Chapter 1: Introduction and Objective

Early and precise disease identification is essential for successful treatment and better patient outcomes in today's healthcare environment. With just five input symptoms, our cutting-edge multi-disease prediction system uses machine learning to forecast a variety of common ailments like fungal infections and jaundice, along with other common diseases like fungal infection, typhoid, jaundice, and many more. With the help of strong machine learning algorithms and an easy-to-use Streamlit-based user interface, this system enables users—from consumers to medical professionals—to rapidly and accurately evaluate any health hazards. Our approach seeks to close the gap between sophisticated medical diagnostics and easily available health information by fusing cutting-edge predictive analytics with an intuitive design, opening the door for proactive healthcare management.

Objectives:

The primary objectives of this project are:

- Develop and train various machine learning algorithms and models for prediction of disease like Diabetes, Parkinson's, Cardiovascular, jaundice, typhoid, and many more.
- Implement a user-friendly web interface in order to input medical parameters as well as obtain the necessary predictions.
- Combine predictions for multiple diseases into a cohesive platform.
- Demonstrating the value of machine learning in the facilitation of early diagnosis as well as proactive health management.
- Enhancing the model's robustness through advanced techniques such as outlier removal, data standardization, and ensemble learning.

Chapter 2: System Analysis

Problem Definition:

One of the biggest causes of death worldwide is still heart disease. While manual diagnosis based on lab data and medical history can be subjective, delayed, or unreliable, early prediction can save lives. A machine learning system uses clinical data to predict whether a patient has heart disease, Parkinson's disease, diabetes, typhoid, fungal infections, or jaundice. The algorithm finds patterns suggestive of these disorders by examining features like blood pressure, glucose, cholesterol, BMI, bilirubin levels, speech data, movement sensor readings, skin lesion photos, and epidemiological data. The cardiovascular risks of heart disease, the neurological symptoms of Parkinson's disease, the metabolic imbalances of diabetes, the infectious indicators of typhoid, the systemic or dermatological signs of fungal infections, and the liver-related problems of jaundice can all be detected early with this method. The approach improves patient outcomes by increasing diagnostic accuracy and expediting intervention across different diseases by processing huge datasets and minimizing human error.

Inputs:

- **Heart Disease Prediction:**
    - Age
    - Sex
    - Chest Pain Type
    - RestingBP
    - Cholesterol
    - Fasting Blood Sugar
    - Resting ECG
    - Max Heart Rate
    - Exercise Angina
    - Oldpeak

- ST Slope

- **Parkinson's Disease Prediction:**
  - MDVP:Fo(Hz)
  - MDVP:Fhi(Hz)
  - MDVP:Flo(Hz)
  - MDVP:Jitter(%)
  - MDVP:Jitter(Abs)
  - MDVP:RAP
  - MDVP:PPQ
  - Jitter:DDP
  - MDVP:Shimmer
  - MDVP:Shimmer(dB)
  - Shimmer:APQ3
  - Shimmer:APQ5
  - MDVP:APQ
  - Shimmer:DDA
  - NHR
  - HNR
  - RPDE
  - DFA
  - spread1
  - spread2
  - D2
  - PPE

- **Diabetes Prediction:**
  - Number of Pregnancies
  - Glucose Level

    - Blood Pressure Value
    - Skin Thickness Value
    - Insulin Level
    - BMI Value
    - Diabetes Pedigree Function Value
    - Age of the Person

- **Disease Prediction Based on Symptoms:** The person can choose 5 main symptoms from a variety of symptoms. These symptoms include: 'back pain', 'constipation', 'abdominal pain', 'diarrhoea', 'mild fever', 'yellow urine', 'yellowing of eyes', 'acute liver failure', 'fluid overload', 'swelling of stomach', 'swelled lymph nodes', 'malaise', 'blurred and distorted vision', 'phlegm', 'throat irritation', 'redness of eyes', 'sinus pressure', 'runny nose', 'congestion', 'chest pain', 'weakness in limbs', 'fast heart rate', 'pain during bowel movements', 'pain in anal region', 'bloody stool', 'irritation in anus', 'neck pain', 'dizziness', 'cramps', 'bruising', 'obesity', 'swollen legs', 'swollen blood vessels', 'puffy face and eyes', 'enlarged thyroid', 'brittle nails', 'swollen extremeties', 'excessive hunger', 'extra marital contacts', 'drying and tingling lips', 'slurred speech', 'knee pain', 'hip joint pain', 'muscle weakness', 'stiff neck', 'swelling joints', 'movement stiffness', 'spinning movements', 'loss of balance', 'unsteadiness', 'weakness of one body side', 'loss of smell', 'bladder discomfort', 'continuous feel of urine', 'passage of gases', 'internal itching', 'toxic look (typhos)', 'depression', 'irritability', 'muscle pain', 'altered sensorium', 'red spots over body', 'belly pain', 'abnormal menstruation', 'watering from eyes', 'increased appetite', 'polyuria', 'family history', 'mucoid sputum', 'rusty sputum', 'lack of concentration', 'visual disturbances', 'receiving blood transfusion', 'receiving unsterile injections', 'coma', 'stomach bleeding', 'distention of abdomen', 'history of alcohol consumption', 'blood in sputum', 'prominent veins on calf', 'palpitations', 'painful walking', 'pus filled pimples', 'blackheads', 'skin peeling', 'silver like dusting', 'small dents in nails', 'inflammatory nails', 'blister', 'red sore around nose', 'yellow crust ooze'.
    - Symptom 1
    - Symptom 2
    - Symptom 3

- Symptom 4
- Symptom 5

Output:

- Heart Disease: Based on the inputs, the model will give the output whether the person has any chances of developing a heart disease or not.
- Parkinson's Disease: Based on the inputs, the model will give the output whether the person has any chances of developing Parkinson's disease or not.
- Diabetes: Based on the inputs provided, the model will give the result whether the person has chances of developing diabetes or not.
- Disease Prediction Based on Symptoms: Based on the inputs provided by the user, the model will give 3 predictions of the possible diseases. The diseases that the model can predict are: 'Fungal infection', 'Allergy', 'GERD', 'Chronic cholestasis', 'Drug Reaction', 'Peptic ulcer disease', 'AIDS', 'Diabetes ', 'Gastroenteritis', 'Bronchial Asthma', 'Hypertension ', 'Migraine', 'Cervical spondylosis', 'Paralysis (brain hemorrhage)', 'Jaundice', 'Malaria', 'Chicken pox', 'Dengue', 'Typhoid', 'hepatitis A', 'Hepatitis B', 'Hepatitis C', 'Hepatitis D', 'Hepatitis E', 'Alcoholic hepatitis', 'Tuberculosis', 'Common Cold ','Pneumonia', 'Dimorphic hemorrhoids(piles)', 'Heart attack', 'Varicose veins', 'Hypothyroidism', 'Hyperthyroidism', 'Hypoglycemia', 'Osteoarthristis', 'Arthritis', '(vertigo) Paroymsal Positional Vertigo', 'Acne', 'Urinary tract infection', 'Psoriasis', 'Impetigo'

Potential Users:

- Data Scientists / Developers (during development)
- Medical professionals (during deployment, via a UI or app)
- Patients (indirect users via apps)

Chapter 3: Data Flow Diagram (DFD)

The Data Flow Diagram focuses on the requirements of the application System and also the flow of structure of the System. It is a graphic representation that shows how data moves through a system or procedure. It illustrates how data is input, processed, stored, and produced using symbols like rectangles, circles, and arrows; it does not specify the order or timing of these steps. DFDs are frequently used to comprehend and enhance systems in business process management, systems analysis, and software engineering.

