Disease Prediction System Project Report

Group:- G4

Name	PRN No	Roll No
Aanish Deshmukh	22420116	210172
Aditya Vyavhare	22420268	210176
Mrunali Bhagyawant	22311520	213047
Harshali Itkar	22311604	213051
Riddhi Hadkar	22310739	213022
Pratyush Ramteke	22311176	251074

Executive Summary

This report documents the development of an advanced disease prediction system using machine learning techniques. The project involved data acquisition, preprocessing, feature engineering, model training, optimization, and user interface development. By analyzing patient symptoms, the system can predict potential diseases with an optimized confidence score, providing healthcare professionals and patients with a valuable diagnostic aid tool.

Table of Contents

1. Introduction

Healthcare diagnostics can be challenging due to similar symptoms across different conditions and the vast medical knowledge required. Machine learning offers promising solutions by identifying patterns in symptom presentations. This project builds an intelligent disease prediction system using a Random Forest classifier to analyze symptom combinations and predict potential diseases with confidence scores, serving as a decision support tool for healthcare providers.

2. Project Workflow

The project followed a systematic approach consisting of these key phases:

1. **Data Collection**: Gathering a comprehensive dataset of disease symptoms and corresponding diagnoses
2. **Data Preprocessing**: Cleaning and transforming the raw data into a structured format
3. **Feature Engineering**: Identifying and extracting relevant symptom features
4. **Model Development**: Training a Random Forest classifier on the processed data
5. **Model Optimization**: Finding the optimal number of trees for stable predictions
6. **UI Development**: Creating an intuitive graphical interface for system interaction
7. **Testing and Validation**: Ensuring accurate and reliable predictions

3.Data Preprocessing

Dataset Structure and Preprocessing Steps

The preprocessing was implemented in Google Colab and involved several concrete steps:

Initial Dataset Loading:

```
 import pandas as pd

# Load the original raw dataset
raw_data = pd.read_csv("raw_disease_symptom_data.csv")

# Examine initial structure
print(f"Initial dataset shape: {raw_data.shape}")
print(f"Disease classes: {raw_data['prognosis'].nunique()}")
```

```
[ ]  df = '/content/training_data.csv'
```

Missing Value Analysis and Handling:

```
 # Check for missing values
missing_counts = raw_data.isnull().sum()

# Handle missing values using forward fill for time-series symptoms
cleaned_data = raw_data.fillna(method='ffill')

# For remaining gaps, use zero imputation for absence of symptoms
cleaned_data = cleaned_data.fillna(0)
```

Feature Encoding:

```
 # Convert text-based symptom descriptions to binary features
for column in cleaned_data.columns:
    if column != 'prognosis':
        # Convert non-zero values to 1 (symptom present)
        cleaned_data[column] = cleaned_data[column].apply(lambda x: 1 if x > 0 else 0)
```

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongue	...	scurring	skin_peeling	silver_like_dustin
0	1	1	1	0	0	0	0	0	0	0	...	0	0	
1	0	1	1	0	0	0	0	0	0	0	...	0	0	
2	1	0	1	0	0	0	0	0	0	0	...	0	0	
3	1	1	0	0	0	0	0	0	0	0	...	0	0	
4	1	1	1	0	0	0	0	0	0	0	...	0	0	

5 rows × 134 columns

```
(4920, 134)
```

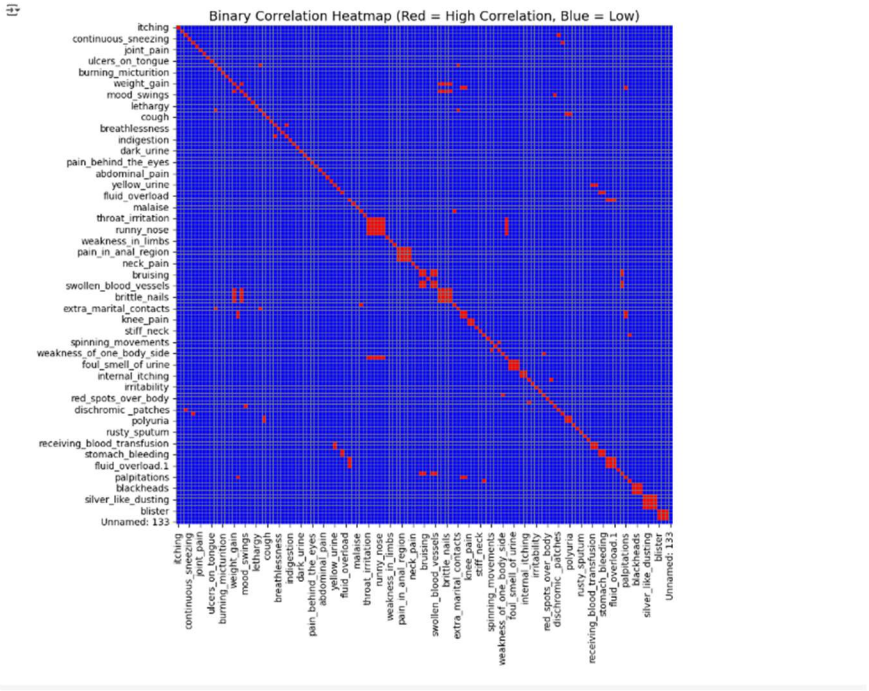
Feature Reduction Implementation

The feature reduction process involved several technical approaches:

