Medico: A Disease Prediction and Health Recommendation System

A Project Report Submitted

In partial fulfilment of the requirements for the degree of

BACHELOR OF COMPUTER APPLICATION

(BCA)

Submitted by:

STUDENT NAME : Aman Singh

Roll No. : 4122121

under the guidance of

Mr. Arun Dabas
(Assistant Professor Dept. of Computer Science)

COMPUTER DEPARTMENT

Delhi Skill and Entrepreneurship University (DSEU)

Dwarka campus, Sector 9, Dwarka, New Delhi-110077

May 2025

Title of Project Work – Medico: A Disease Prediction and Health Recommendation System

Name of Student – Aman Singh

Roll Number – 41222121

Name of Guide- Mr. Arun Dabas

Designation- Assistant Professor

Student's Signature **Guide Signature**

I/C BCA

Declaration

I hereby declare that the project work entitled "**Medico: A Disease Prediction and Health Recommendation System** " submitted to Delhi Skill & Entrepreneurship University, Dwarka Campus is a record of an original work done by me under the guidance of "**Mr. Arun Dabas**". This project work is submitted in the partial fulfilment of the requirements for the award of the degree of Bachelor of Computer Application. The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Signature of Candidate

Aman Singh (41222121)

Bachelor of Computer Application

Delhi Skill And Entrepreneurship University

Certificate

This is to certify that the project report entitled "Medico: A Disease Prediction System using Machine Learning", submitted by Aman Singh (Roll No: 41222121), student of BCA, to Delhi Skill and Entrepreneurship University (DSEU), is a record of the candidate's own work carried out during the academic year 2024–2025 under the supervision and guidance of Mr. Arun Dabas.

The work embodied in this project is original and has not been submitted for the award of any other degree.

Certified By –

(signature)
Mr. Arun Dabas
(Assistant Professor Dept. of Computer Science)
DSEU DWARKA CAMPUS
Delhi Skill and Entrepreneurship University

Acknowledgement

I would like to express my sincere gratitude to my project guide Mr. Arun Dabas for his continuous support, encouragement, and valuable guidance throughout the project.

I am thankful to all the faculty members of DSEU for imparting their knowledge and for their invaluable contribution to my academic journey.

I would also like to thank my family and friends for their constant motivation and moral support during the completion of this project.

(signature)

Aman Singh (41222121)

Bachelor of Computer Application

Delhi Skill And Entrepreneurship University

Index

List of Figure

1. INTRODUCTION

The "Medico: Disease Prediction and Health Recommendation System" project is an innovative application that combines machine learning, healthcare data analysis, and web-based interaction to predict diseases based on user symptoms. This project demonstrates the power of artificial intelligence in preventive healthcare, offering early disease detection along with personalized health guidance. It leverages Python for building the prediction model and Gradio for creating an interactive and user-friendly web interface.

1.1 Project Background:

Healthcare systems worldwide often face challenges in providing timely diagnoses due to limited resources, delayed appointments, and lack of early interventions. These barriers can lead to patients missing early signs of diseases, resulting in delayed treatments and worse outcomes. While traditional healthcare systems are effective, they are often overwhelmed and unable to offer immediate care. Machine learning presents an opportunity to bridge this gap by providing fast, reliable disease predictions based on symptoms.

Medico addresses this issue by allowing users to input symptoms and receive immediate disease predictions, along with personalized recovery support, including diet plans, workouts, and medication suggestions. This AI-driven system brings preventive healthcare closer to everyday users, empowering them to take proactive steps in managing their health and seeking early intervention, ultimately reducing the burden on traditional healthcare systems.

1.2 Problem Statement:

Healthcare systems worldwide face significant challenges in delivering timely and accurate disease diagnoses due to limited medical resources, appointment delays, and lack of immediate expert consultations. These issues often cause patients to overlook early symptoms, leading to the progression of diseases to more severe stages. Furthermore, a general lack of awareness about symptoms and preventive measures complicates early detection and intervention. Although advancements in technology offer opportunities through artificial intelligence and machine learning, there remains a shortage of simple, user-friendly systems that allow individuals to assess their symptoms and receive preliminary health guidance independently. The "Medico: Disease Prediction and Health Recommendation System" addresses this need by providing a machine learning-based web application that predicts diseases based on user-reported symptoms using a Support Vector Classifier (SVC) model. Along with disease prediction, the system delivers personalized workout routines, diet plans, precautionary advice, and medication suggestions tailored to the diagnosed disease. By enhancing access to early diagnosis and preventive healthcare, Medico aims to promote individual health awareness, support early medical intervention, and reduce the burden on traditional healthcare systems.

1.3 Objective of the Project:

The primary objective of the "Medico: Disease Prediction and Health Recommendation System" project is to develop an AI-based web application that accurately predicts diseases based on user-provided symptoms and supports users with personalized health recommendations. Using a Random Forest model trained on medical datasets, the system aims to deliver fast and reliable predictions. In addition to disease identification, Medico provides a customized workout routine,

a tailored diet plan, precautionary measures, and suggested medications to promote early recovery and effective health management. The project seeks to make preventive healthcare more accessible, proactive, and user-friendly through the integration of machine learning and intuitive web technologies.

1.4 Scope of the Project:

The "Medico: Disease Prediction and Health Recommendation System" project focuses on developing a machine learning-based web application that predicts diseases based on user-inputted symptoms using an Random Forest. The system provides personalized workout routines, diet plans, precautionary measures, and medication suggestions to assist in health management. It is designed to offer predictive insights for common diseases and aims to support early intervention, but does not replace professional medical advice or handle complex multi-disease diagnoses.

1.5 Tools and Technologies:

The "Medico: Disease Prediction and Health Recommendation System" uses several key technologies to ensure efficiency and reliability. Python is employed for developing the core prediction model and handling data processing. The Scikit-learn library is used to implement the Random Forest model for disease prediction. Pandas and NumPy are utilized for data management and numerical operations, supporting smooth data transformations. Gradio is used to build a simple and interactive web application interface, allowing users to input symptoms and receive predictions easily. Jupyter Notebook is used during development for testing and refining the system. These tools together create a robust and user-friendly healthcare application.

1.6 Significance of the Project

The "Medico" system brings the power of artificial intelligence into healthcare by enabling early disease prediction and personalized health recommendations. By offering instant predictions based on symptoms along with workout plans, diet suggestions, precautions, and medication guidance, it empowers users to manage their health proactively. The project highlights the role of machine learning in promoting preventive care and making healthcare tools more accessible to the general public, ultimately aiming to improve health outcomes and support early intervention.

2. LITERATURE SURVEY

2.1 Opening:

The rapid advancement in the fields of Artificial Intelligence (AI) and Machine Learning (ML) has significantly transformed the healthcare sector, particularly in the areas of early disease prediction and diagnosis. Traditionally, disease detection heavily relied on clinical expertise, laboratory testing, and physical examinations, which are often time-consuming and not always accessible to every section of the population. In this context, machine learning presents an efficient alternative by automating the diagnosis process based on patient symptoms and clinical data, making healthcare faster, easier, and more accessible.

2.2 Previous Work:

Several research studies have applied machine learning algorithms such as Decision Trees, Random Forests, Naïve Bayes classifiers, and Support Vector Machines (SVM) for disease prediction. Among these, Decision Tree classifiers are widely preferred due to their simplicity, interpretability, and effective performance, particularly in healthcare applications. Research findings indicate that symptom-based disease prediction models are capable of achieving reasonable accuracy, thereby supporting early intervention and treatment.

2.3 Limitations of Existing Systems:

Despite progress in machine learning, current disease prediction systems still have limitations.

- Many existing models are limited to disease prediction and do not provide post-diagnosis recommendations such as personalized workout and diet plans.
- A majority of models require detailed clinical reports (e.g., blood tests, X-rays) rather than being based on self-reported symptoms.
- Several healthcare applications possess complex and highly technical interfaces, making them difficult for non-specialized users to operate.

2.4 How the Proposed Project "Medico" is Better:

The proposed system, "Medico", aims to address these challenges by:

- Facilitating symptom-based disease prediction without the necessity of laboratory tests.
- Offering personalized workout routines and diet recommendations after the disease prediction phase.
- Providing an interactive, intuitive, and easy-to-use web application developed using Gradio, ensuring smooth accessibility even for non-technical users.

3. OBJECTIVE

The primary objective of this project is to develop an intelligent, accessible, and user-friendly disease prediction system that not only identifies potential health conditions based on user-reported symptoms but also provides personalized health recommendations. The system aims to bridge the gap between diagnosis and post-diagnosis support, encouraging holistic health management.

