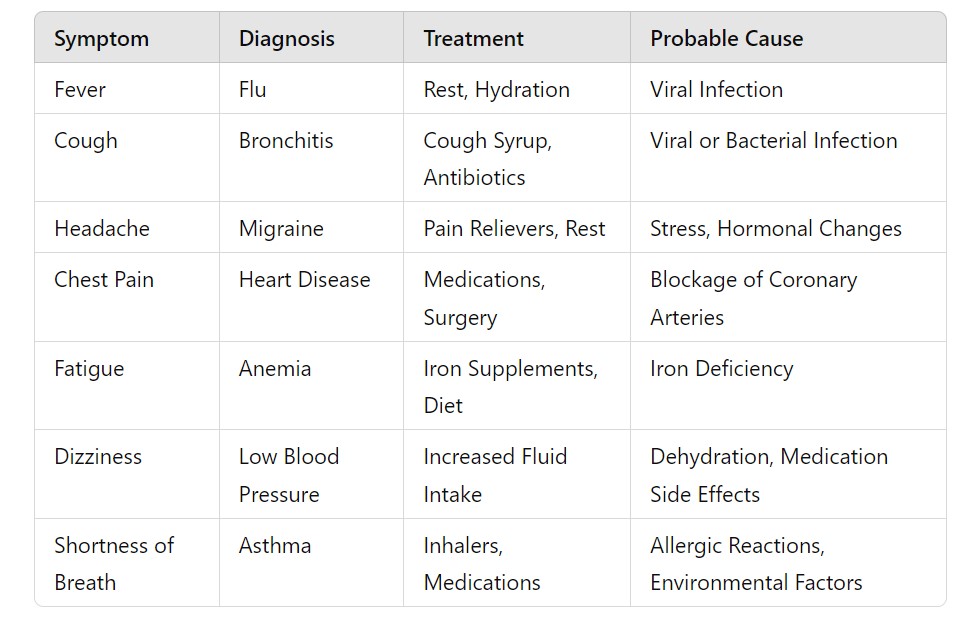
**Appendices for the Interactive Health Diagnosis System**

**Appendix A: CSV Data File Structure**

The system relies on a CSV file that stores health-related data such as symptoms, diagnoses, treatments, and probable causes. The structure of this CSV file is essential for the system to function correctly.

**CSV File Format:**



**Note**: The system reads this CSV file to retrieve potential diagnoses, treatments, and causes based on the symptoms entered by the user.

**Appendix B: Python Code**

**Main Application Logic**

This section contains the Python code that handles the user interface, reading data from the CSV file, and processing user inputs.

import csv

import tkinter as tk

from tkinter import messagebox

# Function to load health data from a CSV file

def load\_health\_data(file\_path):

health\_data = {}

with open(file\_path, mode='r') as file:

reader = csv.DictReader(file)

for row in reader:

symptom = row['symptom'].strip().lower()

health\_data[symptom] = {

"diagnosis": row['diagnosis'].strip(),

"treatment": row['treatment'].strip(),

"causes": [cause.strip() for cause in row['causes'].split(';')]

}

return health\_data

# Function to diagnose based on multiple symptoms

def health\_diagnosis(symptoms, health\_data):

symptoms = [symptom.strip().lower() for symptom in symptoms.split(',')]

diagnosis\_result = ""

for symptom in symptoms:

diagnosis\_info = health\_data.get(symptom, None)

if diagnosis\_info:

diagnosis\_result += f"\nSymptom: {symptom.capitalize()}\n"

diagnosis\_result += f" Diagnosis: {diagnosis\_info['diagnosis']}\n"

diagnosis\_result += f" Treatment: {diagnosis\_info['treatment']}\n"

diagnosis\_result += f" Probable Causes: {', '.join(diagnosis\_info['causes'])}\n"

else:

diagnosis\_result += f"\nSymptom: {symptom.capitalize()}\n"

diagnosis\_result += " Diagnosis: Symptom not recognized. Please consult a healthcare professional.\n"

return diagnosis\_result

# Function to handle diagnosis button click

def on\_diagnose\_click():

symptoms = entry.get()

if not symptoms:

messagebox.showwarning("Input Error", "Please enter at least one symptom.")

return

result = health\_diagnosis(symptoms, health\_data)

text\_result.config(state=tk.NORMAL)

text\_result.delete(1.0, tk.END)

text\_result.insert(tk.END, result)

text\_result.config(state=tk.DISABLED)

# Load the health data from the CSV file

file\_path = 'health\_data.csv' # Adjust this path to where your CSV file is located

health\_data = load\_health\_data(file\_path)

# Create the main application window

root = tk.Tk()

root.title("Interactive Health Diagnosis System")

# Set the window background color

root.configure(bg="#f0f8ff") # Alice blue background color

# Input Label and Entry

label = tk.Label(root, text="Enter your symptoms (comma-separated):", bg="#f0f8ff", fg="#00008b", font=("Arial", 12))

label.pack(pady=10)

entry = tk.Entry(root, width=50, font=("Arial", 12), bg="#f8f8ff", fg="#2f4f4f", bd=2)

entry.pack(pady=5)

