

Data Audit Report

Patient data is given in 2 datasets

1. Health data – Contains demographic details like age, sex; habits like smoking, alcohol consumption etc
2. Activity data – Step count for each day

There is missing data and some times it may require special attention. For example, if gender is male then pregnancy column is always not applicable but for females missing value could mean not pregnant currently or status unknown. Similarly, pregnancy is more likely only in a certain age bracket.

We may have to build data quality checks before feeding the raw data into database.

There is also metadata given about tables which is helpful. It gives a more detailed description of column names plus it has encoding key given for fields.

How to handle the data-

For RAG applications, text data is embedded and stored in a vector DB. The given data is tabular in format with no significant long text in it. We can create a Data layer and abstract the underlying data stored in xlsx, csv, sqlite db etc.

Also, the given data has encodings (e.g. in the sex column, 0 means male & 1 means female). We can store the data after converting encodings back to their labels. Or we can store data as given and specify data transformations that transform this data after retrieval and before feeding to LLM.

Project Development

IDE – VS code due to availability of extensions such as connecting to github, being able to work with WSL

OS – Ubuntu running within WSL and terminal connected through VS code which allows easy access to linux commands and less surprises when moving from development to server deployment

Repo – Code hosted on Github for online backup and version control -> https://github.com/YashKushwaha/chat_ui

LLM Model – Currently using Phi-4 17B params 4 bit mixed quantization model being served through ollama.

