, and easy to use for anyone, even those without technical knowledge. Additionally, existing prediction models sometimes lack accuracy and fail to consider real-world variations in patient symptoms.

Future improvements can focus on:

- Enhancing accuracy by integrating larger and more diverse medical datasets.
- Expanding disease coverage for a more comprehensive health assessment.
- Providing personalized insights by analyzing user history instead of just a single prediction.
- Refining the user interface and experience to ensure better accessibility, particularly on mobile devices.

While this system is not a substitute for professional medical advice, it serves as a first step in self-awareness and encourages users to seek proper medical consultation when needed.

By leveraging technology for early detection, Predict My Health aims to bridge the gap between individuals and preventive healthcare. With further research and refinement, this platform has the potential to significantly contribute to reducing the burden of lifestyle diseases on global healthcare systems.

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Predict My Health

The medical sector has greatly improved in predicting diseases with the aid of artificial intelligence (AI) and machine learning (ML). There are many applications and tools that help users calculate their health risks based on different medical parameters.

Cloud health analytics is a key element in contemporary health prediction. Health platforms use the cloud for safe storage and health information analysis. Real-time disease prediction is facilitated through AI models running on the cloud with electronic health records (EHRs) processing. Cloud-based models are easily accessible and scalable, yet risks to data security, privacy, and internet reliance also exist [2].

Mobile health applications and wearable devices have greatly enhanced real-time monitoring of health. Smartwatches and fitness bands with sensors monitor vital parameters like heart rate, blood pressure, and oxygen saturation. AI-based applications analyze this information to assess potential health hazards [3].

Even with these developments, there are still problems with current health prediction technology. The majority of AI systems are technical in nature and need to be installed and interpreted by technical individuals, thus excluding non-technical individuals. Certain models need huge medical datasets, which are not always easily accessible, resulting in limited access to data. There are accuracy problems with variations in data quality, and privacy risks are posed by cloud-based systems that may expose sensitive health data to security risks.

To tackle these concerns, Predict My Health provides an easy-to-use health forecasting platform. For everyone, it is coded in Python and Streamlit to make instantaneous health checks simple while keeping it easy to use. Unlike most modern health prediction systems based on cloud-hosted machine learning models and deep learning, which require high computing power, Predict My Health is coded in Python and Streamlit. This renders it lightweight and can be run on local machines. Most AI-based healthcare platforms are usually founded on large medical data and complex processing, which makes them difficult to use. Predict My Health simplifies this by enabling users to feed in simple health data and get immediate predictions, and it is therefore accessible to users without technical expertise.

Data privacy is also an important issue. Whereas patient information is retained by certain healthcare prediction programs on cloud servers, thus raising privacy issues, Predict My Health retains user information locally, ensuring individual health information is kept confidential and not disclosed outside.

Most prediction systems rely on deep learning models that demand a lot of labeled data. Predict My Health uses machine learning methods like logistic regression, which are appropriate for structured data and need less resource. Certain health platforms use mobile apps or wearable technology to track health in real time, but Predict My Health uses a web app that can be accessed on any browser-enabled device, making it more accessible. In addition, while most advanced models need.

CHAPTER 2

PROJECT MODULE DESCRIPTION

In today's fast-paced world, health monitoring and early disease detection have become crucial. This project aims to develop a Health Risk Prediction System that utilizes machine learning to predict the likelihood of diabetes and heart disease based on user inputs. By integrating predictive analytics with a user-friendly interface, this system empowers individuals to assess their health risks and take proactive measures. The project consists of three main modules: Diabetes Prediction, Heart Disease Prediction, and a Frontend Interface for user interaction.

Key Steps in the Module:

2.1. Diabetes Prediction Module

- **Objective:**

  To build a model that can predict diabetes risk based on user inputs.

- **Implementation Details:**

  - **Dataset Used:** Dataset from UCI Machine Learning repository where information of patients from of Sylhet Diabetes Hospital is available.
  - **Preprocessing Steps:**
    - Handling missing values and categorical data conversion using Label Encoding.
    - Normalizing continuous features for better model performance.
  - **Exploratory Data Analysis (EDA):**
    - Analysis of key features such as Polyuria, Polydipsia, sudden weight loss, weakness, etc.
    - Correlation analysis between independent variables and the target variable.
    - Feature importance visualization.
  - **Machine Learning Models:**
    - Models trained: Logistic Regression, Random Forest.
    - 10-Fold Cross-Validation applied to improve performance.
    - Model performance measured using accuracy, precision, recall, and ROC-AUC.
  - **Model Deployment:**
    - The best-performing model is saved using pickle for integration into the frontend.

2.2. Heart Disease Prediction Module

- **Objective:**

  To develop a machine learning model that predicts the likelihood of a person having heart disease based on clinical parameters.

- **Implementation Details:**

  - **Dataset Used:** heart.csv (dataset take from kaggle)
  - **Preprocessing Steps:**
    - Checking for missing values and handling data inconsistencies.
    - Encoding categorical variables where necessary.
    - Normalization and feature selection.
  - **Exploratory Data Analysis (EDA):**
    - Statistical analysis of various features like age, chest pain type, blood pressure, cholesterol, etc.
    - Correlation analysis to determine important predictors.
    - Visualization of trends using Seaborn and Matplotlib.
  - **Machine Learning Model:**
    - Logistic Regression is used for classification.
    - Model training on 80% of the dataset, testing on 20%.
    - Performance evaluation using accuracy metrics (precision, recall, F1-score, ROC-AUC score).
  - **Model Deployment:**
    - The trained model is saved using pickle for later use in prediction.

2.3. Frontend Module

- **Objective:**

  To provide an interactive and user-friendly interface for users to input their health details and get disease predictions.

- **Implementation Details:**

  - **Technology Used:** Streamlit
  - **Features Implemented:**
    - **Home Page:** Introduction to the platform.
    - **About Disease Page:** Informational section about heart disease and diabetes.
    - **Prediction Module:**

        - Users input relevant health details.
        - Data is preprocessed and fed into the trained models.
        - Results are displayed along with probability scores.
    - **Background Customization:**
        - Streamlit's UI is enhanced with a background image and CSS styling.
    - **Disclaimer Section:**
        - Users are informed that the predictions should not replace medical consultations.

2.4. Model Integration:

- The saved models (heart_disease_model.sav and diabetes_prediction.sav) are loaded for real-time predictions.
- The frontend processes user input and returns disease risk predictions with an appropriate message [7].

This project integrates machine learning and web technologies to build an accessible health risk prediction system. By leveraging data analytics and predictive models, the system helps users assess their risk for diabetes and heart disease, providing valuable health insights.

CHAPTER 3

DEPLOYMENT ENVIRONMENT

Predict My Health is a user-friendly web application for heart disease and diabetes prediction, built with Python and Streamlit. It integrates scikit-learn, NumPy, pandas, and Matplotlib for data processing, visualization, and machine learning [7].

Development is streamlined with Jupyter Notebook for prototyping and pip for dependency management. The app runs locally with minimal system requirements, ensuring easy deployment without external hosting.

3.1. Programming Language & Framework:

- **Python:**
    - Python acts as the backbone of the application development. It's versatile and robust, making it ideal for tasks such as:
    - Libraries: Using libraries like scikit-learn, for implementing complex models for heart disease or diabetes prediction.
    - Data Preprocessing: Tools like pandas and NumPy allow you to clean, organize, and manipulate health-related data before feeding it into the model.
    - Visualization: Libraries like Matplotlib and Seaborn help you create clear and informative graphs to display trends and results.
    - In the application, Python seamlessly integrates logic and computations, giving you the flexibility to handle everything behind the scenes [5].

- **Streamlit:**
    - Streamlit is specifically designed for creating interactive web applications easily.
    - Minimal Coding: write pure Python code, and Streamlit transforms it into web-ready elements like buttons, sliders, and graphs.
    - Together, Python and Streamlit create a powerful combo for combining backend computations with a front-end interface that's user-friendly and engaging.

3.2. Development Environment:

IDE/Editor: Jupyter Notebook is highly effective for tasks involving data exploration, machine learning, and prototyping due to its interactive and modular nature. Here's why it's a go-to choice for machine learning workflows:

- Interactive Code Execution: You can run code cells independently, view output immediately, and experiment without rerunning the entire script.

- Ease of Debugging: Errors are isolated to specific cells, making it easier to identify and fix issues without disrupting the rest of your workflow.
- Data Visualization Integration: You can directly visualize data with libraries like Matplotlib, Seaborn, or Plotly within cells, enabling instant feedback on your dataset insights.
- Markdown Support: In addition to code, you can write text, explanations, or notes using Markdown cells—great for documenting experiments or explaining model results.
- Notebook Sharing: Jupyter notebooks can be shared with collaborators as .ipynb files or converted to HTML for presentations.
- Extensions and Customizations: Numerous extensions, such as Jupyter Widgets, enhance interactivity, and tools like nbextensions help in customizing notebooks.

This makes Jupyter Notebook ideal for machine learning prototyping, especially when you're working with libraries like pandas, scikit-learn, or NumPy. Once your model is refined, you can transition to production-ready code or frameworks for deployment.

3.3. Package Manager: pip

Pip is the Python Package Index (PyPI) installer, making it indispensable for managing dependencies required for your project. Here's why it's essential and how you can maximize its capabilities:

- Install Libraries: With pip install, you can quickly download and install essential libraries like:
- Streamlit: For front-end web application development.
- pandas & NumPy: For data manipulation and numerical computations.
- scikit-learn: For machine learning models and algorithms.
- Matplotlib & Seaborn: For creating visualizations.
- Manage Dependencies: You can ensure all libraries are compatible and up-to-date using commands like:
- pip freeze > requirements.txt to create a list of dependencies for reproducibility.
- pip install -r requirements.txt to install all dependencies listed in one go.
- Upgrade/Downgrade Packages: Use commands like pip install <library> --upgrade or pip install <library>==version to stay compatible with your application setup [6].
- Uninstall Packages: Remove unnecessary libraries using pip uninstall <library> to keep your environment clean.

pip ensures your development environment is ready and organized, making it easier to integrate libraries into your machine learning models and Streamlit applications.

3.4. Libraries and Dependencies:

- **NumPy & Pandas**
  - These libraries are essential for numerical calculations and efficient data manipulation:[8].
  - NumPy: Provides powerful array operations, mathematical functions, and linear algebra capabilities—great for computations during model training.
  - Pandas: Facilitates data preprocessing, including handling missing values, filtering, and aggregating data. Its intuitive Data Frame structure makes it easy to work with complex datasets.
  - Both are fundamental for preparing healthcare datasets before feeding them into your predictive models.

- **Scikit-learn**
  - Scikit-learn offers a comprehensive suite for implementing machine learning models, such as:
  - Algorithms for classification, regression, and clustering (e.g., decision trees, random forests, etc.).
  - Model evaluation metrics like accuracy, precision, and recall.
  - Tools for feature scaling and selection, ensuring the input data is optimized for performance.
  - Its simplicity and versatility make it ideal for building reliable predictions for heart disease and diabetes.

- **Pickle / Joblib**
  - These libraries are indispensable for model persistence:
  - Pickle: Serializes Python objects, allowing you to save trained models as .pkl files.
  - Joblib: Optimized for handling large arrays or models, making it faster than Pickle for some tasks.
  - By saving your models, you can quickly load and deploy them in your Streamlit app, eliminating the need for retraining.

- **Base64**
  - Base64 is used for image processing in your app:
  - Encodes images into base64 strings, making them easily embeddable in web applications.
  - Ideal for adding background images or watermarks to your Streamlit app.
  - For example, combining health-related themes (e.g., heart or diabetes symbols) with Base64 ensures your application's visual design aligns with its purpose.

3.5. System Requirements:

- Predict My Health functions efficiently across various hardware configurations with minimal resource consumption:
- Operating System: Compatible with Windows.
- Processor: Minimum: Intel i3 or equivalent.
- RAM: 4GB (for running basic functionality).
- Storage: At least 500MB of free disk space is required to store dependencies and datasets.

3.6. Deployment Options:

- Users can run the application locally by executing:
- streamlit run app.py
- No internet connection is required after installing dependencies.
- Ideal for users who prefer a simple setup without external hosting.

CHAPTER 4

MODULES WORKED ON

Predict My Health is a disease prediction tool designed to assess the risk of diabetes and heart disease using machine learning. The system integrates multiple modules, including data preprocessing, model training, exploratory data analysis, and an interactive user interface, ensuring ease of use.

Key Components and Tasks:

4.1. Data Preprocessing Module

This module is responsible for preparing the dataset for analysis and machine learning. It ensures data quality and consistency, which are critical for accurate predictions.

