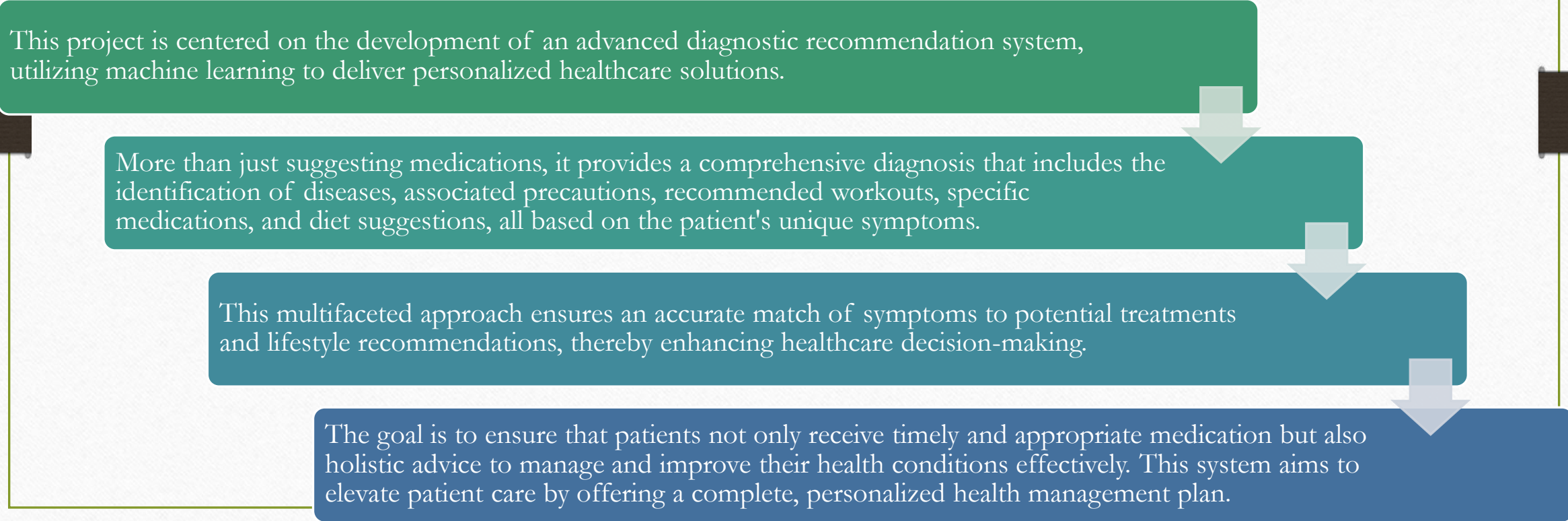
A microscopic view of numerous red blood cells, which are biconcave discs with a reddish-pink hue, set against a light, hazy background. The cells are in various stages of focus, with some appearing sharp and others blurred.

# Disease Prediction and recommendation system

-Kusuma Korada(Sole Contributor)

# OBJECTIVE

This project is centered on the development of an advanced diagnostic recommendation system, utilizing machine learning to deliver personalized healthcare solutions.



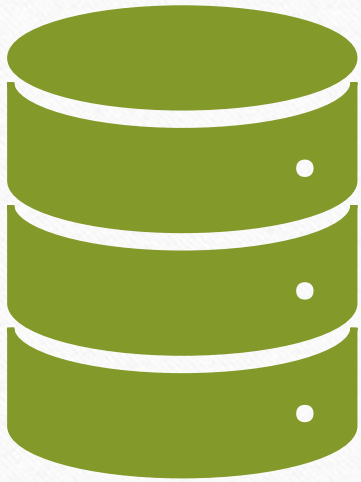
More than just suggesting medications, it provides a comprehensive diagnosis that includes the identification of diseases, associated precautions, recommended workouts, specific medications, and diet suggestions, all based on the patient's unique symptoms.

This multifaceted approach ensures an accurate match of symptoms to potential treatments and lifestyle recommendations, thereby enhancing healthcare decision-making.

The goal is to ensure that patients not only receive timely and appropriate medication but also holistic advice to manage and improve their health conditions effectively. This system aims to elevate patient care by offering a complete, personalized health management plan.

## Dataset Overview:

The project utilizes a comprehensive dataset comprised of various dimensions, including symptoms, medical conditions, drug descriptions, and precautionary measures. This dataset encompasses detailed symptomatology associated with a wide range of medical conditions, alongside corresponding treatment options, thus providing a robust foundation for training our models. Notable components of the dataset include symptom severity, medication effectiveness, dietary considerations, and exercise recommendations, all of which contribute to a holistic approach to drug recommendation.



The data set is taken from kaggle website : <https://www.kaggle.com/datasets/noorsaeed/medicine-recommendation-system-dataset>Links to an external site.



Loading of Data Set and EDA



Data Pre-processing



Model Development, tuning, and evaluation



Logic implementation ,testing, GUI Interface

```
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

['drug_recommendation system']
```

```
# 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
```

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

Unzipping the Dataset: Utilized Python's zipfile module to extract files from 'drug recommendation system.zip'.

Listing Dataset Files: Identified the contents of the extracted folder, revealing files such as 'description.csv', 'diets.csv', 'medications.csv', etc.

### Loading Data:

Imported the 'Training.csv' and 'medications.csv' files using Pandas.