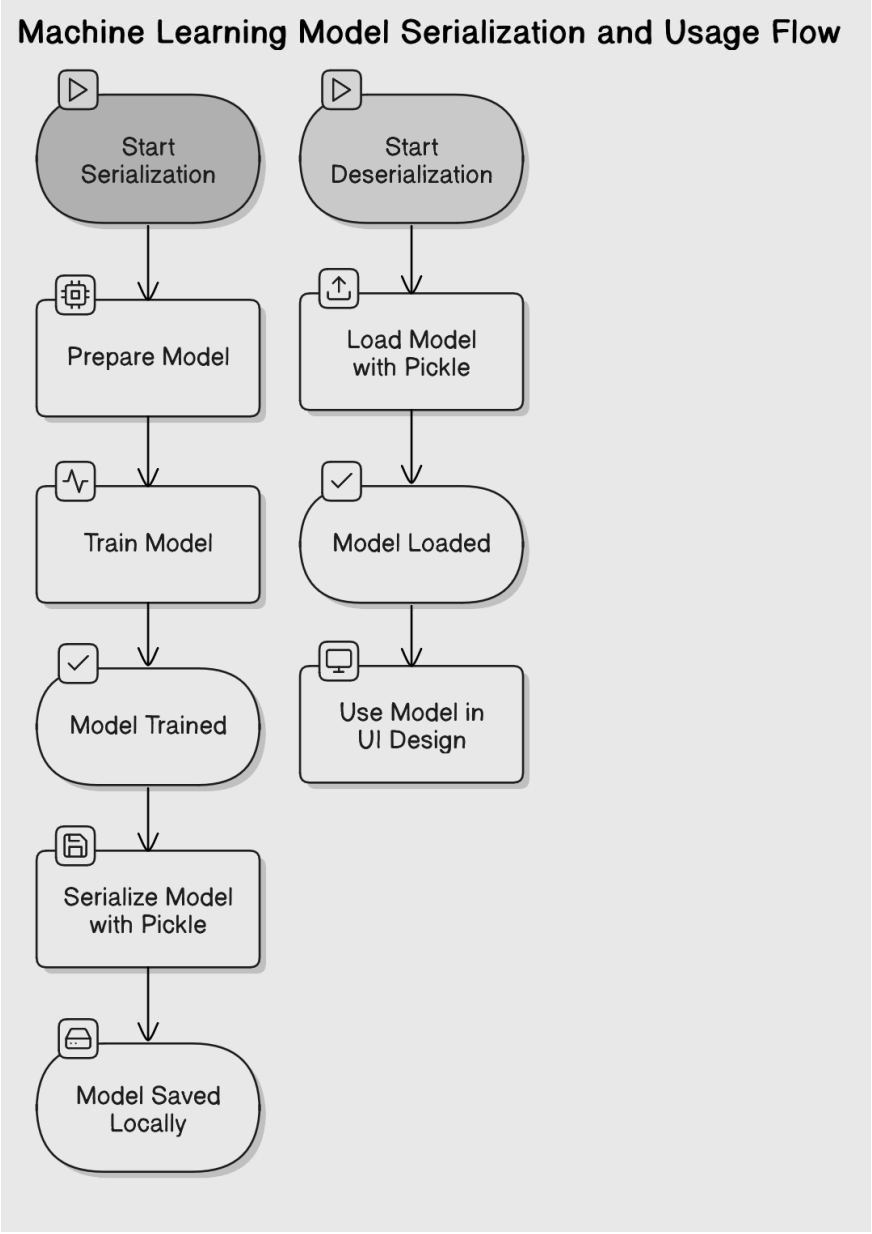


Fig. Machine Learning Model Serialization and Usage Flow

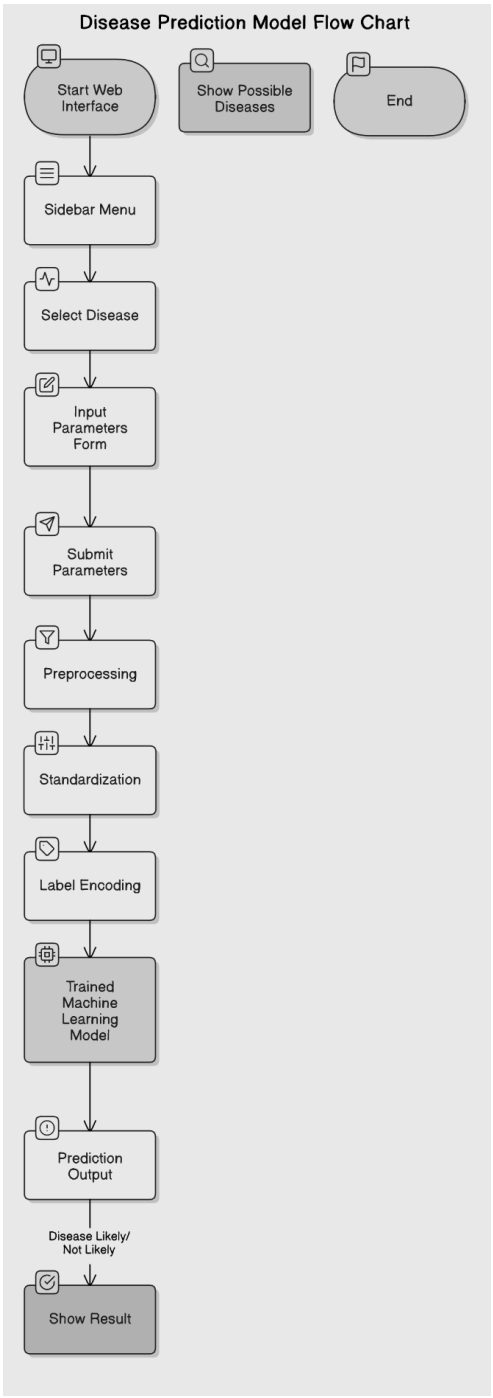


Fig. Disease Prediction Model Flow Chart

Chapter 4: Requirement Specification

Working:

Four machine learning models are used by the Multiple Disease Prediction System to predict various medical disorders. A Support Vector Classifier (SVC) with a linear kernel is used by the diabetes and Parkinson's disease predictors to categorize people according to numerical health measures. The predictions of three base learners—Random Forest, XGBoost, and Logistic Regression—are combined in a complicated stacking classifier for heart disease, and the results are then sent to a final logistic regression estimator. This ensemble method makes use of the advantages of several models to increase forecast accuracy. Three separate algorithms—Decision Tree, Random Forest, and Naive Bayes—are used in the symptoms-based illness prediction model. Each algorithm was trained on a dataset that mapped symptoms to potential diseases.

Python's pickle module serializes the trained model objects so they can be used again without requiring retraining after these models have been trained and saved locally. This system's user interface (UI) is constructed with Streamlit, an open-source Python framework that enables the quick development of interactive web applications straight from Python scripts. Because of its incredibly user-friendly design, even non-programmers may easily utilize Streamlit. Each of the several components that make up the user interface corresponds to a distinct illness prediction model. Depending on the prediction module they are using, users engage with the system by entering their personal health information in dropdowns, sliders, or input forms. This information may include their age, blood pressure, glucose level, or specific symptoms.

Users can enter data into the Streamlit interface and load a model into memory by clicking the "Predict" button. The input data is preprocessed by the system, including scaling and encoding, before being fed into the trained model. One prediction is given back for the diabetes, Parkinson's, and heart disease modules. Three predictions from the Decision Tree, Random Forest, and Naive Bayes models are given in the section on symptoms-based prediction. Users can rapidly comprehend their possible health state thanks to the results' clear labeling on the user interface.

Programming Language: Python

Libraries Used:

- **pandas**: Used for data manipulation and analysis with DataFrame and Series objects.
- **numpy**: Used for numerical computations and handling arrays/matrices efficiently.
- **pickle**: Used for serializing and deserializing Python objects to save/load models or data.
- **sklearn.model_selection.train_test_split**: Used to split datasets into training and testing sets.
- **sklearn.preprocessing.StandardScaler**: Used to standardize features by removing the mean and scaling to unit variance.
- **sklearn.preprocessing.LabelEncoder**: Used to encode categorical labels into numerical values.
- **sklearn.ensemble.RandomForestClassifier**: Used for classification tasks using an ensemble of decision trees.
- **xgboost.XGBClassifier**: Used for gradient boosting-based classification tasks.
- **sklearn.linear_model.LogisticRegression**: Used for binary or multiclass classification using logistic regression.
- **sklearn.ensemble.StackingClassifier**: Used to combine multiple classifiers for improved predictions via stacking.
- **sklearn.metrics.classification_report**: Used to generate a detailed classification performance report.
- **sklearn.metrics.roc_curve**: Used to compute Receiver Operating Characteristic (ROC) curve metrics.
- **sklearn.metrics.auc**: Used to calculate the Area Under the Curve (AUC) for ROC or precision-recall curves.
- **matplotlib.pyplot**: Used for creating visualizations like plots and charts.
- **seaborn**: Used for enhanced statistical data visualization built on matplotlib.
- **sklearn.svm**: Used for Support Vector Machine (SVM) algorithms for classification or regression.
- **sklearn.metrics.accuracy_score**: Used to calculate the accuracy of a model's predictions.
- **sklearn.svm.SVC**: Used for classification tasks using Support Vector Classification.

- **sklearn.tree.DecisionTreeClassifier**: Used for classification tasks using a single decision tree.
- **sklearn.naive_bayes.GaussianNB**: Used for classification tasks using the Naive Bayes algorithm with Gaussian distribution.
- **sklearn.model_selection.cross_val_score**: Used to evaluate a model's performance using cross-validation.
- **streamlit**: Used for creating interactive web applications for data visualization and machine learning with minimal code.
- **streamlit_option_menu**: Used to create customizable option menus in Streamlit applications for navigation or selection.

Machine Learning Models Used:

- **Random Forest:** It is an ensemble machine learning algorithm that builds multiple decision trees and then combines their prediction, either by voting or averaging, in order to improve the efficiency (accuracy) as well as reduce overfitting. This algorithm is suitable for classification and regression tasks.
- **XGBoost:** It is a gradient boosting algorithm that optimizes decision trees sequentially , minimizing errors with high efficiency as well as accuracy. This algorithm is suitable for classification, regression, as well as ranking tasks.
- **Logistic Regression:** It is a statistical method for binary or multiclass classification that predicts probabilities using a logical function. This algorithm is effective for linearly separable data.
- **Stacking Classifier:** It is an ensemble technique that is used to combine predictions from multiple base classes, like SVM or Random Forest, with the help of a meta-classifier that is used to improve the overall performance as well as generalization.
- **Support Vector Machine (SVM):** It is a classification algorithm. SVM finds the optimal hyperplane to separate classes, maximizing the margin between them. It is effective for high-dimensional and non-linear data using kernel tricks.
- **Decision Tree:** It is a tree based model. Decision tree splits data into branches based on feature conditions to make decisions. It is used for classification as well as regression, but is prone to overfitting if not pruned.

- **Gaussian Naive Bayes:** It is a probability classifier based on Bayes' theorem assuming features follow a Gaussian distribution and are conditionally independent. It is suitable for simple and fast classification tasks.

Development Environment: Jupyter notebook was used for model development. Python scripts were used for deployment. Spyder IDE was used for UI designing.

Datasets Used:

- **Diabetes Dataset:** Pima Indians Diabetes Dataset (available at Kaggle)
    - Total Features: 9
    - Labels: 0 (Non-diabetic), 1 (Diabetic)
    - Size: 768 records
- **Parkinson's Disease Dataset:** Voice Measurements Dataset (available at Kaggle)
    - Total Features: 22
    - Labels: 0 (Healthy), 1 (Parkinson's)
    - Size: 195 records
- **Heart Disease Dataset:** Cleveland Heart Disease Dataset (available at Kaggle)
    - Total Features: 12
    - Labels: 0 (Healthy), 1 (Heart Disease)
    - Size: 918
- **Symptoms Based Disease Prediction:** (available at Kaggle)
    - Total Features: 133
    - Labels: 40 different prognosis
    - Size: 4961

Chapter 5: System Architecture and Methodology

This system follows a modular approach:

User Input → Data Preprocessing → Prediction Model → Output Result

Each disease prediction model accepts specific medical parameters via a web interface that is created using Python's Streamlit library. After that, it applies preprocessing, like standardization and label encoding, after which processes inputs through a trained machine learning model. The output predicts whether the person is likely to have a particular disease or not, or tell the person what the disease can be based on the symptoms.

The web interface, built with streamlit, features a sidebar menu in order to select the target disease and input forms for entering parameters.

Methodology:

Date Preprocessing: Data preprocessing is a basic as well as a primary step that converts raw data into some useful information. In most of the cases, data can be incomplete, noisy, or redundant (repetitive). These issues can be resolved by performing data preprocessing, and thus can be used for the generation of machine learning models.

