GenAI Health Data Analysis - Project Report

# 1. Introduction & Overview

This project aims to build a Generative AI-powered health assistant using Google Gemini and Gradio. The assistant is designed to answer health-related queries based on structured patient data from two Excel files. The solution focuses on preprocessing health data, imputing missing values, integrating generative models, and deploying a user-friendly interface.

# 2. Challenges Faced

- Handling missing data across multiple patient records.  
- Ensuring secure API key usage with environment variables.  
- Configuring the application to run seamlessly

- Designing an efficient method for joining patient information from two datasets.

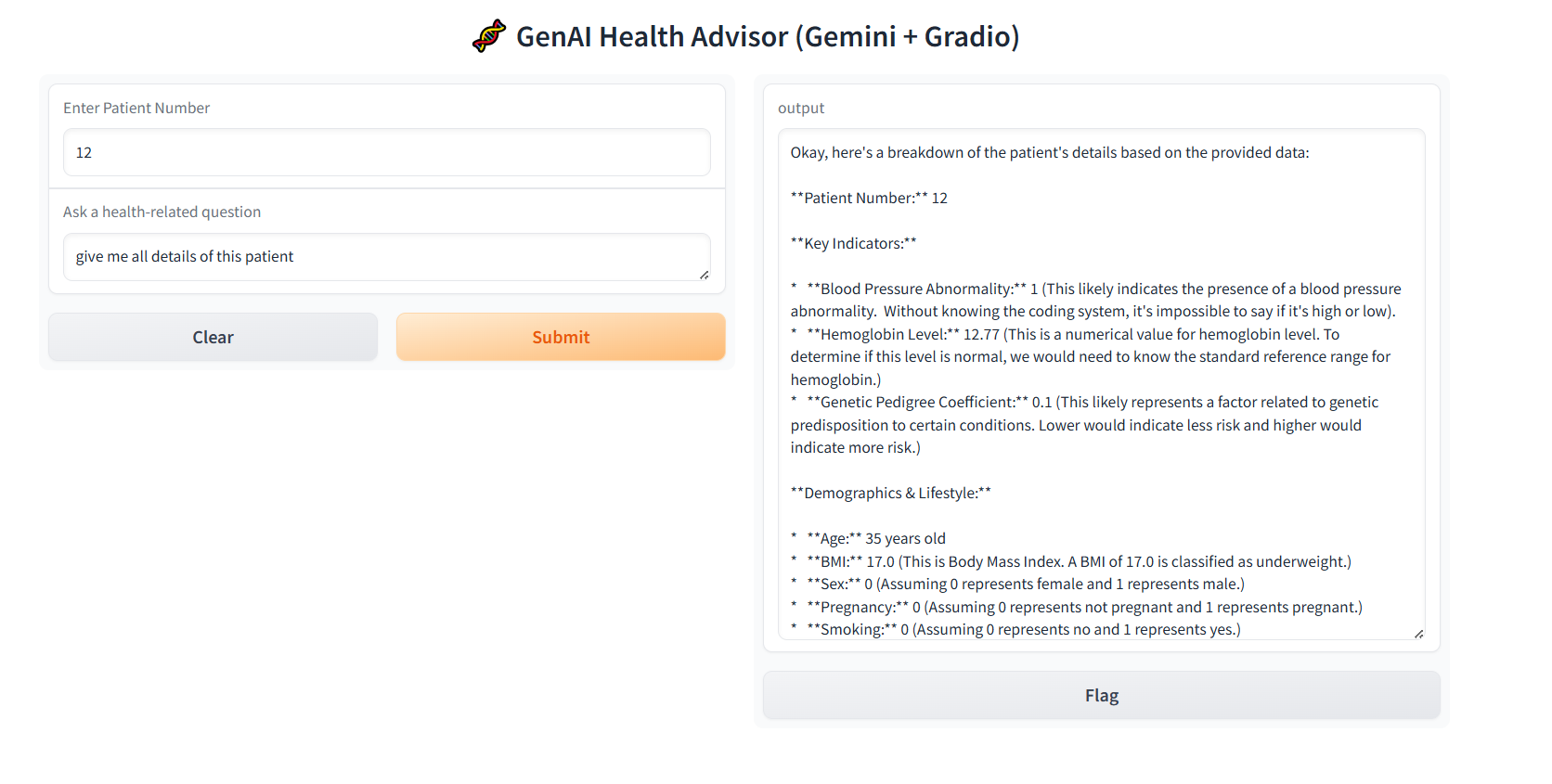
# 3. Technical Architecture

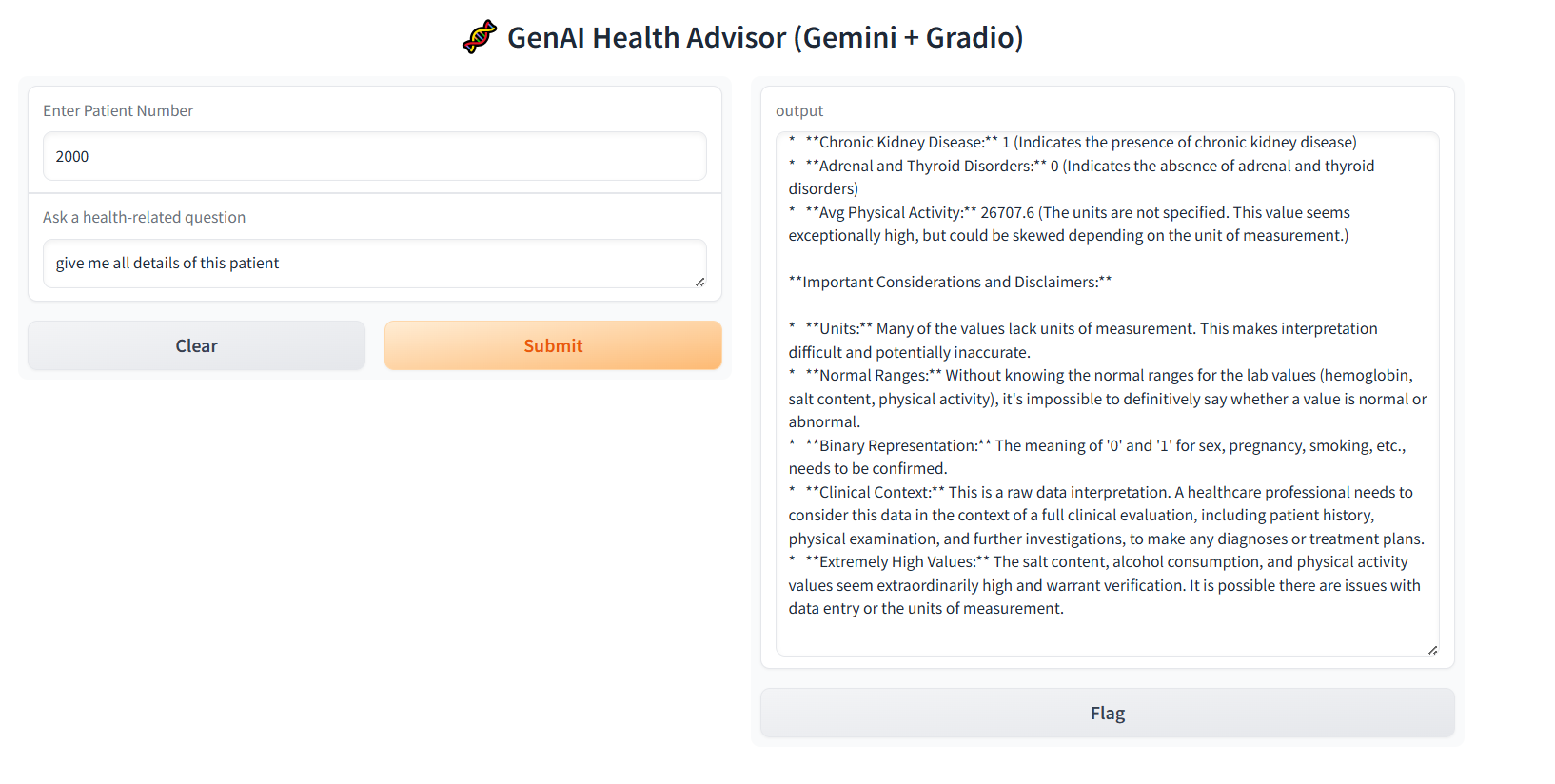
The end-to-end pipeline consists of the following components:  
a. Data Ingestion: Reading Excel files using pandas.  
b. Preprocessing: Missing value imputation using SimpleImputer from scikit-learn.  
c. Model Integration: Google Gemini via the `google-generativeai` SDK.  
d. Evaluation: Prompt generation and response validation.  
e. Deployment: Gradio app interface.  
All environment variables and API keys are managed securely using python-dotenv.