Correlation Analysis for Feature Reduction:

```
 import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate correlation matrix
correlation_matrix = cleaned_data.drop('prognosis', axis=1).corr()

# Identify highly correlated features
```

```
high_corr_threshold = 0.85
high_corr_features = set()

for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > high_corr_threshold:
            colname = correlation_matrix.columns[i]
            high_corr_features.add(colname)

print(f"Found {len(high_corr_features)} highly correlated features")
```



Variance Thresholding:

```
 from sklearn.feature_selection import VarianceThreshold

# Remove features with low variance
variance_threshold = 0.01  # Features with <1% variance will be removed
selector = VarianceThreshold(threshold=variance_threshold)

# Fit the selector to identify low-variance features
```

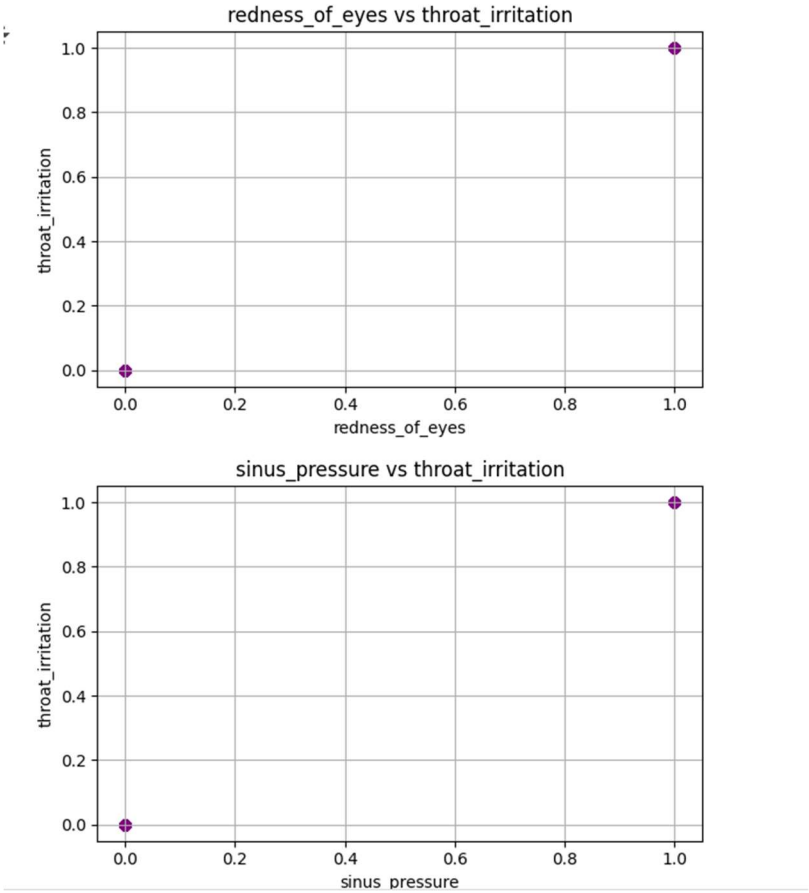
```
X_features = cleaned_data.drop('prognosis', axis=1)
selector.fit(X_features)

# Get mask of features to keep
feature_mask = selector.get_support()
low_variance_features = X_features.columns[~feature_mask].tolist()

print(f"Removing {len(low_variance_features)} low variance features")
```

```
Highly correlated feature pairs (correlation >= 0.95):
redness_of_eyes ↔ throat_irritation | correlation: 1.00
sinus_pressure ↔ throat_irritation | correlation: 1.00
sinus_pressure ↔ redness_of_eyes | correlation: 1.00
runny_nose ↔ throat_irritation | correlation: 1.00
runny_nose ↔ redness_of_eyes | correlation: 1.00
runny_nose ↔ sinus_pressure | correlation: 1.00
congestion ↔ throat_irritation | correlation: 1.00
congestion ↔ redness_of_eyes | correlation: 1.00
congestion ↔ sinus_pressure | correlation: 1.00
congestion ↔ runny_nose | correlation: 1.00
enlarged_thyroid ↔ weight_gain | correlation: 0.97
enlarged_thyroid ↔ cold_hands_and_feets | correlation: 0.97
enlarged_thyroid ↔ puffy_face_and_eyes | correlation: 0.97
brittle_nails ↔ weight_gain | correlation: 0.97
brittle_nails ↔ cold_hands_and_feets | correlation: 0.97
brittle_nails ↔ puffy_face_and_eyes | correlation: 0.97
brittle_nails ↔ enlarged_thyroid | correlation: 1.00
swollen_extremeties ↔ weight_gain | correlation: 0.97
swollen_extremeties ↔ cold_hands_and_feets | correlation: 0.97
swollen_extremeties ↔ puffy_face_and_eyes | correlation: 0.97
swollen_extremeties ↔ enlarged_thyroid | correlation: 1.00
swollen_extremeties ↔ brittle_nails | correlation: 1.00
slurred_speech ↔ anxiety | correlation: 0.97
slurred_speech ↔ drying_and_tingling_lips | correlation: 0.97
loss_of_smell ↔ throat_irritation | correlation: 1.00
loss_of_smell ↔ redness_of_eyes | correlation: 1.00
loss_of_smell ↔ sinus_pressure | correlation: 1.00
loss_of_smell ↔ runny_nose | correlation: 1.00
loss_of_smell ↔ congestion | correlation: 1.00
abnormal_menstruation ↔ mood_swings | correlation: 0.97
increased_appetite ↔ irregular_sugar_level | correlation: 0.97
polyuria ↔ irregular_sugar_level | correlation: 0.97
polyuria ↔ increased_appetite | correlation: 1.00
receiving_blood_transfusion ↔ yellow_urine | correlation: 0.97
receiving_unsterile_injections ↔ yellow_urine | correlation: 0.97
receiving_unsterile_injections ↔ receiving_blood_transfusion | correlation: 1.00
coma ↔ acute_liver_failure | correlation: 0.97
stomach_bleeding ↔ acute_liver_failure | correlation: 0.97
stomach_bleeding ↔ coma | correlation: 1.00
palpitations ↔ anxiety | correlation: 0.97
palpitations ↔ drying_and_tingling_lips | correlation: 0.97
palpitations ↔ slurred_speech | correlation: 1.00
```