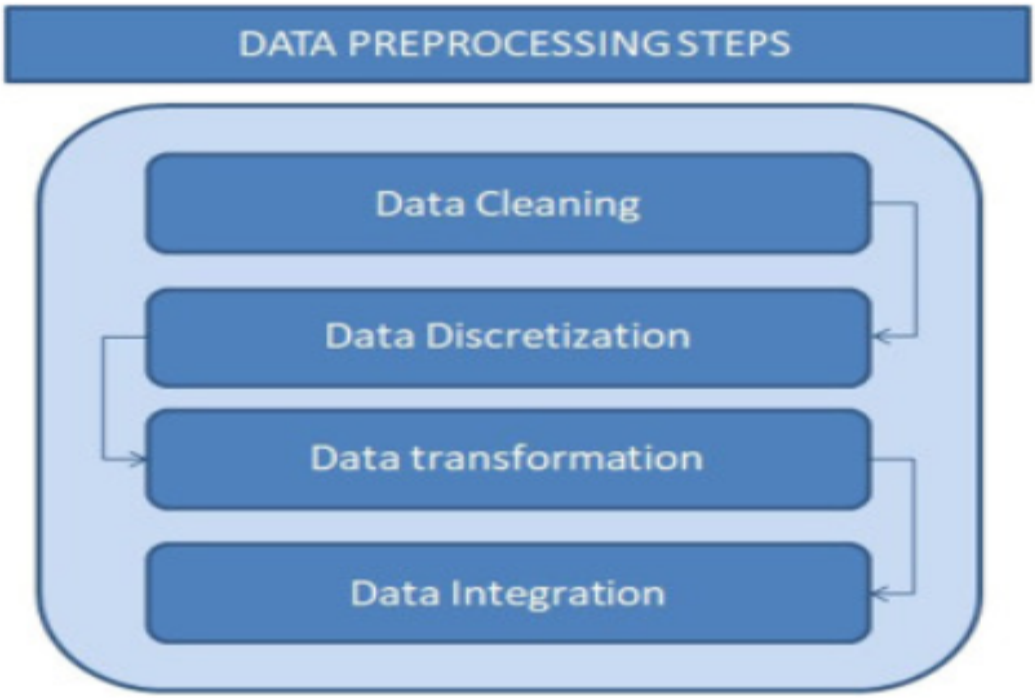


Fig. Data Preprocessing Steps

There are four major steps in data preprocessing. These steps are:

- Data Cleaning: Clean data is essential for reliable machine learning and analysis outcomes, requiring correcting errors and inconsistencies in data, including handling missing values, removing duplicates, and fixing formatting issues.
- Data Discretization: Data discretization converts continuous data into discrete intervals, improving model performance and making interpretation easier. Common methods include binning, histogram analysis, and clustering-based approaches.
- Data Transformation: Data transformation converts data into suitable format for analysis or modeling, including normalization, standardization, encoding, and mathematical functions, improving model convergence and accuracy.
- Data Integration: Data integration consolidates data from various sources for analysis, addressing schema mismatches and inconsistencies through ETL processes and data warehousing, especially in distributed databases and cloud services.

The following steps were applied in the 'Multiple Disease Prediction System' for data preprocessing:

- Missing values: Missing values in the dataset were taken care of by either removing them or changing them using descriptive statistics.
- Outlier removal: Outliers from the dataset were found (using Inter quartile range and box plot), and were dealt with.
- Encoding: Categorical variables from the datasets were encoded using LabelEncoder.
- Standardization: Numerical Features were standardized using StandardScaler in order to ensure consistent scales across features.
- Data Splitting: Datasets were split into training (80%) as well as testing (20%) sets.
- SMOTE (Synthetic Minority Oversampling Technique): SMOTE was applied to deal with class imbalance in the datasets.

Model Training:

- Diabetes Prediction: Trained using Support Vector Classifier (SVC) with a linear kernel.
- Parkinson's Disease Prediction: Trained using Support Vector Classifier (SVC) with a linear kernel.
- Heart Disease Prediction: Utilized a Stacking Classifier combining Random Forest, XGBoost, and Logistic Regression, with Logistic Regression as the final estimator.
- Symptoms Based Disease Prediction: Trained on three distinct disease prediction models: Decision Tree, Random Forest, and Naive Bayes, each trained independently and evaluated on test data, resulting in a total of three predictions.

Model Accuracy:

Disease	**Model(s) Used**	**Accuracy**
Heart Disease Prediction	Stacking Classifier combining Random Forest, XGBoost, and	84.9 % (cross-validation accuracy)

	Logistic Regression Final Estimator: Logistic Regression	
Diabetes Prediction	SVC	78 %
Parkinson's Disease Prediction	SVC	87.2 %
Symptoms Based Disease Prediction	Decision Tree, Random Forest, and Naive Bayes	92.68 % (for all three models)

UI Design: Streamlit, a python framework was used to develop the User Interface for 'Multiple Disease Prediction System'. Streamlit is an open-source Python framework for data scientists and AI/ML engineers to deliver dynamic data apps with only a few lines of code. Because of its user-friendly design, Streamlit can also be used by people without any prior web programming skills. The models were saved on the local device using pickle, which is a module that implements binary protocols for serializing as well as de-serializing a Python object structure. These models were loaded again to use for UI design.

Chapter 6: Code Implementation with Screenshots

Diabetes Prediction Model:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score

diabetes_dataset = pd.read_csv('diabetes.csv')

X = diabetes_dataset.drop(columns = 'Outcome', axis=1)
Y = diabetes_dataset['Outcome']

scaler = StandardScaler()

scaler.fit(X)

standardized_data = scaler.transform(X)

X = standardized_data
Y = diabetes_dataset['Outcome']

X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2, stratify=Y, random_state=2)

classifier = svm.SVC(kernel='linear')
```

```
classifier.fit(X_train, Y_train)

X_train_prediction = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)

X_test_prediction = classifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)

input_data = (5,166,72,19,175,25.8,0.587,51)
input_data_as_numpy_array = np.asarray(input_data)
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)
std_data = scaler.transform(input_data_reshaped)
print(std_data)
prediction = classifier.predict(std_data)
print(prediction)
if (prediction[0] == 0):
  print('The person is not diabetic')
else:
  print('The person is diabetic')

import pickle
filename = "diabetes_model.sav"
pickle.dump(classifier, open(filename, "wb"))

filename = "diabetes_scaler.sav"
pickle.dump(scaler, open(filename, "wb"))
```

Parkinson's Prediction Model:

```
import pandas as pd
import numpy as np

data = pd.read_csv("dataset.csv")

X=data.drop(columns = ['name', 'status'], axis=1)
y=data['status']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=2)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit_transform(X_train)
scaler.fit_transform(X_test)

from sklearn.svm import SVC

model = SVC(kernel='linear')

model.fit(X_train, y_train)

pred_train = model.predict(X_train)
```

```
from sklearn.metrics import accuracy_score

train_acc = accuracy_score(y_train, pred_train)

pred_test = model.predict(X_test)

input_data =
(122.4,148.65,113.819,0.00968,0.00008,0.00465,0.00696,0.01394,0.06134,0.626,0.03134,0.045
18,0.04368,0.09403,0.01929,19.085,0.458359,0.819521,-4.075192,0.33559,2.486855,0.368674)
input_array = np.asarray(input_data)
reshaped_data = input_array.reshape(1,-1)
std_data = scaler.transform(reshaped_data)
prediction = model.predict(std_data)
if prediction[0]==0:
        print("Healthy")
else:
        print("You might have Parkinson Disease")

