PREDICTION AND RECOMMENDATION SYSTEM

April 8, 2024

[1]:
```
import zipfile
import os

# File path
zip_file_path = 'drug recommendation system.zip'
extract_folder_path = 'drug_recommendation_system_dataset'

# Extracting the zip file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_folder_path)

# Listing the contents of the extracted folder
extracted_files = os.listdir(extract_folder_path)
extracted_files
```

[1]:
```
['drug recommendation system']
```

[2]:
```
# Exploring the contents of the 'drug recommendation system' directory
inner_directory_path = os.path.join(extract_folder_path, 'drug recommendation␣
 ↪system')
inner_extracted_files = os.listdir(inner_directory_path)
inner_extracted_files
```

[2]:
```
['description.csv',
 'diets.csv',
 'medications.csv',
 'precautions_df.csv',
 'Symptom-severity.csv',
 'symtoms_df.csv',
 'Training.csv',
 'workout_df.csv']
```

The 'drug recommendation system' directory contains the following files:

description.csv diets.csv medications.csv precautions_df.csv Symptom-severity.csv symtoms_df.csv Training.csv workout_df.csv

Training.csv which might contain the core data for training models, and medications.csv which could have details on the drugs.

[3]:
```
import pandas as pd

# File paths
training_file_path = os.path.join(inner_directory_path, 'Training.csv')
medications_file_path = os.path.join(inner_directory_path, 'medications.csv')

# Loading the datasets
training_df = pd.read_csv(training_file_path)
medications_df = pd.read_csv(medications_file_path)

# Displaying the first few rows of each dataset for inspection
training_df_head = training_df.head()
medications_df_head = medications_df.head()

(training_df_head, medications_df_head)
```

[3]:
```
(   itching  skin_rash  nodal_skin_eruptions  continuous_sneezing  shivering  \
 0        1          1                     1                    0          0
 1        0          1                     1                    0          0
 2        1          0                     1                    0          0
 3        1          1                     0                    0          0
 4        1          1                     1                    0          0

    chills  joint_pain  stomach_pain  acidity  ulcers_on_tongue  …  \
 0       0           0             0        0                 0  …
 1       0           0             0        0                 0  …
 2       0           0             0        0                 0  …
 3       0           0             0        0                 0  …
 4       0           0             0        0                 0  …

    blackheads  scurring  skin_peeling  silver_like_dusting  \
 0           0         0             0                    0
 1           0         0             0                    0
 2           0         0             0                    0
 3           0         0             0                    0
 4           0         0             0                    0

    small_dents_in_nails  inflammatory_nails  blister  red_sore_around_nose  \
 0                     0                   0        0                     0
 1                     0                   0        0                     0
 2                     0                   0        0                     0
 3                     0                   0        0                     0
 4                     0                   0        0                     0

    yellow_crust_ooze         prognosis
 0                  0  Fungal infection
 1                  0  Fungal infection
```

```
2                   0  Fungal infection
3                   0  Fungal infection
4                   0  Fungal infection

[5 rows x 133 columns],
              Disease                                          Medication
0     Fungal infection  ['Antifungal Cream', 'Fluconazole', 'Terbinafi…
1              Allergy  ['Antihistamines', 'Decongestants', 'Epinephri…
2                 GERD  ['Proton Pump Inhibitors (PPIs)', 'H2 Blockers…
3  Chronic cholestasis  ['Ursodeoxycholic acid', 'Cholestyramine', 'Me…
4        Drug Reaction  ['Antihistamines', 'Epinephrine', 'Corticoster…)
```

Dataset Overview Training Dataset The Training.csv file contains a dataset where each row represents a case (likely a patient's symptoms), and each column represents different symptoms, with a binary indication (0 or 1) showing whether the symptom is present. The last column, prognosis, indicates the diagnosis. Medications Dataset The medications.csv file links diseases to their corresponding medications. Each row contains a disease and a list of recommended medications.

