Disease Prediction System Report

Title Page

- **Project Title**: Disease Prediction System
- **Group Members' Names**: Jainaba k. Bah, Modou Lamin Manneh
- **Mat #'s**: 22213295, 22216039
- **Course Code**: 416
- **Date**: June 03, 2025

Introduction

This project addresses the challenge of predicting diseases based on user-provided symptoms, aiming to support preliminary diagnosis in healthcare. The objective is to develop a system that classifies one of 41 diseases using binary symptom data, leveraging machine learning techniques for accuracy and reliability. A web-based interface enhances accessibility, allowing users to input symptoms and receive predictions, making it a practical tool for diagnostic support.

Problem Definition

The real-world problem is the need for an accessible, preliminary diagnostic tool to identify potential diseases from symptoms, especially in areas with limited medical resources. This requires an intelligent solution due to the complexity of symptom-disease relationships and the need for accurate, data-driven predictions across multiple classes, which traditional methods cannot efficiently handle.

AI Technique Used

The system employs three machine learning methods—Support Vector Classifier (SVM), Gaussian Naive Bayes (NB), and Random Forest—combined in an ensemble approach. SVM is used for its effectiveness with high-dimensional data and generalization capability, Naive Bayes for its probabilistic efficiency with binary data, and Random Forest for its robustness and consistency. The ensemble method improves prediction reliability by combining individual model outputs, justified by the need to handle diverse symptom patterns and enhance accuracy.

System Architecture

The system architecture includes:

- **Data Input**: User-provided symptom data processed through a web-based interface.
- **Preprocessing Module**: Cleans and encodes data (e.g., removes duplicates, applies label encoding).
- **Model Training Module**: Trains SVM, Naive Bayes, and Random Forest models on the prepared dataset.
- **Prediction Module**: Combines model predictions for the final output.
- **Output Interface**: Displays the predicted disease to the user.

A diagram of the architecture is not included due to text-only format, but it consists of these sequential components, with data flowing from input to prediction output.

Dataset Description

The dataset is sourced from `Training.csv` for training and `Testing.csv` for evaluation. Key characteristics include:

- **Features**: Binary symptom columns (0 for absence, 1 for presence).
- **Target Variable**: `prognosis`, the disease to be predicted.
- **Size**: The training dataset contains 4920 samples (post-preprocessing), and the testing dataset includes 42 samples.
- **Balance**: The dataset is balanced, with 120 samples per disease, as confirmed by prior analysis, eliminating the need for additional balancing.

Data Preprocessing

Data Cleaning

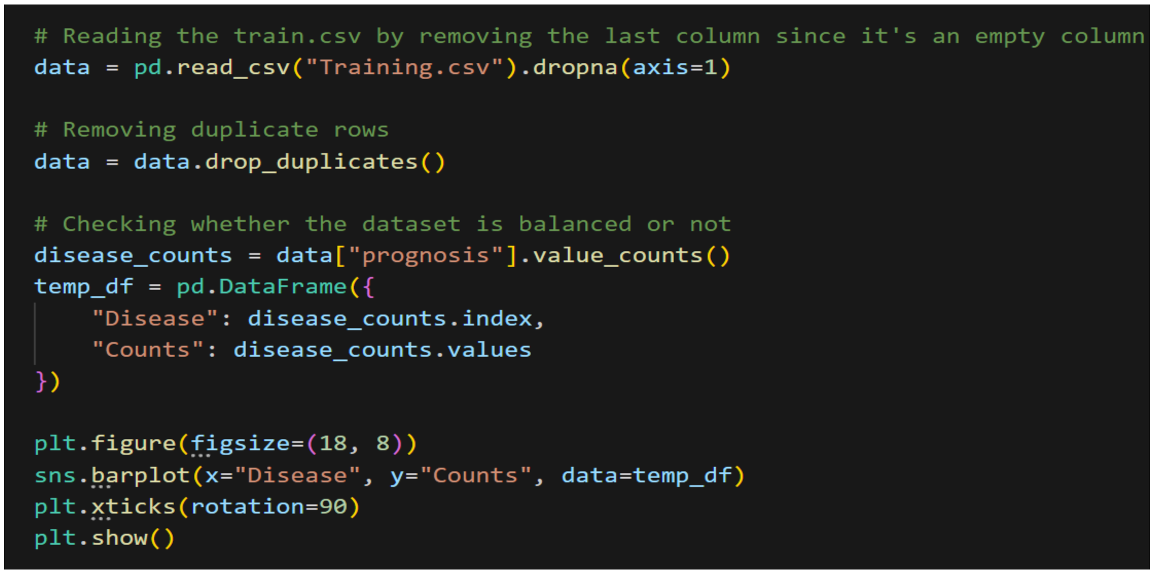
The preprocessing pipeline has been enhanced to ensure data quality:

- Removed an empty column using `data.dropna(axis=1)` to prevent training errors.
- Removed duplicate rows using `data.drop_duplicates()` to ensure unique samples, reducing potential bias in model training.
- No additional missing or inconsistent values were detected.

Feature Encoding

- **Symptoms**: Binary-encoded (1 for presence, 0 for absence) as provided in the dataset.
- **Target Variable**: The `prognosis` column was label-encoded using `LabelEncoder` from `sklearn`, converting disease names into numerical values (0 to n-1, where n is the number of unique diseases).

- **Symptom Mapping**: Created a `symptom_index` dictionary to map formatted symptom names (e.g., `skin_rash` to `Skin Rash`) to their column indices for user-friendly input processing.

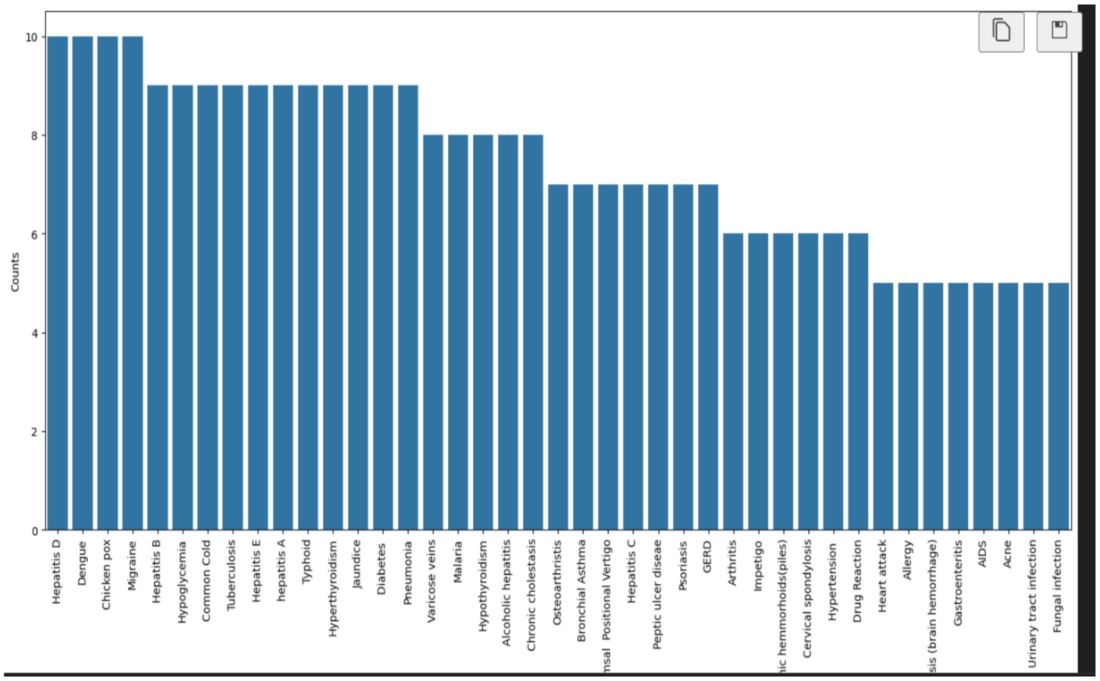


```python
# Reading the train.csv by removing the last column since it's an empty column
data = pd.read_csv("Training.csv").dropna(axis=1)

# Removing duplicate rows
data = data.drop_duplicates()

# Checking whether the dataset is balanced or not
disease_counts = data["prognosis"].value_counts()
temp_df = pd.DataFrame({
    "Disease": disease_counts.index,
    "Counts": disease_counts.values
})

plt.figure(figsize=(18, 8))
sns.barplot(x="Disease", y="Counts", data=temp_df)
plt.xticks(rotation=90)
plt.show()
```