import pickle
filename = "parkinsons_model.sav"
pickle.dump(model, open(filename, "wb"))
filename = "parkinsons_scaler.sav"
pickle.dump(scaler, open(filename, "wb"))
```

Heart Disease Prediction:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
```

```
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import StackingClassifier
from sklearn.metrics import classification_report, roc_curve, auc
import matplotlib.pyplot as plt
import seaborn as sns

heart_failure = pd.read_csv('heart.csv')

def validate_data(df):
    print("Data Validation Report:")

    print("\nAge range:", df['Age'].min(), "to", df['Age'].max())

    print("\nUnique Sex values:", df['Sex'].unique())

    print("\nUnique ChestPainType values:", df['ChestPainType'].unique())

    print("\nRestingBP range:", df['RestingBP'].min(), "to", df['RestingBP'].max())

    print("\nCholesterol range:", df['Cholesterol'].min(), "to", df['Cholesterol'].max())

    print("\nUnique FastingBS values:", df['FastingBS'].unique())

    print("\nUnique RestingECG values:", df['RestingECG'].unique())
```

```
    print("\nMaxHR range:", df['MaxHR'].min(), "to", df['MaxHR'].max())

    print("\nUnique ExerciseAngina values:", df['ExerciseAngina'].unique())

    print("\nUnique ST_Slope values:", df['ST_Slope'].unique())

validate_data(heart_failure)

def remove_outliers(df, columns):
    df_clean = df.copy()

    for column in columns:
    Q1 = df_clean[column].quantile(0.25)
    Q3 = df_clean[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outliers = ((df_clean[column] < lower_bound) | (df_clean[column] > upper_bound))
    print(f"\nNumber of outliers in {column}: {outliers.sum()}")

    df_clean = df_clean[~outliers]

    return df_clean

numerical_columns = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']

plt.figure(figsize=(15, 5))
```

```
heart_failure[numerical_columns].boxplot()
plt.title('Boxplots Before Outlier Removal')
plt.xticks(rotation=45)
plt.show()

heart_failure_clean = remove_outliers(heart_failure, numerical_columns)

plt.figure(figsize=(15, 5))
heart_failure_clean[numerical_columns].boxplot()
plt.title('Boxplots After Outlier Removal')
plt.xticks(rotation=45)
plt.show()

print(f"\nOriginal dataset size: {len(heart_failure)}")
print(f"Dataset size after outlier removal: {len(heart_failure_clean)}")

def clean_cholesterol_improved(df):
    df_clean = df.copy()

    df_clean = df_clean[df_clean['Cholesterol'] >= 100]

    Q1 = df_clean['Cholesterol'].quantile(0.25)
    Q3 = df_clean['Cholesterol'].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = max(Q1 - 1.5 * IQR, 130)
    upper_bound = min(Q3 + 1.5 * IQR, 400)
```

```
        cholesterol_outliers = ((df_clean['Cholesterol'] < lower_bound) |
                (df_clean['Cholesterol'] > upper_bound))

        print(f"Number of Cholesterol outliers: {cholesterol_outliers.sum()}")
        print(f"Range for Cholesterol: {lower_bound:.2f} to {upper_bound:.2f}")

        df_clean = df_clean[~cholesterol_outliers]
        return df_clean

heart_failure_clean = clean_cholesterol_improved(heart_failure)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
bp1 = plt.boxplot(heart_failure['Cholesterol'],
                patch_artist=True,
                medianprops=dict(color="orange"),
                boxprops=dict(facecolor='lightblue'))
plt.title('Cholesterol Before Cleaning')
plt.ylabel('Cholesterol (mg/dl)')

plt.subplot(1, 2, 2)
bp2 = plt.boxplot(heart_failure_clean['Cholesterol'],
                patch_artist=True,
                medianprops=dict(color="orange"),
                boxprops=dict(facecolor='lightblue'))
plt.title('Cholesterol After Cleaning')
plt.ylabel('Cholesterol (mg/dl)')
```

```
plt.tight_layout()
plt.show()

print("\nCholesterol Statistics After Cleaning:")
print(heart_failure_clean['Cholesterol'].describe())
print(f"\nOriginal dataset size: {len(heart_failure)}")
print(f"Dataset size after cleaning: {len(heart_failure_clean)}")

df = heart_failure_clean.copy()

le = LabelEncoder()

categorical_columns = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']
for column in categorical_columns:
        df[column] = le.fit_transform(df[column])

X = df.drop('HeartDisease', axis=1)
y = df['HeartDisease']

numerical_columns = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']
scaler = StandardScaler()
X[numerical_columns] = scaler.fit_transform(X[numerical_columns])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

base_models = [
        ('rf', RandomForestClassifier(random_state=42)),
```

```
        ('xgb', XGBClassifier(random_state=42)),
        ('lr', LogisticRegression(random_state=42))
]

stacking = StackingClassifier(
        estimators=base_models,
        final_estimator=LogisticRegression(),
        cv=5,
        stack_method='predict_proba'
)

stacking.fit(X_train, y_train)

def get_detailed_risk_assessment(patient_data, probability):
        risk_prob = probability * 100

        if risk_prob < 20:
        risk_level = "Low"
        recommendation = "Maintain healthy lifestyle and regular check-ups"
        timeframe = "Annual check-up recommended"
        elif risk_prob < 40:
        risk_level = "Moderate-Low"
        recommendation = "Continue regular monitoring with lifestyle modifications"
        timeframe = "Follow-up in 6 months"
        elif risk_prob < 60:
        risk_level = "Moderate-High"
        recommendation = "Schedule consultation with healthcare provider"
        timeframe = "Follow-up within 1 month"
```

```
elif risk_prob < 80:
risk_level = "High"
recommendation = "Prompt medical evaluation needed"
timeframe = "Follow-up within 1 week"
else:
risk_level = "Very High"
recommendation = "Urgent medical attention required"
timeframe = "Immediate medical attention"

risk_factors = []
concerns = []

if patient_data['Age'] > 60:
risk_factors.append("Advanced age")
elif patient_data['Age'] > 45:
concerns.append("Age-related risk")

if patient_data['RestingBP'] >= 140:
risk_factors.append("High blood pressure")
elif patient_data['RestingBP'] >= 120:
concerns.append("Elevated blood pressure")

if patient_data['Cholesterol'] >= 240:
risk_factors.append("High cholesterol")
elif patient_data['Cholesterol'] >= 200:
concerns.append("Borderline cholesterol")

if patient_data['ST_Slope'] == 'Down':
```

```
        risk_factors.append("Abnormal ST slope")
        elif patient_data['ST_Slope'] == 'Flat':
        concerns.append("Flat ST slope")

        if patient_data['ExerciseAngina'] == 'Y':
        risk_factors.append("Exercise-induced angina")

        if patient_data['ChestPainType'] == 'ASY':
        risk_factors.append("Asymptomatic chest pain")

        return risk_level, recommendation, timeframe, risk_factors, concerns

def predict_comprehensive(patient_data, model, scaler):
        patient_df = pd.DataFrame([patient_data])

        categorical_columns = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina',
'ST_Slope']
        le = LabelEncoder()
        for column in categorical_columns:
        patient_df[column] = le.fit_transform(patient_df[column])

        numerical_columns = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']
        patient_df[numerical_columns] = scaler.transform(patient_df[numerical_columns])

        prediction = model.predict(patient_df)
        probability = model.predict_proba(patient_df)[0][1]

        risk_level, recommendation, timeframe, risk_factors, concerns =
get_detailed_risk_assessment(patient_data, probability)
```

```
        print("\n=== Comprehensive Heart Disease Risk Assessment ===")
        print("\nPatient Profile:")
        print(f"Age: {patient_data['Age']}, Sex: {patient_data['Sex']}")
        print(f"Blood Pressure: {patient_data['RestingBP']} mm Hg")
        print(f"Cholesterol: {patient_data['Cholesterol']} mm/dl")
        print(f"Max Heart Rate: {patient_data['MaxHR']}")

        print("\nCritical Indicators:")
        print(f"Chest Pain Type: {patient_data['ChestPainType']}")
        print(f"Exercise Angina: {'Present' if patient_data['ExerciseAngina']=='Y' else
'Absent'}")
        print(f"ST Depression: {patient_data['Oldpeak']}")
        print(f"ST Slope: {patient_data['ST_Slope']}")

        print("\nRisk Assessment:")
        print(f"Risk Level: {risk_level}")
        print(f"Heart Disease Probability: {probability:.1%}")

        if risk_factors:
        print("\nMajor Risk Factors:")
        for factor in risk_factors:
        print(f"• {factor}")

        if concerns:
        print("\nAreas of Concern:")
        for concern in concerns:
        print(f"• {concern}")
```

```
        print(f"\nRecommendation: {recommendation}")
        print(f"Timeframe: {timeframe}")

        if risk_level in ["High", "Very High"]:
        print("\nWarning: Multiple high-risk factors detected. Immediate medical consultation advised.")

y_pred_final = stacking.predict(X_test)
print("\nModel Performance Metrics:")
print(classification_report(y_test, y_pred_final))

cv_scores = cross_val_score(stacking, X, y, cv=5)
print(f"\nCross-validation accuracy: {cv_scores.mean():.3f} (+/- {cv_scores.std() * 2:.3f})")

import pickle

with open('stacking_model.pkl', 'wb') as f:
        pickle.dump(stacking, f)

with open('scaler.pkl', 'wb') as f:
        pickle.dump(scaler, f)

label_encoders = {}
categorical_columns = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']

for column in categorical_columns:
        le = LabelEncoder()
```

```
        le.fit(heart_failure_clean[column])
        label_encoders[column] = le

with open('label_encoders.pkl', 'wb') as f:
        pickle.dump(label_encoders, f)

print("Model, scaler, and encoders have been saved successfully!")
```

Symptoms Based Disease Prediction:

```
import pandas as pd
import numpy as np
import pickle

l1 = ['back_pain', 'constipation', 'abdominal_pain', 'diarrhoea', 'mild_fever', 'yellow_urine',
'yellowing_of_eyes', 'acute_liver_failure', 'fluid_overload', 'swelling_of_stomach',
'swelled_lymph_nodes', 'malaise', 'blurred_and_distorted_vision', 'phlegm', 'throat_irritation',
'redness_of_eyes', 'sinus_pressure', 'runny_nose', 'congestion', 'chest_pain', 'weakness_in_limbs',
'fast_heart_rate', 'pain_during_bowel_movements', 'pain_in_anal_region', 'bloody_stool',
'irritation_in_anus', 'neck_pain', 'dizziness', 'cramps', 'bruising', 'obesity', 'swollen_legs',
'swollen_blood_vessels', 'puffy_face_and_eyes', 'enlarged_thyroid', 'brittle_nails',
'swollen_extremeties', 'excessive_hunger', 'extra_marital_contacts', 'drying_and_tingling_lips',
'slurred_speech', 'knee_pain', 'hip_joint_pain', 'muscle_weakness', 'stiff_neck', 'swelling_joints',
'movement_stiffness', 'spinning_movements', 'loss_of_balance', 'unsteadiness',
'weakness_of_one_body_side', 'loss_of_smell', 'bladder_discomfort', 'continuous_feel_of_urine',
'passage_of_gases', 'internal_itching', 'toxic_look_(typhos)', 'depression', 'irritability',
'muscle_pain', 'altered_sensorium', 'red_spots_over_body', 'belly_pain', 'abnormal_menstruation',
'watering_from_eyes', 'increased_appetite', 'polyuria', 'family_history', 'mucoid_sputum',
'rusty_sputum', 'lack_of_concentration', 'visual_disturbances', 'receiving_blood_transfusion',
```

```
'receiving_unsterile_injections', 'coma', 'stomach_bleeding', 'distention_of_abdomen',
'history_of_alcohol_consumption', 'blood_in_sputum', 'prominent_veins_on_calf', 'palpitations',
'painful_walking', 'pus_filled_pimples', 'blackheads', 'skin_peeling', 'silver_like_dusting',
'small_dents_in_nails', 'inflammatory_nails', 'blister', 'red_sore_around_nose',
'yellow_crust_ooze']


df = pd.read_csv("Training.csv")

df.replace({'prognosis': {
        'Fungal infection':0,'Allergy':1,'GERD':2,'Chronic cholestasis':3,'Drug Reaction':4,
        'Peptic ulcer diseae':5,'AIDS':6,'Diabetes ':7,'Gastroenteritis':8,'Bronchial Asthma':9,
        'Hypertension ':10,'Migraine':11,'Cervical spondylosis':12,'Paralysis (brain
hemorrhage)':13,
        'Jaundice':14,'Malaria':15,'Chicken pox':16,'Dengue':17,'Typhoid':18,'hepatitis A':19,
        'Hepatitis B':20,'Hepatitis C':21,'Hepatitis D':22,'Hepatitis E':23,'Alcoholic hepatitis':24,
        'Tuberculosis':25,'Common Cold':26,'Pneumonia':27,'Dimorphic hemmorhoids(piles)':28,
        'Heart attack':29,'Varicose
veins':30,'Hypothyroidism':31,'Hyperthyroidism':32,'Hypoglycemia':33,
        'Osteoarthristis':34,'Arthritis':35,'(vertigo) Paroymsal  Positional Vertigo':36,'Acne':37,
        'Urinary tract infection':38,'Psoriasis':39,'Impetigo':40
}}, inplace=True)

X = df[l1]
y = np.ravel(df[['prognosis']])

tr = pd.read_csv("Testing.csv")
tr.replace({'prognosis': {
        'Fungal infection':0,'Allergy':1,'GERD':2,'Chronic cholestasis':3,'Drug Reaction':4,
```

```
        'Peptic ulcer diseae':5,'AIDS':6,'Diabetes ':7,'Gastroenteritis':8,'Bronchial Asthma':9,
        'Hypertension ':10,'Migraine':11,'Cervical spondylosis':12,'Paralysis (brain hemorrhage)':13,
        'Jaundice':14,'Malaria':15,'Chicken pox':16,'Dengue':17,'Typhoid':18,'hepatitis A':19,
        'Hepatitis B':20,'Hepatitis C':21,'Hepatitis D':22,'Hepatitis E':23,'Alcoholic hepatitis':24,
        'Tuberculosis':25,'Common Cold':26,'Pneumonia':27,'Dimorphic hemmorhoids(piles)':28,
        'Heart attack':29,'Varicose veins':30,'Hypothyroidism':31,'Hyperthyroidism':32,'Hypoglycemia':33,
        'Osteoarthristis':34,'Arthritis':35,'(vertigo) Paroymsal  Positional Vertigo':36,'Acne':37,
        'Urinary tract infection':38,'Psoriasis':39,'Impetigo':40
}}, inplace=True)

X_test = tr[l1]
y_test = np.ravel(tr[['prognosis']])

from sklearn.tree import DecisionTreeClassifier
clf_dt = DecisionTreeClassifier()
clf_dt.fit(X, y)

from sklearn.metrics import accuracy_score
y_pred_dt = clf_dt.predict(X_test)
print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))

with open('decision_tree_model.pkl', 'wb') as f:
        pickle.dump(clf_dt, f)

from sklearn.ensemble import RandomForestClassifier
clf_rf = RandomForestClassifier()
clf_rf.fit(X, y)
```

```
y_pred_rf = clf_rf.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))

with open('random_forest_model.pkl', 'wb') as f:
        pickle.dump(clf_rf, f)

from sklearn.naive_bayes import GaussianNB
clf_nb = GaussianNB()
clf_nb.fit(X, y)

y_pred_nb = clf_nb.predict(X_test)
print("Naive Bayes Accuracy:", accuracy_score(y_test, y_pred_nb))

with open('naive_bayes_model.pkl', 'wb') as f:
        pickle.dump(clf_nb, f)

print("All models trained and saved successfully!")
```