- **Handling Missing Values and Cleaning:**
  - Checked for missing or inconsistent values in the dataset and applied suitable imputation techniques to handle them. For example, replacing missing numerical values with mean/median or using forward-fill/backward-fill methods for categorical values.
  - Removed duplicate records or outliers to enhance data integrity.
- **Encoding Categorical Variables:**
  - Converted non-numerical data (e.g., gender, symptoms like Polyuria and Polydipsia) into numerical representations using label encoding or one-hot encoding. For example:
    - Gender: Male → 0, Female → 1
    - Symptoms: No → 0, Yes → 1
- **Normalizing Data**:
  - Scaled numerical features like age, cholesterol, and resting blood pressure using MinMaxScaler or StandardScaler, ensuring that the machine learning models handle the data effectively. This step helps avoid bias due to differences in feature range.
- **Challenge**: Standardizing varied user input formats (numbers, text).
  - **Solution:** Applied data type conversion and validation checks.

4.2. Machine Learning Model Module

This module focuses on building, training, and evaluating machine learning models to predict diabetes and heart disease risk.

- **Implementing Logistic Regression:**
    - Created two separate models for diabetes and heart disease prediction using Logistic Regression. This algorithm was chosen for its simplicity, interpretability, and effectiveness in classification problems.
    - Configured hyperparameters like regularization (L2) for optimized performance.
- **Training and Evaluating Models:**
    - Split the data into training and test sets (e.g., 80%-20%) to assess the models' performance.
    - Evaluated the models using key metrics:
        - Accuracy: Indicates overall correctness of predictions.
        - Precision: Focuses on the proportion of true positive cases among all predicted positives.
        - Recall (Sensitivity): Reflects the model's ability to identify all actual positive cases.
        - ROC-AUC Score: Measures the balance between sensitivity and specificity, providing insight into the model's ability to discriminate between classes.
- **Model Persistence:**
    - Saved the trained models using Pickle or Joblib for future use, ensuring quick deployment without retraining.

- **Challenge:** Loading models quickly without increasing execution time.
    - **Solution:** Leveraged Pickle for efficient model storage and retrieval.

4.3. Exploratory Data Analysis (EDA) Module

This module delves into understanding the dataset, uncovering patterns, and identifying relationships between features and the target variable.

- **Analyzing Key Features:**
    - Explored how features like age, cholesterol, blood pressure, and symptoms correlate with the target variables (diabetes or heart disease risk).
    - Created a correlation heatmap to visualize feature relationships, highlighting key predictive attributes like polyuria, polydipsia, and thalassemia.
- **Visualizing Data Trends:**
    - Used Matplotlib and Seaborn to generate insightful plots, such as:
        - Bar plots and pie charts for categorical feature distributions.
        - Histograms for numerical feature analysis.
        - Scatterplots and boxplots for understanding trends and outliers.
    - Annotated graphs to make findings accessible and easily interpretable for both technical and non-technical audiences.

4.4. Interactive User Interface (UI) Module

This module ensures users can navigate the web application effortlessly and input their health metrics for predictions without requiring technical expertise.

- **Streamlit-Based Design:**
  - Built a sleek and interactive interface using Streamlit, allowing seamless integration of backend machine learning predictions into a user-friendly frontend.
- **Sidebar Menu:**
  - Designed intuitive navigation with options like:
    - Home: Overview of the system and user instructions.
    - About Disease: Educational content for diabetes and heart disease.
    - Disease Prediction: Real-time prediction functionality based on user inputs.
- **Input Forms:**
  - Included dropdowns, sliders, and number fields for collecting health parameters (e.g., chest pain type, fasting blood sugar levels).
- **Challenge:** Simplifying layout while maintaining functionality.
  - **Solution:** Utilized Streamlit's widgets (e.g., buttons, columns) for easy-to-use components.

4.5. Customization Features Module

This module focuses on enhancing the aesthetic and usability aspects of the application for better user engagement.

- **Background Customization:**
  - Embedded a custom background image using Base64 encoding and CSS styling to create a visually appealing interface.
  - Adjusted opacity and alignment for better readability of text over the background.
- **Visual Adjustments:**
  - Enhanced text readability by increasing font sizes for headers and important sections.
- **Layout Optimization:**
  - Structured input fields using Streamlit's column system to maintain a clean and organized design.
- **Challenge:** Text visibility issues due to background images.
  - **Solution:** Adjusted image opacity and text placement for balanced aesthetics

4.6. Disease Information Module

This module educates users on diabetes and heart disease, empowering them to understand health risks and preventative measures.

- **Educational Content:**
  - Added detailed sections covering:
    - Diabetes: Types, causes, symptoms, and prevention strategies.
    - Heart Disease: Risk factors, symptoms, and lifestyle recommendations.

- **Simplified Explanations:**
  - Made medical information accessible to non-technical users by avoiding jargon.

- **Challenge:** Simplifying complex medical concepts.

- **Solution**: Used relatable examples and plain language

CHAPTER 5

SOFTWARE REQUIREMENTS SPECIFICATION (SRS) REPORT

This SRS document outlines the development of Predict My Health, a web-based application that leverages machine learning to predict diabetes and heart disease risk based on user inputs. The system provides an intuitive interface for health metric entry, delivers risk assessments, and offers educational content. Designed for usability, performance, and scalability, Predict My Health ensures accurate predictions while promoting early diagnosis and preventive healthcare.

5.1. Introduction

- **Purpose**

  The purpose of this project is to develop a web-based application, Predict My Health, that uses machine learning models to predict the likelihood of diabetes and heart disease based on user-provided health metrics. The application aims to provide early health risk assessment to empower users to take preventive measures.

- **Scope**

  Predict My Health is a diagnostic tool designed to:

  - Allow users to input their medical details and receive predictions for diabetes and heart disease
  - Provide educational content about these diseases to raise awareness about their risk factors, symptoms, and prevention methods.
  - Offer a user-friendly interface that caters to both technical and non-technical users.

- **Objectives**

  - To create a reliable health risk assessment system.
  - To deliver an intuitive and visually appealing web interface using Streamlit.
  - To educate users about the importance of early diagnosis and lifestyle improvements.

5.2. Functional Requirements

- **User Input**

  - The application must allow users to enter their health metrics, including:

- For Diabetes Prediction: Age, gender, symptoms (e.g., polyuria, polydipsia, obesity).
- For Heart Disease Prediction: Age, blood pressure, cholesterol levels, chest pain type, thalassemia type, and other relevant parameters.

- Inputs must be intuitive, using dropdowns, sliders, and number fields.

➢ **Machine Learning Integration**

- The system must load and utilize pre-trained machine learning models for diabetes and heart disease risk predictions.

➢ **Prediction Results**

- Display results in an easy-to-understand format:
  - "Low Risk" or "High Risk" indicators.
  - Probability scores (for heart disease predictions).
- Include disclaimers that clarify the predictions are not a diagnosis and encourage consulting a doctor for further evaluation.

➢ **Educational Content**

- Provide detailed, user-friendly explanations of diabetes and heart disease, including:
  - Types of diseases.
  - Symptoms, causes, and risk factors.
  - Prevention strategies and lifestyle recommendations.

5.3. Non-Functional Requirements

➢ **Usability**

- The application must have a responsive and visually appealing user interface.
- Navigation should be seamless, with clearly labeled sections.

➢ **Performance**

- Ensure fast loading times for machine learning models and predictions.

➢ **Scalability**

- The system must support scalability to accommodate more features, users, or disease predictions in the future.

➢ **Maintainability**

- Code must be modular and documented for easy updates and debugging.

5.4. Technical Specifications

➢ **Software Components**

- Frontend: Built using Streamlit for developing interactive and dynamic web applications.
- Backend: Python-based backend integrates machine learning models for predictions.
- Machine Learning Models: Pre-trained models saved using Pickle for diabetes and heart disease predictions.

➢ **Tools and Frameworks**

- Libraries: Pandas, NumPy, Scikit-learn, Matplotlib, Seaborn.
- Deployment: Streamlit Cloud for online hosting.

5.5. System Design

➢ **System Architecture**

- User Interface: Users interact with the app through Streamlit's web interface.
- Pre
- diction Workflow:
    - User inputs health parameters.
    - Inputs are preprocessed (categorical values encoded, features scaled).
    - Preprocessed data is fed into machine learning models.
    - Results are displayed along with appropriate recommendations.

➢ **Layout**

- Sidebar Navigation:
    - Home: Overview of the application.
    - About Disease: Educational content about diabetes and heart disease.
    - Disease Prediction: Real-time prediction functionality.

- Main Content Area:
    - Input fields for health metrics.
    - Display area for prediction results.

5.6. Challenges

- Accurate Predictions:
    - Machine learning models must be trained and validated thoroughly to ensure accuracy.
- User Input Handling:
    - Variations in user-provided data formats needed normalization and standardization.
- Design Aesthetics:
    - Background images occasionally affected text visibility.

5.7. Future Scope

- Multi-Disease Predictions: Extend the application to include other diseases like hypertension or cancer.
- Multi-Language Support: Enhance accessibility by adding translations for various languages.
- Mobile Optimization: Ensure a smoother user experience on mobile devices.
- API Integration: Allow healthcare professionals to integrate the app with existing systems for wider usability.

Predict My Health is a comprehensive and user-friendly application that combines machine learning and intuitive design to provide health risk predictions for diabetes and heart disease. Its focus on accessibility, and education makes it a valuable tool for promoting early diagnosis and healthier lifestyles. With its modular design and scalability, the application is poised for future enhancements and broader adoption.

CHAPTER 6

PROJECT DESIGN

The project design focuses on integrating trained machine learning models for diabetes and heart disease predictions with a user-friendly, interactive web interface built using Streamlit. The application provides educational content about these diseases and predictive functionality, prioritizing simplicity.

- **Data Flow Diagram**

  A Data Flow Diagram (DFD) showcases how data moves through a system, detailing the processes, inputs, outputs, and data stores.

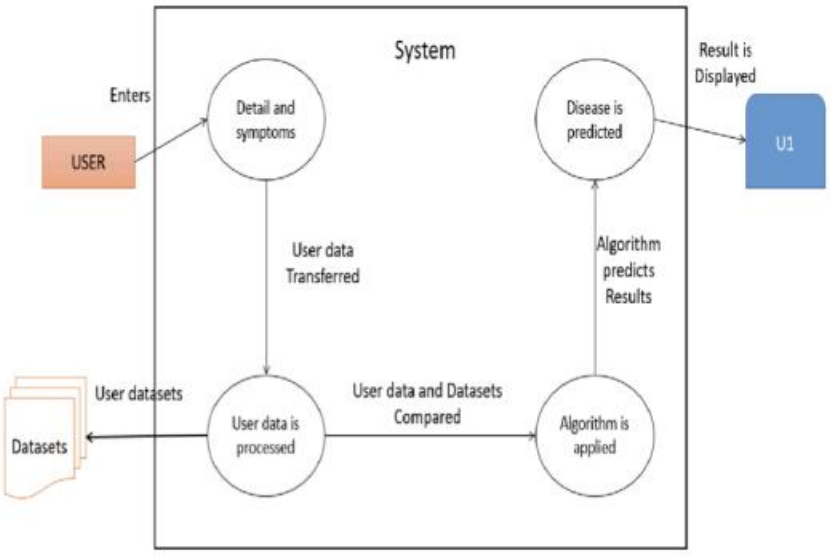


Fig.6.1. Data Flow Diagram

- **Use Case Diagram:**
  A Use Case Diagram represents the interaction between users and the system by illustrating the functionalities (use cases) that the system offers.