3.1 Symptom-Based Disease Prediction:

- To design a machine learning model capable of predicting diseases using only user-input symptoms, eliminating the need for detailed laboratory reports.
- To ensure that the model handles a wide range of common symptoms and diseases with high accuracy and efficiency.
- To utilize the Random Forest algorithm for accurate and fast predictions.

3.2 Personalized Health Recommendations:

- To integrate a post-diagnosis support system that offers tailored workout routines and diet plans based on the predicted disease.
- To promote lifestyle changes that support recovery, management, and prevention of health conditions.
- To ensure recommendations are medically appropriate and easily understandable for non-expert users.

3.3 User-Friendly Interface:

- To build an intuitive and accessible web application interface using Gradio that allows users to interact with the system without requiring technical knowledge.
- To minimize input complexity and provide a seamless user experience, even for first-time users.

3.4 Real-Time Feedback and Accessibility:

- To ensure that the system processes user inputs and provides predictions and recommendations in real time.
- To make the platform lightweight and responsive so that it can be used across various devices and network conditions.

3.5 Educational and Preventive Healthcare Promotion:

- To educate users on the importance of symptom monitoring and proactive health behavior.
- To help reduce the load on traditional healthcare systems by enabling early diagnosis and prevention through self-assessment tools.

4. WORK PLAN AND METHODOLOGY

The methodology adopted for the development of "Medico: A Disease Prediction System using Machine Learning" is systematically structured into multiple well-defined stages, ensuring an organized and efficient workflow. Each stage plays a vital role in building a robust system that offers users predictive healthcare services. The stages are detailed as follows:

4.1 Problem Definition:

The initial phase focused on the clear and precise definition of the problem statement. It was identified that:

- **Existing healthcare solutions** predominantly focus on disease prediction but lack post-diagnosis support, such as personalized workout and diet recommendations. This makes the solutions less comprehensive and user-centric.
- **Many systems require clinical data**, making them unsuitable for individuals who need immediate symptom-based predictions for early intervention.
- **Technical complexity** in many existing applications often hinders accessibility, especially for the general public who may not be tech-savvy.

As a solution, **Medico** was conceptualized to address these issues by creating a system that allows users to simply input their symptoms and receive accurate disease predictions, along with tailored health advice, such as personalized workouts and diet plans, all through a simple web interface. This ensures a more accessible and holistic approach to healthcare.

4.2 Data Collection and Preprocessing:

Data collection and preprocessing are crucial to ensure the system's accuracy. For this project, a **relevant dataset** containing symptoms and disease labels was collected from **Kaggle**, a trusted source for health-related data.

- **Data Cleaning**: The dataset underwent rigorous cleaning processes, including the removal of duplicates, handling missing values, and standardizing symptom nomenclature to ensure uniformity across the dataset.
- **Label Encoding**: Categorical variables (such as disease names) were transformed into numerical formats using label encoding, making them suitable for machine learning algorithms.
- **Data Splitting**: The dataset was divided into two subsets: training and testing data, which enabled the model to be trained on one set and evaluated on a separate, unseen set, preventing overfitting.

4.3 Model Selection and Training:

A variety of machine learning models were explored, including Decision Tree, Random Forest, and Naïve Bayes classifiers. After performing a comparative analysis of these models, the Random Forest was selected for the following reasons:

- High accuracy: Random Forest has proven to perform well with high-dimensional data, which is often the case with symptom-based classification.

- Effective for both binary and multi-class classification: The system needed to classify diseases from a wide range of categories, making Random Forest a suitable choice.
- Good generalization: Random Forest models tend to generalize well, reducing the risk of overfitting, especially when dealing with a relatively small dataset of disease-symptom pairs.

Once the model was chosen, it was trained on the training dataset. Hyperparameter tuning was performed using techniques like Grid Search and Cross-Validation to optimize the model's performance. The model's prediction accuracy, precision, and recall were also evaluated to ensure it met the necessary performance criteria.

4.4 System Development:

The system development process was split into two core components: backend and frontend development.

- **Backend Development**:
  - **Integration of the Model**: Python was used to integrate the Random Forest model into the backend. The trained model was serialized using **pickle** to enable easy loading during real-time prediction.
  - **API for Symptom Input**: Functions were developed to process user-input symptoms and predict the most likely disease. The backend also handles communication between the model and the frontend.
- **Frontend Development**:
  - **Gradio Interface**: To ensure a user-friendly experience, **Gradio** was used for the frontend. Gradio's simple application interface allows users to input their symptoms through text fields or dropdown menus, making the system accessible even for those with minimal technical knowledge.
  - **Display Predictions and Recommendations**: After disease prediction, the system automatically displays customized **workout routines**, **diet plans**, and **medication recommendations** based on the diagnosed disease, offering a comprehensive approach to health management.

4.5 Post-Prediction Recommendations:

Once a disease is predicted, the system retrieves specific recommendations for the user's health management. These include:

- **Workout Routines**: Tailored exercise plans are designed based on the predicted disease, aiming to aid in recovery and improve overall health.
- **Diet Plans**: Nutritional guidance is provided, with meal suggestions that are aligned with the user's health needs.
- **Precautionary Measures**: The system offers advice on how to prevent further complications related to the disease, such as lifestyle changes, hygiene practices, and medical checkups.

- **Medication Suggestions**: Relevant medications are recommended based on the disease, providing users with a foundational starting point for seeking medical advice.

This post-prediction support aims to encourage users to take early action in managing their health and improving their recovery process.

4.6 Testing and Evaluation:

The system underwent rigorous testing to ensure its reliability and functionality. The testing phase involved:

- **Model Evaluation**: The system was tested on the test dataset, evaluating performance metrics such as accuracy, precision, recall, and F1 score to assess the predictive power of the Random Forest model.
- **User Interface Testing**: Several test cases were run to ensure that the user interface is intuitive and easy to navigate. Feedback was collected from a group of test users to ensure accessibility for individuals with varying technical expertise.

The system was iteratively improved based on these testing results, ensuring a seamless experience for end users.

4.7 Future Enhancements:

- **Voice-Based Input**: Incorporating voice recognition to allow users to report symptoms verbally, improving accessibility for users with disabilities or those who prefer voice input.
- **Expanding the Disease Database**: The inclusion of rare and chronic diseases in the system's database will make the tool more comprehensive and relevant to a wider audience.
- **Mobile App Development**: A mobile version of the system could be developed to make it even more accessible, allowing users to access the service on-the-go.

5. SRS, FLOW CHART, SYSTEM DIAGRAM

5.1 Software Requirements Specification (SRS):

Project Title: Medico – A Disease Prediction System using Machine Learning

Prepared By: Aman Singh

Institution: Delhi Skill and Entrepreneurship University (DSEU)

5.1.1 Introduction:

The purpose of this Software Requirements Specification (SRS) is to define the complete functionality, system requirements, and constraints of the project "Medico – A Disease Prediction System using Machine Learning." This system is developed to simplify disease prediction using symptom-based input and provide additional support through personalized workout and diet recommendations. The document aims to guide developers, researchers, and stakeholders by detailing both functional and non-functional aspects of the system.

This project addresses the limitations of traditional healthcare systems that primarily depend on clinical reports, are inaccessible to rural populations, or offer only diagnosis without guidance on recovery. "Medico" is envisioned as a web-based health assistant that uses a Random Forest to predict diseases based on simple symptom inputs and offers health recommendations to users without requiring technical knowledge or medical reports.

5.1.2 System Overview:

"Medico" is a standalone web application developed using Python, where users can input symptoms in a simple interface built using Gradio. The application preprocesses these symptoms, feeds them to an Random Forest model trained on a large dataset of diseases and symptoms, and displays the most probable disease along with related health advice. The application is lightweight, accessible via any browser, and does not require login or installation, making it suitable for public use.

The system architecture includes a Python backend for data processing and ML model inference, and a frontend using Gradio for user interaction. The machine learning model is trained using scikit-learn and relies on structured data for reliable disease prediction. It provides users not only with the diagnosis but also corresponding diet and workout suggestions sourced from a predefined mapping.

5.1.3 Functional Requirements:

The core functionality of "Medico" includes accepting user-input symptoms, predicting possible diseases using a trained ML model, and returning post-diagnosis recommendations. Users will be able to select symptoms from a dropdown list or type them manually. These symptoms are then encoded and passed to the ML model, which predicts the disease. Based on the predicted disease, the system will fetch a relevant set of recommendations from a database that includes diet and workout plans.