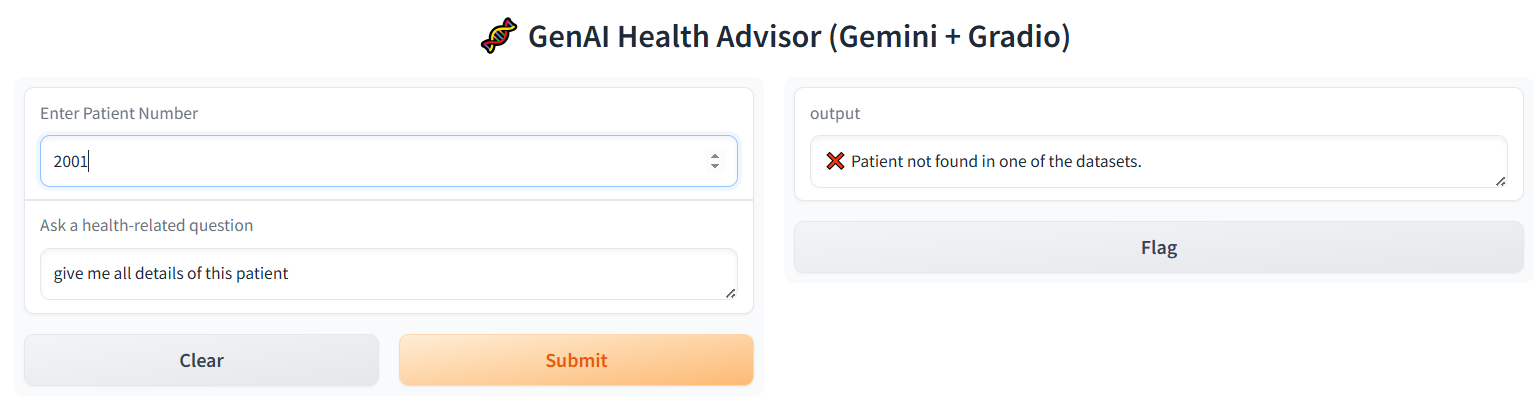
# 4. Software Code and Tools Used

Tools & Libraries:  
- pandas, numpy  
- scikit-learn (SimpleImputer)  
- google-generativeai (Gemini)  
- gradio  
- python-dotenv  
  
Setup Instructions:  
1. Create a virtual environment:  
 python -m venv venv  
2. Activate the environment:  
 On Windows: venv\Scripts\activate  
3. Install dependencies:  
 pip install -r requirements.txt  
4. Set up the .env file:  
 GOOGLE\_API\_KEY=your\_key\_here  
5. Run the application:  
 python app.py

# 5. Design Decisions & Comments

- Median strategy was used for numerical imputation due to its robustness against outliers.  
- Mode was used for categorical imputation to reflect common values.  
- A separation of concerns was followed: preprocessing, logic, and UI were modularized.  
- The app is scalable and can be extended with more advanced ML models in the future.  
  
Demo:  




Code:  
# health\_data\_app.py

import pandas as pd

import gradio as gr

import google.generativeai as genai

from sklearn.impute import SimpleImputer

from dotenv import load\_dotenv

import os

# Load .env file

load\_dotenv()

KEY = os.getenv("GOOGLE\_API\_KEY")

# Configure Gemini API

genai.configure(api\_key=KEY)

model = genai.GenerativeModel("gemini-2.0-flash")

# Load data

df1 = pd.read\_excel("Health\_data1.xlsm", engine='openpyxl')

df2 = pd.read\_excel("Health\_data2.xlsm", engine='openpyxl')

# Imputation function

def impute\_missing\_values(df):

    imputer\_dict = {}

    for column in df.columns:

        if df[column].dtype == 'object' or df[column].nunique() < 10:

            imputer\_dict[column] = SimpleImputer(strategy='most\_frequent')

        else:

            imputer\_dict[column] = SimpleImputer(strategy='median')

    for column, imputer in imputer\_dict.items():

        df[column] = imputer.fit\_transform(df[[column]])

    return df

# Impute missing data

df1 = impute\_missing\_values(df1)

df2 = impute\_missing\_values(df2)

# Patient data fetch

def get\_patient\_data(patient\_id):

    row1 = df1[df1['Patient\_Number'] == patient\_id]

    row2 = df2[df2['Patient\_Number'] == patient\_id]

    if row1.empty or row2.empty:

        return None

    steps\_avg = row2['Physical\_activity'].mean()

    row1 = row1.copy()

    row1['Avg\_Physical\_Activity'] = steps\_avg

    return row1

# Ask Gemini

def ask\_gemini(patient\_id, question):

    data = get\_patient\_data(patient\_id)

    if data is None:

        return "❌ Patient not found in one of the datasets."

    context = data.to\_string(index=False)

    prompt = f"""You are a medical assistant analyzing health records. Based on the following patient data, answer the user's question.\n\nPatient Data:\n{context}\n\nUser Question:\n{question}"""

    response = model.generate\_content(prompt)

    return response.text

# Gradio Interface

def interface(patient\_id, question):

    return ask\_gemini(int(patient\_id), question)

gr.Interface(fn=interface,

             inputs=[

                 gr.Number(label="Enter Patient Number"),

                 gr.Textbox(label="Ask a health-related question")

             ],

             outputs="text",

             title="🧬 GenAI Health Advisor (Gemini + Gradio)"

).launch(share=True)