Fig.6.2. Use Case Diagram

CHAPTER 7

IMPLEMENTATION (SOURCE CODE)

7.1. Diabetes Disease model

#IMPORTING NECESSARY LIBRARIES

#These libraries (Numpy, Pandas, matplotlib, scikit-learn, seaborn) are already installed. Now, importing them to utilize their functionalities for data preprocessing, machine learning model training, evaluation, and visualization.

```
# Warnings Management

import warnings

warnings.filterwarnings('ignore') # Ignores unnecessary warning messages to keep the output clean.

#  Numerical and Data Handling

import numpy as np # Used for numerical operations, arrays, and mathematical functions.

import pandas as pd # Used for handling structured/tabular data (Data Frames).

#  Data Preprocessingg

from sklearn.preprocessing import MinMaxScaler

# Why?: When data has very large or very small values, machine learning models may not perform well because some features dominate others.

# MinMaxScaler helps by scaling all values between 0 and 1, making the model more stable and efficient.

# Data Splitting & Model Evaluation

from sklearn.model_selection import train_test_split, cross_val_score

# train_test_split → Splits dataset into training and testing sets.

# cross_val_score → Performs cross-validation to evaluate model performance.

# Machine Learning Modelss

from sklearn.linear_model import LogisticRegression #Used for classification tasks.

from xgboost import XGBClassifier #A powerful ensemble learning method
```

```
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

#Uses multiple decision trees for better accuracy.

# Improves weak learners using Adaptive Boosting.

from sklearn.svm import SVC #A robust Support Vector Classifier.

# Model Performance Evaluation

from sklearn.metrics import (

accuracy_score, # Measures overall accuracy of predictions.

f1_score, # Balances between precision & recall

precision_score, # Measures correct positive predictions.

recall_score, # Ability to detect all positive instances.

confusion_matrix, # Shows actual vs predicted values.

roc_auc_score # Measures ROC curve area (useful for binary classification).

# Data Visualization

import seaborn as sns

import matplotlib.pyplot as plt

import matplotlib.pyplot as plt # Used for plotting graphs and visualizing data.

%matplotlib inline

#Ensures that plots are displayed within Jupyter Notebook.

# Displaying Images

from IPython.display import Image

#Allows displaying images inside Jupyter Notebook.

# IMPORT COMPLETE: All required libraries are now ready to use.
```

First, we need to check what our dataset looks like before using it to build the diabetes detection model. To do this, we will load the dataset using the pandas library and display its first few rows.

Reading the dataset into a Pandas Data Frame

df = pd.read_csv('diabetes_data_upload.csv')

#Displaying the first few rows of the dataset to understand its structure

df.head()

[3]:

	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring	Itching	Irritability	delayed healing	partial paresis	muscle stiffness	Alopecia	Obesity	class
0	40	Male	No	Yes	No	Yes	No	No	No	Yes	No	Yes	No	Yes	Yes	Yes	Positive
1	58	Male	No	No	No	Yes	No	No	Yes	No	No	No	Yes	No	Yes	No	Positive
2	41	Male	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	No	Yes	Yes	No	Positive
3	45	Male	No	No	Yes	Yes	Yes	Yes	No	Yes	No	Yes	No	No	No	No	Positive
4	60	Male	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Positive

This will help us understand what kind of data we are working with, including the columns and their values.

Checking for Missing Values Before working with the dataset, it's important to check if any values are missing. Missing data can affect the performance of our model, so we need to identify them first. The following code will show the number of missing values in each column:

Checking for missing values

missing_values = df.isna().sum().sum()

df.isna().sum()

[4]:

```
Age                 0
Gender              0
Polyuria            0
Polydipsia          0
sudden weight loss  0
weakness            0
Polyphagia          0
Genital thrush      0
visual blurring     0
Itching             0
Irritability        0
delayed healing     0
partial paresis     0
muscle stiffness    0
Alopecia            0
Obesity             0
class               0
dtype: int64
```

This will display the count of missing values in each column.

If there are missing values, we can decide whether to fill them with appropriate values or remove them.

```
if missing_values == 0:
    print("No missing values found in the dataset! ☑")
else:
    print(f"Missing values found: {missing_values}")
```

```
No missing values found in the dataset! ✅
```

Exploratory Data Analysis (EDA) Now, we will analyze the dataset to understand how different features are distributed and identify patterns in the data. This helps in understanding relationships between variables and their impact on the target variable.

To begin, let's first explore the target variable (i.e., class) to see how the data is distributed between different classes.

1. Distribution of Target Variable

Analyzing the target variable is crucial to check whether the dataset is balanced or imbalanced. If one class has significantly fewer samples (less than 10% of the total), it indicates an imbalance issue, which can affect model performance.

To check this, we use the following code:

```
# Creating Pie Chart

plt.subplot(121) # Create subplot (1 row, 2 columns, 1st plot)

df["class"].value_counts().plot.pie(

autopct="%1.0f%%", # Display percentage with 1 decimal place

colors=sns.color_palette("prism", 7), # Use 'prism' color palette

startangle=60, # Start angle for better visualization

labels=["Positive", "Negative"], # Labels for target classes

wedgeprops={"linewidth": 2, "edgecolor": "k"}, # Styling the wedges

explode=[.1, 0], # Slightly separate the first slice
```

```
shadow=True # Add shadow effect

plt.title("Distribution of Target Variable") # Title for Pie Chart

# Creating Bar Plot

plt.subplot(122) # Create subplot (1 row, 2 columns, 2nd plot)

ax = df["class"].value_counts().plot(kind="barh") # Create horizontal bar plot

# Adding value labels on the bars

for i, j in enumerate(df["class"].value_counts().values):

ax.text(.7, i, j, weight="bold", fontsize=20) # Positioning text on bars

plt.title("Count of Target Variable") # Title for Bar Plot

plt.show() # Display the plots
```

Distribution of Target Variable To understand the dataset better, we analyzed the distribution of the target variable (class) using a pie chart and a bar plot.

Key Observations:

The slight separation (explode effect) highlights the Positive class for better visualization.

Class Count (Bar Chart)

The dataset is not highly imbalanced, as both classes are reasonably well represented. However, the Positive class has a higher count (62%) than the Negative class (37%). This distribution ensures that the model can learn effectively from both classes without requiring significant imbalance-handling techniques. This analysis provides a clear and accurate understanding of the dataset, ensuring there is no ambiguity for further processing.

2. Distribution of Gender

In this step, we analyzed the distribution of the Gender feature, which contains two categories: Male and Female. The purpose of this analysis is to check if there is any imbalance in the dataset regarding gender representation and its relationship with diabetes.

```
# Plotting bar chart for Gender distribution

sns.countplot(x='Gender', hue='class', data=df)
```

hue='class' → Colors the bars based on the class (Diabetes Positive/Negative).

Key Observations:

Gender-wise Distribution (Bar Chart)

The bar chart represents the count of Male and Female cases. The color distinction highlights the proportion of Positive (Diabetes) and Negative (No Diabetes) cases within each gender.

Defining the columns to analyze

plot_criteria = ['Gender', 'class'] # We are analyzing the relationship between Gender and the target variable 'class'.

Creating a color map for better visualization

cm = sns.light_palette("grey", as_cmap=True) # This creates a light grey color gradient for table visualization.

Creating a crosstab to compare Gender distribution across the target variable (class)

---

pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]]) creates a frequency table showing

how many instances of each gender belong to each class (Positive or Negative).

#

normalize='columns' converts these counts into percentages per class.

Multiplying by 100 gives the values in percentage format.

round(..., 2) rounds the values to 2 decimal places for better readability.

#

Finally, .style.background_gradient(cmap=cm) applies the grey gradient to visually

highlight the differences in percentages.

(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100, 2)).style.background_gradient(cmap=cm)

Applying the grey gradient for visualization

Proportion of Diabetes Cases by Gender (Table) A cross-tabulation was performed to compute the percentage of Positive and Negative diabetes cases for each gender. The background gradient makes it easier to identify variations in distribution.

Key Insights:

Males have a much higher proportion of negative diabetes cases compared to females. Females have a slightly higher percentage of positive diabetes cases (54.06%) compared to males (45.94%).

The dataset appears to be somewhat imbalanced towards males in terms of negative diabetes cases. This information suggests that gender could play a significant role in diabetes prevalence in this dataset.

3. Distribution of Polyuria

Polyuria is a condition characterized by excessive urination, where an individual passes more than 3 liters of urine per day, whereas the normal range is between 1 to 2 liters per day.

One of the primary causes of polyuria, especially in both adults and children, is uncontrolled diabetes mellitus. When blood glucose levels become excessively high, the body removes the extra glucose through urine. This process, known as osmotic diuresis, leads to increased urine output, which is a key indicator of diabetes.

In this analysis, we will examine the distribution of the Polyuria feature within the dataset to understand its correlation with the presence of diabetes.

Plotting target variable with respect to (wrt) the Polyuria feature

Define the criteria for the cross-tabulation (categorical variables to compare)

plot_criteria = ['Polyuria', 'class']

Create a color gradient using a light red palette for better visualization

cm = sns.light_palette("grey", as_cmap=True)

Generate a cross-tabulation (pivot table) of Polyuria vs. class (target variable)

Normalize='columns' means we calculate percentages within each class (column-wise)

Multiply by 100 to convert decimal values to percentages

Round to 2 decimal places for better readability

Apply a background color gradient using the color map (cm) to enhance visualization

crosstab_result.style.background_gradient(cmap=cm)

Key Observations:

Only 7.5% of non-diabetic individuals exhibit Polyuria, meaning that while it can occur in some cases, it is much more prevalent in diabetic individuals.

Polyuria is a significant symptom associated with diabetes. The dataset clearly shows that people experiencing Polyuria are more likely to have diabetes compared to those who do not. This feature could be important for diabetes prediction models, as it strongly correlates with the presence of diabetes.

4. Distribution of Polydipsia:

Polydipsia is a condition where a person experiences excessive thirst, often accompanied by a dry mouth. It is one of the early warning signs of diabetes because high blood sugar levels can cause dehydration, triggering persistent thirst.

Plotting target variable distribution with respect to Polydipsia

Step 1: Define the criteria for analysis (Polydipsia vs. Class/Diabetes outcome)

plot_criteria = ['Polydipsia', 'class']

Step 2: Define a color map for visualization (red gradient for emphasis)

cm = sns.light_palette("grey", as_cmap=True)

Step 3: Generate a crosstab (cross-tabulation) to analyze the percentage distribution

- 'normalize="columns"' converts raw counts into percentages (each column sums to 100%)

- Multiplying by 100 converts values into percentage format

- 'round(..., 2)' ensures two decimal places for readability

- 'style.background_gradient(cmap=cm)' applies a heatmap effect for visual clarity

(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100, 2)).style.background_gradient(cmap=cm)

Purpose:

- This visualization helps to understand if Polydipsia (excessive thirst) is a significant predictor of diabetes.

- If 'Yes' for Polydipsia has a much higher percentage in the 'Positive' class, it suggests a strong correlation.

- The color gradient highlights the most important values, making trends easier to interpret.

Insights from the Data

96% of people without diabetes do not experience Polydipsia, while only 4% of them report having it.
70.31% of diabetic patients experience Polydipsia, whereas only 29.69% do not.

Key Takeaways

Polydipsia appears to be strongly correlated with diabetes. A majority of diabetic patients (70.31%) report excessive thirst, which is a well-known symptom of diabetes. In contrast, Polydipsia is quite rare among non-diabetic individuals (only 4%).

Since Polydipsia is significantly more common in diabetic patients, it could serve as a strong predictor for diabetes detection.

5. Distribution of Sudden Weight Loss

Sudden and unexplained weight loss can be a warning sign of diabetes. When the body cannot properly use glucose for energy due to insulin issues, it starts breaking down muscle and fat, leading to rapid weight loss.

Defining the variables for analysis

plot_criteria = ['sudden weight loss', 'class'] # Selecting features for comparison

Defining the color scheme for visualization

cm = sns.light_palette("grey", as_cmap=True)

Creating a cross-tabulation of the selected features

(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100,2)).style.background_gradient(cmap=cm)

Insights:

Among Diabetic Individuals (Positive Cases)

57.75% of people diagnosed with diabetes have reported experiencing sudden weight loss. 41.25% of diabetic individuals have not experienced sudden weight loss.

Among Non-Diabetic Individuals (Negative Cases)

75.50% of non-diabetic people have not experienced sudden weight loss. Only 14.50% of non-diabetic individuals reported sudden weight loss.

Sudden weight loss is a strong indicator of diabetes since a significant 57.75% of diabetic individuals have reported it, compared to only 14.50% of non-diabetics. This suggests that sudden weight loss should be considered a key symptom when predicting diabetes.

6. Distribution of Weakness

Weakness, or fatigue, is a common symptom in diabetes. It occurs due to fluctuating blood sugar levels, insulin resistance, and the body's inability to efficiently convert glucose into energy. When cells do not get enough glucose, it leads to a lack of energy, causing persistent weakness.

```
# Defining the features for analysis

plot_criteria= ['weakness', 'class']

# Creating a red color gradient for better visualization

cm = sns.light_palette("grey", as_cmap=True)

# Generating a crosstab to analyze the relationship between 'weakness' and 'class'

# - Normalizing by 'columns' ensures the percentage distribution within each class (Negative/Positive).

# - Multiplying by 100 converts the values into percentages.

# - Rounding to 2 decimal places improves readability.

# - Applying a background gradient highlights significant values for easier interpretation.

(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100,2)).style.background_gradient(cmap = cm)
```

Key Insights

67.12% of diabetic patients reported weakness, whereas only 31.77% of non-diabetic individuals experienced.
A higher percentage of people without diabetes (56.5%) do not experience weakness. This suggests that weakness is a strong indicator of diabetes, as the majority of diabetic patients report experiencing it.