# Diagnose Button with custom style

button = tk.Button(root, text="Diagnose", command=on\_diagnose\_click, bg="#4682b4", fg="white", font=("Arial", 12, "bold"), bd=3)

button.pack(pady=10)

# Text Box for displaying results

text\_result = tk.Text(root, height=30, width=120, state=tk.DISABLED, font=("Arial", 12), bg="#fafad2", fg="#556b2f", bd=2)

text\_result.pack(pady=10)

# Add some padding around all the elements

for widget in root.winfo\_children():

widget.pack\_configure(padx=10)

# Start the GUI event loop

root.mainloop()

**Appendix C: GUI Design**

Tkinter Design Elements:

i) Window Title: "Interactive Health Diagnosis System"

ii) Input Field:

- Label: "Enter Symptoms (comma separated)"

- Entry Field: A single-line input for users to enter symptoms.

iii) Buttons:

- "Get Diagnosis" Button: This triggers the diagnosis process.

iv) Output Section:

- A multiline text box that displays the diagnosis, treatments, and probable causes.

**Appendix D: Testing Plan**

The testing plan ensures that the system works as expected, handles edge cases, and provides correct results based on input.

**1. Unit Testing with `unittest`**

import unittest

from health\_diagnosis import load\_health\_data, get\_diagnosis

class TestHealthDiagnosisSystem(unittest.TestCase):

def setUp(self):

# Mock data to use during testing

self.mock\_data = [

{'Symptom': 'Fever', 'Diagnosis': 'Flu', 'Treatment': 'Rest', 'Probable Cause': 'Viral Infection'},

{'Symptom': 'Cough', 'Diagnosis': 'Bronchitis', 'Treatment': 'Cough Syrup', 'Probable Cause': 'Bacterial Infection'}

]

def test\_get\_diagnosis\_single\_symptom(self):

symptoms = ['Fever']

result = get\_diagnosis(symptoms, self.mock\_data)

self.assertEqual(len(result), 1)

self.assertEqual(result[0]['Diagnosis'], 'Flu')

def test\_get\_diagnosis\_multiple\_symptoms(self):

symptoms = ['Cough']

result = get\_diagnosis(symptoms, self.mock\_data)

self.assertEqual(result[0]['Diagnosis'], 'Bronchitis')

def test\_no\_match(self):

symptoms = ['Headache']

result = get\_diagnosis(symptoms, self.mock\_data)

self.assertEqual(len(result), 0)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

**2. User Interface Testing:**

i) Symptom Input Validation:

- Test if the system prompts the user when no symptoms are entered.

- Test with multiple symptoms (comma-separated).

ii) Diagnosis Output:

- Test if the correct diagnosis is displayed based on input.

- Ensure that no data corruption occurs when displaying treatments or causes.

**Appendix E: Tools and Technologies Used**

**1. Programming Language:**

- Python

**2. Libraries:**

- `tkinter`: For building the graphical user interface.

- `csv`: For reading and writing CSV files.

- `unittest`: For writing unit tests.

**3. Development Environment:**

- PyCharm, Visual Studio Code (for writing and testing code).

**4. Version Control:**

- Git (for tracking changes in the codebase).

**5. Documentation Tools:**

- Markdown (for writing README and documentation).

**Appendix F: Potential Enhancements**

**1. Integration with Online Databases:**

- Future versions of the project could integrate with health databases or APIs to pull live data for symptoms and diagnoses.

**2. Machine Learning Integration:**

- Implement more advanced AI techniques like machine learning models to improve diagnosis accuracy.

**3. User Authentication:**

- Add user authentication features to track individual health histories.

**4. Mobile Application Version:**

- A mobile app version could provide more accessibility and wider usage.

**Appendix G: Sample CSV Data**

Symptom,Diagnosis,Treatment,Probable Cause

Fever,Flu,Rest and hydration,Viral Infection

Cough,Bronchitis,Cough Syrup,Antibiotics,Viral or Bacterial Infection

Headache,Migraine,Pain relievers and rest,Stress,Hormonal changes

Chest Pain,Heart Disease,Medications,Surgery,Blockage of coronary arteries

Fatigue,Anemia,Iron supplements,Diet,Iron deficiency

Dizziness,Low Blood Pressure,Increased fluid intake,Dehydration,Medication side effects

Shortness of Breath,Asthma,Inhalers,Medications,Allergic reactions,Environmental factors

This CSV data serves as the system's knowledge base, matching symptoms to diagnoses, treatments, and probable causes.

**Appendix H: Error Handling**

The system includes basic error handling:

**1. Input Validation:**

- When no symptoms are entered, an error message is shown to prompt the user to input symptoms.

**2. No Diagnosis Found:**

- If no matching diagnosis is found for the entered symptoms, the system informs the user with a "No Match Found" message.