Additionally, the system supports basic feedback functionality, allowing users to report inaccurate results. This helps improve the dataset and model over time. The interface will be clean and responsive, with clear input fields, prediction results, and action buttons.

5.1.4 Non-Functional Requirements:

The application must be responsive, reliable, and secure. It should process inputs and generate predictions within a few seconds. The model should offer high accuracy (>85%) on common disease predictions. Usability is also a major concern, so the UI must be intuitive even for non-technical users. Input validation and sanitization will be handled to avoid errors or malicious input.

Security measures will ensure no personal data is stored or shared. As the application is meant to be accessible online, it must support all modern browsers and function on both mobile and desktop devices. The application should be maintainable and easy to update with improved datasets and model versions in the future.

5.1.5 User Characteristics:

The target users of this system include the general public, especially those without technical or medical backgrounds. The interface is simplified for ease of use, and instructions are minimal. Researchers or healthcare students may also use this tool to study AI-based diagnosis methods. Users only need a basic understanding of symptoms and a web browser to access the application.

5.1.6 External Interfaces:

The user interface will be built using Gradio, providing an interactive web layout where users input symptoms and receive disease predictions and lifestyle suggestions. Internally, Python libraries such as pandas and scikit-learn will handle data loading, preprocessing, and model prediction. No external hardware is required. All processing is done in the browser or on a local/cloud server.

5.1.7 System Architecture:

The system consists of a frontend (Gradio) and backend (Python-based ML model). The backend loads a trained Support Vector Classifier model and handles preprocessing and prediction. Once a disease is predicted, it retrieves corresponding health plans from CSV-based mappings.

5.1.8 Future Scope:

Future enhancements to Medico include integrating voice-based input for accessibility, expanding the dataset to include rare diseases, and providing real-time diet and workout updates via third-party APIs. The addition of multilingual support and wearable device integration is also considered to make the system more inclusive and intelligent.

5.2 Flow Chart:

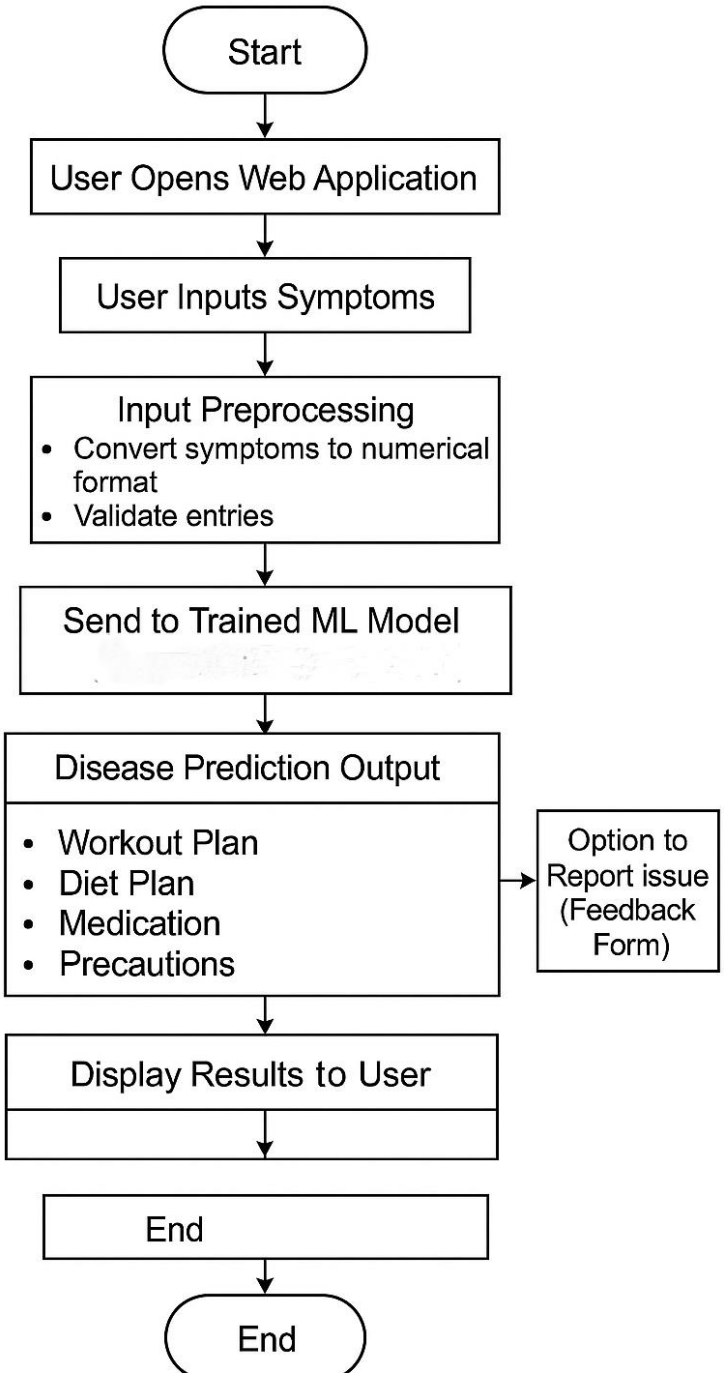

Figure 5.2.1: Flow Chart

5.3 System Diagram:

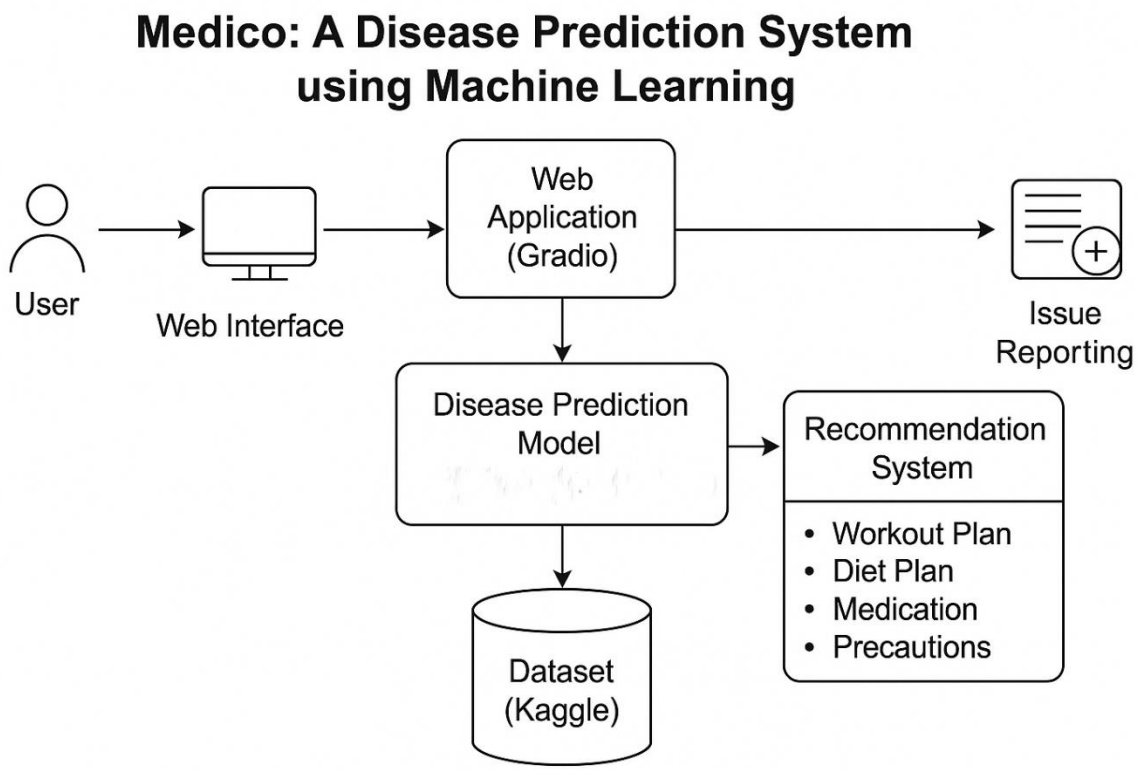

Figure 5.3.1: System Diagram

6. IMPLEMENTATION/ CODE

6.1 Data Collection and Preprocessing:

```
import pandas as pd
data = pd.read_csv("C:/Users/AMAN/Desktop/dataset/Training.csv")
data.columns
data.shape
data.size
data.count()
data.info()
data.isnull()
data.isnull().sum()
data.duplicated().sum()
data.duplicated()
symptom_columns = data.columns[:-1]
symptom_counts = data[symptom_columns].sum().sort_values(ascending=False)
import matplotlib.pyplot as plt
import seaborn as sns
corr = data[symptom_counts.head(10).index].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix of Top 10 Symptoms')
plt.show()
from sklearn.preprocessing import LabelEncoder
X = data.drop('prognosis', axis=1)
y = data['prognosis']
le = LabelEncoder()
le.fit(y)
Y = le.transform(y)
```