Since a significantly higher percentage of diabetic individuals experience weakness, this feature can be a key predictor in diabetes detection models.

7.Distribution of genital thrush

Genital thrush, also known as candidiasis, is a fungal infection caused by the overgrowth of Candida yeast. It commonly affects individuals with high blood sugar levels, making diabetics more susceptible. Excess glucose provides a breeding ground for the yeast, leading to infection, irritation, and discomfort.

```
# Defining the criteria for cross-tabulation

plot_criteria = ['Genital thrush', 'class']

# Creating a color map using a red gradient for better visualization

cm = sns.light_palette("grey", as_cmap=True)

# Generating a cross-tabulation table to analyze the relationship between "Genital thrush" and "class"

# 1. `pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]])` creates a frequency table of occurrences

# 2. `normalize='columns'` ensures that the values are shown as percentages of the respective class (Negative/Positive)

# 3. Multiplying by 100 converts the fraction into percentage format

# 4. `round(..., 2)` ensures that the percentage values are rounded to two decimal places for better readability

# 5. `.style.background_gradient(cmap=cm)` applies a red color gradient to highlight variations in distribution

(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100, 2)).style.background_gradient(cmap=cm)
```

Insights from the Data From the given data distribution:

Non-Diabetic Individuals (Negative Class) 73.5% of people without genital thrush do not have diabetes. 16.5% of non-diabetics experience genital thrush.

Key Observations

The occurrence of genital thrush is higher in diabetic patients (25.94%) compared to non-diabetics (16.5%). This suggests that genital thrush may be a potential indicator of diabetes, as the body's weakened immune response in diabetes can lead to an increased risk of fungal infections. However, most diabetic patients (74.06%) do not have genital thrush, so it cannot be the sole factor in diagnosing diabetes.
The higher prevalence of genital thrush in diabetics supports the medical understanding that fungal infections are more common in people with diabetes. While not every diabetic person will have genital thrush, its presence could be a warning sign, especially if accompanied by other symptoms like frequent urination and excessive thirst.

8. Distribution of visual blurring

Visual blurring is a common diabetes symptom caused by fluctuating blood sugar levels. High sugar levels can swell the eye lens, while low sugar reduces eye energy supply, both leading to blurry vision. Long-term diabetes may cause diabetic retinopathy, risking permanent vision loss. Regular eye check-ups and sugar control help prevent complications.

```
plot_criteria = ['visual blurring', 'class']

cm = sns.light_palette("grey", as_cmap=True)
```

This supports the medical understanding that high blood sugar levels can cause fluid imbalances in the eyes, leading to temporary or long-term visual disturbances.

Visual blurring is a common symptom among diabetic patients, making it an important factor in early diagnosis.

9. Distribution of Itching

Itching (pruritus) can be linked to diabetes due to dry skin, fungal infections, poor circulation, or nerve damage (neuropathy). Persistent itching, especially with other symptoms, may indicate high blood sugar levels.

```
plot_criteria= ['Itching', 'class']
```

```
cm = sns.light_palette("grey", as_cmap=True)

(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100,2)).style.background_gradient(cmap = cm)
```

The table shows the distribution of Itching among individuals with and without diabetes:

50.5% of non-diabetic individuals (Negative class) do not experience itching, while 49.5% of them do.
51.77% of diabetic individuals (Positive class) do not have itching, whereas 47.12% do.

Interpretation:

The distribution is nearly equal in both groups, indicating that itching alone is not a strong predictor of diabetes. However, when combined with other symptoms like polyuria, polydipsia, and sudden weight loss, it might contribute to identifying potential diabetic cases.

10. Distribution of Irritability

Irritability in diabetes is often due to blood sugar fluctuations, leading to mood swings and agitation. It can be an early sign of diabetes when combined with other symptoms like weakness and fatigue. Now, let's examine its distribution.

```
plot_criteria= ['Irritability', 'class']

cm = sns.light_palette("grey", as_cmap=True)

(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100,2)).style.background_gradient(cmap = cm)
```

The distribution of Irritability in diabetes cases shows that:

92% of negative cases (non-diabetic individuals) do not experience irritability, while 7% do. 65.62% of positive cases (diabetic individuals) also do not report irritability, but 34.37% do. This suggests that while irritability is not a definitive symptom, it is more commonly reported in individuals with diabetes compared to those without it.

11.Distribution of Delayed Healing

Delayed healing refers to the slowed recovery of wounds, cuts, or injuries. It is a common symptom in diabetes due to high blood sugar levels affecting circulation and the immune system.Poor blood flow reduces oxygen supply to tissues, making healing slower and increasing the risk of infections.

```
plot_criteria= ['delayed healing', 'class']

cm = sns.light_palette("grey", as_cmap=True)

(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100,2)).style.background_gradient(cmap = cm)
```

This distribution shows that a significant portion of diabetic individuals (47.71%) experience delayed healing, compared to 43% of non-diabetic individuals.This suggests that while delayed healing is not exclusive to diabetes, it is more frequently observed in diabetic patients.

12. Distribution of Partial Paresis

Partial paresis refers to muscle weakness or partial loss of voluntary movement in certain body parts. It is often linked to nerve damage (diabetic neuropathy) caused by prolonged high blood sugar levels. This condition can lead to difficulty in movement and coordination, significantly affecting daily activities.

```
plot_criteria= ['partial paresis', 'class']

cm = sns.light_palette("grey", as_cmap=True)

 (round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100,2)).style.background_gradient(cmap = cm)
```

60% of diabetic individuals experience partial paresis, while only 16% of non-diabetic individuals are affected. This suggests that diabetes significantly increases the risk of nerve-related muscle weakness, making early diagnosis and blood sugar control crucial in preventing such complications.

13. Distribution of Muscle Stiffness

Muscle stiffness refers to tightness, rigidity, or reduced flexibility in muscles, often making movement uncomfortable. In diabetes, high blood sugar can lead to inflammation and nerve damage, which may contribute to muscle stiffness and pain, especially in the limbs.

```
plot_criteria= ['muscle stiffness', 'class']

 cm = sns.light_palette("grey", as_cmap=True)

(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100,2)).style.background_gradient(cmap = cm)
```

42.19% of diabetic individuals report muscle stiffness, compared to 30% in non-diabetic individuals. This suggests that diabetes may increase the likelihood of muscle discomfort and restricted movement, making regular exercise, physiotherapy, and proper diabetes management essential for reducing muscle-related complications.

14. Distribution of Obesity

Obesity is a major risk factor for diabetes. Excess body fat, particularly around the abdomen, leads to insulin resistance, where the body's cells do not respond effectively to insulin. This increases blood sugar levels, significantly raising the risk of Type 2 diabetes.

```
plot_criteria= ['Obesity', 'class']

 cm = sns.light_palette("grey", as_cmap=True)

 (round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100,2)).style.background_gradient(cmap = cm)
```

we observe that only 19.06% of diabetic individuals are obese, while 70.94% are not, suggesting that while obesity is a contributing factor, diabetes can occur in non-obese individuals as well. Hence, other factors like genetics, diet, and lifestyle also play a crucial role in diabetes development.

15. Alopecia

Alopecia, or sudden hair loss, can sometimes be linked to diabetes due to poor blood circulation, hormonal imbalances, and immune system dysfunction. High blood sugar levels can damage hair follicles, leading to hair thinning or bald patches.

```
plot_criteria= ['Alopecia', 'class']

cm = sns.light_palette("grey", as_cmap=True)

 (round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100,2)).style.background_gradient(cmap = cm)
```

we observe that 75.62% of diabetic individuals do not experience alopecia, while 24.37% of them do. This suggests that while alopecia is not a major symptom of diabetes, it can still be present in some cases. Managing blood sugar levels and maintaining a healthy lifestyle can help reduce the risk of hair loss.

Data Pre-processing

Before building a model, we need to pre-process the data to ensure it is in the correct format. The target variable, which currently consists of categorical values ("Positive" and "Negative"), must be converted into a numerical format (1 and 0) to be properly interpreted by machine learning models.

```
#transforming target column from string to numeric format

df['class'] = df['class'].apply(lambda x: 0 if x=='Negative' else 1)
```

Before building the model, we need to separate the input features and the target variable. The input features (X) will include all columns except the target variable, while the target variable (Y) will contain only the class label, which we aim to predict.

```
# creating feature and target variable

X= df.drop(['class'],axis=1)

y=df['class']
```

Label Encoding is a technique used to convert categorical string values into numerical format. This is necessary because machine learning models

typically work with numerical data rather than text. By applying Label Encoding, each unique categorical value is assigned a unique integer.

This ensures that the dataset is properly formatted for model training.

```
#creating a list of object datatypes

objList = X.select_dtypes(include = "object").columns

#Label Encoding for object to numeric conversion

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

for feat in objList:

X[feat] = le.fit_transform(X[feat].astype(str))

print (X.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 520 entries, 0 to 519
Data columns (total 16 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Age                 520 non-null    int64
 1   Gender              520 non-null    int32
 2   Polyuria            520 non-null    int32
 3   Polydipsia          520 non-null    int32
 4   sudden weight loss  520 non-null    int32
 5   weakness            520 non-null    int32
 6   Polyphagia          520 non-null    int32
 7   Genital thrush      520 non-null    int32
 8   visual blurring     520 non-null    int32
 9   Itching             520 non-null    int32
 10  Irritability        520 non-null    int32
 11  delayed healing     520 non-null    int32
 12  partial paresis     520 non-null    int32
 13  muscle stiffness    520 non-null    int32
 14  Alopecia            520 non-null    int32
 15  Obesity             520 non-null    int32
dtypes: int32(15), int64(1)
memory usage: 34.7 KB
None
```

X.head()

[29]:

	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring	Itching	Irritability	delayed healing	partial paresis	muscle stiffness	Alopecia	Obesity
0	40	1	0	1	0	1	0	0	0	1	0	1	0	1	1	1
1	58	1	0	0	0	1	0	0	1	0	0	0	1	0	1	0
2	41	1	1	0	0	1	1	0	0	1	0	1	0	1	1	0
3	45	1	0	0	1	1	1	1	0	1	0	1	0	0	0	0
4	60	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1

Now that all categorical features have been successfully converted into numeric format (int32), we can inspect the dataset to ensure it is properly formatted for model training.

Correlation

Correlation measures the strength and direction of the relationship between two variables. In this case, we are checking how each feature (independent variable) is related to the target variable (dependent variable, i.e., diabetes presence).

Types of Correlation:

Positive Correlation (+1 to 0): If the correlation value is positive, it means that as the feature value increases, the target value is also likely to increase.

Negative Correlation (-1 to 0): If the correlation value is negative, it means that as the feature value increases, the target value is likely to decrease.

No Correlation (≈0): A correlation value near zero suggests that there is no significant relationship between the feature and the target variable.

calculating correlation of all the input variables with target variable

X.corrwith(y)

```
[31]: Age                  0.108679
      Gender              -0.449233
      Polyuria             0.665922
      Polydipsia           0.648734
      sudden weight loss   0.436568
      weakness             0.243275
      Polyphagia           0.342504
      Genital thrush       0.110288
      visual blurring      0.251300
      Itching             -0.013384
      Irritability         0.299467
      delayed healing      0.046980
      partial paresis      0.432288
      muscle stiffness     0.122474
      Alopecia            -0.267512
      Obesity              0.072173
      dtype: float64
```

Data from your output

correlation_values = {

'Age': 0.107679,

'Gender': -0.449233,

'Polyuria': 0.665922,

'Polydipsia': 0.647734,

'sudden weight loss': 0.436567,

'weakness': 0.243275,

'Polyphagia': 0.342504,

'Genital thrush': 0.110277,

'visual blurring': 0.251300,

'Itching': -0.013374,

'Irritability': 0.299467,

'delayed healing': 0.046970,

```
'partial paresis': 0.432277,

'muscle stiffness': 0.122474,

'Alopecia': -0.267512,

'Obesity': 0.072173

}

# Extracting keys and values for plotting

features = list(correlation_values.keys())

values = list(correlation_values.values())

# Plotting

plt.figure(figsize=(10, 6))

plt.bar(features, values, color='skyblue')

plt.xticks(rotation=90) # Rotate labels for better readability

plt.title('Correlation of Features with Target Variable')

plt.xlabel('Features')

plt.ylabel('Correlation Coefficient')

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()

plt.show()
```

Strongest indicators: Polyuria (frequent urination) and Polydipsia (excessive thirst) are highly linked to diabetes.