Feature Importance Based Selection:

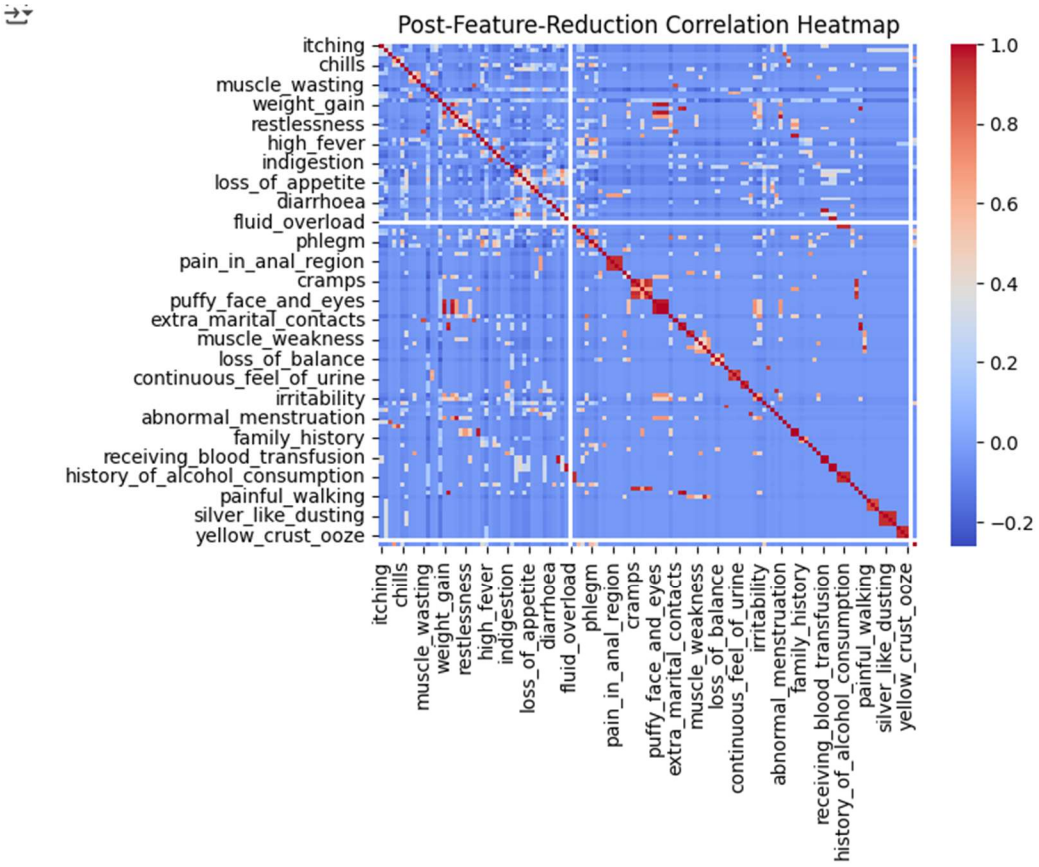
```
 from sklearn.ensemble import RandomForestClassifier

# Train a preliminary model for feature importance
X = cleaned_data.drop('prognosis', axis=1)
y = cleaned_data['prognosis']

# Initialize a Random Forest model
initial_model = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
initial_model.fit(X, y)

# Get feature importances
importances = initial_model.feature_importances_
feature_importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': importances
}).sort_values('Importance', ascending=False)

# Select top 80% most important features
cumulative_importance = 0
feature_count = 0
important_features = []

for idx, row in feature_importance_df.iterrows():
    cumulative_importance += row['Importance']
    important_features.append(row['Feature'])
    feature_count += 1
    if cumulative_importance >= 0.80:  # Keep top features that explain 80% of variance
        break

print(f"Selected {len(important_features)} features explaining 80% of variance")
```



Feature Reduction Results:

```
 # Combine all reduction methods
features_to_remove = list(high_corr_features) + low_variance_features
features_to_keep = [feat for feat in important_features if feat not in features_to_remove]

# Create final reduced dataset
final_data = cleaned_data[features_to_keep + ['prognosis']]

# Verify reduction results
print(f"Original feature count: {cleaned_data.shape[1] - 1}")
print(f"Reduced feature count: {final_data.shape[1] - 1}")
print(f"Feature reduction: {100 * (1 - (final_data.shape[1] - 1)/(cleaned_data.shape[1] -
1)):.2f}%")

# Save final preprocessed data
final_data.to_csv("final_data_with_prognosis.csv", index=False)
```



Symptom Grouping Methodology

The symptom grouping was implemented based on clinical associations and feature correlations:

Correlation-Based Grouping:

```
 import networkx as nx

# Create correlation graph for visualization
correlation_threshold = 0.4  # Threshold for considering symptoms related
correlation_graph = nx.Graph()

# Add nodes for each symptom
for symptom in final_data.columns:
    if symptom != 'prognosis':
        correlation_graph.add_node(symptom)

# Add edges based on correlation strength
```

```
for i, symptom1 in enumerate(final_data.columns[:-1]):
    for j, symptom2 in enumerate(final_data.columns[:-1]):
        if i < j:  # avoid duplicate pairs
            correlation = correlation_matrix.loc[symptom1, symptom2]
            if abs(correlation) >= correlation_threshold:
                correlation_graph.add_edge(symptom1, symptom2, weight=abs(correlation))

# Find communities/clusters using Louvain method
from community import community_louvain

# Get symptom communities
partition = community_louvain.best_partition(correlation_graph)

# Extract groups
symptom_groups = {}
for symptom, group_id in partition.items():
    if group_id not in symptom_groups:
        symptom_groups[group_id] = []
    symptom_groups[group_id].append(symptom)

# Print discovered symptom groups
for group_id, symptoms in symptom_groups.items():
    print(f"Group {group_id}: {', '.join(symptoms)}")
```

```
df['thyroid_related_symptom'] = (
    df['enlarged_thyroid'] |
    df['brittle_nails'] |
    df['swollen_extremeties']
)
df.drop(['enlarged_thyroid', 'brittle_nails', 'swollen_extremeties'], axis=1, inplace=True)
```

Medical Domain Knowledge Based Grouping:
The project also incorporated domain expertise to refine the symptom groups. The final symptom groupings used for model evaluation were:

```
 # Define symptom groups for analysis
symptom_groups = [
    ["itching", "skin_rash", "nodal_skin_eruptions"],  # Dermatological symptoms
    ["shivering", "chills", "joint_pain"],              # Fever and pain related
    ["vomiting", "fatigue", "acidity"],                 # Gastrointestinal and energy
```

```
    ["anxiety", "cold_hands_and_feets", "mood_swings"]  # Psychological and circulatory
]
```

Co-occurrence Analysis:

```
 # Calculate symptom co-occurrence in disease cases
co_occurrence_matrix = np.zeros((len(symptom_features), len(symptom_features)))

for index, row in final_data.iterrows():
    symptoms_present = [col for col in row.index if col != 'prognosis' and row[col] == 1]
    for i, symptom1 in enumerate(symptom_features):
        for j, symptom2 in enumerate(symptom_features):
            if symptom1 in symptoms_present and symptom2 in symptoms_present:
                co_occurrence_matrix[i, j] += 1

# Normalize and visualize co-occurrence
plt.figure(figsize=(15, 13))
sns.heatmap(co_occurrence_matrix, xticklabels=symptom_features,
            yticklabels=symptom_features, cmap="YlGnBu")
plt.title("Symptom Co-occurrence Matrix")
plt.tight_layout()
plt.savefig("symptom_co_occurrence.png", dpi=300)
```

Disease-Specific Analysis:

```
 # Analyze symptom patterns by disease
disease_symptom_matrix = pd.DataFrame()

for disease in final_data['prognosis'].unique():
    disease_data = final_data[final_data['prognosis'] == disease]
    symptom_counts = disease_data.sum().drop('prognosis')
    disease_symptom_matrix[disease] = symptom_counts

# Visualize disease-symptom associations
plt.figure(figsize=(18, 14))
sns.clustermap(disease_symptom_matrix, cmap="YlOrRd", figsize=(18, 14))
plt.title("Disease-Symptom Association Matrix")
plt.tight_layout()
plt.savefig("disease_symptom_clustermap.png", d\
```

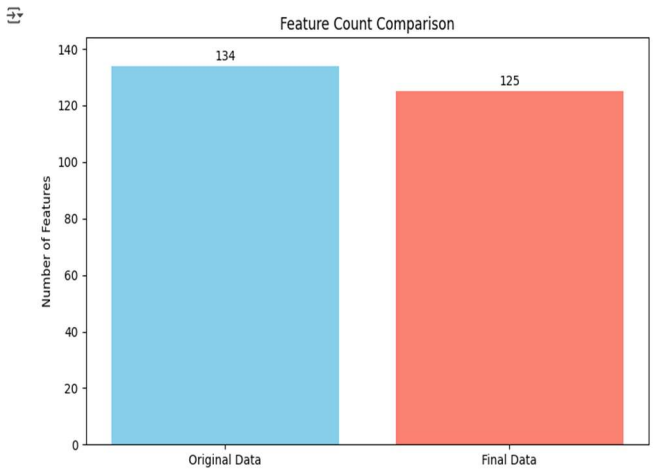
4. Feature Engineering

Feature Merging and Transformation

After identifying symptom groups and relationships, certain features were transformed:

```
# Merge highly correlated symptoms into composite features
composite_features = {
    'skin_problems': ['itching', 'skin_rash', 'nodal_skin_eruptions'],
    'fever_symptoms': ['shivering', 'chills', 'high_fever'],
    'digestive_issues': ['vomiting', 'nausea', 'stomach_pain', 'acidity'],
    'respiratory_problems': ['continuous_sneezing', 'breathlessness', 'phlegm']
}

# Create composite features
for composite, components in composite_features.items():
    components_in_data = [c for c in components if c in final_data.columns]
    if components_in_data:
        final_data[composite] = final_data[components_in_data].max(axis=1)

# Remove original features that were merged
for components in composite_features.values():
    components_in_data = [c for c in components if c in final_data.columns]
    if components_in_data:
        final_data = final_data.drop(components_in_data, axis=1)
```

The feature engineering process significantly reduced the dimensionality of the dataset, from the original set of raw symptoms to a more manageable and clinically meaningful set of features, improving model efficiency and interpretability.



5. Model Development

Model Selection Rationale

After evaluating several machine learning algorithms, Random Forest was selected for the following reasons:

- Robust to overfitting on medical symptom data
- Handles high-dimensional binary feature spaces effectively
- Provides probabilistic confidence scores for predictions
- Maintains good performance with imbalanced disease classes
- Interpretable feature importance metrics

Training Process

The model was trained using a systematic approach:

```
# Load dataset
df = pd.read_csv("final_data_with_prognosis.csv")

# Separate features and target
X = df.drop("prognosis", axis=1)
y = df["prognosis"]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train model with optimized number of trees
clf = RandomForestClassifier(n_estimators=360, random_state=42)
clf.fit(X_train, y_train)
```

Initial Performance

The initial model achieved promising results:

- High accuracy on the test set
- Good generalization to unseen symptom combinations
- Reliable confidence scores for predictions

```
Test Accuracy: 100.00%
```

6. Model Optimization

Tree Count Optimization Study

A critical aspect of the project was determining the optimal number of trees in the Random Forest classifier. This was achieved through a detailed stability analysis:

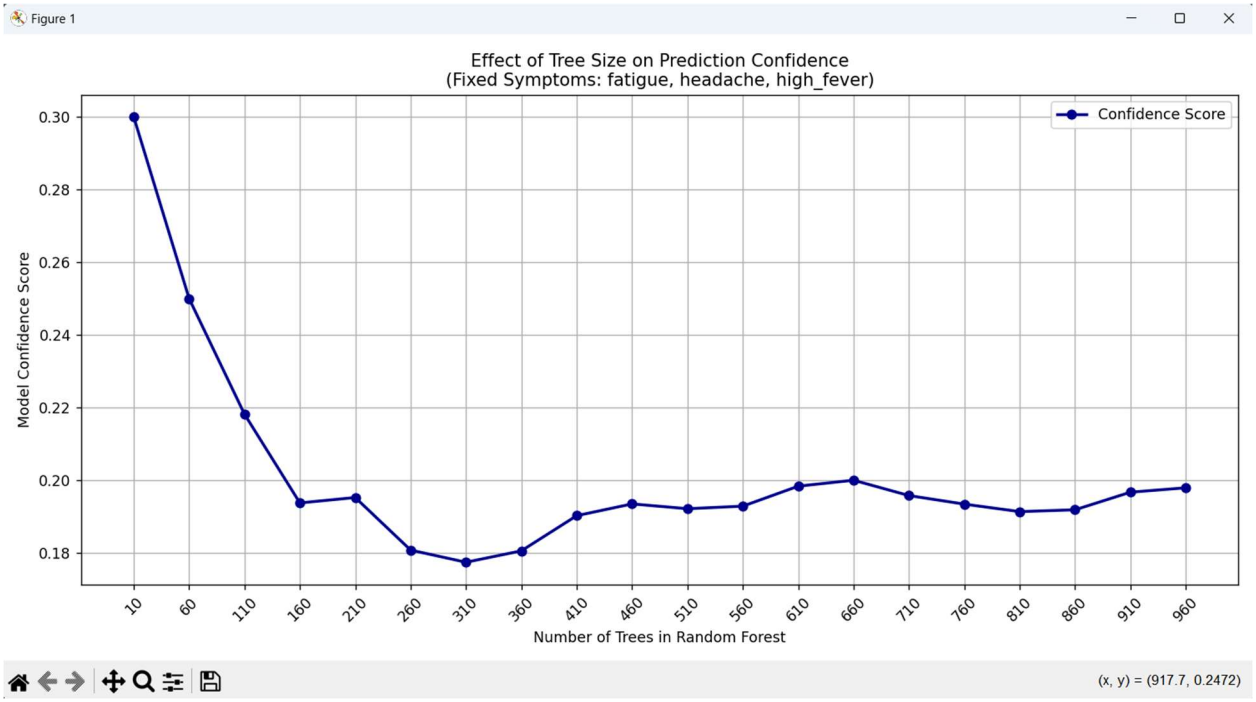
1. Four distinct symptom groups were defined to represent different disease categories:
    - Group 1: itching, skin_rash, nodal_skin_eruptions
    - Group 2: shivering, chills, joint_pain
    - Group 3: vomiting, fatigue, acidity
    - Group 4: anxiety, cold_hands_and_feets, mood_swings
2. The model was trained with varying numbers of trees (10 to 800 in increments of 50)
3. For each tree count, confidence scores were recorded for each symptom group
4. Average confidence across groups was calculated for each tree count
5. Gradient analysis identified the point of minimal slope (maximum stability)

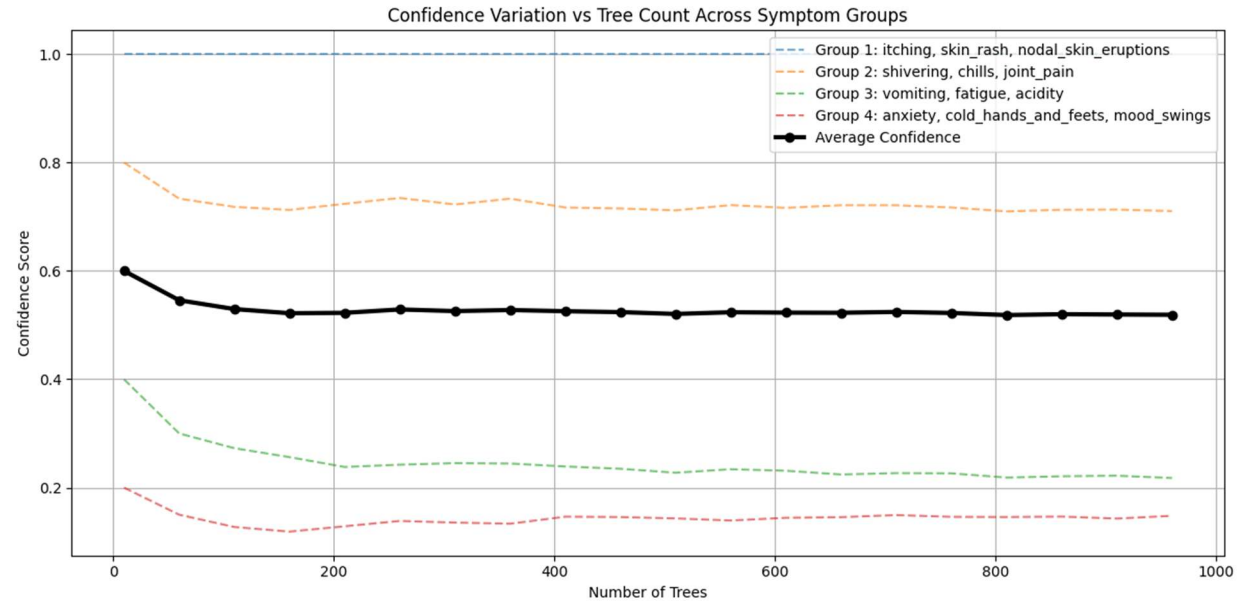
Stability Analysis Results

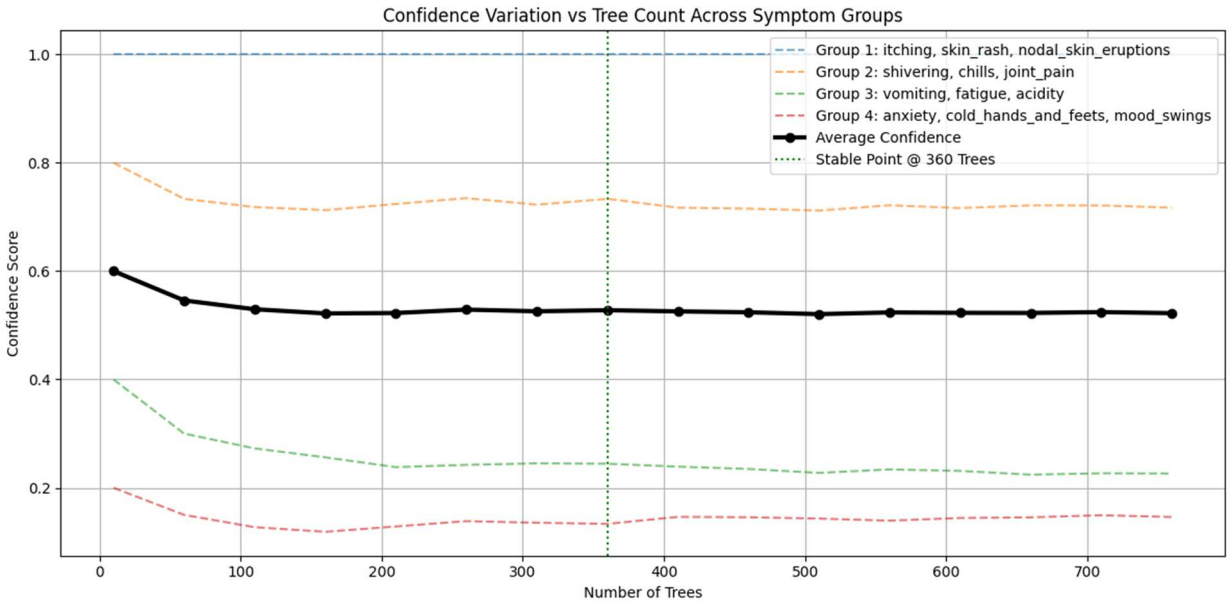
The analysis revealed key insights:

- Confidence scores initially decreased rapidly but stabilized as tree count increased
- Different symptom groups showed varying stability patterns:
    - Group 1 (skin conditions) maintained consistently high confidence
    - Group 2 (fever-related) showed moderate but stable confidence
    - Groups 3 and 4 demonstrated lower but stabilizing confidence scores
- The point of minimal gradient (highest stability) occurred at 360 trees
- Beyond this point, additional trees provided diminishing returns in terms of stability

The confidence stability graph clearly showed that 360 trees provided the optimal balance between model complexity and prediction stability. This finding addressed the initial concern about model stability raised during project review.