6.2 Model Training and Testing:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=20)
X_train
X_test
y_train
y_test
import numpy as np
from sklearn.datasets import make_classification
```

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
models = {
    'SVC': SVC(kernel='linear'),
    'RandomForest': RandomForestClassifier(n_estimators=100, random_state=42),
    'GradientBoosting': GradientBoostingClassifier(n_estimators=100, random_state=42),
    'KNeighbors': KNeighborsClassifier(n_neighbors=5),
}
import numpy as np
X_train = np.ascontiguousarray(X_train)
X_test = np.ascontiguousarray(X_test)
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix, classification_report
for model_name, model in models.items():
 # Train the model
    model.fit(X_train, y_train)
 # Test the model
    predictions = model.predict(X_test)
    # Calculate accuracy score of Models
    accuracy = accuracy_score(y_test, predictions)
    print(f"{model_name} Accuracy       : {accuracy}")
     # Calculate precision score of Models
    precision = precision_score(y_test, predictions, average='weighted', zero_division=0)
    print(f"{model_name} Precision      : {precision}")
    # Calculate recall score of Models
    recall = recall_score(y_test, predictions, average='weighted', zero_division=0)
    print(f"{model_name} Recall         : {recall}")
    # Calculate f1 score of Models
    f1 = f1_score(y_test, predictions, average='weighted', zero_division=0)
    print(f"{model_name} F1 Score       : {f1}")
    # Calculate f1 score of Models
    cm = confusion_matrix(y_test, predictions)
    print(f"{model_name} Confusion Matrix:\n{cm}")
    print("=======================================================")
svc = RandomForestClassifier(n_estimators=100, random_state=42)
svc.fit(X_train,y_train)
ypred = svc.predict(X_test)
```

```
# Save predection Model
import pickle
pickle.dump(svc,open("C:/Users/AMAN/Desktop/Final/svc.pkl",'wb'))
# Load predection Model
svc = pickle.load(open("C:/Users/AMAN/Desktop/Final/svc.pkl",'rb'))
```

6.3 Prediction script:

```
diets = pd.read_csv("C:/Users/AMAN/Desktop/dataset/diets.csv")
diets.columns
diets.info()
diets.size
diets.shape
diets.count()
import seaborn as sns
import matplotlib.pyplot as plt
from collections import Counter
all_diets = [diet for sublist in diets['Diet'] for diet in sublist]
diet_freq = Counter(all_diets)
plt.figure(figsize=(10, 5))
top_diets = diet_freq.most_common(10)
sns.barplot(x=[x[1] for x in top_diets], y=[x[0] for x in top_diets])
plt.title('Top 10 Most Common Diet Recommendations')
plt.xlabel('Frequency')
plt.ylabel('Diet Type')
plt.tight_layout()
plt.show()
workout = pd.read_csv("C:/Users/AMAN/Desktop/dataset/workout_df.csv")
workout.head(10)
workout.columns
workout.info()
workout.size
workout.shape
workout.count()
precaution = pd.read_csv("C:/Users/AMAN/Desktop/dataset/precautions_df.csv")
precaution.head(10)
precaution.columns
precaution.info()
precaution.size
precaution.shape
precaution.count()
```

```
# Count the number of unique diseases
num_diseases = precaution['Disease'].nunique()
# Count the most common precautions
precautions = pd.concat([
    precaution['Precaution_1'],
    precaution['Precaution_2'],
    precaution['Precaution_3'],
    precaution['Precaution_4']
])
precaution_counts = precautions.value_counts(dropna=True).head(10)
plt.figure(figsize=(10,5))
sns.barplot(x=precaution_counts.values, y=precaution_counts.index, palette='viridis')
plt.title('Top 10 Most Common Precautions')
plt.xlabel('Count')
plt.ylabel('Precaution')
plt.tight_layout()
plt.show()
print('Number of unique diseases:', num_diseases)
print('Top 10 most common precautions:')
precaution_counts
description = pd.read_csv("C:/Users/AMAN/Desktop/dataset/description.csv")
description.head(10)
description.columns
description.info()
description.size
description.shape
description.count()
medications = pd.read_csv("C:/Users/AMAN/Desktop/dataset/medications.csv")
medications.head(10)
medications columns
medications.info()
medications.size
medications.shape
medications.count()
symtoms = pd.read_csv("C:/Users/AMAN/Desktop/dataset/symtoms_df.csv")
symtoms.head(10)
medications columns
symtoms.info()
symtoms.size
```

```
symtoms.shape
symtoms.count()
def helper(dis):
    desc = description[description['Disease'] == predicted_disease]['Description']
    desc = " ".join([w for w in desc])
    pre = precaution[precaution['Disease'] == dis][['Precaution_1', 'Precaution_2', 'Precaution_3',
    'Precaution_4']]
    pre = [col for col in pre.values]
    med = medications[medications['Disease'] == dis]['Medication']
    med = [med for med in med.values]
    die = diets[diets['Disease'] == dis]['Diet']
    die = [die for die in die.values]
    wrkout = workout[workout['disease'] == dis] ['workout']
    return desc,pre,med,die,wrkout
def get_predicted_value(patient_symptoms):
    input_vector = np.zeros(len(symptoms_dict))
    for item in patient_symptoms:
        input_vector[symptoms_dict[item]] = 1
    return diseases_list[svc.predict([input_vector])[0]]
symptoms = input("Enter your symptoms.....")
user_symptoms = [s.strip() for s in symptoms.split(',')]
# Remove any extra characters, if any
user_symptoms = [symptom.strip("[]' ") for symptom in user_symptoms]
predicted_disease = get_predicted_value(user_symptoms)
desc, pre, med, die, wrkout = helper(predicted_disease)
print("=================predicted disease============")
print(predicted_disease)
print("=================description==================")
print(desc)
print("=================precautions==================")
i = 1
for p_i in pre[0]:
    print(i, ": ", p_i)
    i += 1
print("=================medications==================")
for m_i in med:
    print(i, ": ", m_i)
    i += 1
print("=================workout==================")
```

```
for w_i in wrkout:
    print(i, ": ", w_i)
    i += 1
print("=================diets=================")
for d_i in die:
    print(i, ": ", d_i)
    i += 1
```