Moderate links: Sudden weight loss, partial paresis (muscle weakness), and Polyphagia (increased appetite) also show a noticeable connection.

Weak or no link: Delayed healing and itching don’t seem to relate much to diabetes.

Opposite relationship: Gender and Alopecia (hair loss) are inversely connected, meaning they might reduce the chance of diabetes.

Train Test Split

To clarify, we want to split your dataset into training and test sets while maintaining the same proportion of classes in both sets. This is especially useful if you have an imbalanced dataset, where some classes are more prevalent than others. Both sets should maintain this 70%-30% ratio

```
from sklearn.model_selection import train_test_split

# Assuming 'df' is your Data Frame, and 'target' is your target column

# Replace 'target' with the actual name of your target column

X = df.drop(columns=['class']) # Feature columns

y = df['class'] # Target variable (the column you want to predict)

# Check the distribution of the target variable (class labels)

print("Class distribution in the dataset:")

print(y.value_counts())

# If one class is underrepresented, reduce the test size to make sure we have enough instances in both sets

# If the classes are imbalanced, you might want to adjust the test size or perform resampling.

#If there is a significant class imbalance (e.g., one class has far more samples than the other), you may want to reduce the size of the test set

#(test_size) to ensure that both classes are represented adequately in both the training and test sets.

#This could also be an indicator to explore techniques like oversampling (increasing instances of the underrepresented class) or

#undersampling (decreasing instances of the overrepresented class) to balance the classes.

# Perform the train-test split with stratification to maintain the class distribution

try:

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=1234)

#test_size=0.2 means 20% of the data will be used as the test set, and 70% will be used for training.
```

#Stratification (stratify=y) ensures that the class distribution in the training and test sets is the same as in the original dataset,

#maintaining the proportion of classes in both sets.

#random_state=1234 ensures the split is reproducible (i.e., the split will be the same each time you run the code).

Check the distribution of the target variable in the training set

print('Distribution of target variable in training set:')

print(y_train.value_counts())

Check the distribution of the target variable in the test set

print('Distribution of target variable in test set:')

print(y_test.value_counts())

except ValueError as e:

print("Error during split:", e)

print("Adjusting test size...")

If there was an error due to small class sizes, try reducing test_size further

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, stratify=y, random_state=1234)

Check the distribution of the target variable again after the split

print('Distribution of target variable in training set (after adjustment):')

print(y_train.value_counts())

print('Distribution of target variable in test set (after adjustment))

print(y_test.value_counts())

```
Class distribution in the dataset:
class
1    320
0    200
Name: count, dtype: int64
Distribution of target variable in training set:
class
1    256
0    160
Name: count, dtype: int64
Distribution of target variable in test set:
class
1    64
0    40
Name: count, dtype: int64
```

This process is crucial for ensuring that both the training and test sets have the same number of examples for each class (like class 0 and class 1). This helps the model learn and test fairly without favoring one class. If there was any issue due to imbalanced data, you reduced the test size to fix it.

Data Normalization

Normalization is used to scale the Age feature so that its values are on the same scale as the other features, which helps the model learn better and faster. Since Age has larger values compared to other features, normalizing it ensures the model doesn't give it too much importance. You can normalize Age using methods like Min-Max (scaling between 0 and 1) or Standardization (scaling to have a mean of 0 and a standard deviation of 1). This makes the model perform more consistently.

```
# instantiating minmax scaling object

minmax = MinMaxScaler()

#apply minmax scaling on Age feature

X_train[['Age']] = minmax.fit_transform(X_train[['Age']])

X_test[['Age']] = minmax.transform(X_test[['Age']])

X_train.head()
```

[43]:

	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring	Itching	Irritability	delayed healing	partial paresis	muscle stiffness	Alopecia	Obesity
246	0.405405	Male	No	No	No	Yes	No	No	No	Yes	No	Yes	No	No	Yes	No
274	0.554054	Male	No	No	No	No	Yes	No	Yes	No	No	No	No	Yes	No	No
424	0.364865	Male	Yes	Yes	Yes	Yes	No	Yes	No	No	No	No	No	No	No	No
316	0.527027	Female	No	No	No	Yes	No	Yes	No	Yes	No	Yes	Yes	No	Yes	No
159	0.297297	Female	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No

Model Building

Logistic Regression is a simple, linear model used for binary classification tasks. It predicts the probability of an event occurring (e.g., whether a person has diabetes) based on the relationship between input features and the target variable. It's easy to implement and interpret.

```
# Import necessary libraries

from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

# Initialize the LabelEncoder to convert categorical values into numerical ones

label_encoder = LabelEncoder()

# Automatically detect all categorical columns (those of type 'object')

categorical_columns = df.select_dtypes(include=['object']).columns

# Loop through each categorical column and apply label encoding

# Label Encoding converts string categories into numerical values

for col in categorical_columns:

df[col] = label_encoder.fit_transform(df[col]) # Transform each column with LabelEncoder

# Now that all categorical variables are encoded, we define our feature matrix 'X' and target vector 'y'

X = df.drop(columns=['class']) # Features are all columns except 'class'

y = df['class'] # 'class' is the target variable (the one we're trying to predict)
```

'random_state=0': This ensures reproducibility of results, meaning every time you run the model with the same data, you'll get the same outcome.

'penalty='l2'': This specifies the type of regularization to apply. 'l2' stands for L2 regularization (Ridge). It penalizes large coefficients by

#adding the sum of their squared values to the loss function.

The L2 penalty helps prevent overfitting by reducing the complexity of the model, particularly when the model has many features. It encourages the

#model to keep the weights smaller.

Regularization is important to avoid overfitting, where the model might perform well on the training data but poorly on unseen data.

If the penalty were 'l1', it would use L1 regularization (Lasso), which could also lead to some coefficients being reduced to zero, effectively

#removing those features from the model.

Split the data into training and test sets (70% training, 20% testing)

The 'stratify=y' ensures that both training and test sets maintain the same class distribution as in the original dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=1234)

Initialize the Logistic Regression model with L2 regularization (default) and random state for reproducibility

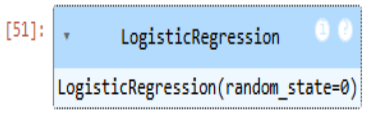
logi = LogisticRegression(random_state=0, penalty='l2')

Train the Logistic Regression model using the training data

The model will now learn the relationship between the features (X_train) and the target (y_train)

logi.fit(X_train, y_train)

After fitting the model, it is ready for making predictions or further evaluation.



10-Fold Cross-Validation

10-Fold Cross-Validation splits the dataset into 10 equal parts (folds) and uses each fold for testing while the remaining 9 are used for training This process is repeated 10 times, with each fold used once as the test set. The results are averaged to give a reliable estimate of model performance.

```
from sklearn import model_selection

from sklearn.model_selection import cross_val_score

# Set up K-Fold cross-validation with 10 splits and shuffle=True for randomness, with a fixed random_state

kfold = model_selection.KFold(n_splits=10, shuffle=True, random_state=7)

# Define the scoring method as accuracy (i.e., evaluating the correct predictions out of total predictions)

scoring = 'accuracy'

# Perform K-Fold cross-validation on the logistic regression model

# 'estimator=logi' specifies the model, 'X_train' and 'y_train' are the features and target labels,

# 'cv=kfold' specifies that 10-fold cross-validation will be used,

# 'scoring=scoring' specifies that accuracy will be the evaluation metric.

acc_logi = cross_val_score(estimator=logi, X=X_train, y=y_train, cv=kfold, scoring=scoring)

# Calculate and print the average accuracy score across all 10 folds

# This gives an overall performance metric for the logistic regression model.

print(acc_logi.mean())
```

```
0.913240418118467
```

```
# Use the trained logistic regression model to predict the target variable (class labels) for the test set (X_test)

y_predict_logi = model.predict(X_test)

# Calculate the accuracy score - proportion of correctly predicted labels
```

```
acc = accuracy_score(y_test, y_predict_logi)

# Calculate the ROC AUC score - measures the model's ability to distinguish between classes

roc = roc_auc_score(y_test, y_predict_logi)

# Calculate the precision score - proportion of predicted positive cases that are actually positive

prec = precision_score(y_test, y_predict_logi)

# Calculate the recall score - proportion of actual positive cases correctly identified by the model

rec = recall_score(y_test, y_predict_logi)

# Calculate the F1 score - harmonic mean of precision and recall, balancing both metrics

f1 = f1_score(y_test, y_predict_logi)

# Create a pandas Data Frame to store the evaluation metrics for Logistic Regression

# The 'results' Data Frame includes the model name, accuracy, cross-validation accuracy, precision, recall, F1 score, and ROC AUC score

results = pd.Data Frame([['Logistic Regression', acc, acc, prec, rec, f1, roc]],

columns = ['Model', 'Accuracy', 'Cross Val Accuracy', 'Precision', 'Recall', 'F1 Score', 'ROC'])

# Print the results Data Frame to see the output

print(results)
```

```
                Model  Accuracy  Cross Val Accuracy  Precision   Recall  \
0  Logistic Regression  0.971154             0.91324   0.984127  0.96875

   F1 Score       ROC
0  0.976378  0.971875
```

```
# Plotting feature importance to understand which features have the highest impact on diabetes prediction

# Extract the absolute values of the coefficients from the logistic regression model.

# These values indicate the strength and direction of the relationship between each feature and the target (diabetes).

feature_importance = abs(logi.coef_[0])
```

```
# Normalize the feature importances by dividing them by the maximum importance value

# This scales all the importances to a 0-100% range, making it easier to compare relative importance.

feature_importance = 100.0 * (feature_importance / feature_importance.max())

# Sort the features by their importance (in ascending order).

# np.argsort returns the indices that would sort the array.

sorted_idx = np.argsort(feature_importance)

# Set up the positions of the bars in the horizontal bar chart.

pos = np.arange(sorted_idx.shape[0]) + .3

# Create a figure for the plot with a size of 12x7 inches for better visibility.

featfig = plt.figure(figsize=(12,7))

# Add a subplot to the figure.

featax = featfig.add_subplot(1, 1, 1)

# Create a horizontal bar chart showing the relative feature importance.

# The bar lengths represent how important each feature is in predicting diabetes.

featax.barh(pos, feature_importance[sorted_idx], align='center')

# Set the y-ticks (the positions of the features) and assign the feature names to the y-ticks.

# The labels are the feature names sorted according to their importance.

featax.set_yticks(pos)

featax.set_yticklabels(np.array(X_train.columns)[sorted_idx], fontsize=7)

# Label the x-axis as 'Relative Feature Importance' to indicate that the values show each feature's importance.

featax.set_xlabel('Relative Feature Importance')

# Adjust the layout of the plot to make it look clean and fit within the figure space.
```

```
plt.tight_layout()

# Display the plot.

plt.show()
```

Based on the plot above, the top 5 most important features contributing to the logistic regression model's prediction of diabetes are: Polydipsia (excessive thirst),Polyuria (frequent urination),Gender,Itching,Irritability These features play a significant role in determining whether a person is likely to have diabetes, as shown by their higher importance values in the model.