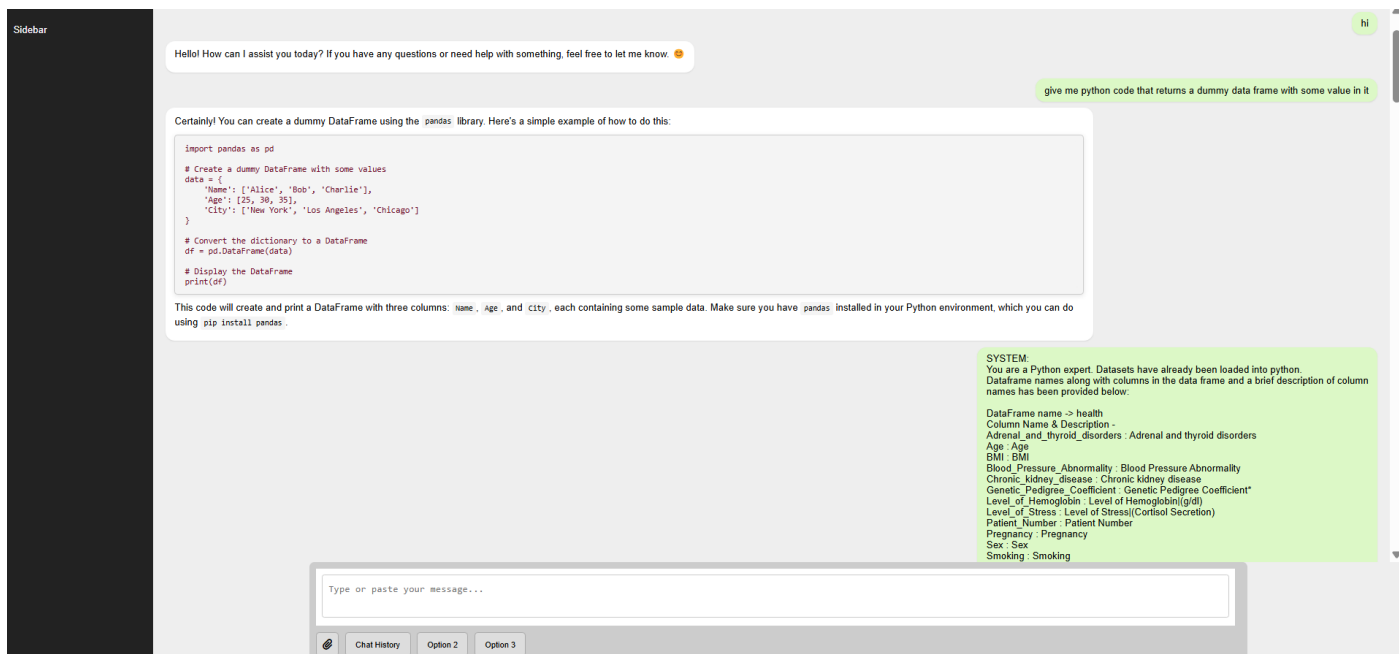
Advantage –

- Data remains offline and secure
- Also gives a sense of reasoning capabilities required for the tasks. If current model not capable then we can go for Phi-4 model with more parameters or lesser quantization (e.g 8 bit or 16 bit). We can try other open source models (llmama, mistral etc) or go for commercial LLMs (e.g. ChatGPT)
- Backend – FastAPI used to develop routes. We can have endpoints for simple LLM calls, RAG, Agents and test routes as well
- Front end – Designed using HTML, CSS & simple JS. Web frameworks like Node.js not used to keep the interface simple and light weight. Key features -
 - Messaging app like design to distinguish user input and bot/LLM response
 - UI can render markdown as well leading to better formatting plus separate box for codes
 - Sidebar for keeping track of conversation (to be implemented in future)

Architecture Used

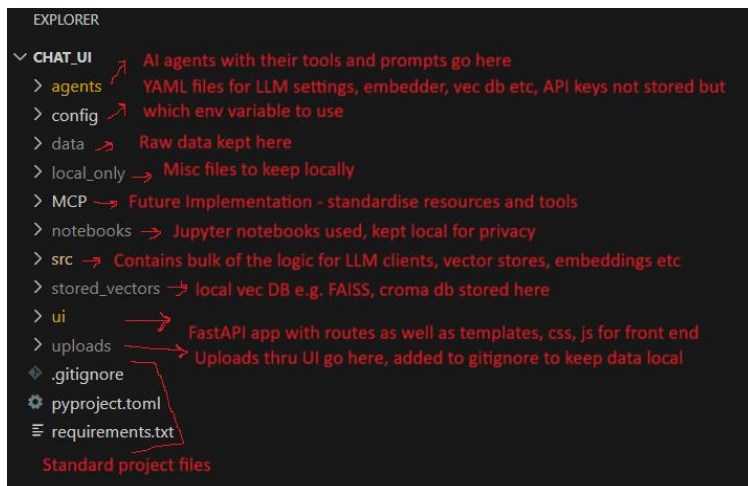
I have used router agent architecture where a master agent receives the user query & then interprets it and classifies the query. The query is then delegated to specialized sub agent best suited for the query. Currently 2 sub agents are implemented –

- 1) Agent 1 – Takes as input a patient number and then retrieves patient data from the database and analyses the data and creates a report highlighting the finding
- 2) Agent 2 – Converts user queries into python code which allows for joining multiple tables. This python code is then executed and then analysed by the LLM.



Project Structure

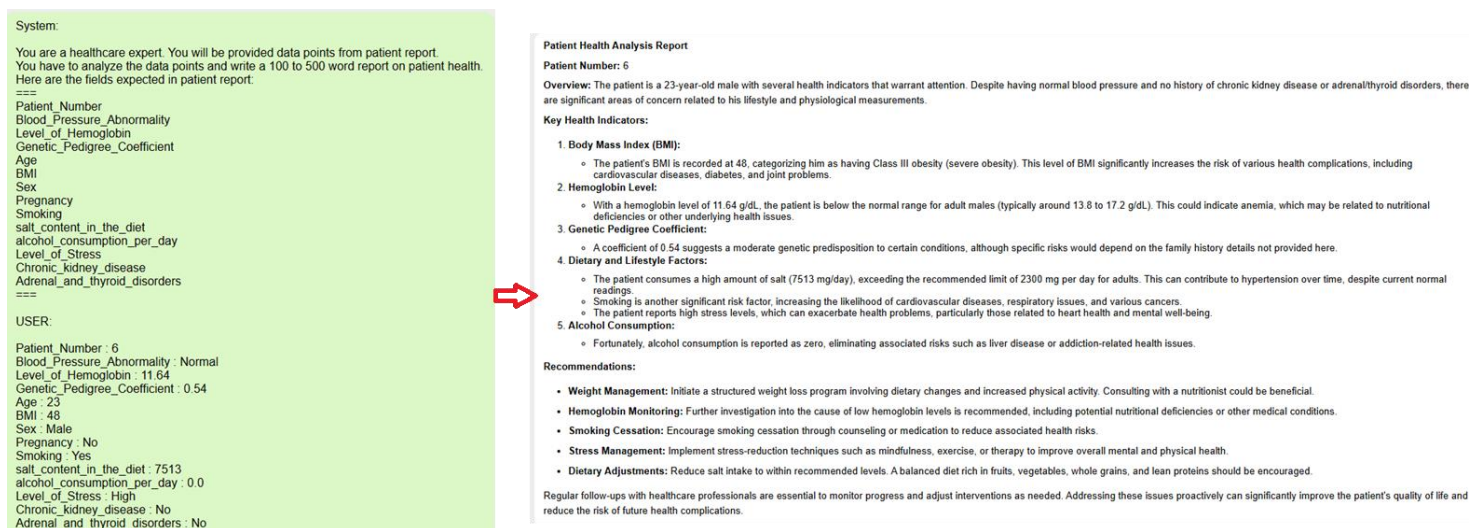
Designed to keep code modular and easy to extend