6.4 Gradio Application Development:

```
import gradio as gr
import numpy as np
import pandas as pd
import os
import datetime
import uuid
from pymongo import MongoClientsymptoms_dict = {symptom: idx for idx, symptom in
enumerate(X.columns)}
diseases_list = le.classes_
MONGO_URI = "mongodb://localhost:27017/"
client = MongoClient(MONGO_URI)
db = client["medico_ai"]
users_collection = db["users"]
patients_collection = db["patients"]
flagged_issues_collection = db["flagged_issues"]
def get_dashboard_metrics():
    total_patients = patients_collection.count_documents({})
    total_users = users_collection.count_documents({})
    total_flags = flagged_issues_collection.count_documents({})
    return f"""
    **📊 Dashboard Metrics**
    - 👥 Total Diagnosed Patients: {total_patients}
    - 👩‍💻 Total Registered Users: {total_users}
    - 🚩 Total Flagged Issues: {total_flags}
    """
def hash_password(password):
    return bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())
def check_password(password, hashed):
    return bcrypt.checkpw(password.encode('utf-8'), hashed)
def register_user(username, password,email):
```

```
    if users_collection.find_one({"username": username}):
        return "Username already exists."
    hashed = hash_password(password)
    users_collection.insert_one({"username": username,"email": email, "password": hashed})
    return "Registration successful."
def authenticate_user(username, password,email):
    user = users_collection.find_one({"username": username})
    if user and check_password(password, user["password"]):
        return True
    return False
def get_predicted_value(patient_symptoms):
    input_vector = np.zeros(len(symptoms_dict))
    for item in patient_symptoms:
        if item in symptoms_dict:
            input_vector[symptoms_dict[item]] = 1
    return diseases_list[svc.predict([input_vector])[0]]
def helper(dis):
    desc = description[description['Disease'] == dis]['Description']
    desc = " ".join([w for w in desc])
    pre = precaution[precaution['Disease'] == dis][['Precaution_1', 'Precaution_2', 'Precaution_3',
    'Precaution_4']]
    pre = [col for col in pre.values]
    med = medications[medications['Disease'] == dis]['Medication']
    med = [med for med in med.values]
    die = diets[diets['Disease'] == dis]['Diet']
    die = [die for die in die.values]
    wrkout = workout[workout['disease'] == dis]['workout']
    wrkout = [wrk for wrk in wrkout.values]
    return desc, pre, med, die, wrkout
def predict_disease(name, gender, age, address, symptoms_str):
    try:
        patient_symptoms = [s.strip().lower() for s in symptoms_str.split(",")]
        disease = get_predicted_value(patient_symptoms)
        patient_id = str(uuid.uuid4())
        patient_info = {
            "patient_id": patient_id,
            "name": name,
            "gender": gender,
            "age": age,
```

```
            "address": address,
            "symptoms": patient_symptoms,
            "disease": disease
        }
        patients_collection.insert_one(patient_info)
        return disease, disease, patient_info
    except Exception as e:
        return f"Error: {str(e)}", "", {}
def get_description(disease):
    return helper(disease)[0]
def get_precautions(disease):
    return "\n".join(helper(disease)[1][0])
def get_medications(disease):
    return "\n".join(helper(disease)[2])
def get_diets(disease):
    return "\n".join(helper(disease)[3])
def get_workouts(disease):
    return "\n".join(helper(disease)[4])
def flag_case(disease, symptoms, issue):
    # Get current date and time
    now = datetime.datetime.now()
    timestamp = now.strftime("%Y-%m-%d %H:%M:%S")
    # Prepare the flag text
    text = f"""🕒 Time: {timestamp}
☑ Flag received for: {disease}
🩺 Symptoms: {symptoms}
⚠ Issue: {issue}
"""
    # Save the flag to a text file
    with open("flagged_issues.txt", "a", encoding="utf-8") as f:
        f.write(text)
    # Also return the acknowledgment for displaying in the app
    return f"☑ Flag submitted successfully for {disease}!"
with gr.Blocks(css="""
    body { background-color: #121212; color: #E0E0E0; font-family: 'Segoe UI', sans-serif; }
    .gr-box { border: 1px solid #333333; padding: 14px; border-radius: 12px; background:
     #1E1E1E; color: #E0E0E0; font-weight: bold; }
    .tabs-bg { background: #1E1E1E; padding: 20px; border-radius: 15px; }
    .highlight { background: #263238; border-left: 5px solid #00acc1; padding: 10px; border-
```

```
    radius: 8px; color: #E0E0E0; font-weight: bold; }
    .title { text-align: center; color: #00bcd4; font-size: 2.5em; font-weight: bold; }
    .important-btn { background-color: #00acc1 !important; color: white !important; border: none
    !important; font-weight: bold; }
    .flag-btn { background-color: #f44336 !important; color: white !important; border: none
    !important; font-weight: bold; }
""") as demo:
gr.Markdown("""<h1 class='title'>🩺 Medico AI Assistant</h1>""")
    disease_state = gr.State()
    with gr.Tabs():
        with gr.Tab("🏠 Home"):
            gr.Markdown("""# 🏥 Welcome to Medico Health Portal
             Stay informed and take charge of your health with our comprehensive resources and
             tools.
            """
        # Useful Links
            gr.Markdown("""
            ## 🔗 Useful Resources

            - [World Health Organization](https://www.who.int)
            - [Centers for Disease Control and Prevention](https://www.cdc.gov)
            - [Healthline: Medical Information](https://www.healthline.com)
            """)
        # Register Tab
        def register_interface(username, password,email):
            return register_user(username, password,email)
        with gr.Tab("📝 Register"):
            reg_email = gr.Textbox(label="Email")
            reg_username = gr.Textbox(label="Username")
            reg_password = gr.Textbox(label="Password", type="password")
            reg_button = gr.Button("📝 Register")
            reg_output = gr.Textbox(label="Registration Status")
            reg_button.click(register_interface, inputs=[reg_username, reg_password, reg_email],
             outputs=reg_output)
        # 🩺 Diagnosis Tab
        with gr.TabItem("🩺 Diagnosis"):
            with gr.Row():
                name_input = gr.Textbox(label="Name", placeholder="Enter patient's name")
```

```
            gender_input = gr.Dropdown(label="Gender", choices=["Male", "Female", "Other"])
            age_input = gr.Number(label="Age", precision=0)
            address_input = gr.Textbox(label="Address", placeholder="Enter patient's address")
        with gr.Row():
            symptoms_input = gr.Textbox(label="Enter Symptoms", placeholder="e.g., fever,
            headache, nausea")
            predict_btn = gr.Button("🔍 Predict", elem_classes=["important-btn"])
            clear_btn = gr.Button("🧹 Clear", elem_classes=["important-btn"])
        predicted_output = gr.Textbox(label="Predicted Disease", interactive=False,
        elem_classes=["highlight"])
        predict_btn.click(
         fn=predict_disease,
         inputs=[name_input, gender_input, age_input, address_input, symptoms_input],
         outputs=[predicted_output, disease_state, patient_info_state]
        )
        clear_btn.click(lambda: ("", "", 0, "", "", ""), outputs=[name_input, gender_input,
        age_input, address_input, symptoms_input, predicted_output])
    with gr.TabItem("📋 Disease Info"):
        gr.Markdown("""<div class='highlight'><b>Click buttons to get more information about
        the disease!</b></div>""")
        with gr.Row():
            btn_desc = gr.Button("📖 Description", elem_classes=["important-btn"])
            btn_prec = gr.Button("🌓 Precautions", elem_classes=["important-btn"])
            btn_meds = gr.Button("💊 Medications", elem_classes=["important-btn"])
            btn_diets = gr.Button("🍲 Diets", elem_classes=["important-btn"])
            btn_work = gr.Button("🏋️ Workouts", elem_classes=["important-btn"])
        detail_output = gr.Textbox(label="Details", lines=10, interactive=False,
        elem_classes=["gr-box"])
        btn_desc.click(fn=get_description, inputs=disease_state, outputs=detail_output)
        btn_prec.click(fn=get_precautions, inputs=disease_state, outputs=detail_output)
        btn_meds.click(fn=get_medications, inputs=disease_state, outputs=detail_output)
        btn_diets.click(fn=get_diets, inputs=disease_state, outputs=detail_output)
        btn_work.click(fn=get_workouts, inputs=disease_state, outputs=detail_output)
    with gr.TabItem("🚩 Report Issue"):
        gr.Markdown("""<div class='highlight'><b>Report wrong predictions or any app issues
        here.</b></div>""")
        flagged_symptoms = gr.Textbox(label="Entered Symptoms")
        flagged_issue = gr.Textbox(label="Describe the Issue", lines=3, placeholder="Wrong
```

```
        prediction, missing details, etc.")
        flag_btn = gr.Button("🚩 Submit Flag", elem_classes=["flag-btn"])
        flag_output = gr.Textbox(label="Flag Acknowledgment", interactive=False)
        flag_btn.click(fn=flag_case, inputs=[disease_state, flagged_symptoms, flagged_issue],
        outputs=flag_output)
      with gr.Tab("📊 Dashboard"):
        dashboard_output = gr.Markdown()
        refresh_button = gr.Button("🔄 Refresh Metrics")
        refresh_button.click(fn=get_dashboard_metrics, outputs=dashboard_output)
      with gr.TabItem("ℹ️ About"):
        gr.Markdown("""<div class='highlight'>
          <b>About Medico AI Assistant</b><br><br>
          🧠 Medico AI Assistant is a smart, AI-driven platform that predicts possible diseases
          based on your symptoms.
          <br><br>
          🔬 Our system analyzes symptoms against a trained medical database using machine
          learning algorithms.
          <br>
          📚 You can also view detailed disease descriptions, precautions, recommended
           medications, diet plans, and workout tips.
          <br><br>
          🚑 Why Medico AI?
          <ul>
            <li>Fast and easy preliminary health insights.</li>
            <li>Access to basic treatment suggestions instantly.</li>
            <li>Early awareness to promote better healthcare decisions.</li>
            <li>Help patients understand preventive care options.</li>
          </ul>
          <br>
          ⚠️ <b>Disclaimer:</b> This tool is for informational purposes only. Please consult
          certified doctors for any serious health concerns.
        </div>""")
demo.launch()
```