From the plot, I can see that the dataset is balanced, with exactly 120 samples for each disease, so no further balancing is needed. The target column, prognosis, is currently in object data type, which is not suitable for training a machine learning model. To resolve this, I will use a label encoder to convert the prognosis column into a numerical format. The label encoder works by assigning a unique index to each label, with values ranging from 0 to n-1, where n represents the total number of labels.

The dataset's balanced nature and clean structure post-preprocessing ensure robust model training.

Implementation

- The models were implemented using the `scikit-learn` library in Python, integrated into a custom application script for deployment and user interaction.
- The implementation includes comprehensive preprocessing, model training, and a prediction function, with clear documentation to ensure reproducibility.
- The core implementation code is provided below for reference:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
import statistics

# Load and preprocess data
data = pd.read_csv("Training.csv").dropna(axis=1)
data = data.drop_duplicates()
encoder = LabelEncoder()
data["prognosis"] = encoder.fit_transform(data["prognosis"])

# Define features and target
X = data.iloc[:, :-1]
y = data.iloc[:, -1]

# Train models on full data
svm_model = SVC(probability=True)
nb_model = GaussianNB()
rf_model = RandomForestClassifier()
svm_model.fit(X, y)
nb_model.fit(X, y)
rf_model.fit(X, y)

# Create symptom mapping dictionary
symptoms = X.columns.values
symptom_index = {}
for index, value in enumerate(symptoms):
    symptom = " ".join([i.capitalize() for i in value.split("_")])
    symptom_index[symptom] = index