Initial Inspection:

Displayed the first few rows of each dataset to understand their structure.

```
# 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()
```

## DATA PRE-PROCESSING

- Checked missing values and Data Types. No missing values are found and Prognosis column contain categorical data.
- Label Encoding: Converted the 'prognosis' column to numerical format using LabelEncoder.
- Splitting Data: Divided the dataset into features (symptoms) and target (diagnosis), then further split into training and testing sets.

```
# 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
```

```
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)

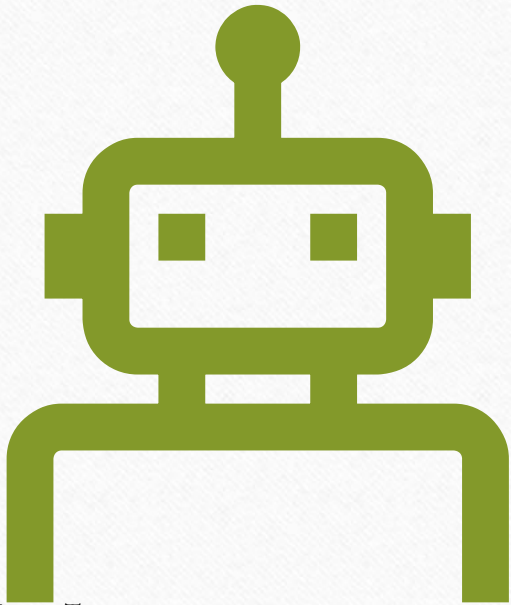
((3936, 132), (984, 132), (3936,), (984,))
```

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 AND EVALUATION

1. Random Forest
2. Decision Trees
3. XG Boost
4. Support Vector Machines





## RANDOM FOREST:

1. Imported necessary modules like RandomForestClassifier from sklearn.ensemble, GridSearchCV from sklearn.model\_selection
2. A RandomForestClassifier object is created with a random\_state of 42
3. Set Hyperparameter Grid
4. Done Hyperparameter tuning using Grid Search
5. Outputted the best parameters, imported evaluation metrics for model evaluation and printed classification report.

```
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}

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)
```

```
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	20

## DECISION TREES:

Decision Tree classifier is selected for its ability to handle a multiclass classification problem, with a grid search implemented to fine-tune the hyperparameters, yielding an unconstrained `max_depth`, and requiring at least one sample per leaf and two to split a node. The model's robustness is affirmed by near-perfect cross-validation scores, which suggest a strong fit to the training data

```
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        18
```

## XG BOOST:

- An XGBoost classifier is prepared with predefined settings and the 'mlogloss' metric for a multiclass classification task.
- The model's hyperparameters, including 'n\_estimators', 'max\_depth', and 'learning\_rate', are optimized using GridSearchCV with 5-fold cross-validation.
- The optimal parameters are identified, and the corresponding best model achieves a perfect accuracy of 1.0 on the test dataset, as reflected in the classification report's detailed metrics.

```
from xgboost import XGBClassifier

# 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
```



## SUPPORT VECTOR MACHINE:

- An SVM classifier is created with a fixed random state to ensure consistent results across runs.
- A parameter grid is set up for C (regularization parameter), gamma (kernel coefficient), and kernel type.
- GridSearchCV conducts a search across the parameter space with 5-fold cross-validation, and identifies the best SVM configuration as {'C': 0.1, 'gamma': 'scale', 'kernel': 'rbf'}.

This optimal model is then evaluated on a test set, achieving an accuracy score of 1.0, indicating perfect classification. The corresponding classification report suggests that all predictions match the true labels exactly, with precision, recall, and F1-score all at 1.0 for each class.

```
# 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



## LOGIC IMPLEMENTATION AND TESTING

Various CSV files containing data on symptoms, precautions, workouts, descriptions, medications, and diets are being loaded into corresponding pandas DataFrames.

```
sym_des = pd.read_csv("symptoms_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")
```

```
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]
```

```
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'
    workout = 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, workout
```

- get\_predicted\_value for diagnosing diseases based on symptoms using a machine learning model and translating the output into a readable disease name;
- get\_recommendations for fetching health advice related to the diagnosed disease from various datasets, including descriptions and specific recommendations for medications, diets, and workouts.

```

# 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

```

- Tests the machine learning pipeline with hypothetical symptoms to diagnose a disease and provide relevant health recommendations.
- Given symptoms 'itching', 'skin rash', and 'nodal skin eruptions', the model predicts the disease as 'Fungal infection'.
- Then, it retrieves health recommendations, including a description of the condition, precautions to take, medications to use, dietary suggestions, and workout advice. The results are printed out, providing a comprehensive guide for managing the diagnosed condition, Fungal infection, with details such as bathing with detol or neem water and using antifungal cream.

# INTEGRATION WITH GUI INTERFACE

Simple graphical user interface (GUI) is created using the Tkinter library in Python for a drug recommendation system.

The `on_predict` function is defined to read user input symptoms from a text entry widget, process these symptoms to predict the disease using the `get_predicted_value` function, and then retrieve health recommendations through the `get_recommendations` function.

Once the disease is predicted and recommendations are fetched, the results are displayed to the user in a message box. The main window of the GUI is set up with a title, a label, an entry widget for symptom input, and a button that triggers the prediction when clicked. The application is run with a loop that waits for user events.

Sample Footer Text

```
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 Recom
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()
```



Drug Recommendation...

Enter Symptoms (comma-separated):  
vomiting,headache,nausea, spinning\_movements  
Predict and Get Recommendations

Recommendations

i

Predicted Disease: (vertigo) Paroysmal Positional Vertigo

Description: No description available

Precautions: (vertigo) Paroysmal Positional Vertigo, lie down, avoid sudden change in body, avoid abrupt head movment, relax

Medications: No medication available

Diet Recommendations: No diet available

Workout Recommendations: Avoid trigger foods (caffeine, alcohol)

OK



THANK YOU

