ML- Mini Project

DISEASE PREDICTION AND REMEDY SUGGESTION

BY:
Swastishri Desai (UCE2022448)
Vedika Desai (UCE2022449)
Batch: A3

Problem Statement:

The project aims to address the critical need for accurate disease prediction and effective remedy suggestions using machine learning techniques, particularly in healthcare settings. Early disease detection and personalized remedy recommendations are crucial for improving patient outcomes and reducing healthcare costs.

Introduction:

The project focuses on leveraging machine learning algorithms to predict diseases based on symptoms and provide tailored remedy suggestions. The significance lies in enhancing healthcare decision-making and ensuring timely interventions for patients.

Data Set Information:

- *The project utilizes a comprehensive dataset containing symptom profiles, disease labels, and corresponding remedy suggestions. The dataset is sourced from reputable healthcare databases or repositories, ensuring data quality and reliability.*

- *Link to dataset: https://www.kaggle.com/datasets/kaushil268/disease-prediction-using-machine-learning*
- *As this dataset was very large and not feasible for our GUI based model, we had to reduce this data using data preprocessing methos, like data reduction, feature subset selection, etc. this preprocessed dataset also contains rememdy for each disease, helping with remedy suggestion.*
- *Few values from preprocessed data:*

Runny nos	cough	sore throa	fever	bodyache	Fatigue	abdomina	diarrhea	nausea/vc	head pain	light sens	itch	skin rash	blood pre	difficulty i	difficulty	Disease	REMEDY
2	2	2	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	Common C	Rest, drink fluids, and take over-the-counter cold medications.
2	2	2	3	2	1	-1	-1	-1	2	-1	-1	-1	-1	-1	-1	Influenza	Rest. drink fluids. and take antiviral medications if prescribed by a doctor.

Methodology:

- *First part of the project focuses on comparing the models that can be used to predict disease, namely random forest, decision tree and naïve bayes algorithm.*

- *From the first part, we conclude that, although the best model for any data depends on dataset size, computational resources, training time, etc., the best model for this project would be random forest.*
- *Not only random forest is versatile and robust, it also provide better accuracy and reduce overfitting.*
- *Thus in the 2nd part, we implement disease prediction using random forest along with GUI to make it more interactive. this project gives user an intuitive interface to input their symptoms and receive predictions about potential diseases using the Random Forest algorithm, decision tree and naïve bayes. Although it predicts using all 3 algorithms, it would always give priority to random forest algorithm.*

Code and Output:

- ***Part 1 code:***

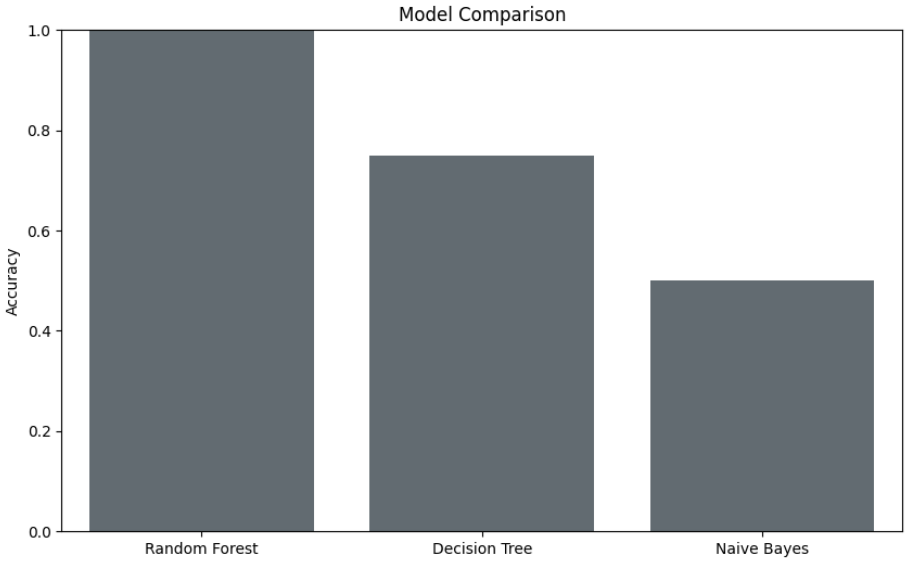
```
#import necessary libraries
import pandas as pd
```

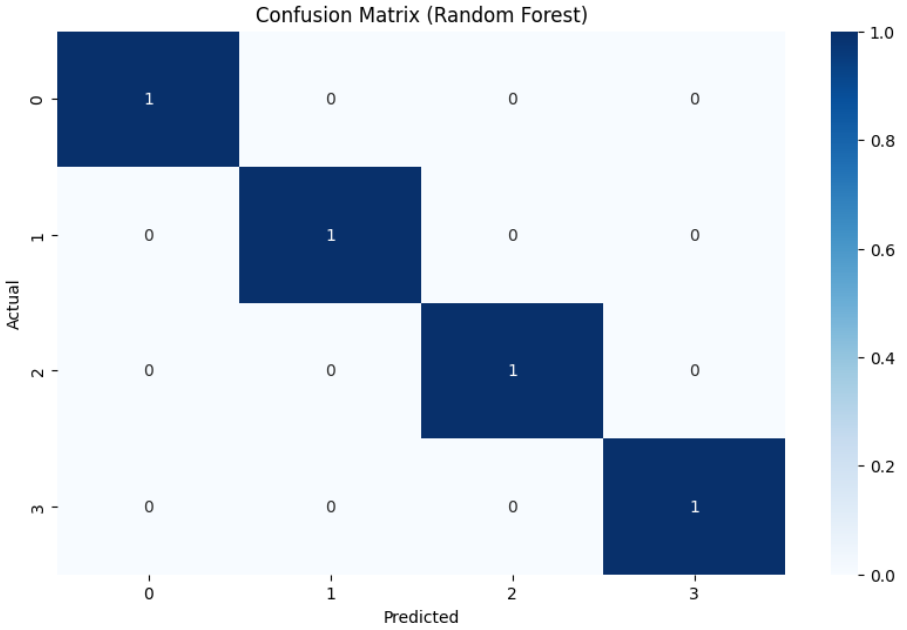
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix

#load dataset
data = pd.read_csv('/content/ml_final.csv', encoding='latin1')

# Split data into symptoms (X) and target disease (y)
X = data.drop(['Disease', 'REMEDY'], axis=1)
y = data['Disease']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.0606)

# Initialize classifiers
random_forest = RandomForestClassifier()
decision_tree = DecisionTreeClassifier()
naive_bayes = GaussianNB()

# Train the classifiers
random_forest.fit(X_train, y_train)
decision_tree.fit(X_train, y_train)
naive_bayes.fit(X_train, y_train)

# Make predictions using each classifier on the testing set
predicted_disease_rf = random_forest.predict(X_test)
predicted_disease_dt = decision_tree.predict(X_test)
predicted_disease_nb = naive_bayes.predict(X_test)

print("Actual output:", y_test)
print("Predicted Disease (Random Forest):", predicted_disease_rf)
print("Predicted Disease (Decision Tree):", predicted_disease_dt)
print("Predicted Disease (Naive Bayes):", predicted_disease_nb)

# Calculate accuracy for each model
accuracy_rf = accuracy_score(y_test, predicted_disease_rf)
accuracy_dt = accuracy_score(y_test, predicted_disease_dt)
accuracy_nb = accuracy_score(y_test, predicted_disease_nb)

# Display the accuracy of each model
```

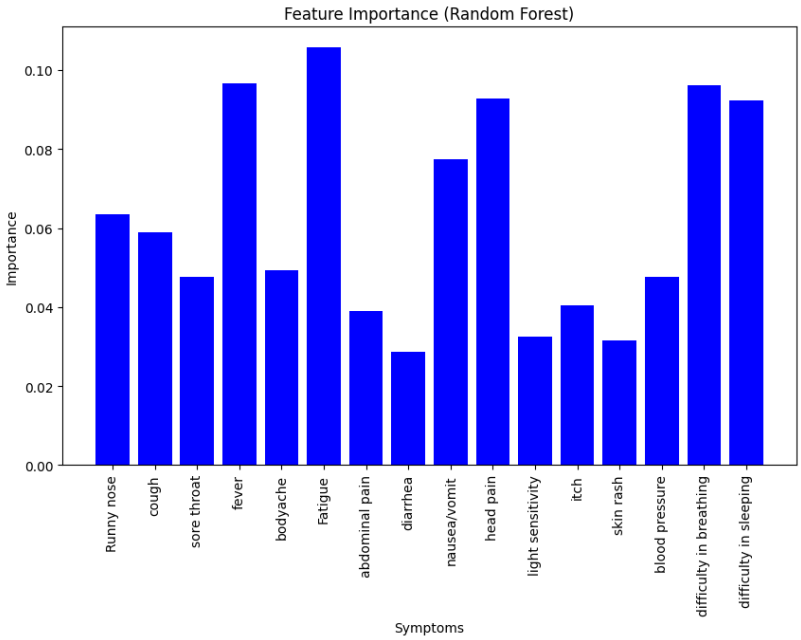
```
print("Accuracy (Random Forest):", accuracy_rf)
print("Accuracy (Decision Tree):", accuracy_dt)
print("Accuracy (Naive Bayes):", accuracy_nb)

# Model Comparison Plot
plt.figure(figsize=(10, 6))
sns.barplot(x=['Random Forest', 'Decision Tree', 'Naive Bayes'],
y=[accuracy_rf, accuracy_dt, accuracy_nb], saturation=0.1)
plt.title('Model Comparison')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
plt.show()

# Confusion Matrix
plt.figure(figsize=(10, 6))
sns.heatmap(confusion_matrix(y_test, predicted_disease_rf),
annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix (Random Forest)')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Feature Importance Plot for Random Forest
plt.figure(figsize=(10, 6))
importances = random_forest.feature_importances_
indices = range(len(X.columns))
plt.bar(indices, importances, color='b', align='center')
plt.xlabel('Symptoms')
plt.ylabel('Importance')
plt.title('Feature Importance (Random Forest)')
plt.xticks(indices, X.columns, rotation='vertical')
plt.show()
```

- ***Part 1 output:***

```
Accuracy (Random Forest): 1.0
Accuracy (Decision Tree): 0.75
Accuracy (Naive Bayes): 0.5
```







- ***Part 2 code snippets:***

```
import tkinter as tk
from tkinter import messagebox, Scale
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from PIL import Image, ImageTk
from sklearn.preprocessing import LabelEncoder
from collections import Counter

# Loading dataset
data = pd.read_csv("E:/VEDIKA DESAI/OneDrive/Desktop/ml.csv",
encoding='latin1')

# Perform label encoding on the 'REMEDY' column
label_encoder = LabelEncoder()
data['REMEDY'] = label_encoder.fit_transform(data['REMEDY'])

# Initialize classifiers
naive_bayes = GaussianNB()
decision_tree = DecisionTreeClassifier()
random_forest = RandomForestClassifier()

# Fit the classifiers
```

```
naive_bayes.fit(data.drop(['Disease', 'REMEDY'], axis=1), data['Disease'])
decision_tree.fit(data.drop(['Disease', 'REMEDY'], axis=1),
data['Disease'])
random_forest.fit(data.drop(['Disease', 'REMEDY'], axis=1),
data['Disease'])

# Create GUI window
window = tk.Tk()
window.title("Disease Prediction and Remedy Suggestion")
window.configure(bg='lightblue')  # Set background color

# Set the window size to fit the entire screen and adjust position
window.geometry("{0}x{1}+0+0".format(window.winfo_screenwidth(),
window.winfo_screenheight() - 100))

# Function to open main page
def open_main_page():
    initial_page_frame.pack_forget()  #To Hide initial page
    main_page_frame.pack()  #To Show main page

def predict_disease_and_suggest_remedy():
    try:
        symptoms = [int(slider_values[label].get()) for label in labels]

        # Ensure that the feature names match the ones used during
training
        user_input = pd.DataFrame([symptoms],
columns=data.drop(['Disease', 'REMEDY'], axis=1).columns)

        # Predictions using RandomForestClassifier, Naive Bayes, and
Decision Tree
        predicted_disease_rf = random_forest.predict(user_input)[0]
        predicted_disease_nb = naive_bayes.predict(user_input)[0]
        predicted_disease_dt = decision_tree.predict(user_input)[0]

        predicted_label.config(text=f"Predicted Disease (Random Forest):
{predicted_disease_rf}\n"
                                    f"Predicted Disease (Naive Bayes):
{predicted_disease_nb}\n"
                                    f"Predicted Disease (Decision Tree):
{predicted_disease_dt}",
                                    fg="green", font=("Arial", 12,
"bold"))

        # Implement voting mechanism to get the majority prediction
```

```
        predictions = [predicted_disease_rf, predicted_disease_nb,
predicted_disease_dt]
        majority_prediction = Counter(predictions).most_common(1)[0][0]

        main_prediction_label.config(text=f"Main Prediction:
{majority_prediction}", fg="darkblue", font=("Arial", 16, "bold"))

        # Check if the main predicted disease is in the dataset
        if majority_prediction in data['Disease'].values:
            # Get the actual remedy description using the decoded label
            remedy_description = data.loc[data['Disease'] ==
majority_prediction, 'REMEDY'].values[0]
            remedy_label.config(text=f"Remedy:
{label_encoder.inverse_transform([remedy_description])[0]}", fg="blue",
font=("Arial", 14))  # Decode remedy
        else:
            remedy_label.config(text="Remedy not found for the main
predicted disease.", fg="orange", font=("Arial", 14))

        # Ask if the user wants to make another prediction
        another_prediction = messagebox.askyesno("Another Prediction", "Do
you want to make another prediction?")
        if another_prediction:
            predicted_label.config(text="")  # Clear the predicted disease
label
            main_prediction_label.config(text="")  # Clear the main
prediction label
            remedy_label.config(text="")  # Clear the remedy suggestion
label
            for label_text in labels:
                slider_values[label_text].set(0)  # Reset sliders to
default position
        else:
            window.destroy()  # Close the GUI window

    except ValueError:
        messagebox.showwarning("Input Error", "Please enter valid integer
values for symptoms (-1 to 3).", icon="error")

# Initial Page Frame
initial_page_frame = tk.Frame(window)
initial_page_frame.pack(fill=tk.BOTH, expand=True)

# Load background image for initial page
bg_image = Image.open("C:/Users/HPW/OneDrive/Desktop/assets/ml.jpg")
```

```
bg_photo = ImageTk.PhotoImage(bg_image)
bg_label = tk.Label(initial_page_frame, image=bg_photo)
bg_label.image = bg_photo  # Keep a reference to prevent garbage
collection
bg_label.pack(fill=tk.BOTH, expand=True)

open_main_button = tk.Button(
    initial_page_frame,
    text="Disease Prediction",
    command=open_main_page,
    width=30,
    height=5,
    bg="lightblue",
    fg="darkblue",
font=("Arial", 14, "bold"),
relief=tk.RAISED,  # Add a raised border effect
borderwidth=3,  # Increase the border width
)
open_main_button.pack(pady=20)
```

```
# Main Page Frame
main_page_frame = tk.Frame(window)
```

```
# Create a frame for symptom levels
symptom_levels_frame = tk.LabelFrame(main_page_frame, text="Symptom
Levels", padx=10, pady=10, font=("Arial", 12, "bold"), bg='lightblue',
fg='black')
symptom_levels_frame.pack(side=tk.LEFT, padx=10, pady=10,
fill=tk.Y)  # Pack to the left side
```

```
# Labels for different symptom levels
symptom_levels = [
"3 - High",
"2 - Moderate",
"1 - Mild",
"-1 - None"
]
```

```
# Display labels for symptom levels
for level in symptom_levels:
level_label = tk.Label(symptom_levels_frame, text=level,
font=("Arial", 10), bg='lightblue', fg='navy')
level_label.pack(anchor=tk.W, padx=5, pady=5)
```

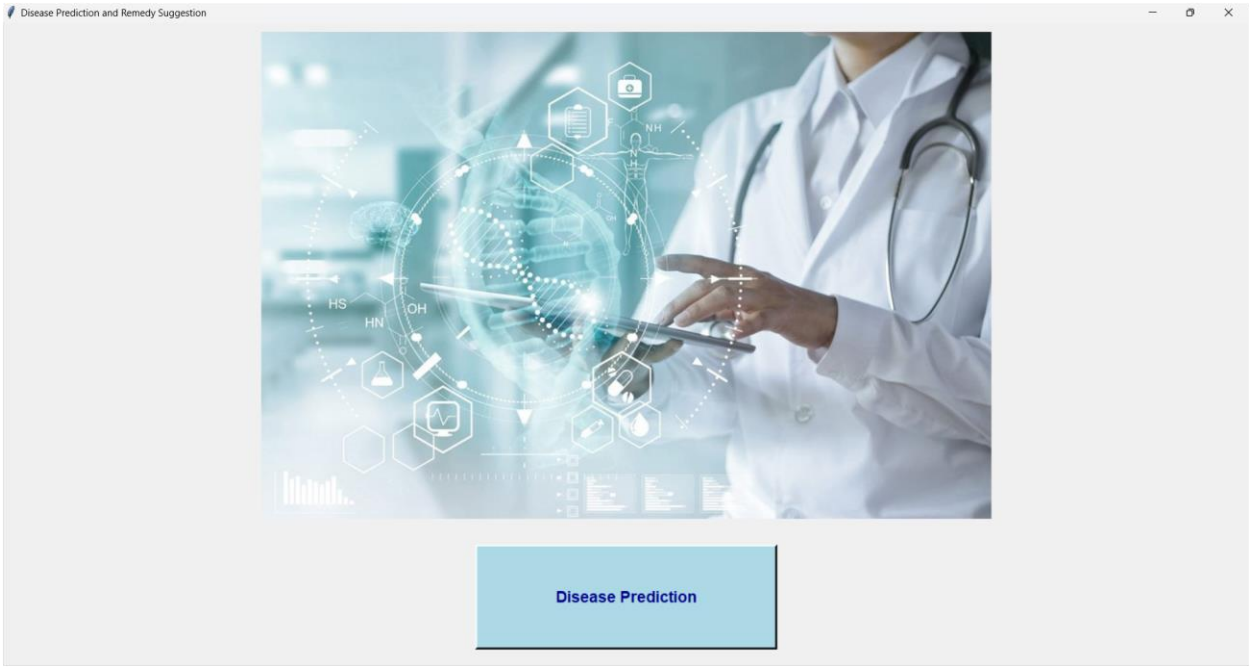
```
# Create a frame for symptoms entry
```

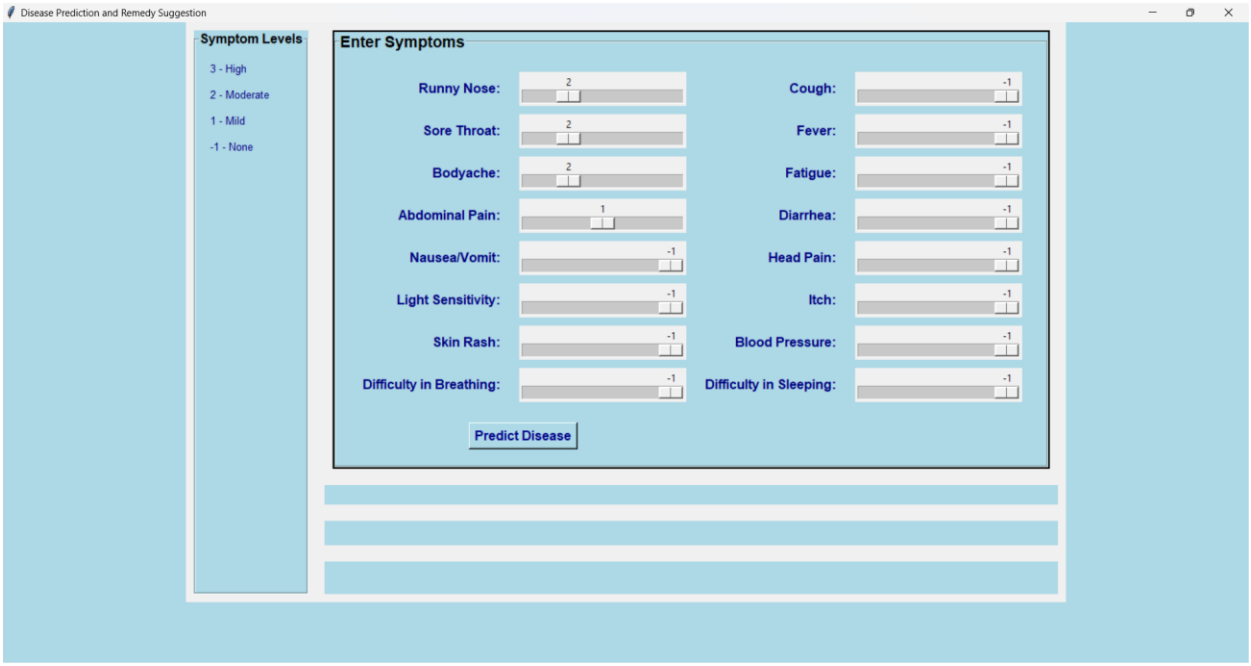
```
symptoms_frame = tk.LabelFrame(main_page_frame, text="Enter Symptoms",
padx=20, pady=20, font=("Arial", 14, "bold"), bg='lightblue', fg='black')
symptoms_frame.pack(padx=20, pady=10, anchor=tk.NW)

labels = [
'Runny Nose', 'Cough', 'Sore Throat', 'Fever', 'Bodyache','Fatigue',
'Abdominal Pain', 'Diarrhea', 'Nausea/Vomit', 'Head Pain',
'Light Sensitivity', 'Itch', 'Skin Rash', 'Blood Pressure',
'Difficulty in Breathing', 'Difficulty in Sleeping'
]
# Variables to hold user input for symptoms
slider_values = {}
for label_text in labels:
label = tk.Label(symptoms_frame, text=label_text + ":", font=("Arial", 12,
"bold"), fg="navy", bg='lightblue')
label.grid(row=labels.index(label_text)//2,
column=(labels.index(label_text)%2)*2, sticky=tk.E, padx=10, pady=5)
slider = Scale(symptoms_frame, from_=3, to=-1, orient=tk.HORIZONTAL,
length=200)
slider.grid(row=labels.index(label_text)//2,
column=(labels.index(label_text)%2)*2+1, padx=10, pady=5)
slider_values[label_text] = slider

# Set different colors for the inside and outside of the symptoms entry
box
symptoms_frame.config(highlightbackground='black', highlightthickness=2)

# Button to predict disease and suggest remedy
predict_button = tk.Button(symptoms_frame, text="Predict Disease",
command=predict_disease_and_suggest_remedy, font=("Arial", 12, "bold"),
bg="lightblue", fg="darkblue")
predict_button.grid(row=len(labels)//2, columnspan=2, pady=(20, 0))

# Label to display predicted disease from each algorithm
predicted_label = tk.Label(main_page_frame, text="", font=("Arial", 12),
bg='lightblue', justify=tk.LEFT)
predicted_label.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

# Label to display main predicted disease
main_prediction_label = tk.Label(main_page_frame, text="", font=("Arial",
16, "bold"), bg='lightblue', fg='darkblue')
main_prediction_label.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

# Label to display suggested remedy for the main predicted disease
```

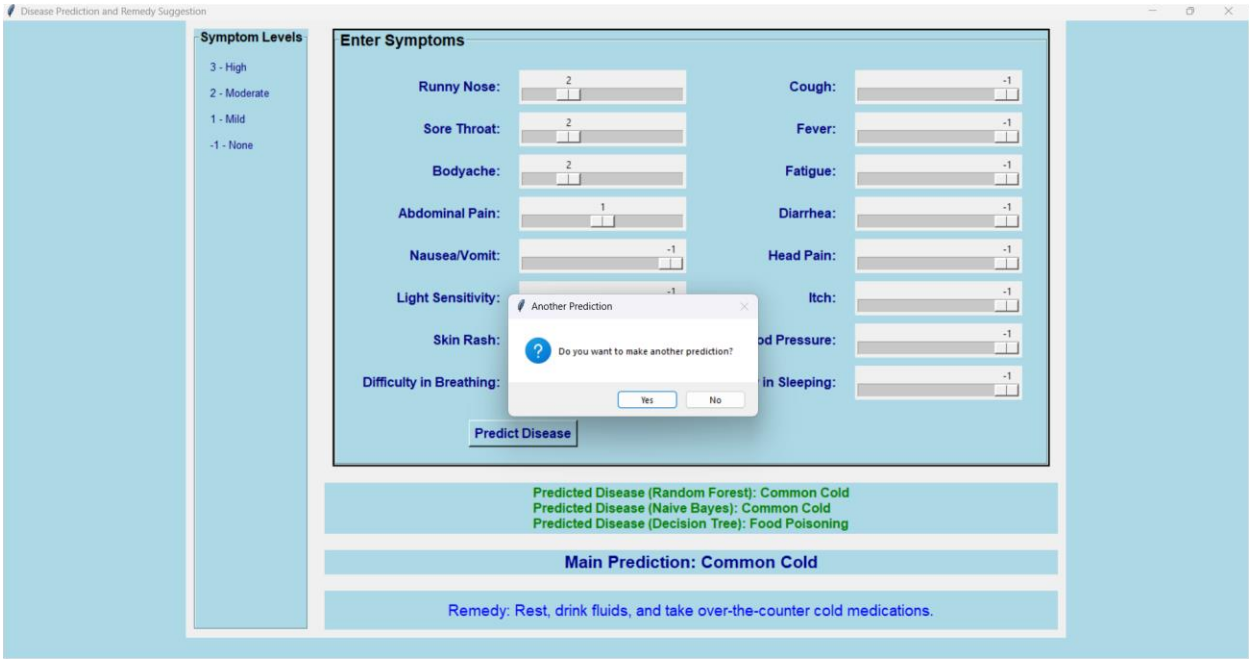
```
remedy_label = tk.Label(main_page_frame, text="", font=("Arial", 8),
bg='lightblue', wraplength=800, justify='left')
remedy_label.pack(padx=10, pady=10, fill=tk.BOTH, expand=True, ipady=10)
```

```
# Run the GUI
window.mainloop()
```

- ***Part 2 output:***







Remedy Recommendation System:

The preprocessed dataset contains remedy to be suggested for the predicted disease. The model will predict the disease based on the symptoms entered by the user using the

algorithms and then suggest remedy for the predicted diseases. this would provide following advantages:

1. ***Personalized Recommendations***
2. ***Patient Engagement***
3. ***Efficient Resource Allocation***
4. ***Support for Healthcare Professionals***
5. ***Research and Insights***

Results and Evaluation:

The project evaluates disease prediction accuracy of 3 algorithms and then evaluate the best using metrics like confusion matrices. It also depicts the importance of symptoms while predicting using graph (3). This would give us insight into how random forest works in classifying the disease. This would then help in the GUI based model when user can interact with model. The remedy recommendation system's effectiveness can assessed through user feedback or domain expert validation. Although no machine learning model has 100% accuracy, it is important to evaluate the existing model and provide the best model with relatively higher accuracy to the users.

Conclusion:

The project demonstrates the feasibility and effectiveness of using machine learning for disease prediction and remedy suggestion. It contributes to improving healthcare outcomes and decision-making processes, with potential for future enhancements and applications.

References:

1. https://www.ijraset.com/research-paper/disease-prediction-and-treatment-recommendation-using-machine-learning
2. https://github.com/anujdutt9/Disease-Prediction-from-Symptoms
3. https://www.researchgate.net/publication/357449131_THE_PREDICTION_OF_DISEASE_USING_MACHINE_LEARNING
4. https://www.ije.ir/article_169090_5525e34b7bd485c6f9f9cc710f62522f.pdf