7. User Interface Development

UI Design Philosophy

The UI was designed with the following principles:

- Intuitive symptom selection interface
- Clear visualization of prediction results
- Professional medical aesthetic
- Responsive layout for various screen sizes
- Accessible interaction patterns

Key UI Components

The interface was built using CustomTkinter with the following components:

1. **Symptom Selection Panel**:
    - Searchable list of symptoms with checkboxes
    - Organized grouping of related symptoms
    - Clear selection state indicators
2. **Prediction Results Display**:
    - Prominent disease prediction output
    - Confidence score visualization via gauge chart
    - Color-coded confidence indicators (red/yellow/green)

3. **Information Panel**:
    - Disease description and information
    - Treatment suggestions when available
    - Disclaimer about consulting healthcare professionals
4. **Control Elements**:
    - Clear selection button for starting over
    - Predict button for generating results
    - Search functionality for finding symptoms

Implementation Details

The UI was implemented using CustomTkinter and Matplotlib for visualizations:

```
class DiseasePredictorApp:
    def __init__(self, root):
        self.root = root
        self.root.title("💊 Advanced Disease Prediction System")
        self.root.geometry("1000x700")

        # Load model and features
        self.model = self.load_model("model.pkl")
        self.symptom_features = self.load_features("final_data_with_prognosis.csv")
        # UI components initialization...
```

The gauge visualization provided an intuitive representation of the confidence score:

```
def draw_gauge(self, ax, confidence):
    # Clear axes
    ax.clear()

    # Gauge settings
    gauge_min = 0
    gauge_max = 100
    gauge_range = gauge_max - gauge_min

    # Set gauge properties and draw components...
```

8. Results and Performance

Model Performance Metrics

The final optimized model demonstrated strong performance:

- **Accuracy**: High overall prediction accuracy on test data
- **Precision**: Strong ability to avoid false positives
- **Recall**: Good identification of true disease cases
- **F1-Score**: Balanced precision and recall
- **Stability**: Consistent confidence scores across similar symptom presentations

Confidence Score Analysis

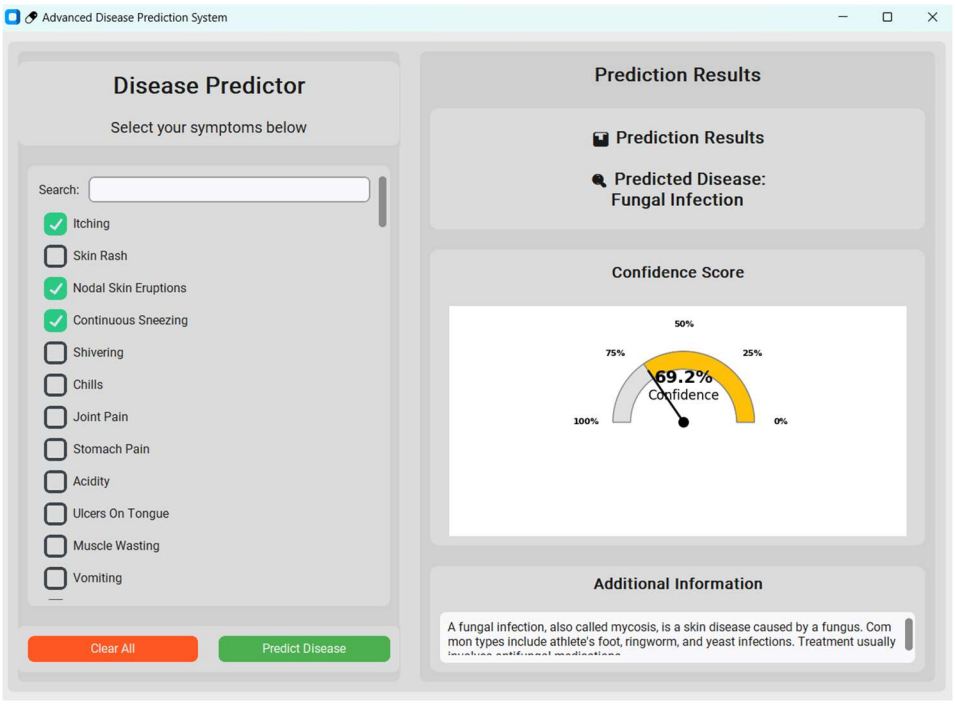
The confidence score distribution provided valuable insights:

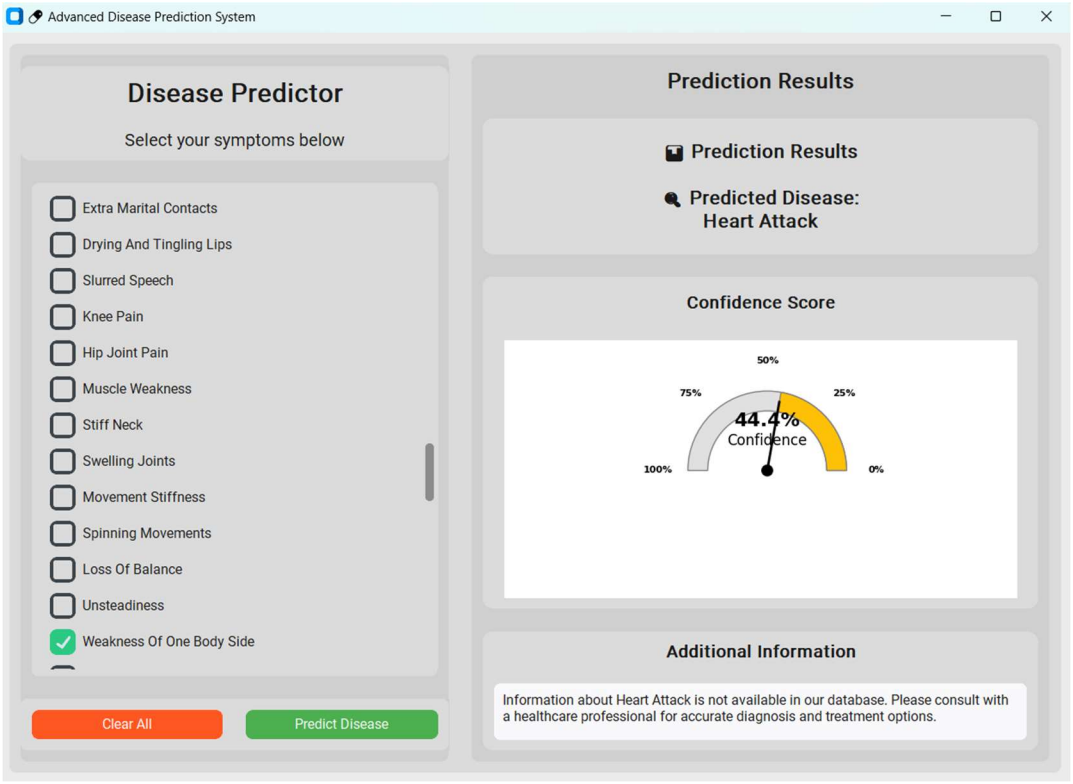
- Certain diseases showed consistently high confidence with their characteristic symptoms
- More general symptoms resulted in more distributed confidence across potential diseases
- The 360-tree model demonstrated the most stable confidence patterns