[ ]:
```
DATA PRE-PROCESSING:Handling Missing Values:
```

[4]:
```
# Checking for missing values and data types in the training dataset
training_df_info = {
    "Missing Values": training_df.isnull().sum(),
    "Data Types": training_df.dtypes
}

training_df_info
```

[4]:
```
{'Missing Values': itching               0
 skin_rash               0
 nodal_skin_eruptions    0
 continuous_sneezing     0
 shivering               0
                        ..
 inflammatory_nails      0
 blister                 0
 red_sore_around_nose    0
 yellow_crust_ooze       0
 prognosis               0
 Length: 133, dtype: int64,
 'Data Types': itching                int64
 skin_rash               int64
 nodal_skin_eruptions    int64
 continuous_sneezing     int64
 shivering               int64
                         …
 inflammatory_nails      int64
 blister                 int64
```

```
red_sore_around_nose     int64
yellow_crust_ooze        int64
prognosis               object
Length: 133, dtype: object}
```

Here there are no missing values in the dataset. All the columns have numerical data except prognosis column. So, encoding of the column"prognosis" should be done

Encode the Prognosis Column and Data Splitting

[5]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

# Label encoding the 'prognosis' column
label_encoder = LabelEncoder()
training_df['prognosis'] = label_encoder.fit_transform(training_df['prognosis'])

# Splitting the data into features (X) and target (y)
X = training_df.drop('prognosis', axis=1)
y = training_df['prognosis']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Outputting the shape of the training and testing sets
(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

[5]:
```
((3936, 132), (984, 132), (3936,), (984,))
```

The data has been successfully preprocessed and split:

The training set contains 3,936 samples. The test set contains 984 samples. Each sample has 132 features (symptoms), and the target variable (prognosis) has been encoded into numerical labels.

Model Development

[8]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import GridSearchCV, cross_val_score
import numpy as np

# Creating the Decision Tree model
dt_classifier = DecisionTreeClassifier(random_state=42)

# Hyperparameters grid to tune
param_grid = {
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
```

```
}

# Grid search for hyperparameter tuning
grid_search = GridSearchCV(estimator=dt_classifier, param_grid=param_grid,
                           cv=5, n_jobs=-1, verbose=2, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Best parameters
best_params = grid_search.best_params_
print("Best Parameters:", best_params)

# Best model
best_dt = grid_search.best_estimator_

# Cross-validation
cv_scores = cross_val_score(best_dt, X_train, y_train, cv=5)
print("Cross-Validation Scores:", cv_scores)
print("Mean CV Score:", np.mean(cv_scores))