7.2. Heart Disease Code

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

import warnings

warnings.filterwarnings('ignore')

dataset = pd.read_csv("heart.csv")

type(dataset)
```

```
[3]: pandas.core.frame.DataFrame
```

```
dataset.shape
```

```
[4]: (303, 14)
```

```
dataset.head(5)
```

[5]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

```
dataset.sample(5)
```

[6]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
282	59	1	2	126	218	1	1	134	0	2.2	1	1	1	0
200	44	1	0	110	197	0	0	177	0	0.0	2	1	2	0
76	51	1	2	125	245	1	0	166	0	2.4	1	0	2	1
102	63	0	1	140	195	0	1	179	0	0.0	2	2	2	1
129	74	0	1	120	269	0	0	121	1	0.2	2	1	2	1

dataset.describe()

[9]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.313531	
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.612277	
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.000000	
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.000000	
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.000000	
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	

dataset.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

Luckily, we have no missing values, now see columns

info = ["age","1: male, 0: female","chest pain type, 1: typical angina, 2: atypical angina, 3: non-anginal pain, 4: asymptomatic","resting blood pressure"," serum cholestoral in mg/dl","fasting blood sugar > 120 mg/dl","resting electrocardiographic results (values 0,1,2)"," maximum heart rate achieved","exercise induced angina","oldpeak = ST depression induced by exercise relative to rest","the slope of the peak exercise ST segment","number of major vessels (0-3) colored by flourosopy","thal: 3 = normal; 6 = fixed defect; 7 = reversable defect"]

```
age:                age
sex:                1: male, 0: female
cp:                 chest pain type, 1: typical angina, 2: atypical angina, 3: non-anginal pain, 4: asymptomatic
trestbps:                   resting blood pressure
chol:                serum cholestoral in mg/dl
fbs:                fasting blood sugar > 120 mg/dl
restecg:                    resting electrocardiographic results (values 0,1,2)
thalach:                     maximum heart rate achieved
exang:              exercise induced angina
oldpeak:                    oldpeak = ST depression induced by exercise relative to rest
slope:              the slope of the peak exercise ST segment
ca:                 number of major vessels (0-3) colored by flourosopy
thal:               thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
```

dataset["target"].describe()

```
count    303.000000
mean       0.544554
std        0.498835
min        0.000000
25%        0.000000
50%        1.000000
75%        1.000000
max        1.000000
Name: target, dtype: float64
```

dataset["target"].unique()

```
[14]: array([1, 0], dtype=int64)
```

Clearly, this is a classification problem, with the target variable having values '0' and '1' Checking correlation between columns.

print(dataset.corr()["target"].abs().sort_values(ascending=False))

```
target      1.000000
exang       0.436757
cp          0.433798
oldpeak     0.430696
thalach     0.421741
ca          0.391724
slope       0.345877
thal        0.344029
sex         0.280937
age         0.225439
trestbps    0.144931
restecg     0.137230
chol        0.085239
fbs         0.028046
Name: target, dtype: float64
```

#This shows that most columns are moderately correlated with target, but 'fbs' is very weakly correlated.

Exploratory Data Analysis (EDA)

First, analysing the target variable:

#Count the values in the target column

target_temp = dataset["target"].value_counts() print(target_temp)

#Create the bar plot

sns.barplot(x=target_temp.index, y=target_temp.values)

#Add labels and title for clarity

plt.xlabel("Target") plt.ylabel("Count") plt.title("Distribution of Target") plt.show()

print("Percentage of patience without heart problems: "+str(round(target_temp[0]*100/303,2)))

print("Percentage of patience with heart problems: "+str(round(target_temp[1]*100/303,2)))

#Alternatively,

print("Percentage of patience with heart problems: "+str(y.where(y==1).count()*100/303))

print("Percentage of patience with heart problems: "+str(y.where(y==0).count()*100/303))

#Or,

countNoDisease = len(df[df.target == 0])

countHaveDisease = len(df[df.target == 1])

```
Percentage of patience without heart problems: 45.54
Percentage of patience with heart problems: 54.46
```

We'll analyse 'sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca' and 'thal' features Analysing the 'Sex' feature

dataset["sex"].unique()

```
[23]: array([1, 0], dtype=int64)
```

#Count the values of target for each sex category grouped_data = dataset.groupby("sex")["target"].value_counts().unstack()

Plot the data

grouped_data.plot(kind="bar", stacked=True, color=["#1f77b4", "#ff7f0e"], figsize=(7, 6))

Add labels and title

plt.xlabel("Sex (0: Female, 1: Male)") plt.ylabel("Count") plt.title("Target Distribution by Sex") plt.legend(title="Target", labels=["No Disease", "Disease"]) plt.show()

The chart shows the distribution of disease (target) by gender: Females (0): A significant proportion are dealing with health challenges, showcasing opportunities for focused health interventions.

Males (1): Many individuals are maintaining good health, highlighting positive trends in their well-being.

Analysing the 'Chest Pain Type' feature

dataset["cp"].unique()

```
[30]: array([3, 2, 1, 0], dtype=int64)
```

#Aggregate data: Calculate the mean of target for each chest pain type (cp)

grouped_data = dataset.groupby("cp")["target"].mean().reset_index()

#Create the bar plot

sns.barplot(x="cp", y="target", data=grouped_data)

#Add labels and title

plt.xlabel("Chest Pain Type (CP)") plt.ylabel("Mean Target (Disease Probability)") plt.title("Mean Disease Probability by Chest Pain Type")

#Add annotations with improved spacing and alignment

annotations = { 0: "Typical Angina\n(Heart-related pain)", 1: "Atypical Angina\n(Not heart-related)", 2: "Non-Anginal Pain\n(Other causes)", 3: "Asymptomatic\n(No pain)" }

for index, label in annotations.items(): plt.text(index, grouped_data.loc[index, "target"] , label, ha="center", fontsize=7, bbox=dict(facecolor="white", alpha=0.7, edgecolor="none"))

#Adjust layout for better visibility

plt.tight_layout() plt.show()

The chart shows:

- Typical Angina (0): Lowest disease probability.
- Atypical Angina (1): Moderate disease probability.
- Non-Anginal Pain (2): Higher disease probability.
- Asymptomatic (3): Highest disease probability despite no chest pain.

It highlights how chest pain types correlate with heart disease risks.

Analysing the FBS feature

dataset["fbs"].describe()

```
[46]: count    303.000000
      mean       0.148515
      std        0.356198
      min        0.000000
      25%        0.000000
      50%        0.000000
      75%        0.000000
      max        1.000000
      Name: fbs, dtype: float64
```

dataset["fbs"].unique()

```
[47]: array([1, 0], dtype=int64)
```

#Create the bar plot

sns.barplot(x="fbs", y="target", data=grouped_data)

#Add labels and title

plt.xlabel("Fasting Blood Sugar (FBS)") plt.ylabel("Mean Target (Disease Probability)") plt.title("Mean Disease Probability by FBS")

#Add annotations for explanation

annotations = { 0: "Normal (< 120 mg/dl)", 1: "High (≥ 120 mg/dl)" }

for index, label in annotations.items(): plt.text(index, grouped_data.loc[index, "target"], label, ha="center", fontsize=10, bbox=dict(facecolor="white", alpha=0.7, edgecolor="none"))

#Show the plot

plt.show()

The chart shows that individuals with normal fasting blood sugar (FBS < 120 mg/dl) have a slightly higher probability of disease compared to those with high fasting blood sugar (FBS ≥ 120 mg/dl) in this dataset. This indicates that higher blood sugar levels do not always correlate with an increased risk of disease here.

Analysing the restecg feature

dataset["restecg"].unique()

Aggregate data: Calculate the mean of target for each restecg category

grouped_data = dataset.groupby("restecg")["target"].mean().reset_index()

#Create the bar plot

sns.barplot(x="restecg", y="target", data=grouped_data)

#Add labels and title

plt.xlabel("Resting ECG Results (restecg)") plt.ylabel("Mean Target (Disease Probability)") plt.title("Mean Disease Probability by Resting ECG Results")

#Add annotations for explanations

annotations = { 0: "Normal ECG\n(No issues)", 1: "ST-T Wave Abnormality\n(T wave changes)", 2: "Left Ventricular Hypertrophy\n(Heart muscle thickening)" }

for index, label in annotations.items(): plt.text(index, grouped_data.loc[index, "target"] , label, ha="center", fontsize=9, bbox=dict(facecolor="white", alpha=0.7, edgecolor="none"))

#Show the plot

plt.tight_layout() plt.show()

The chart shows how different resting ECG results relate to disease probability:

Normal ECG (0): Moderate disease probability (~45%), indicating a balanced risk.

ST-T Wave Abnormality (1): Higher disease probability (~60%), suggesting increased heart strain or potential issues.

Left Ventricular Hypertrophy (2): Lowest disease probability (~25%), implying this condition in isolation may not strongly correlate with the disease.

Analysing the 'exang' feature

```
dataset["exang"].unique()
array([0, 1], dtype=int64)
#Aggregate data: Calculate the mean of target for each exang category
grouped_data = dataset.groupby("exang")["target"].mean().reset_index()
#Create the bar plot
sns.barplot(x="exang", y="target", data=grouped_data)
#Add labels and title
plt.xlabel("Exercise-Induced Angina (exang)") plt.ylabel("Mean Target (Disease Probability)") plt.title("Mean Disease Probability by Exercise-Induced Angina")
#Add annotations for explanations
annotations = { 0: "No Angina\n(No chest pain)", 1: "Exercise-Induced Angina\n(Chest pain during exercise)" }
for index, label in annotations.items(): plt.text(index, grouped_data.loc[index, "target"], label, ha="center", fontsize=9, bbox=dict(facecolor="white", alpha=0.7, edgecolor="none"))
#Show the plot
plt.tight_layout() plt.show()
```

No Angina (0): Higher disease probability (~65%).

Exercise-Induced Angina (1): Lower disease probability (~22%).

This suggests individuals without exercise-induced chest pain may have a higher risk of disease in this dataset.

Analysing the Slope feature

```
dataset["slope"].unique()
```

array([0, 2, 1], dtype=int64)

```
#Aggregate data: Calculate the mean of target for each slope category
grouped_data = dataset.groupby("slope")["target"].mean().reset_index()
#Create the bar plot
sns.barplot(x="slope", y="target", data=grouped_data)
#Add labels and title
plt.xlabel("Slope of ST Segment (slope)") plt.ylabel("Mean Target (Disease Probability)") plt.title("Mean Disease Probability by Slope of ST Segment")
#Add annotations for explanations
```

annotations = { 0: "Upsloping\n(Better heart health)", 1: "Flat\n(Possible blood flow issues)", 2: "Downsloping\n(Poor heart health)" }

for index, label in annotations.items(): plt.text(index, grouped_data.loc[index, "target"], label, ha="center", fontsize=9, bbox=dict(facecolor="white", alpha=0.7, edgecolor="none"))

#Show the plot

plt.tight_layout() plt.show()

Upsloping (0): Moderate disease probability (~45%).

Flat (1): Lowest disease probability (~30%).

Downsloping (2): Highest disease probability (~72%).

This highlights that a downsloping ST segment is strongly linked with higher heart disease risk.

Analysing the 'ca' feature #number of major vessels (0-3) colored by flourosopy

dataset["ca"].unique()

array([0, 2, 1, 3, 4], dtype=int64)

#Create the count plot

sns.countplot(x="ca", data=dataset, palette="viridis")

#Add labels and title

plt.xlabel("Number of Major Vessels (ca)") plt.ylabel("Count") plt.title("Distribution of Major Vessels Detected")

#Add annotations for explanations

annotations = { 0: "No vessels detected", 1: "One vessel detected", 2: "Two vessels detected", 3: "Three vessels detected", 4: "Four vessels detected" }

for index, label in annotations.items(): plt.text(index, dataset["ca"].value_counts().get(index, 0) , label, ha="center", fontsize=9, bbox=dict(facecolor="white", alpha=0.7, edgecolor="none"))

#Show the plot

plt.tight_layout() plt.show()

0 (No vessels): Most patients (~175) have no major vessels detected, indicating a common occurrence.

1 (One vessel): A moderate number (~50) have one major vessel detected.

2 (Two vessels): Fewer patients (~25) show two major vessels.

3 (Three vessels): Even fewer (~15) have three major vessels.

4 (Four vessels): Rare (~5) cases with four vessels detected.

This suggests that the majority of patients have no major vessels detected during fluoroscopy, with progressively fewer patients as the number of vessels increases

#Aggregate data: Calculate the mean of target for each ca category

grouped_data = dataset.groupby("ca")["target"].mean().reset_index()

#Create the bar plot

sns.barplot(x="ca", y="target", data=grouped_data)

#Add labels and title

plt.xlabel("Number of Major Vessels (ca)") plt.ylabel("Mean Target (Disease Probability)") plt.title("Mean Disease Probability by Number of Major Vessels")

#Add annotations for explanation

annotations = { 0: "No vessels detected", 1: "One vessel detected", 2: "Two vessels detected", 3: "Three vessels detected", 4: "Four vessels detected" }

for index, label in annotations.items(): plt.text(index, grouped_data.loc[index, "target"], label, ha="center", fontsize=9, bbox=dict(facecolor="white", alpha=0.7, edgecolor="none"))

#Show the plot

plt.tight_layout() plt.show()

0 vessels detected: High disease probability (~75%), indicating significant risk.

1 vessel detected: Moderate disease probability (~35%), showing a reduction in risk.

2 vessels detected: Lower disease probability (~20%), suggesting better heart health.

3 vessels detected: Lowest disease probability (~15%), indicating the least risk.

4 vessels detected: Disease probability increases again (~75%), showing outliers or specific cases.

This suggests that the number of major vessels plays an essential role in assessing heart health, with lower counts generally linked to higher risks, except for cases with ca=4

Analysing the 'thal' feature

dataset["thal"].unique()

array([1, 2, 3, 0], dtype=int64)

#Aggregate data: Calculate the mean of target for each thal category

grouped_data = dataset.groupby("thal")["target"].mean().reset_index()

#Create the bar plot

sns.barplot(x="thal", y="target", data=grouped_data)

#Add labels and title

plt.xlabel("Thalassemia Type (thal)") plt.ylabel("Mean Target (Disease Probability)") plt.title("Mean Disease Probability by Thalassemia Type")

#Add annotations for explanations

annotations = { 0: "No information/missing", 1: "Normal blood flow", 2: "Fixed defect (serious issues)", 3: "Reversible defect (manageable)" }

for index, label in annotations.items(): plt.text(index, grouped_data.loc[index, "target"], label, ha="center", fontsize=9, bbox=dict(facecolor="white", alpha=0.7, edgecolor="none"))

#Show the plot

plt.tight_layout() plt.show()

0 (No information/missing): Moderate disease probability (~45%), possibly indicating incomplete data.

1 (Normal blood flow): Lower disease probability (~25%), indicating a healthier status.

2 (Fixed defect): Highest disease probability (~75%), showing severe and permanent heart-related issues.

3 (Reversible defect): Lowest disease probability (~15%), suggesting manageable heart conditions.

This highlights that individuals with fixed defects face the highest risk, whereas those with reversible defects have the lowest probability of heart disease.

#Using histplot to visualize the distribution

sns.histplot(dataset["thal"], kde=True, bins=10, color='blue')

#Add labels and title

plt.xlabel("Thalassemia Type (thal)") plt.ylabel("Frequency") plt.title("Distribution of Thalassemia Types") plt.show()

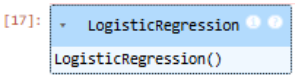
- Type 2 (Fixed Defect): Most common, indicating that permanent blood flow issues are prevalent.
- Type 3 (Reversible Defect): Second most frequent, suggesting manageable conditions are also significant.
- Type 1 (Normal): Less frequent, indicating fewer cases with normal blood flow.
- Type 0 (No Information): Rare, likely representing missing or incomplete data.

Train Test split