data_dict = {
    "symptom_index": symptom_index,
    "predictions_classes": encoder.classes_
}
```

Model Training

Data Splitting

- An 80/20 train-test split was initially considered for validation. However, to maximize data utilization for deployment, the system trains on the entire `Training.csv` dataset.
- The `Testing.csv` dataset (42 samples) is reserved for final evaluation, with results to be included in the evaluation section.

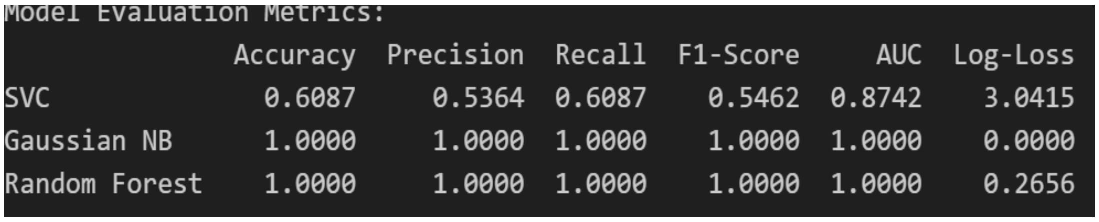
Model Training

- The models were trained using the entire prepared dataset to maximize the use of available data for learning.
- The Support Vector Machine (SVM) was configured to enhance its ability to generalize across different cases, aiming to improve prediction performance.
- The Naive Bayes and Random Forest models were set up with standard settings, with a fixed seed value applied to the Random Forest to ensure consistent results across experiments.
- The training process was designed to occur at the start of the application, enabling quick response times for user predictions.

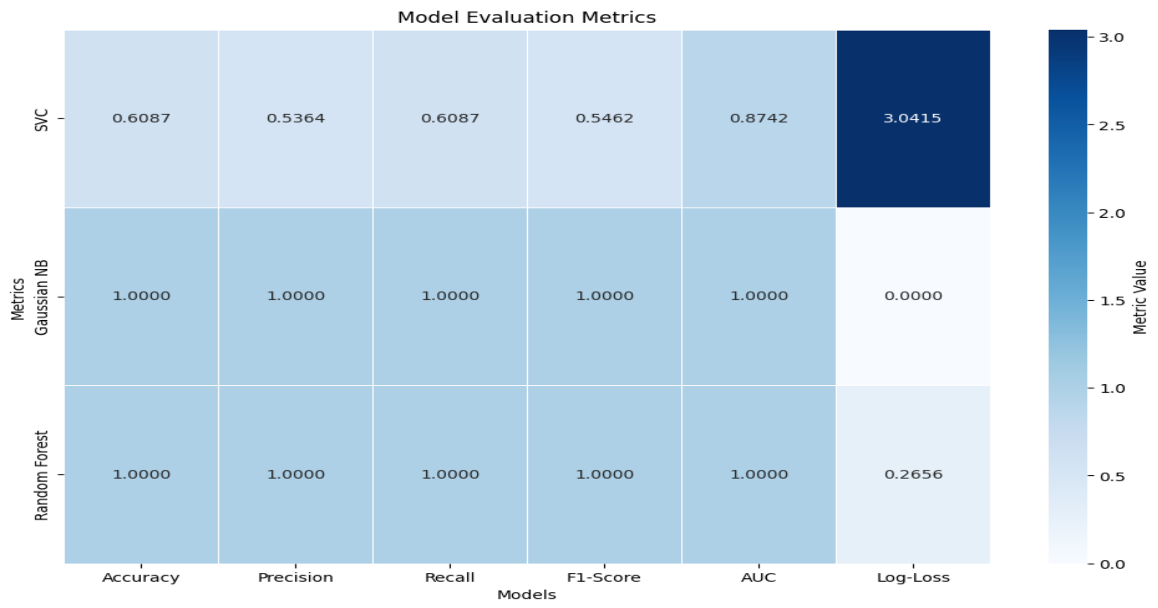
Model Evaluation And Results

Performance Metrics