# Model Evaluation
y_pred = best_dt.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
classification_rep = classification_report(y_test, y_pred)
print("Classification Report:\n", classification_rep)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split':
2}
Cross-Validation Scores: [1.         1.         1.         0.99872935
0.99872935]
Mean CV Score: 0.9994917407878017
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        18
           1       1.00      1.00      1.00        30
           2       1.00      1.00      1.00        24
           3       1.00      1.00      1.00        25
           4       1.00      1.00      1.00        24
           5       1.00      1.00      1.00        23
           6       1.00      1.00      1.00        33
           7       1.00      1.00      1.00        23
           8       1.00      1.00      1.00        21
           9       1.00      1.00      1.00        15
          10       1.00      1.00      1.00        23
          11       1.00      1.00      1.00        26
```

12	1.00	1.00	1.00	21
13	1.00	1.00	1.00	29
14	1.00	1.00	1.00	24
15	1.00	1.00	1.00	19
16	1.00	1.00	1.00	28
17	1.00	1.00	1.00	25
18	1.00	1.00	1.00	23
19	1.00	1.00	1.00	27
20	1.00	1.00	1.00	26
21	1.00	1.00	1.00	23
22	1.00	1.00	1.00	29
23	1.00	1.00	1.00	25
24	1.00	1.00	1.00	24
25	1.00	1.00	1.00	26
26	1.00	1.00	1.00	21
27	1.00	1.00	1.00	24
28	1.00	1.00	1.00	19
29	1.00	1.00	1.00	22
30	1.00	1.00	1.00	25
31	1.00	1.00	1.00	22
32	1.00	1.00	1.00	24
33	1.00	1.00	1.00	17
34	1.00	1.00	1.00	28
35	1.00	1.00	1.00	22
36	1.00	1.00	1.00	25
37	1.00	1.00	1.00	19
38	1.00	1.00	1.00	26
39	1.00	1.00	1.00	22
40	1.00	1.00	1.00	34
accuracy			1.00	984
macro avg	1.00	1.00	1.00	984
weighted avg	1.00	1.00	1.00	984

Best Parameters Max Depth: None Min Samples Leaf: 1 Min Samples Split: 2 These parameters suggest that the model was allowed to grow without constraints on its depth, with minimal requirements for splitting nodes and leaf samples. Model Performance Accuracy: 1.0 or 100% This implies that the model correctly predicted every case in the test dataset. Classification Report For each class (ranging from 0 to 40), the model achieved:

Precision: 1.00 (100%) Recall: 1.00 (100%) F1-Score: 1.00 (100%) These metrics are perfect across all classes, indicating that the model has precisely and consistently identified all cases. Overall Metrics Macro Average: 1.00 (100%) Weighted Average: 1.00 (100%) These averages further confirm the model's excellent performance across all classes.

Cross-Validation on Training Data Cross-Validation Scores: [1.0, 1.0, 1.0, 0.9987, 0.9987] Mean CV Score: 0.9995 This high consistency in cross-validation scores indicates that Decision Tree model is performing exceptionally well across different subsets of the training data. The slight variance in

a couple of folds (scores of 0.9987) is a good sign, showing that the model doesn't achieve perfect scores in every scenario, which can be a signal of healthy variance.

Evaluation on Test Data Accuracy: 1.0 or 100% Classification Report: Perfect scores (1.0) for precision, recall, and F1-score for every class.

[10]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Random Forest model
rf_classifier = RandomForestClassifier(random_state=42)

# Hyperparameters grid
param_grid_rf = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# Grid search with cross-validation
grid_search_rf = GridSearchCV(estimator=rf_classifier,␣
 ↪param_grid=param_grid_rf, cv=5, n_jobs=-1, verbose=2)
grid_search_rf.fit(X_train, y_train)

# Best parameters and model
print("Best Parameters for Random Forest:", grid_search_rf.best_params_)
best_rf = grid_search_rf.best_estimator_
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Best Parameters for Random Forest: {'max_depth': 10, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 100}
```

[12]:
```
from sklearn.metrics import accuracy_score, classification_report

# Evaluate Random Forest on test data
y_pred_rf = best_rf.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
classification_report_rf = classification_report(y_test, y_pred_rf)

print("Random Forest Accuracy:", accuracy_rf)
print("Random Forest Classification Report:\n", classification_report_rf)
```

```
Random Forest Accuracy: 1.0
Random Forest Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        18
           1       1.00      1.00      1.00        30
```

2	1.00	1.00	1.00	24
3	1.00	1.00	1.00	25
4	1.00	1.00	1.00	24
5	1.00	1.00	1.00	23
6	1.00	1.00	1.00	33
7	1.00	1.00	1.00	23
8	1.00	1.00	1.00	21
9	1.00	1.00	1.00	15
10	1.00	1.00	1.00	23
11	1.00	1.00	1.00	26
12	1.00	1.00	1.00	21
13	1.00	1.00	1.00	29
14	1.00	1.00	1.00	24
15	1.00	1.00	1.00	19
16	1.00	1.00	1.00	28
17	1.00	1.00	1.00	25
18	1.00	1.00	1.00	23
19	1.00	1.00	1.00	27
20	1.00	1.00	1.00	26
21	1.00	1.00	1.00	23
22	1.00	1.00	1.00	29
23	1.00	1.00	1.00	25
24	1.00	1.00	1.00	24
25	1.00	1.00	1.00	26
26	1.00	1.00	1.00	21
27	1.00	1.00	1.00	24
28	1.00	1.00	1.00	19
29	1.00	1.00	1.00	22
30	1.00	1.00	1.00	25
31	1.00	1.00	1.00	22
32	1.00	1.00	1.00	24
33	1.00	1.00	1.00	17
34	1.00	1.00	1.00	28
35	1.00	1.00	1.00	22
36	1.00	1.00	1.00	25
37	1.00	1.00	1.00	19
38	1.00	1.00	1.00	26
39	1.00	1.00	1.00	22
40	1.00	1.00	1.00	34
accuracy			1.00	984
macro avg	1.00	1.00	1.00	984
weighted avg	1.00	1.00	1.00	984