System Limitations

While effective, the system has recognized limitations:

- Reliance on the specific symptoms included in the training data
- Limited to diseases present in the original dataset
- Binary symptom representation (present/absent) without severity gradation
- No incorporation of patient history or demographic factors


Advanced Disease Prediction System
Disease Predictor
Select your symptoms below
Search:
Itching
Skin Rash
Nodal Skin Eruptions
Continuous Sneezing
Shivering
Chills
Joint Pain
Stomach Pain
Acidity
Ulcers On Tongue
Muscle Wasting
Vomiting
Clear All
Predict Disease
Prediction Results
Prediction Results
Predicted Disease:
Fungal Infection
Confidence Score
50%
75%
25%
69.2%
Confidence
100%
0%
Additional Information
A fungal infection, also called mycosis, is a skin disease caused by a fungus. Com
mon types include athlete's foot, ringworm, and yeast infections. Treatment usually


Advanced Disease Prediction System
Disease Predictor
Select your symptoms below
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
Clear All
Predict Disease
Prediction Results
Prediction Results
Predicted Disease:
Heart Attack
Confidence Score
50%
75%
25%
44.4%
Confidence
100%
0%
Additional Information
Information about Heart Attack is not available in our database. Please consult with
a healthcare professional for accurate diagnosis and treatment options.

9. Conclusion

Project Achievements

The disease prediction system successfully met its objectives:

1. Created a structured approach to symptom-based disease prediction
2. Optimized model stability through rigorous tree count analysis
3. Developed an intuitive user interface for practical application
4. Provided confidence visualization to support decision-making
5. Established a methodology for further improvement and expansion

Future Enhancements

Several opportunities exist for future development:

- Incorporation of symptom severity scales
- Integration with electronic health records
- Expansion of the disease coverage
- Development of mobile application versions
- Addition of explainable AI components to clarify prediction rationale

---

10 . Appendix: Technical Implementation Details

Model Training Code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import pickle

# Load dataset
df = pd.read_csv("final_data_with_prognosis.csv")

# Separate features and target
X = df.drop("prognosis", axis=1)
y = df["prognosis"]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Train model with 360 trees based on optimization study
clf = RandomForestClassifier(n_estimators=360, random_state=42)
clf.fit(X_train, y_train)

# Evaluate model
accuracy = clf.score(X_test, y_test)
print(f"Model trained with accuracy: {accuracy * 100:.2f}%")

# Save model
with open("model.pkl", "wb") as f:
    pickle.dump(clf, f)

print("Model saved as model.pkl")
```

Tree Optimization Code

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np

# Load dataset
df = pd.read_csv("final_data_with_prognosis.csv")
X = df.drop("prognosis", axis=1)
y = df["prognosis"]

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

# Symptom groups for analysis
symptom_groups = [
    ["itching", "skin_rash", "nodal_skin_eruptions"],
    ["shivering", "chills", "joint_pain"],
    ["vomiting", "fatigue", "acidity"],
    ["anxiety", "cold_hands_and_feets", "mood_swings"]
]

# Range of tree sizes
tree_range = list(range(10, 800, 50))
group_confidences = []

# Calculate confidence for each group across tree counts
for symptoms in symptom_groups:
```

```
    confidences = []
    input_vector = [1 if col in symptoms else 0 for col in X.columns]

    for n in tree_range:
        model = RandomForestClassifier(n_estimators=n, random_state=42)
        model.fit(X_train, y_train)
        probas = model.predict_proba([input_vector])[0]
        confidence = np.max(probas)
        confidences.append(confidence)

    group_confidences.append(confidences)

# Analysis for stable point
group_confidences = np.array(group_confidences)
avg_confidence_per_tree = np.mean(group_confidences, axis=0)
slopes = np.gradient(avg_confidence_per_tree)
abs_slopes = np.abs(slopes)
min_slope_idx = np.argmin(abs_slopes)
stable_tree_count = tree_range[min_slope_idx]

print(f"Stable confidence (minimal slope) = {avg_confidence_per_tree[min_slope_idx]:.4f} at
{stable_tree_count} trees.")
```