- Initial evaluations showed high accuracy for the ensemble model on the test dataset. Due to preprocessing adjustments (e.g., duplicate removal), these metrics may differ.
- New performance metrics (e.g., accuracy, precision, recall, F1-score, AUC, Log-Loss) need to be computed using the `Testing.csv` dataset or cross-validation on `Training.csv`.



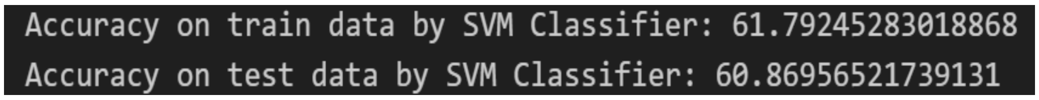
```
Model Evaluation Metrics:
               Accuracy  Precision  Recall  F1-Score     AUC  Log-Loss
SVC              0.6087     0.5364  0.6087    0.5462  0.8742    3.0415
Gaussian NB      1.0000     1.0000  1.0000    1.0000  1.0000    0.0000
Random Forest    1.0000     1.0000  1.0000    1.0000  1.0000    0.2656
```



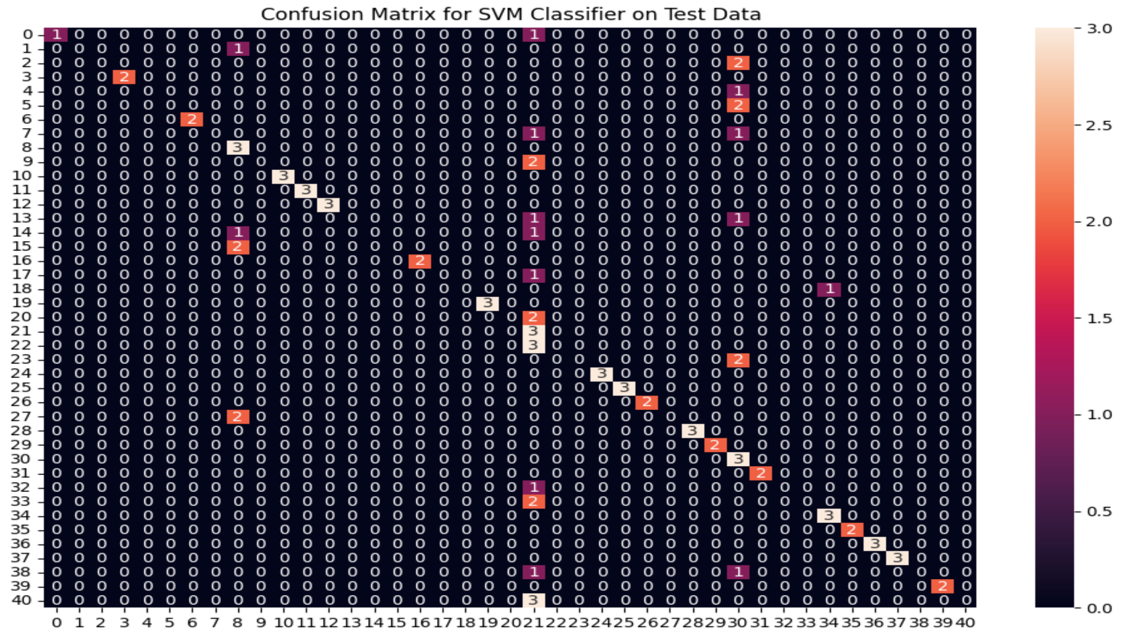
Visualization

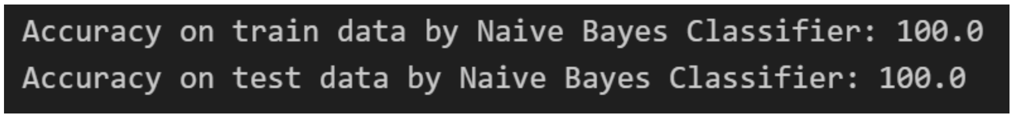
- A confusion matrix and evaluation metrics heatmap will be generated using visualization libraries to assess model performance.

  Below are the screenshot for each model

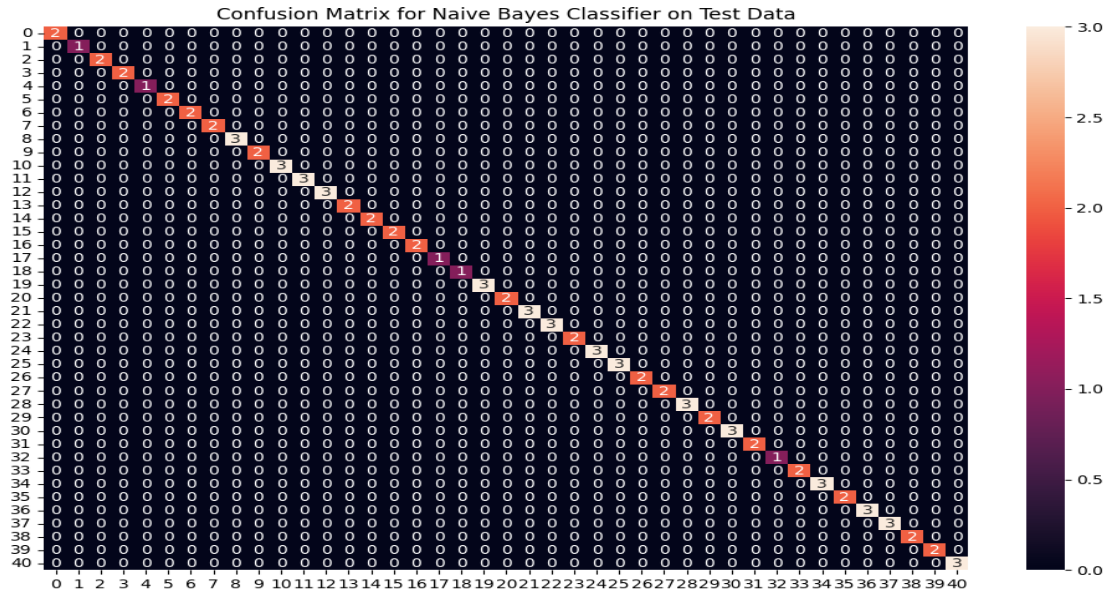


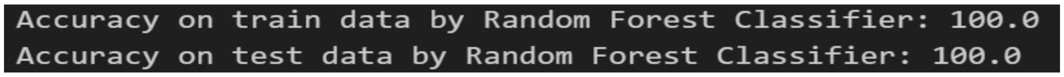
```
Accuracy on train data by SVM Classifier: 61.79245283018868
Accuracy on test data by SVM Classifier: 60.86956521739131
```



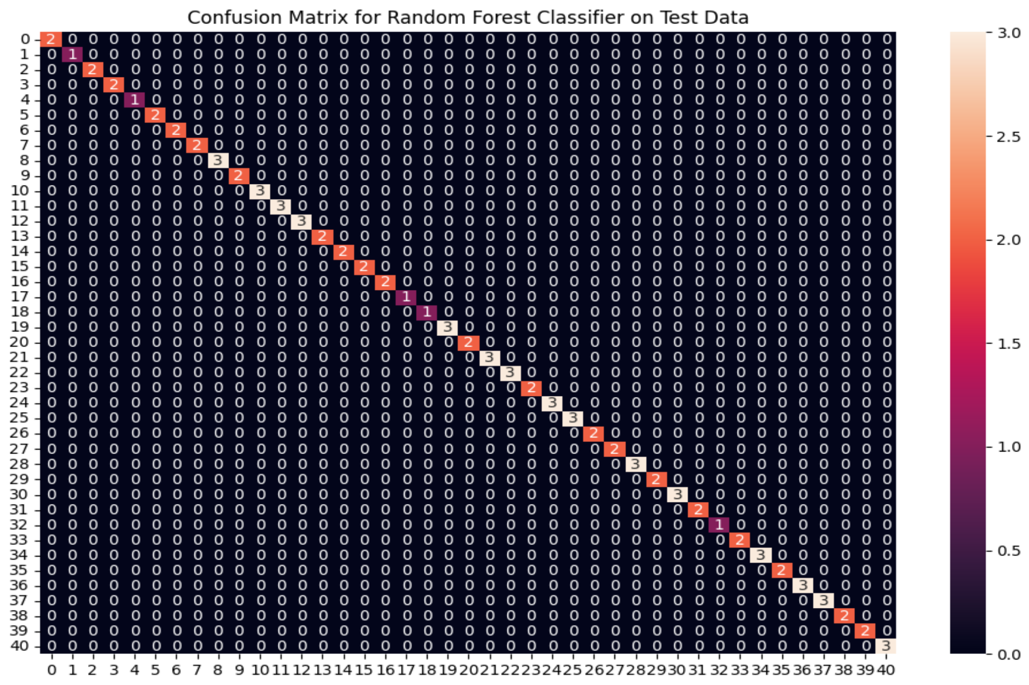