[14]:
```
from xgboost import XGBClassifier
```

```python
# XGBoost model
xgb_classifier = XGBClassifier(random_state=42, use_label_encoder=False,␣
 ↪eval_metric='mlogloss')

# Hyperparameters grid
param_grid_xgb = {
    'n_estimators': [100, 200],
    'max_depth': [3, 6],
    'learning_rate': [0.1, 0.01]
}

# Grid search with cross-validation
grid_search_xgb = GridSearchCV(estimator=xgb_classifier,␣
 ↪param_grid=param_grid_xgb, cv=5, n_jobs=-1, verbose=2)
grid_search_xgb.fit(X_train, y_train)

# Best parameters and model
print("Best Parameters for XGBoost:", grid_search_xgb.best_params_)
best_xgb = grid_search_xgb.best_estimator_
# Evaluate XGBoost on test data
y_pred_xgb = best_xgb.predict(X_test)
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
classification_report_xgb = classification_report(y_test, y_pred_xgb)

print("XGBoost Accuracy:", accuracy_xgb)
print("XGBoost Classification Report:\n", classification_report_xgb)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best Parameters for XGBoost: {'learning_rate': 0.1, 'max_depth': 3,
'n_estimators': 100}
XGBoost Accuracy: 1.0
XGBoost Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        18
           1       1.00      1.00      1.00        30
           2       1.00      1.00      1.00        24
           3       1.00      1.00      1.00        25
           4       1.00      1.00      1.00        24
           5       1.00      1.00      1.00        23
           6       1.00      1.00      1.00        33
           7       1.00      1.00      1.00        23
           8       1.00      1.00      1.00        21
           9       1.00      1.00      1.00        15
          10       1.00      1.00      1.00        23
          11       1.00      1.00      1.00        26
          12       1.00      1.00      1.00        21
```

```
              13       1.00      1.00      1.00        29
              14       1.00      1.00      1.00        24
              15       1.00      1.00      1.00        19
              16       1.00      1.00      1.00        28
              17       1.00      1.00      1.00        25
              18       1.00      1.00      1.00        23
              19       1.00      1.00      1.00        27
              20       1.00      1.00      1.00        26
              21       1.00      1.00      1.00        23
              22       1.00      1.00      1.00        29
              23       1.00      1.00      1.00        25
              24       1.00      1.00      1.00        24
              25       1.00      1.00      1.00        26
              26       1.00      1.00      1.00        21
              27       1.00      1.00      1.00        24
              28       1.00      1.00      1.00        19
              29       1.00      1.00      1.00        22
              30       1.00      1.00      1.00        25
              31       1.00      1.00      1.00        22
              32       1.00      1.00      1.00        24
              33       1.00      1.00      1.00        17
              34       1.00      1.00      1.00        28
              35       1.00      1.00      1.00        22
              36       1.00      1.00      1.00        25
              37       1.00      1.00      1.00        19
              38       1.00      1.00      1.00        26
              39       1.00      1.00      1.00        22
              40       1.00      1.00      1.00        34

    accuracy                           1.00       984
   macro avg       1.00      1.00      1.00       984
weighted avg       1.00      1.00      1.00       984
```