Streamlit Code for UI Design:

```
import pickle
import streamlit as st
from streamlit_option_menu import option_menu
import numpy as np
import pandas as pd

diabetes_model = pickle.load(open("C:/Users/shubh/OneDrive/Desktop/Multiple Disease Prediction/saved models/diabetes_model.sav", 'rb'))
```

```
diabetes_scaler = pickle.load(open("C:/Users/shubh/OneDrive/Desktop/Multiple Disease Prediction/saved models/diabetes_scaler.sav", 'rb'))

parkinsons_model = pickle.load(open("C:/Users/shubh/OneDrive/Desktop/Multiple Disease Prediction/saved models/parkinsons_model.sav", 'rb'))
parkinsons_scaler = pickle.load(open("C:/Users/shubh/OneDrive/Desktop/Multiple Disease Prediction/saved models/parkinsons_scaler.sav", 'rb'))

decision_tree_model = pickle.load(open("C:/Users/shubh/OneDrive/Desktop/Multiple Disease Prediction/saved models/decision_tree_model.pkl", 'rb'))
random_forest_model = pickle.load(open("C:/Users/shubh/OneDrive/Desktop/Multiple Disease Prediction/saved models/random_forest_model.pkl", 'rb'))
naive_bayes_model = pickle.load(open("C:/Users/shubh/OneDrive/Desktop/Multiple Disease Prediction/saved models/naive_bayes_model.pkl", 'rb'))

stacking_model = pickle.load(open('C:/Users/shubh/OneDrive/Desktop/Multiple Disease Prediction/saved models/stacking_model.pkl', 'rb'))
scaler = pickle.load(open('C:/Users/shubh/OneDrive/Desktop/Multiple Disease Prediction/saved models/scaler.pkl', 'rb'))
label_encoders = pickle.load(open('C:/Users/shubh/OneDrive/Desktop/Multiple Disease Prediction/saved models/label_encoders.pkl', 'rb'))

autism_model = pickle.load(open('C:/Users/shubh/OneDrive/Desktop/Multiple Disease Prediction/saved models/autism_model.pkl', 'rb'))
autism_encoders = pickle.load(open('C:/Users/shubh/OneDrive/Desktop/Multiple Disease Prediction/saved models/autism_encoders.pkl', 'rb'))


l1 = ['back_pain', 'constipation', 'abdominal_pain', 'diarrhoea', 'mild_fever', 'yellow_urine',
```

'yellowing_of_eyes', 'acute_liver_failure', 'fluid_overload', 'swelling_of_stomach', 'swelled_lymph_nodes',

'malaise', 'blurred_and_distorted_vision', 'phlegm', 'throat_irritation', 'redness_of_eyes', 'sinus_pressure',

'runny_nose', 'congestion', 'chest_pain', 'weakness_in_limbs', 'fast_heart_rate', 'pain_during_bowel_movements',

'pain_in_anal_region', 'bloody_stool', 'irritation_in_anus', 'neck_pain', 'dizziness', 'cramps', 'bruising',

'obesity', 'swollen_legs', 'swollen_blood_vessels', 'puffy_face_and_eyes', 'enlarged_thyroid', 'brittle_nails',

'swollen_extremeties', 'excessive_hunger', 'extra_marital_contacts', 'drying_and_tingling_lips', 'slurred_speech',

'knee_pain', 'hip_joint_pain', 'muscle_weakness', 'stiff_neck', 'swelling_joints', 'movement_stiffness',

'spinning_movements', 'loss_of_balance', 'unsteadiness', 'weakness_of_one_body_side', 'loss_of_smell',

'bladder_discomfort', 'continuous_feel_of_urine', 'passage_of_gases', 'internal_itching', 'toxic_look_(typhos)',

'depression', 'irritability', 'muscle_pain', 'altered_sensorium', 'red_spots_over_body', 'belly_pain',

'abnormal_menstruation', 'watering_from_eyes', 'increased_appetite', 'polyuria', 'family_history', 'mucoid_sputum',

'rusty_sputum', 'lack_of_concentration', 'visual_disturbances', 'receiving_blood_transfusion', 'receiving_unsterile_injections',

'coma', 'stomach_bleeding', 'distention_of_abdomen', 'history_of_alcohol_consumption', 'blood_in_sputum',

'prominent_veins_on_calf', 'palpitations', 'painful_walking', 'pus_filled_pimples', 'blackheads', 'skin_peeling',

'silver_like_dusting', 'small_dents_in_nails', 'inflammatory_nails', 'blister', 'red_sore_around_nose', 'yellow_crust_ooze']

```
disease_map = {
    0: 'Fungal infection', 1: 'Allergy', 2: 'GERD', 3: 'Chronic cholestasis', 4: 'Drug Reaction',
    5: 'Peptic ulcer diseae', 6: 'AIDS', 7: 'Diabetes', 8: 'Gastroenteritis', 9: 'Bronchial Asthma',
    10: 'Hypertension', 11: 'Migraine', 12: 'Cervical spondylosis', 13: 'Paralysis (brain hemorrhage)',
    14: 'Jaundice', 15: 'Malaria', 16: 'Chicken pox', 17: 'Dengue', 18: 'Typhoid', 19: 'hepatitis A',
    20: 'Hepatitis B', 21: 'Hepatitis C', 22: 'Hepatitis D', 23: 'Hepatitis E', 24: 'Alcoholic hepatitis',
    25: 'Tuberculosis', 26: 'Common Cold', 27: 'Pneumonia', 28: 'Dimorphic hemmorhoids(piles)', 29: 'Heart attack',
    30: 'Varicose veins', 31: 'Hypothyroidism', 32: 'Hyperthyroidism', 33: 'Hypoglycemia', 34: 'Osteoarthristis',
    35: 'Arthritis', 36: '(vertigo) Paroymsal  Positional Vertigo', 37: 'Acne', 38: 'Urinary tract infection',
    39: 'Psoriasis', 40: 'Impetigo'
}



with st.sidebar:
    selected = option_menu("Multiple Disease Prediction System",

        ["Diabetes Prediction",
            "Parkisons Prediction",
            "Disease Prediction using ML Models",
            "Heart Disease Prediction"],
```

```
                icons = ['activity', 'person', 'hospital', 'heart-pulse-fill'],

                        default_index = 0)

if (selected == 'Diabetes Prediction'):
        st.title("Diabetes Prediction using ML")



        col1, col2, col3 = st.columns(3)

        with col1:
        Pregnancies = st.text_input("Number of Pregnancies")

        with col2:
        Glucose = st.text_input("Glucose Level")

        with col3:
        BloodPressure = st.text_input("Blood Pressure Value")

        with col1:
        SkinThickness = st.text_input("Skin Thickness Value")

        with col2:
        Insulin = st.text_input("Insulin Level")

        with col3:
        BMI = st.text_input("BMI Value")
```

```
with col1:
DiabetesPedigreeFunction = st.text_input("Diabetes Pedigree Function Value")

with col2:
Age = st.text_input("Age of the Person")



diab_diagnosis = ""

if st.button("Diabetes Test Result"):
input_data = np.array([[Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age]], dtype=float)

scaled_input_data = diabetes_scaler.transform(input_data)

diab_prediction = diabetes_model.predict(scaled_input_data)

if (diab_prediction[0] == 1):
diab_diagnosis = 'The Person is diabetic'
else:
diab_diagnosis = 'The Person is not diabetic'

st.success(diab_diagnosis)
```

```
if (selected == 'Parkisons Prediction'):
        st.title("Parkisons Prediction using ML")

        col1, col2, col3, col4, col5 = st.columns(5)

        with col1:
        MDVP_Fo_Hz = st.text_input("MDVP:Fo(Hz)")

        with col2:
        MDVP_Fhi_Hz = st.text_input("MDVP:Fhi(Hz)")

        with col3:
        MDVP_Flo_Hz = st.text_input("MDVP:Flo(Hz)")

        with col4:
        MDVP_Jitter_percent = st.text_input("MDVP:Jitter(%)")

        with col5:
        MDVP_Jitter_Abs = st.text_input("MDVP:Jitter(Abs)")

        with col1:
        MDVP_RAP = st.text_input("MDVP:RAP")

        with col2:
        MDVP_PPQ = st.text_input("MDVP:PPQ")

        with col3:
        Jitter_DDP = st.text_input("Jitter:DDP")
```

```
with col4:
MDVP_Shimmer = st.text_input("MDVP:Shimmer")

with col5:
MDVP_Shimmer_dB = st.text_input("MDVP:Shimmer(dB)")

with col1:
Shimmer_APQ3 = st.text_input("Shimmer:APQ3")

with col2:
Shimmer_APQ5 = st.text_input("Shimmer:APQ5")

with col3:
MDVP_APQ = st.text_input("MDVP:APQ")

with col4:
Shimmer_DDA = st.text_input("Shimmer:DDA")

with col5:
NHR = st.text_input("NHR")

with col1:
HNR = st.text_input("HNR")

with col2:
RPDE = st.text_input("RPDE")
```

```
with col3:
DFA = st.text_input("DFA")

with col4:
spread1 = st.text_input("spread1")

with col5:
spread2 = st.text_input("spread2")

with col1:
D2 = st.text_input("D2")

with col2:
PPE = st.text_input("PPE")



park_diagnosis = ""

if st.button("Parkinson's Test Result"):
input_data = np.array([[MDVP_Fo_Hz, MDVP_Fhi_Hz, MDVP_Flo_Hz, MDVP_Jitter_percent, MDVP_Jitter_Abs, MDVP_RAP, MDVP_PPQ, Jitter_DDP, MDVP_Shimmer, MDVP_Shimmer_dB, Shimmer_APQ3, Shimmer_APQ5, MDVP_APQ, Shimmer_DDA, NHR, HNR, RPDE, DFA, spread1, spread2, D2, PPE]], dtype=float)

scaled_input_data = parkinsons_scaler.transform(input_data)

park_prediction = parkinsons_model.predict(scaled_input_data)
```

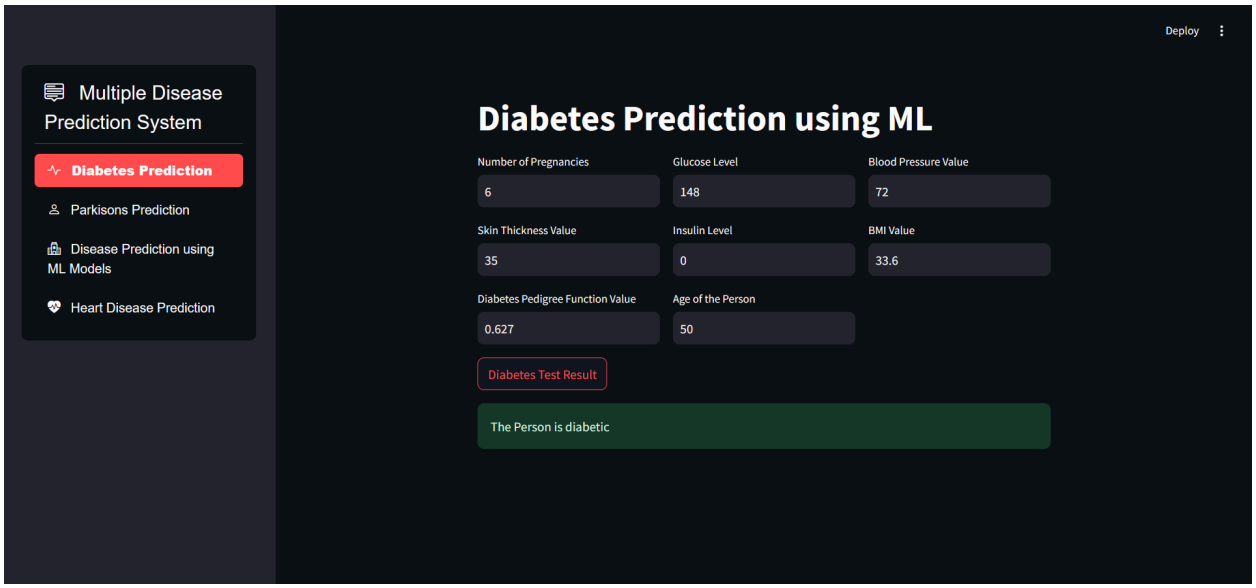
```
        if (park_prediction[0] == 1):
        park_diagnosis = 'The Person is having Parkinsons'
        else:
        park_diagnosis = 'The Person is not having Parkinsons'

        st.success(park_diagnosis)


if selected == "Disease Prediction using ML Models":
        st.title("Disease Prediction by Symptoms")

        symptoms_input = []

        for i in range(5):  # Loop to show 5 symptoms dropdowns
        symptom = st.selectbox(f"Choose symptom {i+1}", l1)
        symptoms_input.append(symptom)

        symptoms_input_binary = [1 if symptom in symptoms_input else 0 for symptom in l1]

        symptoms_input_array = np.array([symptoms_input_binary]).astype(int)

        if st.button("Prediction 1"):
        prediction_dt = decision_tree_model.predict(symptoms_input_array)
        disease_name = disease_map.get(prediction_dt[0], "Unknown disease")
        st.success(f"Predicted Disease: {disease_name}")

        if st.button("Prediction 2"):
        prediction_rf = random_forest_model.predict(symptoms_input_array)
```

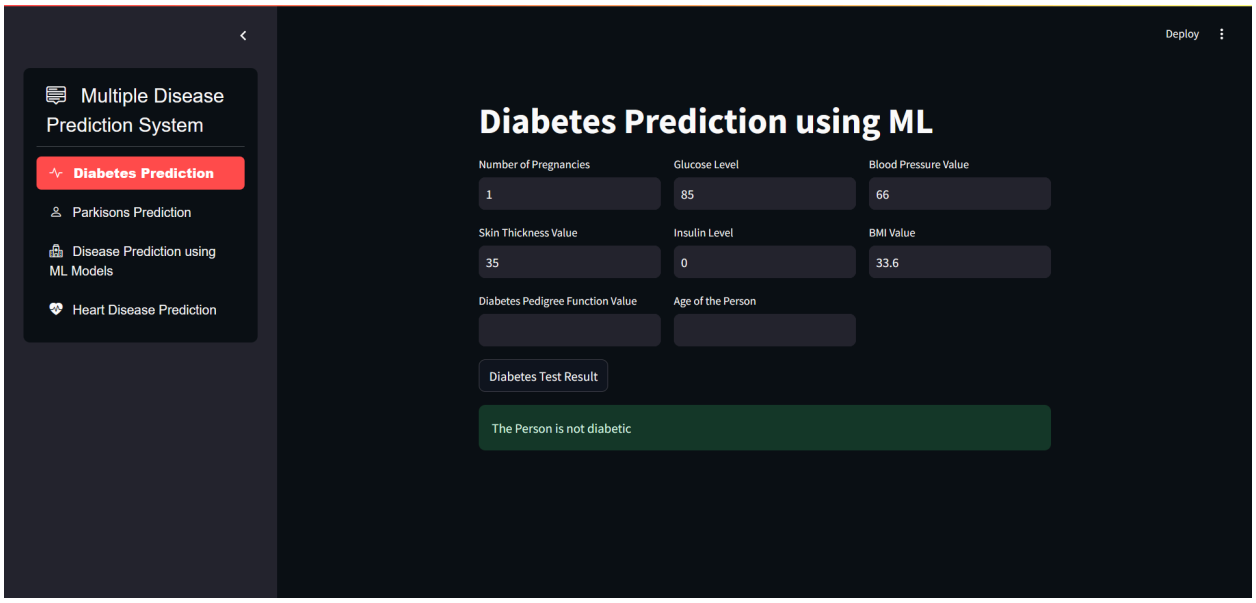
```
        disease_name = disease_map.get(prediction_rf[0], "Unknown disease")
        st.success(f"Predicted Disease: {disease_name}")

        if st.button("Prediction 3"):
        prediction_nb = naive_bayes_model.predict(symptoms_input_array)
        disease_name = disease_map.get(prediction_nb[0], "Unknown disease")
        st.success(f"Predicted Disease: {disease_name}")



if selected == "Heart Disease Prediction":

        st.title("Heart Disease Prediction using Stacking Model")

        col1, col2, col3 = st.columns(3)

        with col1:
        Age = st.text_input("Age")
        with col2:
        Sex = st.selectbox("Sex", ['M', 'F'])
        with col3:
        ChestPainType = st.selectbox("ChestPainType", ['TA', 'ATA', 'NAP', 'ASY'])

        with col1:
        RestingBP = st.text_input("Resting Blood Pressure")
        with col2:
        Cholesterol = st.text_input("Cholesterol")
        with col3:
```

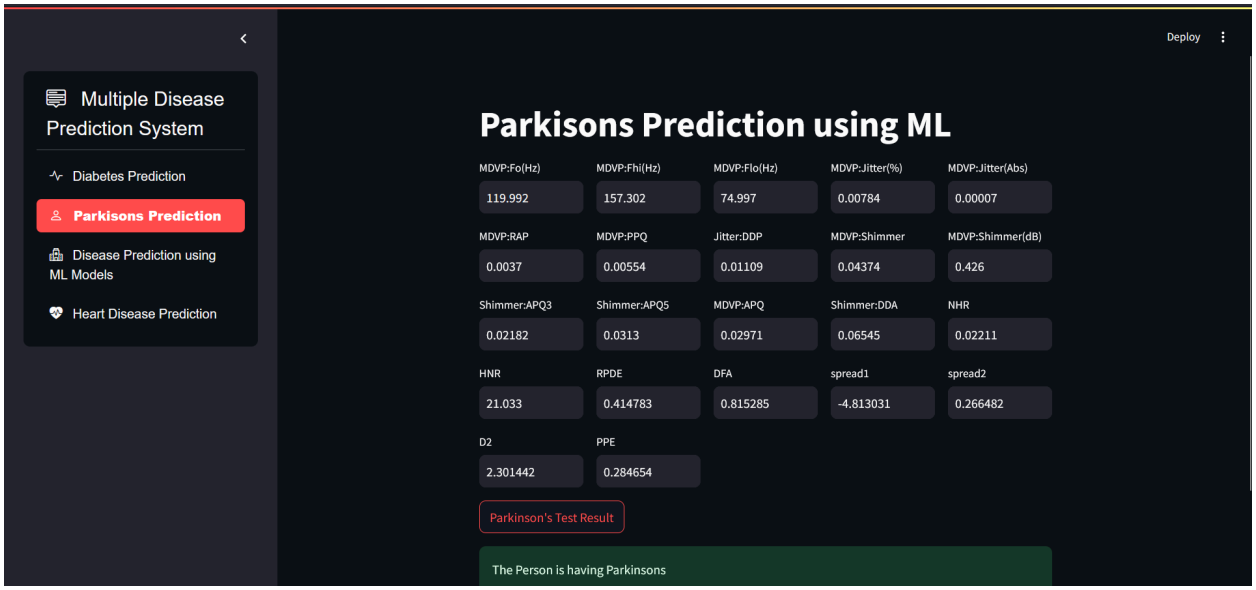
```
FastingBS = st.selectbox("Fasting Blood Sugar > 120 mg/dl", [1, 0])

with col1:
RestingECG = st.selectbox("RestingECG", ['Normal', 'ST', 'LVH'])
with col2:
MaxHR = st.text_input("Max Heart Rate")
with col3:
ExerciseAngina = st.selectbox("ExerciseAngina", ['Y', 'N'])

with col1:
Oldpeak = st.text_input("Oldpeak (Depression)")
with col2:
ST_Slope = st.selectbox("ST_Slope", ['Up', 'Flat', 'Down'])


if st.button("Heart Disease Test Result"):

input_data = {
'Age': [Age],
'Sex': [Sex],
'ChestPainType': [ChestPainType],
'RestingBP': [RestingBP],
'Cholesterol': [Cholesterol],
'FastingBS': [FastingBS],
'RestingECG': [RestingECG],
'MaxHR': [MaxHR],
'ExerciseAngina': [ExerciseAngina],
'Oldpeak': [Oldpeak],
```

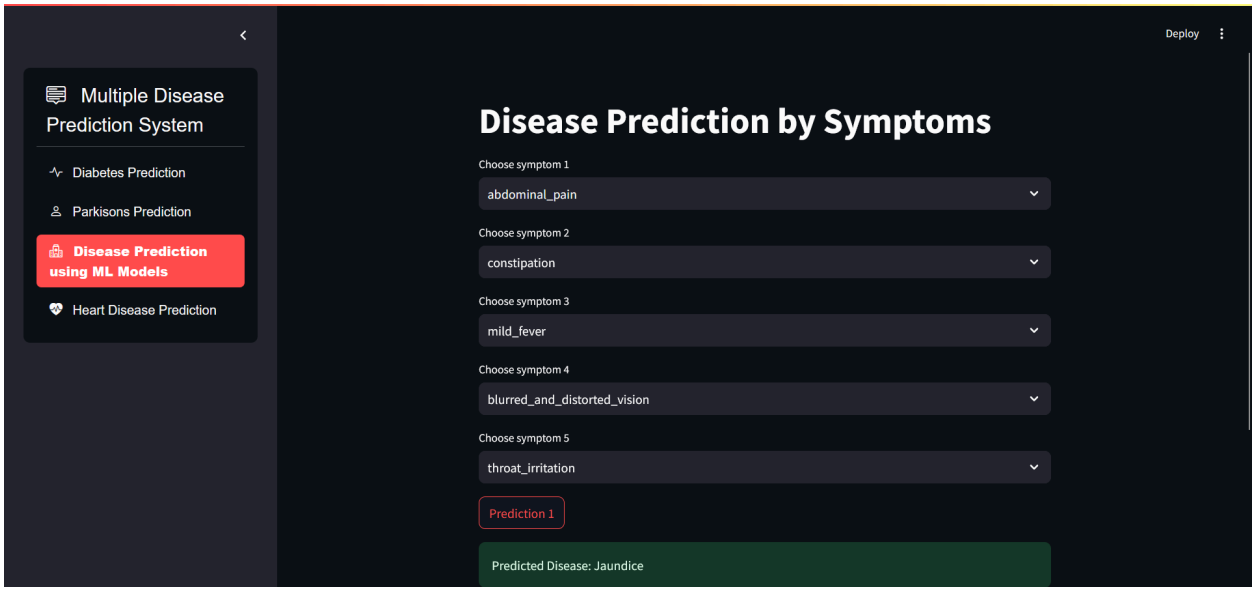
```
'ST_Slope': [ST_Slope]
}

input_df = pd.DataFrame(input_data)

categorical_columns = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']
for col in categorical_columns:
input_df[col] = label_encoders[col].transform(input_df[col])

numerical_columns = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']
input_df[numerical_columns] = scaler.transform(input_df[numerical_columns].astype(float))

expected_order = ['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol',
'FastingBS', 'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope']

input_df = input_df[expected_order]

prediction = stacking_model.predict(input_df)

if prediction[0] == 1:
st.success("The Person is likely to have heart disease")
else:
st.success("The Person is unlikely to have heart disease")
```

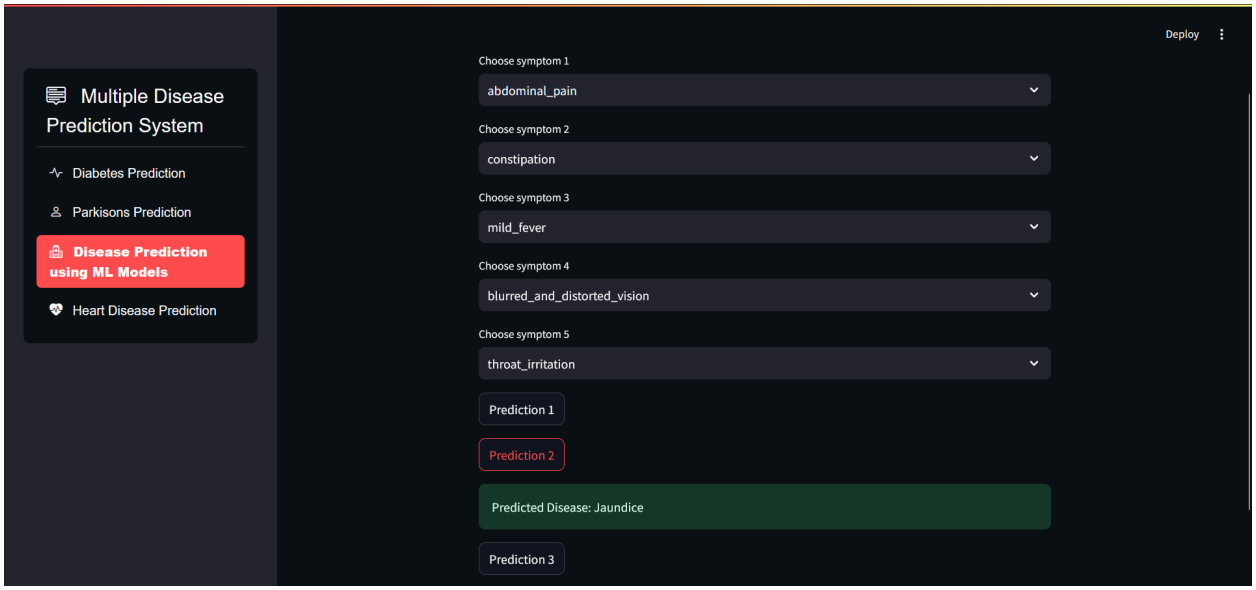
UI Output:

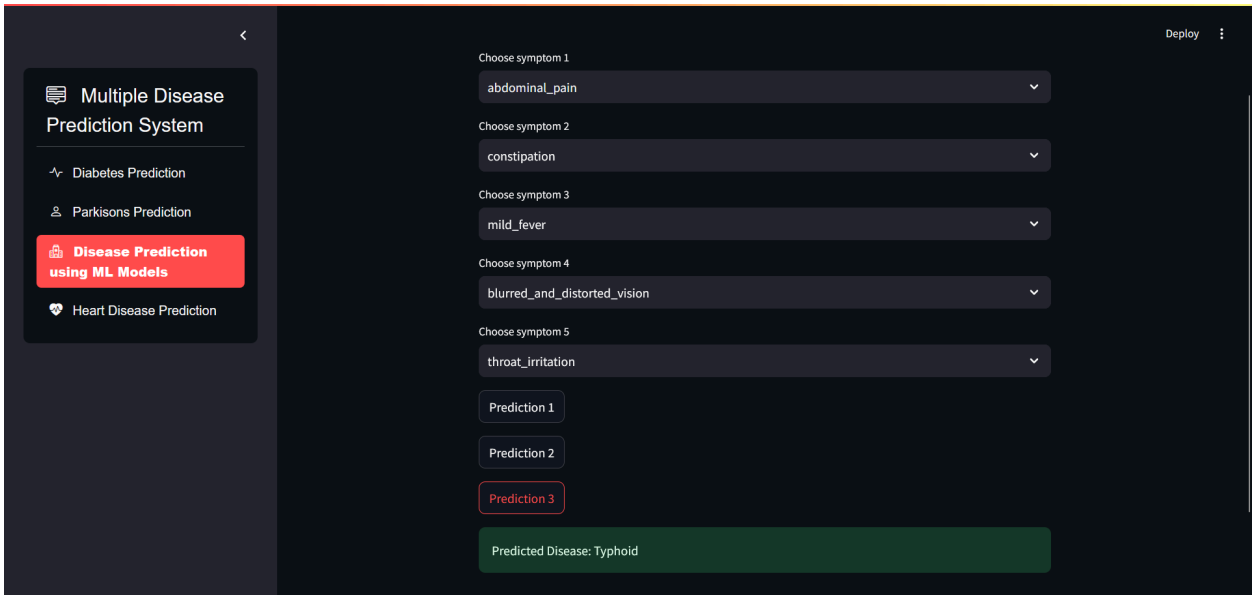

Deploy
Multiple Disease Prediction System
Diabetes Prediction
Parkisons Prediction
Disease Prediction using ML Models
Heart Disease Prediction
Diabetes Prediction using ML
Number of Pregnancies
6
Glucose Level
148
Blood Pressure Value
72
Skin Thickness Value
35
Insulin Level
0
BMI Value
33.6
Diabetes Pedigree Function Value
0.627
Age of the Person
50
Diabetes Test Result
The Person is diabetic