```
from sklearn.model_selection import train_test_split

predictors = dataset.drop("target",axis=1) target = dataset["target"]

X_train.shape

(242, 13)

X_test.shape

(61, 13)

Y_train.shape

(242,)

Y_test.shape

(61,)
```

Model Fitting

```
model = LogisticRegression()

#training the LogisticRegression model with Training data

model.fit(X_train, Y_train)
```




```
#accuracy on training data

X_train_prediction = model.predict(X_train)

training_data_accuracy = accuracy_score(X_train_prediction, Y_train)

print('Accuracy on Training data : ', training_data_accuracy)

Accuracy: 0.7512396694214777
#accuracy on test data

X_test_prediction=model.predict(X_test)

test_data_accuracy = accuracy_score(X_test_prediction, Y_test)

print('Accuracy on Test data : ', test_data_accuracy)

Accuracy on Test data :  0.719672131147541

Saving the trained model

import pickle
```

```
filename = 'heart_diseasemodel.sav'

pickle.dump(model, open(filename, 'wb'))

#loading the saved model

loaded_model = pickle.load(open('heart_diseasemodel.sav', 'rb'))
```

7.3. Frontend Code

```
from PIL import Image

 import joblib import os

import pickle

import streamlit as st

import pandas as pd

 import numpy as np

from streamlit_option_menu

import option_menu

import base64

#Set page configuration

st.set_page_config( page_title="Predict My Health", layout="wide", page_icon=" 👨‍⚕️ " )

#Get the working directory of the script

working_dir = os.path.dirname(os.path.abspath(file))

#Load the trained machine learning models

diabetes_model = pickle.load(open(f'{working_dir}/diabetes_prediction.sav', 'rb'))

heart_disease_model = pickle.load(open(f'{working_dir}/heart_disease_model.sav', 'rb'))

#Function to set background image

def set_bg_from_local(image_path): with open(image_path, "rb") as img_file: encoded_string = base64.b64encode(img_file.read()).decode()# Inject CSS for background image #Set background image (Update the file name if necessary)

set_bg_from_local("watermark14.jpg")

#Sidebar menu

st.sidebar.title("Menu") menu = st.sidebar.radio("Go to", ["Home", "About Disease", "Disease Prediction"])

#Home Page
```

```
if menu == "Home": st.title("Welcome to Predict My Health") st.write(""" This web application helps users predict the likelihood of developing certain diseases based on input features. It uses trained machine learning models to offer predictions for Diabetes and Heart Disease. """)

st.subheader("How to Use:")
st.write("""
1. Open the Main Menu (>) in the top-left corner.
2. Select a category: 'Diabetes Prediction' or 'Heart Disease Prediction'.
3. Enter the required details.
4. Click 'Predict' to get the results.
""")

st.subheader("Disclaimer:")
st.write("""
- This web app may not always provide accurate predictions. Please verify the results.
- This tool is for informational purposes only. Always consult a doctor for medical advice.
""")

st.subheader("Thank You for Visiting! 💚")

#About Disease Page

elif menu == "About Disease": st.title("About Diseases") disease_type = st.selectbox("Select Disease", ["Diabetes", "Heart Disease"])# Information about Diabetes
if disease_type == "Diabetes":
    st.subheader("Causes and Risk Factors:")
    st.write("""
    - **Genetics:** Family history of diabetes.
    - **Obesity:** Excess fat, especially in the abdomen.
    - **Physical Inactivity:** Lack of exercise.
    - **Poor Diet:** High sugar and processed food intake.
    - **Age & Ethnicity:** Higher risk for people over 45 and certain ethnic groups.
    """)

    st.subheader("Symptoms of Diabetes:")
    st.write("""
    - Frequent urination
    - Excessive thirst and hunger
    - Unexplained weight loss
    - Fatigue and blurred vision
    - Slow-healing wounds and infections
```

```
    - Numbness or tingling in hands/feet
    """)
# Information about Heart Disease
elif disease_type == "Heart Disease":
    st.subheader("Heart Disease"
    st.subheader("Common Types of Heart Disease:")
    st.write("""
    - **Coronary Artery Disease (CAD):** Blockage of coronary arteries leading to heart attacks.
    - **Heart Attack (Myocardial Infarction):** Oxygen deprivation causes heart muscle damage.
    - **Heart Failure:** The heart cannot pump blood efficiently.
    - **Arrhythmias:** Irregular heartbeats affecting circulation.
    - **Valvular Heart Disease:** Damage to heart valves affecting blood flow.
    """)

    st.subheader("Key Risk Factors:")
    st.write("""
    - **Age & Gender:** Older individuals and men have a higher risk.
    - **High Blood Pressure & Cholesterol:** Leads to artery blockages.
    - **Diabetes & Obesity:** Increases stress on the heart.
    - **Smoking & Inactivity:** Damages arteries and increases risk.
    - **Family History:** Genetics play a role.
    """)

    st.subheader("Symptoms of Heart Disease:")
    st.write("""
    - Chest pain or discomfort
    - Shortness of breath
    - Fatigue and weakness
    - Irregular heartbeats or palpitations
    - Swelling in legs, ankles, or feet
    """)
```

Disease Prediction Page

elif menu == "Disease Prediction": st.title("Disease Prediction Coming Soon! 🚀") st.write("This section will allow users to input their medical details and get predictions for diabetes and heart disease.") prediction_type = st.radio("Select Prediction Type", ["Diabetes Prediction", "Heart Disease Prediction"])

```
# **Diabetes Prediction**
if prediction_type == "Diabetes Prediction":
```

```
st.subheader("Diabetes Prediction")

 # Define input fields for user data
 age = st.number_input("Age", min_value=1, max_value=120, step=1)
 gender = st.selectbox("Gender", ['Male', 'Female'])
 polyuria = st.selectbox("Polyuria (Excessive urination)", ['No', 'Yes'])
 polydipsia = st.selectbox("Polydipsia (Excessive thirst)", ['No', 'Yes'])
 sudden_weight_loss = st.selectbox("Sudden Weight Loss", ['No', 'Yes'])
 weakness = st.selectbox("Weakness", ['No', 'Yes'])
 polyphagia = st.selectbox("Polyphagia (Excessive hunger)", ['No', 'Yes'])
 genital_thrush = st.selectbox("Genital Thrush", ['No', 'Yes'])
 visual_blurring = st.selectbox("Visual Blurring", ['No', 'Yes'])
 itching = st.selectbox("Itching", ['No', 'Yes'])
 irritability = st.selectbox("Irritability", ['No', 'Yes'])
 delayed_healing = st.selectbox("Delayed Healing", ['No', 'Yes'])
 partial_paresis = st.selectbox("Partial Paresis (Muscle weakness)", ['No', 'Yes'])
 muscle_stiffness = st.selectbox("Muscle Stiffness", ['No', 'Yes'])
 alopecia = st.selectbox("Alopecia (Hair Loss)", ['No', 'Yes'])
 obesity = st.selectbox("Obesity", ['No', 'Yes'])

 # Convert categorical inputs to numerical values
 gender = 0 if gender == 'Male' else 1
 inputs = [age, gender, polyuria, polydipsia, sudden_weight_loss, weakness, polyphagia,
        genital_thrush, visual_blurring, itching, irritability, delayed_healing,
        partial_paresis, muscle_stiffness, alopecia, obesity]

 # Map categorical features to numerical values
 inputs = [1 if val == 'Yes' else 0 if val == 'No' else val for val in inputs]

 # Prediction button
 if st.button("Predict"):
    # Reshape the inputs and make a prediction
    input_data = np.array(inputs).reshape(1, -1)
    prediction = diabetes_model.predict(input_data)[0]

    # Display the prediction result
    if prediction == 1:
       st.error("⚠ You may have Diabetes disease. Consult a doctor for further evaluation.")
    else:
       st.success("☑ You are unlikely to have heart disease. Stay healthy!")
```

```
elif prediction_type == "Heart Disease Prediction":
    st.subheader("Heart Disease Prediction")

    # User input fields
    # Collect user input
    col1, col2, col3 = st.columns(3)


  if st.button("Predict Heart Disease"):
        try:
                    # Scale input data
            input_data_scaled = scaler.transform(input_data_df)

            # Make prediction
            prediction = heart_disease_model.predict(input_data_scaled)

            # Display prediction
            if probability > 0.7:  # Use probability threshold instead of 0.5
                st.error("⚠ You may have heart disease. Consult a doctor for further evaluation.")
            else:
                st.success("☑ You are unlikely to have heart disease. Stay healthy!")

            st.write(f"**Prediction Probability:** {probability:.4f}")

        except Exception as e:
            st.error(f"Error: {e}")
```

CHAPTER 8

INTERFACE

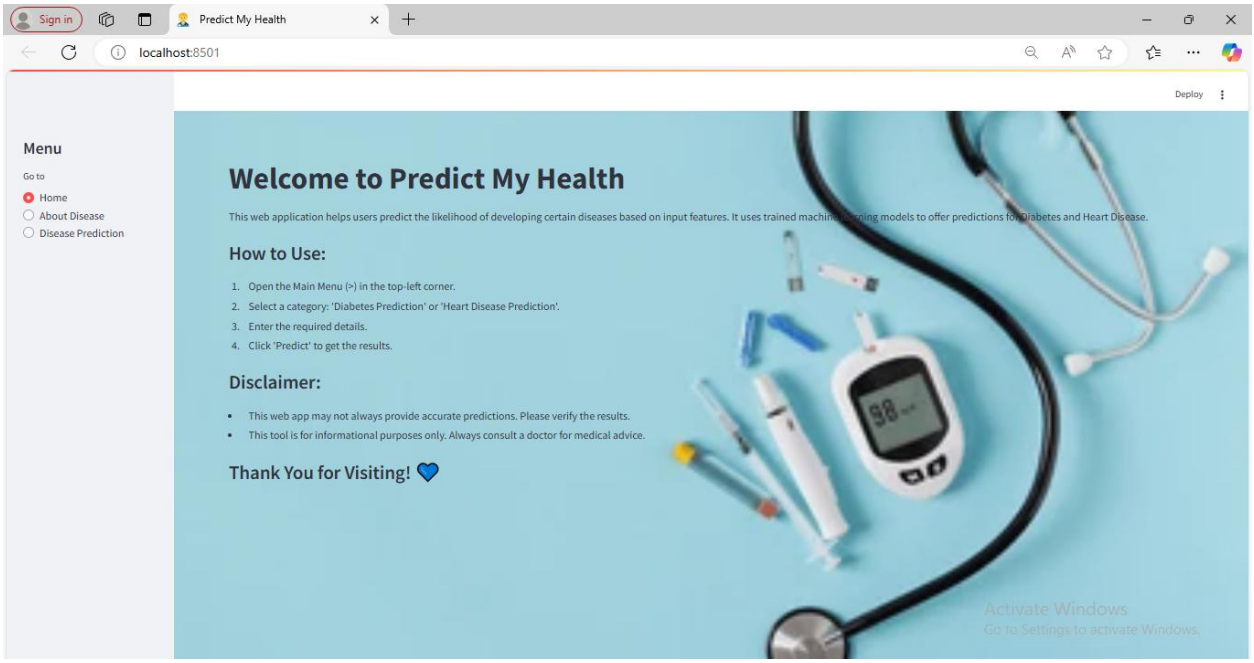


Fig.8.1. Home page

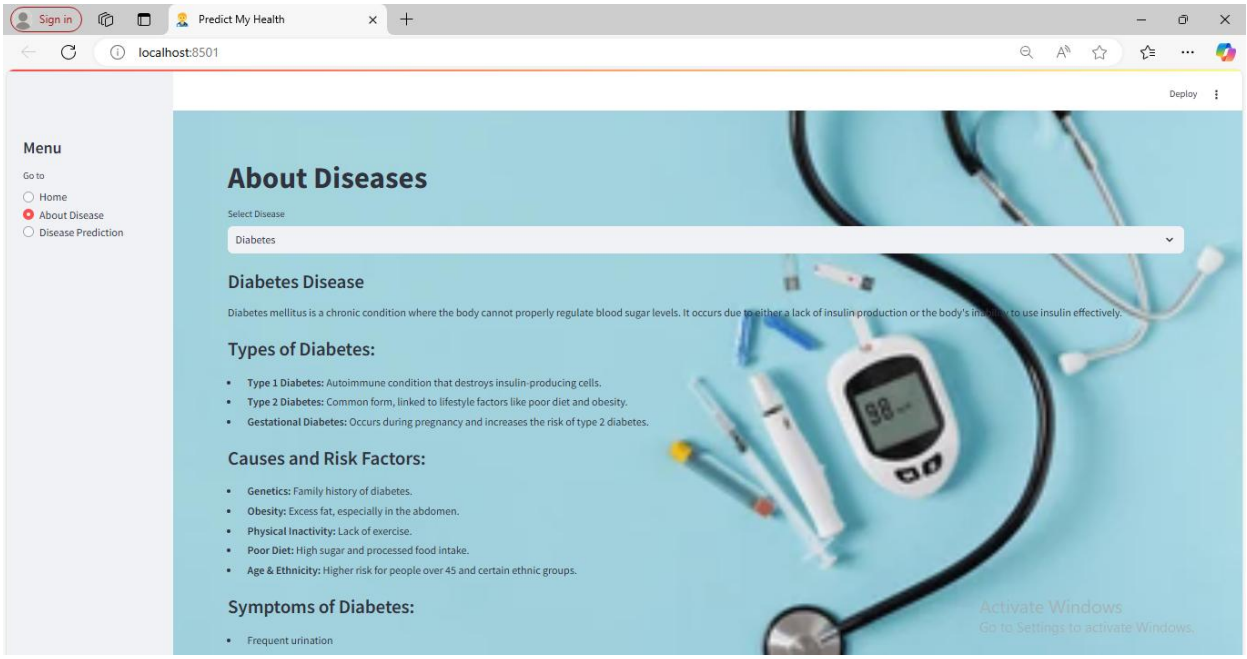


Fig.8.2. About Disease

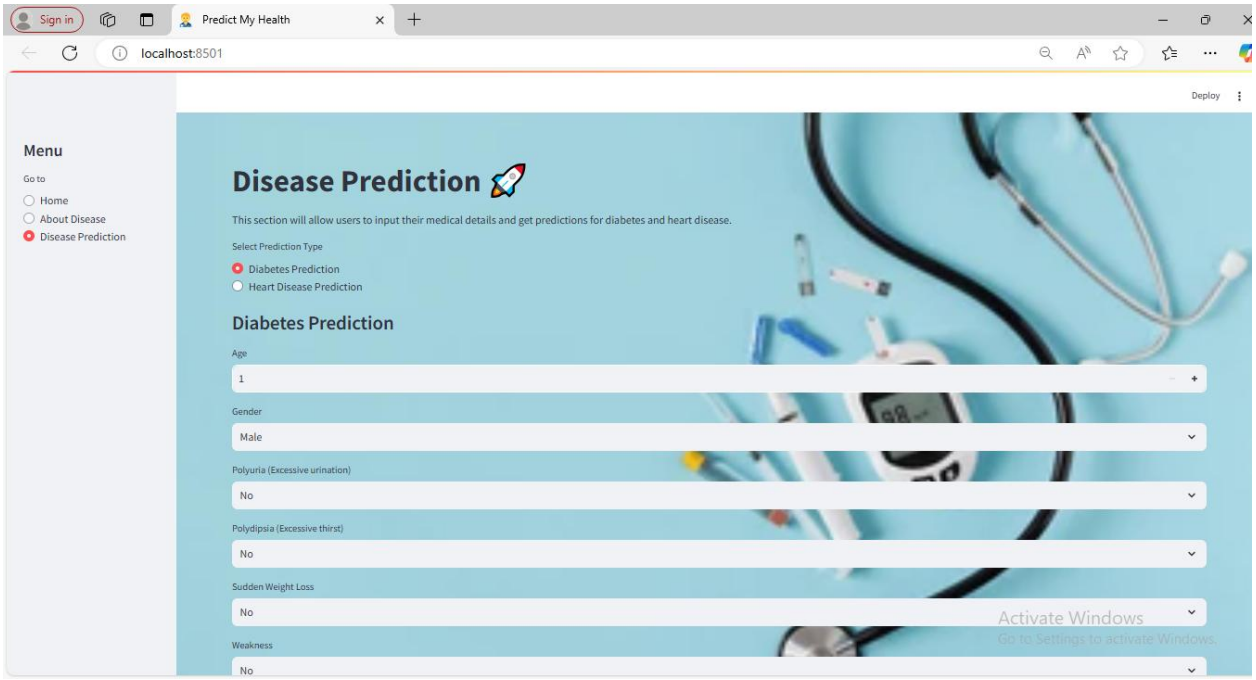


Fig.8.3. Diabetes Prediction

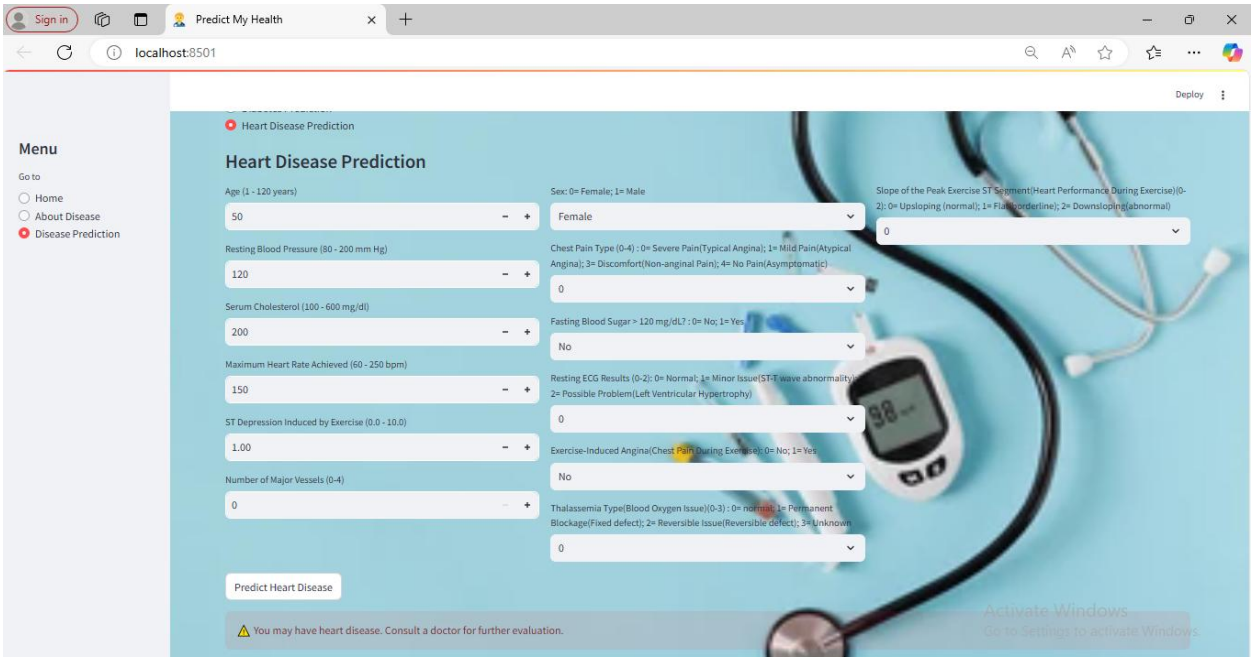


Fig.8.4. Heart Prediction

CHAPTER 9

FUTURE WORK FOR PREDICT MY HEALTH

Here are some potential directions for extending and enhancing the capabilities of Predict My Health application:

- **Expand Disease Prediction**

  - Additional Health Conditions:
    - Include predictions for other diseases such as hypertension, chronic kidney disease, or mental health conditions like depression.
    - Develop models for cancers such as breast cancer or skin cancer using relevant datasets.

  - Multi-Disease Predictions:

    - Allow users to input comprehensive health details to receive risk assessments for multiple diseases at once.

- **Improve Machine Learning Models**

  - Deep Learning Integration:
    - Experiment with advanced models such as neural networks for more complex disease predictions.
    - Incorporate frameworks like TensorFlow or PyTorch for better performance with larger datasets.

- **Interactive Features**

  - Personalized Reports:
    - Generate downloadable health reports summarizing prediction results and recommendations.
  - Data Visualization:
    - Create dynamic graphs and charts to show trends in user health metrics over time.
    - Integrate libraries like Plotly or Altair for interactive visualization.

  - Feedback Integration:
    - Allow users to provide feedback on predictions, helping improve model performance and user experience.

- **Multi-Language Support**

  - Develop a multilingual interface to make the application accessible to users across different regions.
  - Use language detection and translation APIs (e.g., Google Translate API) for real-time translations of prediction results and educational content.

- **Mobile Optimization**

  - Build a mobile-friendly version of the app with enhanced responsiveness for smartphones and tablets.
  - Create a dedicated mobile application using frameworks like React Native or Flutter.

- **Integration with External Systems**

  - Wearables and IoT:
    - Connect with devices such as smartwatches and fitness trackers to collect real-time health data (e.g., heart rate, blood pressure).
  - Medical APIs:
    - Integrate with medical databases and APIs (e.g., FHIR, HealthKit) to pull user-specific records and refine predictions.

- **Predictive Insights and Notifications**

  - Lifestyle Recommendations:
    - Offer personalized health tips based on user predictions (e.g., fitness routines, dietary suggestions).
  - Early Warning System:
    - Send alerts for significant health risks based on user data trends or predictions.

- **Real-Time Collaboration**

  - Professional Integration:
    - Allow users to share prediction results with healthcare providers directly through the app.
    - Include functionality for doctors to input additional clinical observations and refine predictions.

- **Continuous Learning**

  - User Data Insights:

    - Apply techniques like transfer learning to incorporate insights from user-provided data for continual model improvement.
- Community Contributions:
    - Create a platform for medical researchers to upload datasets or share findings that improve app capabilities.

Predict My Health provides a user-friendly, efficient, and secure deployment environment, making it an invaluable tool for health assessments. Whether deployed locally for personal use or on the cloud for broader accessibility, the application ensures seamless operation with minimal technical requirements. With its scalable architecture and continuous enhancements, Predict My Health is set to become an indispensable tool in AI-powered health diagnostics.

REFERENCES

[1] https://www.who.int/health-topics/cardiovascular-diseases#tab=tab_1

[2] https://www.netguru.com/blog/cloud-computing-in-healthcare

[3] https://gearedapp.co.uk/the-impact-of-wearable-devices-mobile-apps-on-health-monitoring/

[4] https://archive.ics.uci.edu/dataset/34/diabetes

[5] https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset

[6] https://machinelearningmastery.com/save-load-machine-learning-models-python-scikit-learn

[7] https://stackoverflow.com/questions/5226311/installing-specific-package-version-with-pip

[8] https://docs.python.org/3/library/index.html