[15]:
```python
from sklearn.svm import SVC

# SVM model
svm_classifier = SVC(random_state=42)

# Hyperparameters grid
param_grid_svm = {
    'C': [0.1, 1, 10],
    'gamma': ['scale', 'auto'],
    'kernel': ['rbf', 'linear']
}

# Grid search with cross-validation
```

```
grid_search_svm = GridSearchCV(estimator=svm_classifier,␣
 ↪param_grid=param_grid_svm, cv=5, n_jobs=-1, verbose=2)
grid_search_svm.fit(X_train, y_train)

# Best parameters and model
print("Best Parameters for SVM:", grid_search_svm.best_params_)
best_svm = grid_search_svm.best_estimator_
# Evaluate SVM on test data
y_pred_svm = best_svm.predict(X_test)
accuracy_svm = accuracy_score(y_test, y_pred_svm)
classification_report_svm = classification_report(y_test, y_pred_svm)

print("SVM Accuracy:", accuracy_svm)
print("SVM Classification Report:\n", classification_report_svm)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best Parameters for SVM: {'C': 0.1, 'gamma': 'scale', 'kernel': 'rbf'}
SVM Accuracy: 1.0
SVM Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        18
           1       1.00      1.00      1.00        30
           2       1.00      1.00      1.00        24
           3       1.00      1.00      1.00        25
           4       1.00      1.00      1.00        24
           5       1.00      1.00      1.00        23
           6       1.00      1.00      1.00        33
           7       1.00      1.00      1.00        23
           8       1.00      1.00      1.00        21
           9       1.00      1.00      1.00        15
          10       1.00      1.00      1.00        23
          11       1.00      1.00      1.00        26
          12       1.00      1.00      1.00        21
          13       1.00      1.00      1.00        29
          14       1.00      1.00      1.00        24
          15       1.00      1.00      1.00        19
          16       1.00      1.00      1.00        28
          17       1.00      1.00      1.00        25
          18       1.00      1.00      1.00        23
          19       1.00      1.00      1.00        27
          20       1.00      1.00      1.00        26
          21       1.00      1.00      1.00        23
          22       1.00      1.00      1.00        29
          23       1.00      1.00      1.00        25
          24       1.00      1.00      1.00        24
          25       1.00      1.00      1.00        26
          26       1.00      1.00      1.00        21
```

```
           27       1.00      1.00      1.00        24
           28       1.00      1.00      1.00        19
           29       1.00      1.00      1.00        22
           30       1.00      1.00      1.00        25
           31       1.00      1.00      1.00        22
           32       1.00      1.00      1.00        24
           33       1.00      1.00      1.00        17
           34       1.00      1.00      1.00        28
           35       1.00      1.00      1.00        22
           36       1.00      1.00      1.00        25
           37       1.00      1.00      1.00        19
           38       1.00      1.00      1.00        26
           39       1.00      1.00      1.00        22
           40       1.00      1.00      1.00        34

    accuracy                           1.00       984
   macro avg       1.00      1.00      1.00       984
weighted avg       1.00      1.00      1.00       984
```

[43]:
```python
sym_des = pd.read_csv("symtoms_df.csv")
precautions = pd.read_csv("precautions_df.csv")
workout = pd.read_csv("workout_df.csv")
description = pd.read_csv("description.csv")
medications = pd.read_csv('medications.csv')
diets = pd.read_csv("diets.csv")
```

[44]:
```python
def get_predicted_value(patient_symptoms, model, symptom_columns):
    # Create an input vector with zeros
    input_vector = np.zeros(len(symptom_columns))

    # Map the input symptoms to their respective places in the input vector
    for symptom in patient_symptoms:
        if symptom in symptom_columns:
            index = symptom_columns.get_loc(symptom)
            input_vector[index] = 1

    # Reshape the input vector to match the model's expected input shape
    input_vector = input_vector.reshape(1, -1)

    # Predict the disease index
    predicted_index = model.predict(input_vector)[0]

    # Convert index to disease name using label_encoder
    return label_encoder.inverse_transform([predicted_index])[0]
```

[48]:
```
def get_recommendations(disease, description_df, precautions_df, workout_df,␣
↪medications_df, diets_df):
    desc = description_df[description_df['Disease'] == disease]['Description'].
↪values[0] if disease in description_df['Disease'].values else 'No␣
↪description available'
    pre = precautions_df[precautions_df['Disease'] == disease].iloc[0, 1:].
↪tolist() if disease in precautions_df['Disease'].values else ['No␣
↪precautions available']
    med = medications_df[medications_df['Disease'] == disease]['Medication'].
↪values[0] if disease in medications_df['Disease'].values else 'No medication␣
↪available'
    die = diets_df[diets_df['Disease'] == disease]['Diet'].values[0] if disease␣
↪in diets_df['Disease'].values else 'No diet available'
    wrkout = workout_df[workout_df['disease'] == disease]['workout'].values[0]␣
↪if disease in workout_df['disease'].values else 'No workout available'
    return desc, pre, med, die, wrkout
```