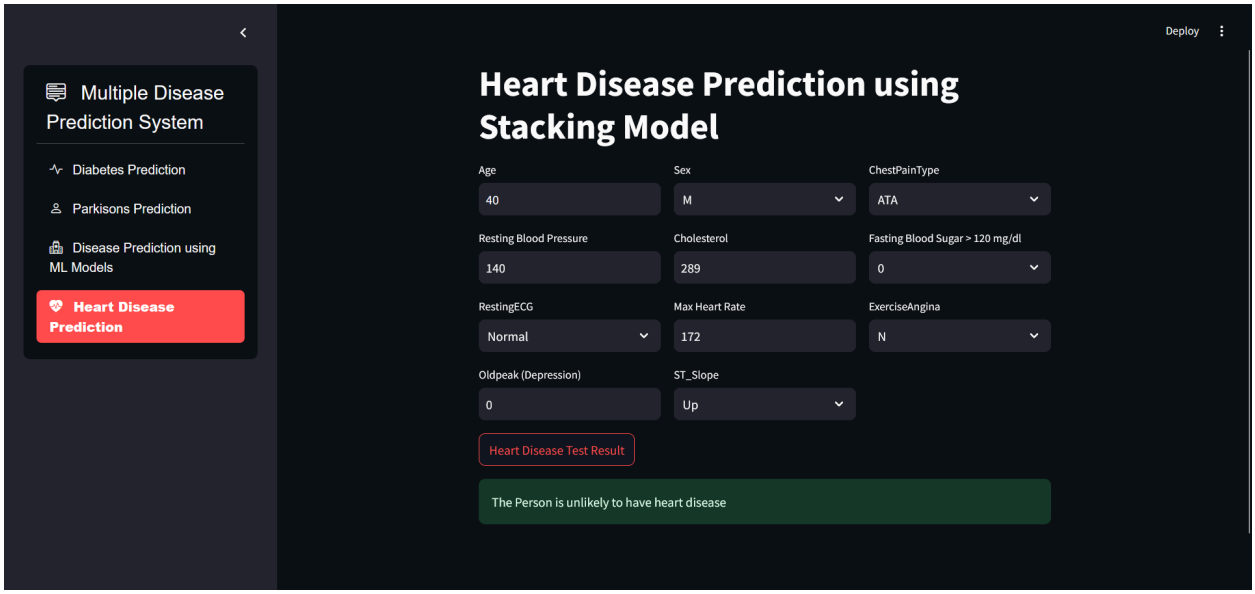

Deploy
Multiple Disease Prediction System
Diabetes Prediction
Parkisons Prediction
Disease Prediction using ML Models
Heart Disease Prediction
Diabetes Prediction using ML
Number of Pregnancies
1
Glucose Level
85
Blood Pressure Value
66
Skin Thickness Value
35
Insulin Level
0
BMI Value
33.6
Diabetes Pedigree Function Value
Age of the Person
Diabetes Test Result
The Person is not diabetic


Multiple Disease Prediction System
Diabetes Prediction
Parkisons Prediction
Disease Prediction using ML Models
Heart Disease Prediction
Deploy
Parkisons Prediction using ML
MDVP:Fo(Hz)
119.992
MDVP:Fhi(Hz)
157.302
MDVP:Flo(Hz)
74.997
MDVP:Jitter(%)
0.00784
MDVP:Jitter(Abs)
0.00007
MDVP:RAP
0.0037
MDVP:PPQ
0.00554
Jitter:DDP
0.01109
MDVP:Shimmer
0.04374
MDVP:Shimmer(dB)
0.426
Shimmer:APQ3
0.02182
Shimmer:APQ5
0.0313
MDVP:APQ
0.02971
Shimmer:DDA
0.06545
NHR
0.02211
HNR
21.033
RPDE
0.414783
DFA
0.815285
spread1
-4.813031
spread2
0.266482
D2
2.301442
PPE
0.284654
Parkinson's Test Result
The Person is having Parkinsons


Multiple Disease Prediction System
Diabetes Prediction
Parkisons Prediction
Disease Prediction using ML Models
Heart Disease Prediction
Deploy
Disease Prediction by Symptoms
Choose symptom 1
abdominal_pain
Choose symptom 2
constipation
Choose symptom 3
mild_fever
Choose symptom 4
blurred_and_distorted_vision
Choose symptom 5
throat_irritation
Prediction 1
Predicted Disease: Jaundice


Deploy
Multiple Disease Prediction System
Diabetes Prediction
Parkisons Prediction
Disease Prediction using ML Models
Heart Disease Prediction
Choose symptom 1
abdominal_pain
Choose symptom 2
constipation
Choose symptom 3
mild_fever
Choose symptom 4
blurred_and_distorted_vision
Choose symptom 5
throat_irritation
Prediction 1
Prediction 2
Predicted Disease: Jaundice
Prediction 3


Deploy
Multiple Disease Prediction System
Diabetes Prediction
Parkisons Prediction
Disease Prediction using ML Models
Heart Disease Prediction
Choose symptom 1
abdominal_pain
Choose symptom 2
constipation
Choose symptom 3
mild_fever
Choose symptom 4
blurred_and_distorted_vision
Choose symptom 5
throat_irritation
Prediction 1
Prediction 2
Prediction 3
Predicted Disease: Typhoid


Multiple Disease Prediction System
Diabetes Prediction
Parkisons Prediction
Disease Prediction using ML Models
Heart Disease Prediction
Deploy
Heart Disease Prediction using Stacking Model
Age
40
Sex
M
ChestPainType
ATA
Resting Blood Pressure
140
Cholesterol
289
Fasting Blood Sugar > 120 mg/dl
0
RestingECG
Normal
Max Heart Rate
172
ExerciseAngina
N
Oldpeak (Depression)
0
ST_Slope
Up
Heart Disease Test Result
The Person is unlikely to have heart disease

Chapter7: Key Features

- **Multiple Disease Support:** The system predicts 43 diseases, each with specific symptoms, covering common infections, chronic illnesses, skin conditions, and complex disorders. It supports a wide range of medical conditions, making it versatile for users seeking health insights. Disease labels are numerically encoded for efficient model training and prediction.
- **Fast Prediction:** Machine learning models like Decision Tree, Random Forest, and Naive Bayes provide real-time prediction capabilities, processing user symptoms in seconds. Efficiency is enhanced by preloading models, making it suitable for time-sensitive environments or limited computational resources.
- **Lightweight and Resource Efficient:** The system is designed for standard local systems, requiring no high-end GPU or cloud computing infrastructure, and uses efficient machine learning models. Its lightweight architecture allows seamless integration with Python-based frameworks like Streamlit for web deployment.
- **Expandable and Modular Design:** The system is modular, allowing developers to add or replace diseases or models as needed. It can incorporate new symptoms, disease classes, and machine learning algorithms with minimal changes, making it future-proof and scalable for evolving medical datasets and prediction needs.

Chapter 8: Limitations

- **Not a Substitute for Professional Medical Advice:** The system's predictions are based on statistical patterns, lacking clinical judgment and addressing real-world medical diagnoses. It should not replace consulting a qualified healthcare provider and serves as a support tool, not a diagnostic authority.
- **Predictions Depend Heavily on Input Quality:** The system's accuracy relies on user-inputted symptoms, making it highly sensitive to human error during data entry, as incorrect inputs can lead to misleading predictions.
- **Lack of Real-Time Learning or Adaptation:** The current implementation employs static, pre-trained models that may not adapt over time, potentially leading to performance degradation or failure to reflect current disease trends without regular retraining.
- **Privacy and Ethical Concerns:** Users may unintentionally share sensitive health information using a tool, raising concerns over data privacy and ethical compliance due to inadequate data governance and encryption.

Chapter 9: Conclusion and Future Scope

Conclusion:

The project showcases a user-friendly, lightweight disease prediction system using machine learning and Python's Streamlit framework. The system uses trained models like Decision Tree, Random Forest, and Naive Bayes to predict over 40 diseases based on user-provided symptoms. The web-based interface ensures accessibility, and the underlying models provide real-time predictions without heavy computational resources. While not a substitute for professional medical advice, it serves as a valuable preliminary diagnostic aid in remote or resource-limited settings. The system's modular architecture allows for easy expansion and adaptability for future healthcare applications.

Future Scope:

- **Cloud Deployment for Broader Accessibility:** Cloud hosting of disease prediction systems like AWS, Azure, or Google Cloud allows global accessibility, eliminates local installations, and offers scalability, load balancing, and easier model updates. It also enables integration with centralized health databases or APIs for real-time patient data access.
- **Integration with IoT Health Monitoring Devices:** The system can be connected to IoT-enabled health devices like smartwatches, pulse oximeters, glucose monitors, or wearable ECG sensors, enabling real-time physiological data analysis, continuous health monitoring, early warning alerts, and improved data accuracy for personalized diagnosis.
- **Expansion to Include Additional Diseases and Advanced Deep Learning Models:** The current system supports around 43 diseases, but can be expanded to include rare or complex diseases. Including more comprehensive medical datasets and using deep learning models like neural networks can improve prediction accuracy. Additionally, ensemble learning and transfer learning can enhance model robustness and generalizability.
- **Enhanced Data Handling for Improved Model Performance:** Improving training data

quality, diversity, and volume is crucial for better model outcomes. Including structured and unstructured data enhances context-awareness. Techniques like data augmentation, synthetic data generation, and anomaly detection strengthen the training process. Implementing better preprocessing pipelines boosts model efficiency and accuracy.

References

https://www.sciencedirect.com/topics/engineering/data-preprocessing

https://docs.streamlit.io/

https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://www.kipi.ai/insights/building-a-streamlit-application-with-custom-components-a-step-by-step-guide/&ved=2ahUKEwjs---JtoKNAxX5yDgGHXdIJ3sQFnoECAUQAw&usg=AOvVaw2RP_mq7MhzcKDOxHqRX3AR

https://docs.python.org/3/library/pickle.html

https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction

https://www.kaggle.com/datasets/vikasukani/parkinsons-disease-data-set

https://www.kaggle.com/datasets/kaushil268/disease-prediction-using-machine-learning

https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

https://scikit-learn.org/

https://archive.ics.uci.edu/