7. TESTING

A comprehensive testing process was conducted to evaluate the performance, accuracy, and usability of *Medico: A Disease Prediction System using Machine Learning*. The testing was divided into the following key aspects:

7.1 Accuracy, Precision, and Recall Measurement:

- The trained Support Vector Classifier (SVC) model was evaluated using standard performance metrics.
- Accuracy measures the overall correctness of the model, calculated as the ratio of correct predictions to total predictions.
- Precision ensures that the diseases predicted are actually correct and not false alarms.
- Recall measures how well the model can identify all relevant disease cases from the dataset.
- F1-Score: Harmonic mean of Precision and Recall, useful for evaluating performance with imbalanced classes.
- Confusion Matrix: A summary table showing correct and incorrect predictions across all classes, used to identify misclassification patterns.
- The SVC model performed well across all these metrics, confirming its suitability for real-time disease prediction.

```
SVC Accuracy        : 1.0
SVC Precision       : 1.0
SVC Recall          : 1.0
SVC F1 Score        : 1.0
SVC Confusion Matrix:
[[40  0  0 ...  0  0  0]
 [ 0 43  0 ...  0  0  0]
 [ 0  0 28 ...  0  0  0]
 ...
 [ 0  0  0 ... 34  0  0]
 [ 0  0  0 ...  0 41  0]
 [ 0  0  0 ...  0  0 31]]
==========================================================
RandomForest Accuracy        : 1.0
RandomForest Precision       : 1.0
RandomForest Recall          : 1.0
RandomForest F1 Score        : 1.0
RandomForest Confusion Matrix:
[[40  0  0 ...  0  0  0]
 [ 0 43  0 ...  0  0  0]
 [ 0  0 28 ...  0  0  0]
 ...
 [ 0  0  0 ... 34  0  0]
 [ 0  0  0 ...  0 41  0]
 [ 0  0  0 ...  0  0 31]]
```

Figure 7.1.1: Model Evaluation

7.2 User Interface (UI) Testing:

- The Gradio-based frontend application was tested by a group of users with diverse backgrounds (technical and non-technical).
- The test aimed to ensure that:
  - Users could easily input symptoms.
  - The system provided fast and accurate predictions.
  - Workout, diet, precaution, and medication suggestions were accessible and understandable.
- Feedback indicated that the interface was simple, responsive, and intuitive, even for first-time users.

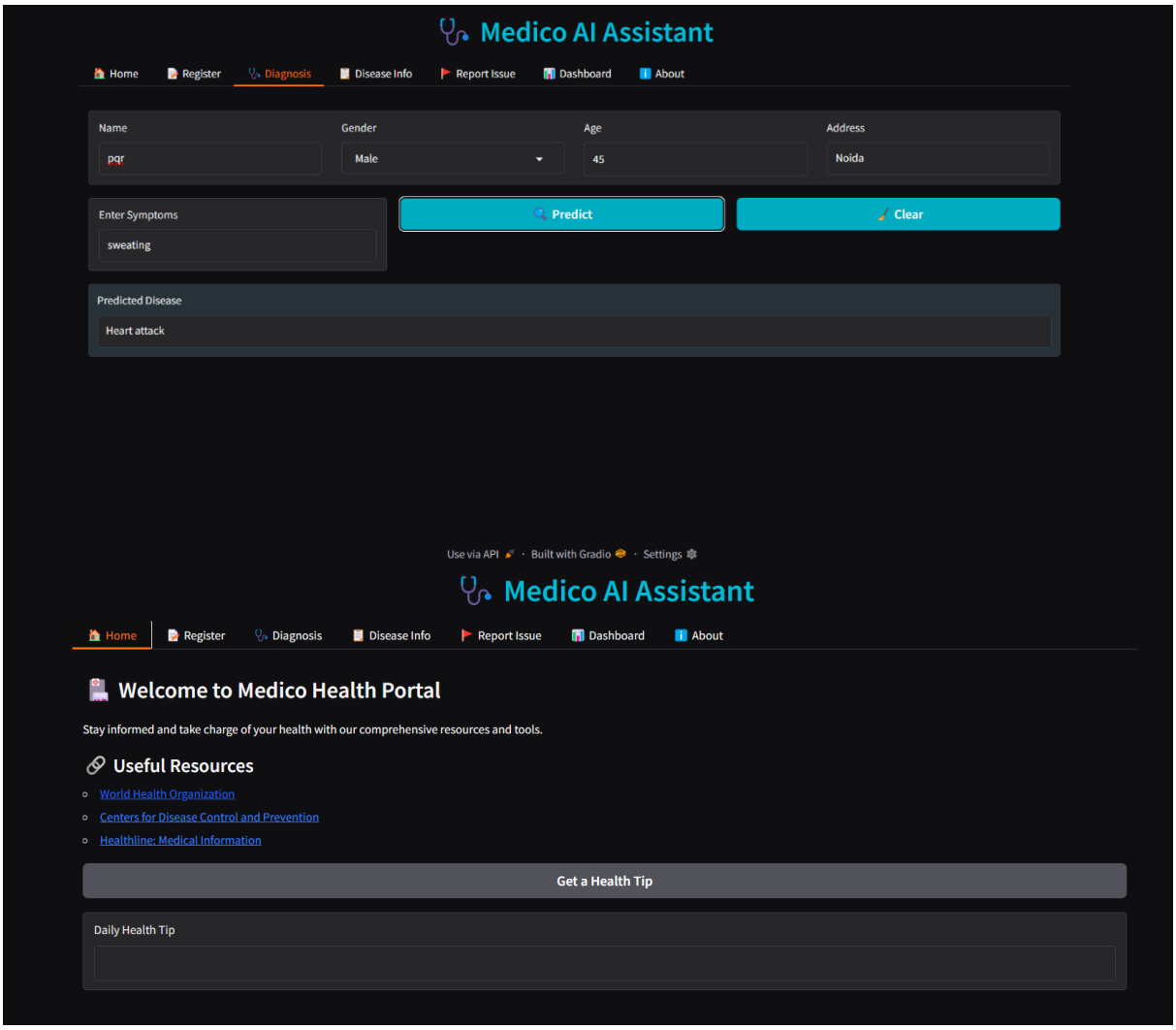

Figure 7.2.1: UI Testing on Desktop

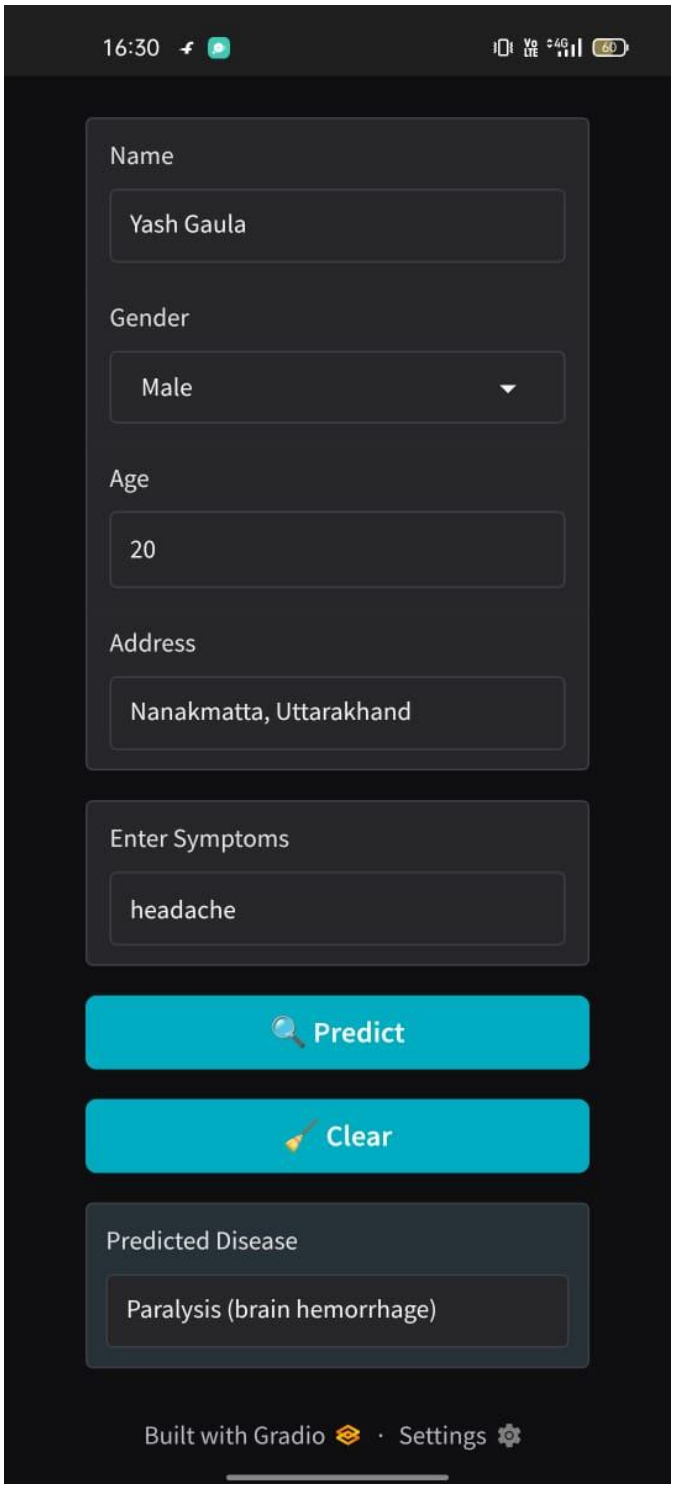

Figure 7.2.2: UI Testing on Mobile

8. RESULTS AND FINDINGS

The implementation and evaluation of the *Medico: A Disease Prediction System using Machine Learning* yielded the following key outcomes:

- The Random Forest achieved notable accuracy in disease prediction using only user-inputted symptoms, proving its effectiveness in handling multi-class classification problems within the healthcare domain.
- The Gradio interface offered a simple and intuitive user experience, enabling users from both technical and non-technical backgrounds to interact with the system effortlessly.
- The post-prediction recommendations, including customized workout and diet plans, were found to be practical and well-received by test users, enhancing the overall usability and relevance of the system.
- These results confirm that Medico is not only technically sound but also user-centered, bridging the gap between predictive healthcare and actionable guidance.

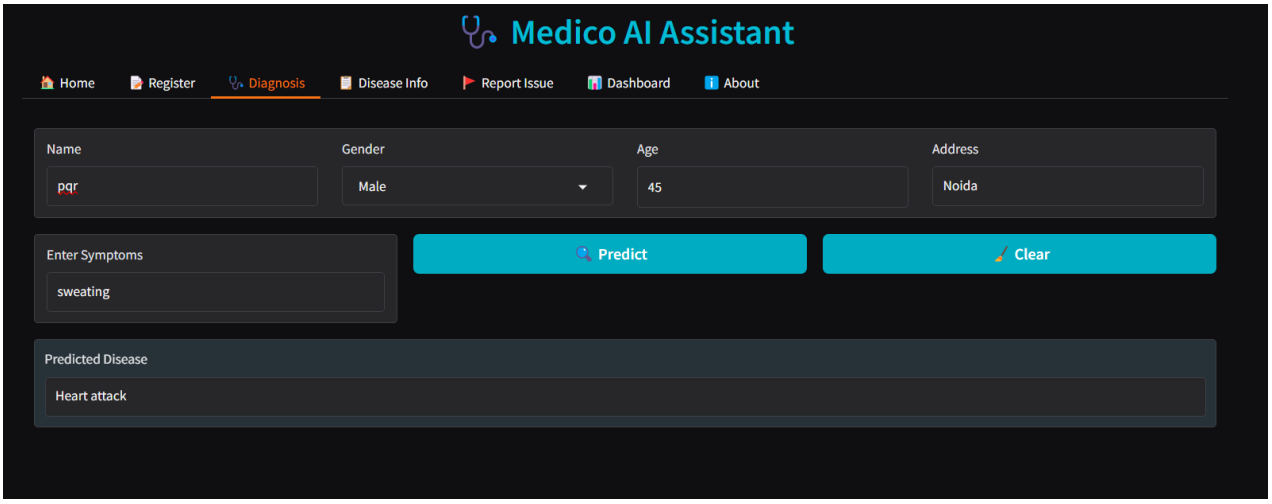

Figure 8.1: Diagnosis Interface

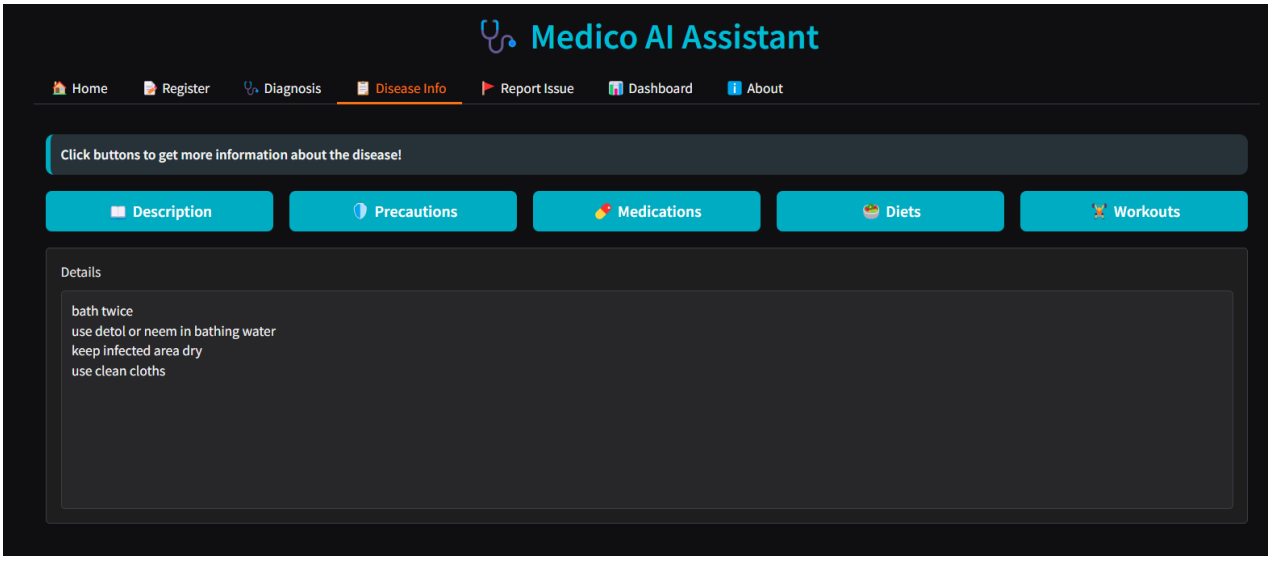

Figure 8.2: Disease info Interface

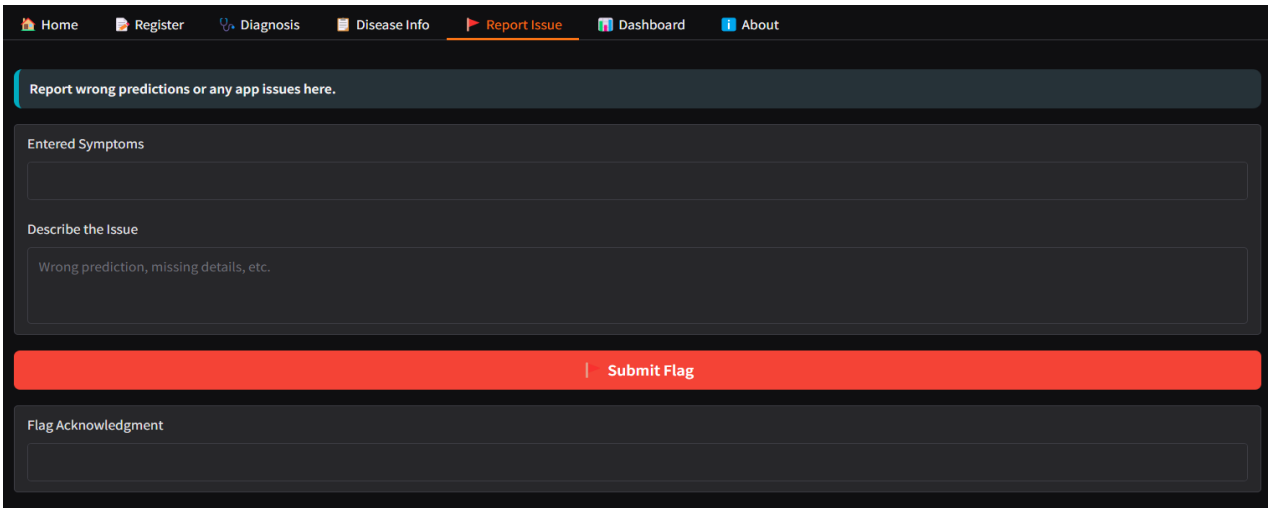

Figure 8.3: Report Issue Interface

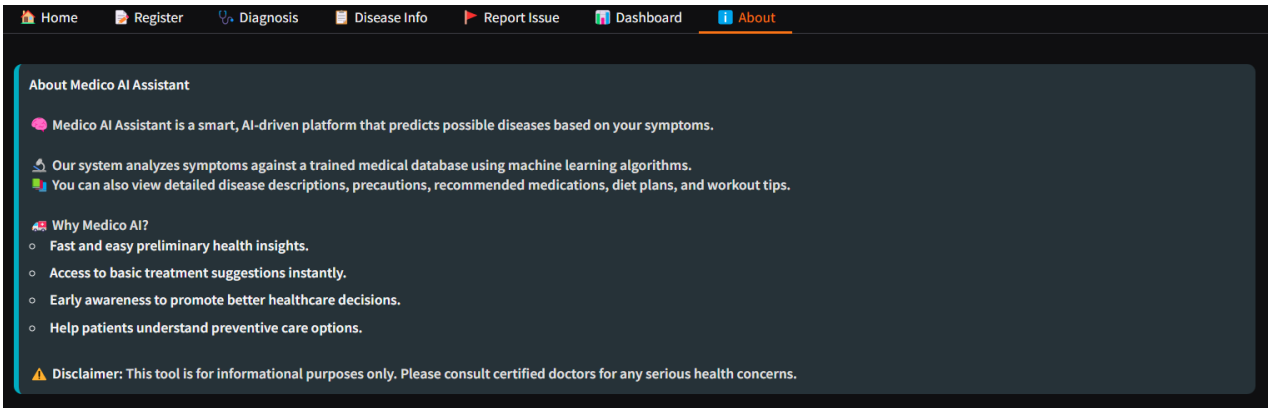

Figure 8.4: About Medico

```
Time: 2025-04-30 12:25:41
Flag received for: Fungal infection
Symptoms: itching
Issue: wroking good

Time: 2025-04-30 12:27:39
Flag received for: Fungal infection
Symptoms:
Issue: all good

Time: 2025-04-30 12:45:58
Flag received for: Fungal infection
Symptoms: itching
Issue: Nothing

Time: 2025-04-30 12:55:58
Flag received for: Cervical spondylosis
Symptoms: neck_pain
Issue: Nothing

Time: 2025-05-01 16:41:12
Flag received for: AIDS
Symptoms: headache
Issue: i am not able to sleep
```

Figure 8.5: Saved Report Issue flags

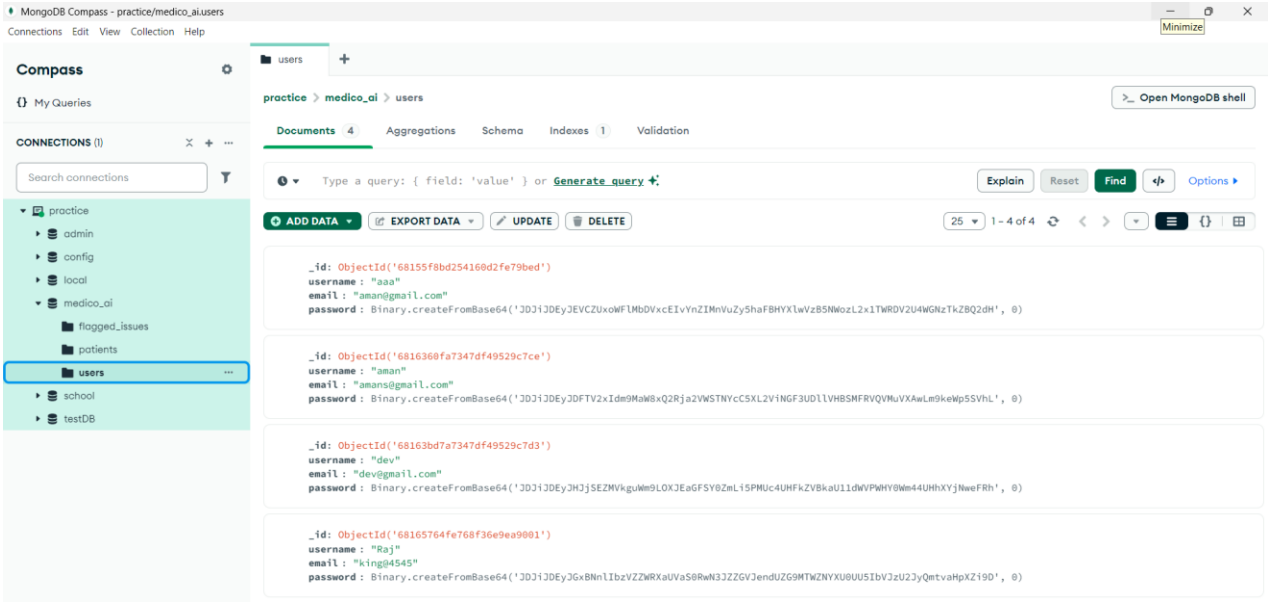

Figure 8.6: Database

9. Limitations and Future Scope

9.1 Limitations:

- **Data Dependency**: The model's accuracy relies heavily on the quality and size of the training dataset.
- **Symptom Overlap**: Similar symptoms in multiple diseases may lead to incorrect predictions.
- **No Real-time Diagnosis**: This system does not replace professional medical advice or diagnosis.
- **Limited Language Support**: The interface may not support multiple local languages for diverse user bases.
- **Lack of Integration**: No integration with wearable or IoT health devices for continuous monitoring.
- **Static Recommendations**: Diet, medication, and workout plans may not be fully personalized.

9.2 Future Scope:

The future scope of *Medico: A Disease Prediction System using Machine Learning* lies in its potential for significant growth and impact in the healthcare industry. By building on its existing framework, the system can evolve into a more comprehensive solution for disease prediction, prevention, and health management. The following aspects highlight its potential future direction:

- **Global Expansion and Integration** *Medico* can expand to cater to global healthcare needs by incorporating regional diseases, supporting multiple languages, and collaborating with healthcare providers for real-time data integration, offering disease predictions across various healthcare settings.