Evaluation

Manual evaluation to check quality of responses

Check 1 – Can the model understand the data provided? LLM response is good



Check 2 – Can the LLM generate python code to join tables and perform data analysis based on user query? Yes

SYSTEM:
You are a Python expert. You have been given access to following tables:

DataFrame -> health
Adrenal_and_thyroid_disorders : Adrenal and thyroid disorders
Age : Age
BMI : BMI
Blood_Pressure_Abnormality : Blood Pressure Abnormality
Chronic_kidney_disease : Chronic kidney disease
Genetic_Pedigree_Coefficient : Genetic Pedigree Coefficient
Level_of_Hemoglobin : Level of Hemoglobin(g/dl)
Level_of_Stress : Level of Stress(Cortisol Secretion)
Patient_Number : Patient Number
Pregnancy : Pregnancy
Sex : Sex
Smoking : Smoking
alcohol_consumption_per_day : Alcohol consumption per day (ml/day)
salt_content_in_the_diet : Salt content in the diet(mg/per day)

DataFrame -> activity
Day_Number : Day Number
Patient_Number : Patient Number
Physical_activity : Physical activity (no. of steps/day) in the last 10 days

Your job is to convert user query into python statements that can be executed.

USER:
How do I get the list of male patients who smoke but also walk a lot every day ?

To retrieve a list of male patients who smoke and take a lot of steps each day, we need to perform the following operations:

1. Filter the `health` DataFrame for male patients who smoke.
2. Determine what constitutes "a lot of steps" by analyzing the `activity` DataFrame (e.g., you might choose an average or specific threshold).
3. Merge the two filtered DataFrames on `Patient_Number` to find common entries that satisfy both conditions.

Below is a Python code snippet using pandas to achieve this:

```
import pandas as pd

# Assuming 'health' and 'activity' are pre-loaded pandas DataFrame objects

# Step 1: Filter male patients who smoke from the health DataFrame
male_smokers = health[(health['Sex'] == 'Male') & (health['Smoking'] == True)]

# If smoking is stored in a different format, you might need to adjust this condition.

# Step 2: Determine what constitutes "a lot of steps"
# This example assumes that "a lot" means more than the 75th percentile of daily steps
threshold_steps = activity['Physical_activity'].quantile(0.75)

# Filter patients in the activity DataFrame who walk a lot every day
active_patients = activity[activity['Physical_activity'] > threshold_steps]

# Step 3: Merge filtered DataFrames on 'Patient_Number' to get male smokers who are also active
result = pd.merge(male_smokers, active_patients, on='Patient_Number')

# Optionally, drop duplicates if a patient appears more than once in the activity data
result = result.drop_duplicates(subset=['Patient_Number'])

# Get the list of unique Patient Numbers or full rows depending on your need
male_smoking_active_patients_list = result['Patient_Number'].unique()

print(male_smoking_active_patients_list)
```

Note:

- Adjust the threshold for "a lot" based on your specific criteria.
- Ensure that `health` and `activity` DataFrames are properly loaded before executing this code. If necessary, you can load them using pandas read functions like `pd.read_csv()`.
- The condition `(health['Smoking'] == True)` assumes 'Smoking' is a boolean column, adjust according to the actual data format (e.g., if it's categorical or coded differently).