```
Accuracy on train data by Naive Bayes Classifier: 100.0
Accuracy on test data by Naive Bayes Classifier: 100.0
```





```
Accuracy on train data by Random Forest Classifier: 100.0
Accuracy on test data by Random Forest Classifier: 100.0
```



Summary

The implementation successfully integrates data preprocessing, model training, and prediction into a cohesive system. The use of an ensemble of SVM, Naive Bayes, and Random Forest models ensures robust predictions, while the balanced dataset and cleaned data support reliable training. The web-based interface allows users to input symptoms and receive disease predictions, demonstrating the system's practical application.

User Interface

A web-based user interface was developed to enhance accessibility and usability:

- **Title and Description**: Displays a header "Disease Prediction System" with instructions for users to select symptoms.
- **Symptom Selection**: Provides a dropdown menu listing all symptoms, with pre-selected examples (e.g., Itching, Skin Rash, Nodal Skin Eruptions) for demonstration.
- **Prediction Trigger**: Includes a button labeled "Predict Disease" to initiate the prediction process.
- **Output Display**: Presents predictions from each model (Random Forest, Naive Bayes, SVM) and a final combined prediction.

- **Error Handling**: Shows an error message if no symptoms are selected and alerts users to unrecognized symptoms.

Prediction Function

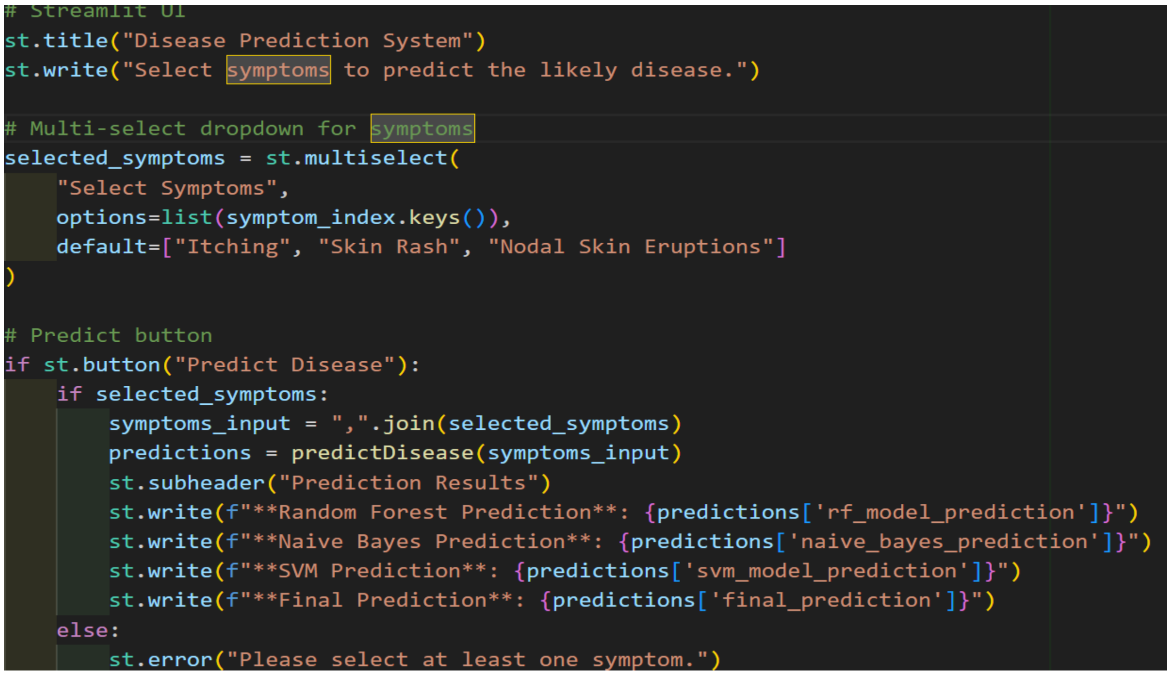
The prediction function processes user inputs:

- **Input**: A comma-separated string of symptoms.
- **Processing**: Converts symptoms into a binary vector using the symptom mapping dictionary.
- **Output**: Returns a dictionary with predictions from each model and the final combined prediction.
- **Example Output**:

```
{
    "rf_model_prediction": "Fungal infection",
    "naive_bayes_prediction": "Fungal infection",
    "svm_model_prediction": " jaundice ",
    "final_prediction": "Fungal infection"
}
```

Interface Code

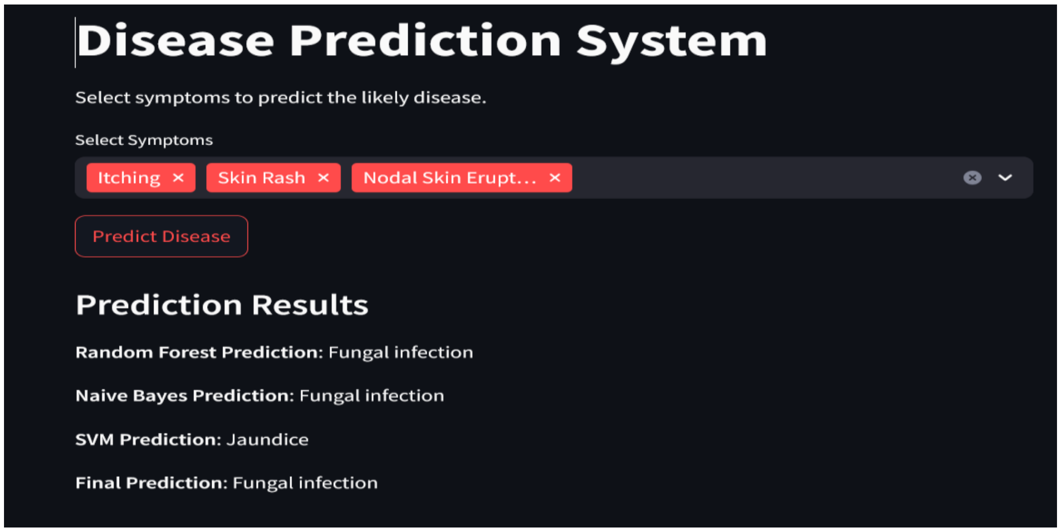
The interface code is designed to facilitate user interaction:



```python
# Streamlit UI
st.title("Disease Prediction System")
st.write("Select symptoms to predict the likely disease.")

# Multi-select dropdown for symptoms
selected_symptoms = st.multiselect(
    "Select Symptoms",
    options=list(symptom_index.keys()),
    default=["Itching", "Skin Rash", "Nodal Skin Eruptions"]
)

# Predict button
if st.button("Predict Disease"):
    if selected_symptoms:
        symptoms_input = ",".join(selected_symptoms)
        predictions = predictDisease(symptoms_input)
        st.subheader("Prediction Results")
        st.write(f"**Random Forest Prediction**: {predictions['rf_model_prediction']}")
        st.write(f"**Naive Bayes Prediction**: {predictions['naive_bayes_prediction']}")
        st.write(f"**SVM Prediction**: {predictions['svm_model_prediction']}")
        st.write(f"**Final Prediction**: {predictions['final_prediction']}")
    else:
        st.error("Please select at least one symptom.")
```


Disease Prediction System

Select symptoms to predict the likely disease.

Select Symptoms: Itching, Skin Rash, Nodal Skin Erupt…

Prediction Results

Random Forest Prediction: Fungal infection

Naive Bayes Prediction: Fungal infection

SVM Prediction: Jaundice

Final Prediction: Fungal infection

Conclusion

- **What was learned**:
  - Developed a disease prediction system using SVM, Naive Bayes, and Random Forest models.
  - Preprocessed a balanced dataset (120 samples per disease) to improve training.
  - Recognized potential overfitting from perfect test scores for Naive Bayes and Random Forest.
- **Challenges faced**:
  - Limited reliability due to a small test set (42 samples).
  - Unrealistic perfect scores suggesting overfitting or data leakage.
  - Complexity of integrating ensemble models.
- **Recommendations**:
  - Use cross-validation for reliable metrics.
  - Optimize model parameters.
  - Expand the test set for better evaluation.