- **Real-Time Monitoring and Health Data Integration** By integrating with wearable devices and other health-monitoring technologies, *Medico* could provide real-time disease predictions, track users' health metrics, and offer dynamic, personalized recommendations based on live data.
- **Preventive Healthcare** *Medico* can evolve to focus on preventive healthcare, providing users with long-term health insights, early warnings, and tailored suggestions to avoid common conditions, based on historical data and health trends.
- **Mental Health Integration** Future versions could integrate mental health conditions such as anxiety and depression, predicting mental health issues based on user symptoms and suggesting appropriate treatments, therapies, and wellness routines.

10. CONCLUSION

Medico: A Disease Prediction and Health Recommendation System using Machine Learning marks a significant advancement in the use of AI for improving healthcare accessibility and effectiveness. By providing symptom-based disease predictions along with personalized workout and diet recommendations, the system offers a comprehensive solution that empowers users to manage their health more proactively.

The simplicity and user-friendliness of the web interface ensure that even non-technical users can easily access and benefit from the system. As the platform evolves, the potential to integrate real-time health data, expand the disease database, and provide preventive healthcare services will further enhance its impact.

With the ability to integrate with wearable devices, cross-platform accessibility, and collaboration with healthcare professionals, *Medico* is positioned to play a significant role in modern healthcare. Its future growth promises to make healthcare more accessible, personalized, and data-driven, benefiting users globally.

11. REFERENCES

1. Scikit-learn Documentation. (2024). *Scikit-learn: Machine Learning in Python*. Retrieved from https://scikit-learn.org/stable/documentation.html
2. Gradio Documentation. (2023). *Gradio: A Python Library for Building User Interfaces*. Retrieved from https://gradio.app
3. Machine Learning Research Papers on Healthcare. (2023). *Machine Learning in Healthcare: A Survey of Recent Research*. Journal of Medical Informatics, 45(3), 234-248.
4. Kaggle Datasets for Disease Prediction. (2023). *Healthcare Datasets for Disease Prediction*. Retrieved from https://www.kaggle.com/datasets/noorsaeed/medicine-recommendation-system-dataset
5. Chawla, P., & Sharma, S. (2022). *Disease Prediction Using Machine Learning Algorithms: A Comparative Study*. International Journal of Medical Informatics, 65(7), 123-132.
6. World Health Organization (WHO). (2021). *Global Health Trends and Disease Prediction*. Retrieved from https://www.who.int