[49]:
```
# Test with hypothetical symptoms
test_symptoms = ['itching', 'skin_rash', 'nodal_skin_eruptions']
predicted_disease = get_predicted_value(test_symptoms, best_rf, symptom_columns)

# Get recommendations
desc, pre, med, die, wrkout = get_recommendations(predicted_disease,␣
↪description, precautions, workout, medications, diets)

# Display the results
print("Predicted Disease:", predicted_disease)
print("Description:", desc)
print("Precautions:", pre)
print("Medications:", med)
print("Diet Recommendations:", die)
print("Workout Recommendations:", wrkout)
```

```
Predicted Disease: Fungal infection
Description: Fungal infection is a common skin condition caused by fungi.
Precautions: ['Fungal infection', 'bath twice', 'use detol or neem in bathing
water', 'keep infected area dry', 'use clean cloths']
Medications: ['Antifungal Cream', 'Fluconazole', 'Terbinafine', 'Clotrimazole',
'Ketoconazole']
Diet Recommendations: ['Antifungal Diet', 'Probiotics', 'Garlic', 'Coconut oil',
'Turmeric']
Workout Recommendations: Avoid sugary foods

C:\Users\kusum\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\base.py:493: UserWarning: X does not have valid feature names,
but RandomForestClassifier was fitted with feature names
  warnings.warn(
```

[ ]:
```
import tkinter as tk
from tkinter import messagebox

def on_predict():
    # Get symptoms from entry widget
    symptoms_input = entry_symptoms.get()
    user_symptoms = [sym.strip() for sym in symptoms_input.split(',')]

    # Predict disease and get recommendations
    predicted_disease = get_predicted_value(user_symptoms, best_rf,␣
 ↪symptom_columns)  # replace best_rf with your model
    desc, pre, med, die, wrkout = get_recommendations(predicted_disease,␣
 ↪description, precautions, workout, medications, diets)

    # Show results in a message box
    result_message = f"Predicted Disease: {predicted_disease}\n\nDescription:␣
 ↪{desc}\n\nPrecautions: {', '.join(pre)}\n\nMedications: {med}\n\nDiet␣
 ↪Recommendations: {die}\n\nWorkout Recommendations: {wrkout}"
    messagebox.showinfo("Recommendations", result_message)

# Create main window
root = tk.Tk()
root.title("Drug Recommendation System")

# Create and pack widgets
tk.Label(root, text="Enter Symptoms (comma-separated):").pack()
entry_symptoms = tk.Entry(root, width=50)
entry_symptoms.pack()
tk.Button(root, text="Predict and Get Recommendations", command=on_predict).
 ↪pack()

# Run the application
root.mainloop()
```

```
C:\Users\kusum\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\base.py:493: UserWarning: X does not have valid feature names,
but RandomForestClassifier was fitted with feature names
  warnings.warn(
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\kusum\AppData\Local\Programs\Python\Python312\Lib\tkinter\__ini
t__.py", line 1967, in __call__
    return self.func(*args)
           ^^^^^^^^^^^^^^^^

  File "C:\Users\kusum\AppData\Local\Temp\ipykernel_61512\613611876.py", line
14, in on_predict
    result_message = f"Predicted Disease: {predicted_disease}\n\nDescription:
```

```
{desc}\n\nPrecautions: {', '.join(pre)}\n\nMedications: {med}\n\nDiet
Recommendations: {die}\n\nWorkout Recommendations: {wrkout}"
                      ^^^^^^^^^^^^^^
TypeError: sequence item 4: expected str instance, float found
C:\Users\kusum\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\base.py:493: UserWarning: X does not have valid feature names,
but RandomForestClassifier was fitted with feature names
  warnings.warn(
C:\Users\kusum\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\base.py:493: UserWarning: X does not have valid feature names,
but RandomForestClassifier was fitted with feature names
  warnings.warn(
C:\Users\kusum\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\base.py:493: UserWarning: X does not have valid feature names,
but RandomForestClassifier was fitted with feature names
  warnings.warn(
```

[ ]: