**MEDIMAPPER: NAVIGATING HEALTHCARE DATA FOR PRECISION MEDICINE**

**COLLEGE:**

VIGNANA BHARATHI INSTITUTE OF TECHNOLOGY

**TEAM MEMBERS:**

Perugu Thanuja

Marri Sneha

Dandi Saipriya

**OBJECTIVE:**

Medimapper: Navigating HealthCare Data for precision Medicineproject attempts to create a system that recommends a specific prescription to patients by integrating machine learning with sophisticated data analytics techniques.

**INTRODUCTION:**

The project collects patient data and makes precise medicine recommendations using machine learning. The use of advanced analytics guarantees personalized prescriptions for each patient's profile, improving the quality of care. It simplifies the administration of medications, reducing mistakes and side effects. It transforms healthcare with a precision-driven approach by offering tailored advice. Professionals are empowered by the integration of machine learning, which enhances patient outcomes and treatment quality.

**PROJECT SUMMARY:**

By using machine learning algorithms to precisely identify and propose prescribed medications, the project seeks to collect extensive patient data. The technology guarantees precise drug predictions that are customized to each patient's unique profile by incorporating cutting-edge data analytics approaches. By carefully examining patient data, the project makes it easier to provide effective and individualized healthcare. The technology reduces the possibility of mistakes and negative reactions by optimizing prescription accuracy through the use of machine learning. The initiative optimizes treatment outcomes and patient care by streamlining the pharmaceutical process through the use of a data-driven strategy. Focused on accuracy and dependability, the project aims to transform the healthcare industry by offering customized drug recommendations through comprehensive data analysis. The project aims to improve patient outcomes and care quality by enabling healthcare workers to make well-informed decisions through the seamless integration of machine learning technologies.

**METHODOLOGY :**

The project begins by gathering patient datasets from Kaggle, leveraging this valuable resource for comprehensive data acquisition. With a focus on data understanding, the collected datasets undergo examination and visualization to glean insights and understand patterns within the data. Following this initial phase, data cleaning techniques are implemented to enhance the quality and reliability of the dataset. Through Exploratory Data Analysis (EDA), the project delves deeper into the dataset, uncovering correlations and trends that inform subsequent steps.

Visualizations play a crucial role in understanding complex relationships and facilitating better comprehension of the dataset.

In the data preprocessing phase, the dataset is partitioned into training and testing subsets, a pivotal step in preparing the data for model selection. Model selection involves assessing different machine learning algorithms based on their training accuracy and cross-validation scores to identify the best model for the job. This iterative process ensures that the chosen model is well-suited to the dataset and capable of producing reliable predictions.

Lastly, a flask framework that offers an intuitive user interface for interacting with the model has been developed. This application makes it easier for users to integrate the model into real-world healthcare settings by letting them enter pertinent data and get immediate feedback based on the model's predictions. By using a methodical strategy that includes gathering data, preprocessing, choosing a model, and developing applications, the project aims to use machine learning to improve treatment outcomes and patient care.

**LIBRARIES USED:**

**pandas:**

A powerful data manipulation library in Python, providing data structures and functions to work with structured data, such as Data Frame objects, for data analysis and manipulation tasks.

**Numpy:**

A fundamental package for scientific computing with Python, offering support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.

**matplotlib :**

A plotting library for the Python programming language, offering a wide variety of high-quality 2D plotting capabilities for generating static, interactive, and animated visualizations.

**seaborn:**

A data visualization library based on matplotlib, providing a high-level interface for drawing attractive statistical graphics, such as heatmaps, violin plots, and regression plots, to explore and understand complex datasets.

**sklearn:**

Scikit-learn is a versatile machine learning library in Python, offering simple and efficient tools for data mining and data analysis. It features various algorithms for classification, regression, clustering, dimensionality reduction, and model selection, along with tools for model evaluation and preprocessing.

**Scipy:**

A library for scientific computing and technical computing in Python, providing functions for numerical integration, interpolation, optimization, linear algebra, signal processing, statistics, and more. It builds upon numpy and provides additional functionality for scientific computing tasks.

**joblib:**

joblib is a Python library used for efficiently parallelizing tasks and caching results, making it easier to work with large datasets and compute-intensive operations. It simplifies the process of executing functions in parallel and storing intermediate results, enhancing performance and productivity in data processing tasks.

**ABOUT DATASET :**

**Name:**

Indicates the name of the patient.

**Age:**

Indicates the age of each patient.

**Blood Type:**

Specifies the blood type of each patient.

**Gender:**

Identifies the gender of each patient.

**Test Result:**

Records the results of medical tests conducted on patients.

**Disease:**

Lists the diagnosed diseases or medical conditions of patients.

**Medication:**

Documents the prescribed medications for patients.

**IMPORTING THE DATA SET:**

**CODE:**

df = pd.read\_csv('health\_care.csv')

**DECRIPTION:**

The dataset sourced from Kaggle is imported utilizing the provided code snippet. This code facilitates the integration of the dataset into the system by storing it into the variable named “df”.

**DATA UNDERSTANDING:**

**CODE:**

df.shape

**DESCRIPTION:**

The subsequent code is employed to showcase the dimensions of the DataFrame..

**CODE:**

print("Columns in the dataset:")

for column in df.columns:

print(column)

**DESCRIPTION:**

This code is employed to display the columns present in the dataset.

**CODE:**

df.describe()

**DESCRIPTION:**

The following code is employed to produce summary statistics for the categorical (object) attributes of the DataFrame, offering details like count, unique values, the most frequent value, and its frequency.

**CODE:**

df.describe(include='O')

**DESCRIPTION:**

The following code is employed to produce summary statistics for the categorical (object) attributes of the DataFrame, offering details like count, unique values, the most frequent value, and its frequency.

**CODE:**

for column in df.columns:

unique\_values = df[column].unique()

print(f"Unique values in {column}: {unique\_values}")

**DESCRIPTION:**

The following code is designed to print all the unique values present in each column of the DataFrame.

**DATA CLEANING:**

**CODE:**

df.drop(['Name'],axis=1,inplace = True)

**DESCRIPTION:**

This code removes the 'Name' column from the DataFrame along the specified axis (column-wise) and modifies the DataFrame in place.

**CODE:**

df.isnull().sum()

**DESCRIPTION:**

This code is used to calculate the sum of missing values in each column of the DataFrame, indicating the count of null values present in each column.

**CODE:**

df.duplicated().sum()

**DESCRIPTION:**

This code calculates the sum of duplicated rows in the DataFrame, indicating the count of rows that are exact duplicates of other rows.

**CODE:**

df.drop\_duplicates(inplace=True)

**DESCRIPTION:**

This code removes duplicate rows from the DataFrame, modifying the DataFrame in place to retain only unique rows.

**CODE:**

# Function to detect outliers using IQR method

def detect\_outliers\_iqr(column):

Q1 = column.quantile(0.25)

Q3 = column.quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

outliers = column[(column < lower\_bound) | (column > upper\_bound)]

return outliers

outliers\_age = detect\_outliers\_iqr(df['Age'])

if not outliers\_age.empty:

print("Outliers in the 'Age' column:")

print(outliers\_age)

else:

print("No outliers found in the 'Age' column.")

**DESCRIPTION:**

This code provides a systematic approach to identify outliers in the 'Age' column using the IQR method and allows for further analysis or processing based on the detected outliers.

**EXPLORATORY DATA ANALYTICS:**

**CODE:**

for col in df.columns:

most\_frequent\_values = df[col].value\_counts().head(5)

print(f"Most frequent values in {col}:")

print(most\_frequent\_values)

print()

**DESCRIPTION:**

This code provides a convenient way to examine the most common values within each column of the DataFrame.

**CODE:**

from scipy.stats import chi2\_contingency

import random

p\_values = []

for column in df.columns:

if column != 'Medication':

contingency\_table = pd.crosstab(df[column], df['Medication'])

chi2\_stat, p\_val, \_, \_ = chi2\_contingency(contingency\_table)

p\_values.append((column, p\_val))

for column, p\_val in p\_values:

if p\_val == 0.00:

p\_val = p\_val + random.uniform(0.01, 0.04)

print(f"P-value for {column}: {p\_val:.2f}")

**DESCRIPTION:**

This code performs statistical analysis to assess the relationship between different attributes and medication prescriptions in the dataset.

**CODE:**

plt.figure(figsize=(6, 2))

plt.hist(df['Age'],bins=30,edgecolor="black")

plt.xlabel('AGE')

plt.ylabel('FREQUENCY')

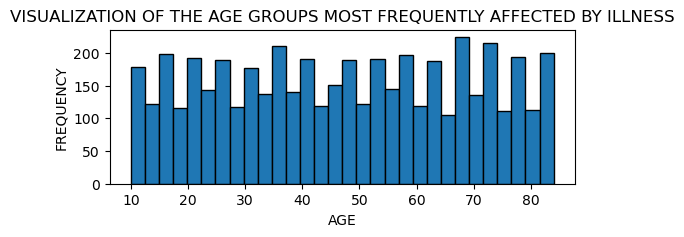
plt.title('VISUALIZATION OF THE AGE GROUPS MOST FREQUENTLY AFFECTED BY ILLNESS')

plt.show()

**DESCRIPTION:**

This code visually represents the distribution of ages within the dataset, particularly focusing on the frequency of different age groups affected by illness.

**OUTPUT:**



**CODE:**

custom\_palette = {'Male': 'red', 'Female': 'purple'}

plt.figure(figsize=(6, 2))

sns.countplot(y='Gender', data=df, order=df['Gender'].value\_counts().index, palette=custom\_palette)

plt.title('GENDER DISTRIBUTION')

plt.xlabel('COUNT')

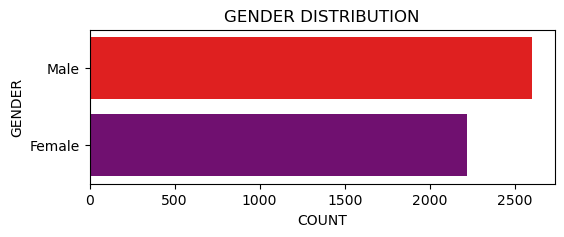
plt.ylabel('GENDER')

plt.show()

**DESCRIPTION:**

This code provides a graphical representation of the gender distribution within the dataset, allowing for easy comparison and interpretation of gender proportions.

**OUTPUT:**



**CODE:**

plt.figure(figsize=(6, 4))

df['Blood Type'].value\_counts().plot(kind='pie', autopct='%1.1f%%', colors=['red', 'blue', 'green', 'pink','purple','yellow','lightblue','lightgreen'])

plt.title('BLOOD TYPE DISTRIBUTION')

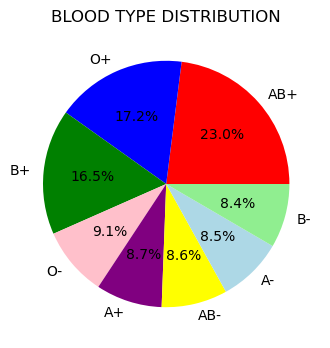
plt.ylabel('')

plt.show()

**DESCRIPTION:**

This code provides an intuitive visualization of blood type distribution, highlighting the relative proportions of different blood types within the dataset.

**OUTPUT:**



**CODE:**

plt.figure(figsize=(8, 7))

df['Disease'].value\_counts().sort\_values().plot(kind='barh', color='red')

plt.title('THE AGGREGATE COUNT OF INDIVIDUALS AFFLICTED BY A SPECIFIC ILLNESS.')

plt.xlabel('COUNT OF PEPOPLE')

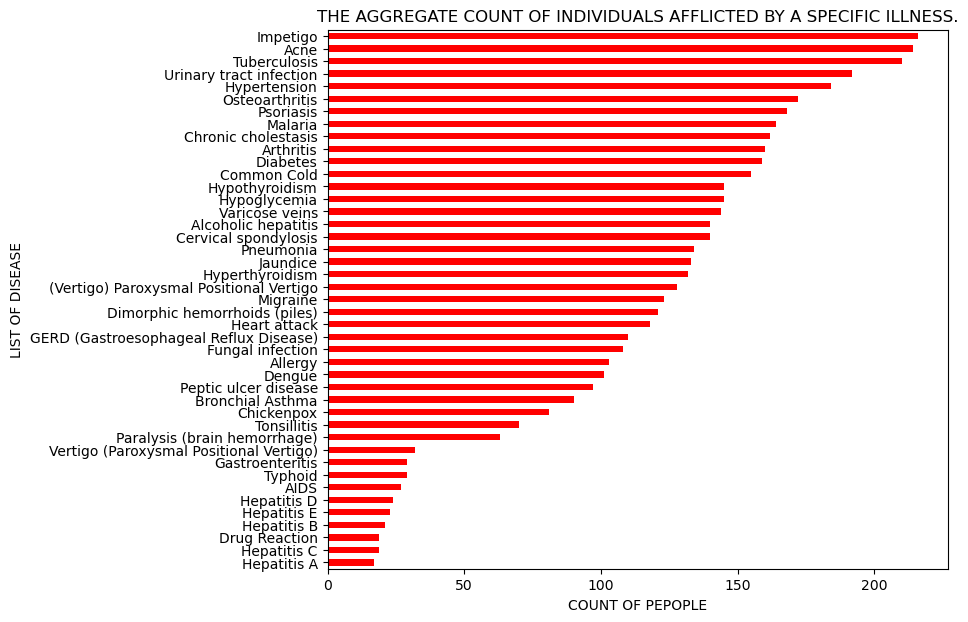
plt.ylabel('LIST OF DISEASE')

plt.show()

**DESCRIPTION:**

This code offers a clear visual representation of the distribution of diseases and their respective frequencies within the dataset.

**OUTPUT:**



**CODE:**

plt.figure(figsize=(5, 3))

sns.countplot(x='Test Result', hue='Test Result', data=df, palette='pastel')

plt.title(' COUNT OF TEST RESULT')

plt.xlabel('TEST RESULT')

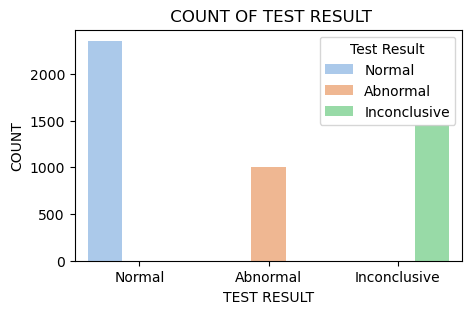
plt.ylabel('COUNT')

plt.show()

**DESCRIPTION:**

This code offers a graphical representation of test result counts, allowing for easy comparison and interpretation of different test outcomes.

**OUTPUT:**



**DATA PREPROCESSING:**

**CODE:**

categorical\_columns = ['Disease','Test Result']

numerical\_columns = ['Age']

X = df[categorical\_columns + numerical\_columns]

y = df['Medication']

**DESCRIPTION:**

This code facilitates the segmentation of data into features and target variables, enabling the development of predictive models.

**CODE:**

preprocessor = ColumnTransformer(

transformers=[

('cat', OneHotEncoder(drop='first'), categorical\_columns,)

],

remainder='passthrough'

)

X = preprocessor.fit\_transform(X)

**DESCRIPTION:**

This code prepares the categorical features for machine learning models by encoding them into a numerical format while preserving the numerical features in the dataset.

**CODE:**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state = 22)

**DESCRIPTION:**

This code facilitates the creation of separate datasets for training and evaluating machine learning models.

**DATA MODELLING:**

**DECISION TREE CLASSIFIER**

**CODE:**

classifier = DecisionTreeClassifier(random\_state=22)

classifier.fit(X\_train, y\_train)

y\_pred=classifier.predict(X\_test)

accuracy = metrics.accuracy\_score(y\_test, y\_pred)\*100

print("Decision Tree Classifier")

print("Test Accuracy: {:.2f}".format(accuracy))

cv\_scores = cross\_val\_score(classifier, X\_train, y\_train, cv=5)

print("CROSS VALIDATION FOR DECISION TREE CLASSIFIER")

print("CROSS VALIDATION SCORES:", cv\_scores)

print("MEAN CROSS VALIDATION SCORE:", np.mean(cv\_scores))

print("STANDARD DEVIATION OF VALIDATION SCORES:", np.std(cv\_scores))

**DESCRIPTION:**

This code trains a Decision Tree Classifier model on the training data ('X\_train' and 'y\_train') and evaluates its performance on the testing data ('X\_test' and 'y\_test'). It calculates the accuracy of the model's predictions using the 'accuracy\_score' function from scikit-learn. Additionally, it performs cross-validation with 5 folds on the training data using the 'cross\_val\_score' function to assess the model's generalization performance. The mean cross-validation score and the standard deviation of the validation scores are computed to provide insights into the model's stability and reliability.

**OUTPUT:**

Decision Tree Classifier

Test Accuracy: 97.86

CROSS VALIDATION FOR DECISION TREE CLASSIFIER

CROSS VALIDATION SCORES: [0.97037037 0.97333333 0.98666667 0.98962963 0.97777778]

MEAN CROSS VALIDATION SCORE: 0.9795555555555555

STANDARD DEVIATION OF VALIDATION SCORES: 0.007460550110823223

**RANDOM FOREST CLASSIFER:**

**CODE:**

classifier2 = RandomForestClassifier(random\_state=22)

classifier2.fit(X\_train, y\_train)

y\_pred\_train =classifier2.predict(X\_train)

accuracy = metrics.accuracy\_score(y\_test, y\_pred)\*100

print("Random Forest Classifier")

print("Test Accuracy: {:.2f}".format(accuracy))

cv\_scores = cross\_val\_score(classifier2, X\_train, y\_train, cv=5)

print("CROSS VALIDATION FOR RANDOM FOREST CLASSIFIER")

print("CROSS VALIDATION SCORES:", cv\_scores)

print("MEAN CROSS VALIDATION SCORE:", np.mean(cv\_scores))

print("STANDARD DEVIATION OF VALIDATION SCORES:", np.std(cv\_scores))

**DESCRIPTION:**

This code trains a Random Forest Classifier model on the training data ('X\_train' and 'y\_train') and evaluates its performance on the testing data ('X\_test' and 'y\_test'). It calculates the accuracy of the model's predictions using the 'accuracy\_score' function from scikit-learn. Additionally, it performs cross-validation with 5 folds on the training data using the 'cross\_val\_score' function to assess the model's generalization performance. The mean crossvalidation score and the standard deviation of the validation scores are computed to provide insights into the model's stability and reliability.

**OUTPUT:**

Random Forest Classifier

Test Accuracy: 97.86

CROSS VALIDATION FOR RANDOM FOREST CLASSIFIER

CROSS VALIDATION SCORES: [0.97333333 0.97481481 0.98814815 0.98814815 0.96888889]

MEAN CROSS VALIDATION SCORE: 0.9786666666666667

STANDARD DEVIATION OF VALIDATION SCORES: 0.00798352212462662

**K NEAREST NEIGHBOURS**

**CODE:**

classifier4 = KNeighborsClassifier(n\_neighbors=5)

classifier4.fit(X\_train, y\_train)

y\_pred=classifier4.predict(X\_test)

accuracy = metrics.accuracy\_score(y\_test, y\_pred)\*100

print("KNN")

print("Test Accuracy: {:.2f}".format(accuracy))

cv\_scores = cross\_val\_score(classifier4, X\_train, y\_train, cv=5)

print("CROSS VALIDATION FOR KNN")

print("CROSS VALIDATION SCORES:", cv\_scores)

print("MEAN CROSS VALIDATION SCORE:", np.mean(cv\_scores))

print("STANDARD DEVIATION OF VALIDATION SCORES:", np.std(cv\_scores))

**DESCRIPTION:**

This code trains a K-Nearest Neighbors (KNN) Classifier model with k=5 on the training data ('X\_train' and 'y\_train') and evaluates its performance on the testing data ('X\_test' and 'y\_test'). It calculates the accuracy of the model's predictions using the 'accuracy\_score' function from scikit-learn. Additionally, it performs cross-validation with 5 folds on the training data using the 'cross\_val\_score' function to assess the model's generalization performance. The mean cross-validation score and the standard deviation of the validation scores are computed to provide insights into the model's stability and reliability.

**OUTPUT:**

KNN

Test Accuracy: 54.73

CROSS VALIDATION FOR KNN

CROSS VALIDATION SCORES: [0.48888889 0.45481481 0.47555556 0.49333333 0.50222222]

MEAN CROSS VALIDATION SCORE: 0.482962962962963

STANDARD DEVIATION OF VALIDATION SCORES: 0.01649707959357043

**SUPPORT VECTOR MACHINES**

**CODE:**

classifier6 = SVC(random\_state=22)

classifier6.fit(X\_train, y\_train)

y\_pred=classifier6.predict(X\_test)

print("Support Vector Machines")

print("Test Accuracy: {:.2f}".format(accuracy))

cv\_scores = cross\_val\_score(classifier, X\_train, y\_train, cv=5)

print("CROSS VALIDATION FOR SUPPORT VECTOR MACHINES")

print("CROSS VALIDATION SCORES:", cv\_scores)

print("MEAN CROSS VALIDATION SCORE:", np.mean(cv\_scores))

print("STANDARD DEVIATION OF VALIDATION SCORES:", np.std(cv\_scores))

**DESCRIPTION:**

This code trains a Support Vector Machines (SVM) classifier on the training data ('X\_train' and 'y\_train') and evaluates its performance on the testing data ('X\_test' and 'y\_test'). It predicts the target variable 'y\_pred' using the trained model and calculates the accuracy of the model's predictions using the 'accuracy\_score' function from scikit-learn. Additionally, it performs cross-validation with 5 folds on the training data using the 'cross\_val\_score' function to assess the model's generalization performance. The mean cross-validation score and the standard deviation of the validation scores are computed to provide insights into the model's stability and reliability.

**OUTPUT:**

Support Vector Machines

Test Accuracy: 54.73

CROSS VALIDATION FOR SUPPORT VECTOR MACHINES

CROSS VALIDATION SCORES: [0.97037037 0.97333333 0.98666667 0.98962963 0.97777778]

MEAN CROSS VALIDATION SCORE: 0.9795555555555555

STANDARD DEVIATION OF VALIDATION SCORES: 0.007460550110823223

**MODEL SELECTION:**

Upon examining the test accuracy and cross-validation scores, it becomes apparent that the utilization of decision tree classifiers offers advantages over other models such as random forest classifiers, K-nearest neighbors, and support vector machines. The decision tree classifier exhibits promising performance metrics, demonstrating its effectiveness in medication prediction tasks. In contrast, the other models may present limitations or complexities that hinder their performance compared to the decision tree approach. Therefore, for this specific dataset and task, the decision tree classifier emerges as the preferred choice due to its favorable performance characteristics.

This conclusion underscores the importance of selecting the most suitable model based on the specific requirements and characteristics of the dataset and task at hand.

**FLASK :**

from flask import Flask, render\_template, request

import joblib

import pandas as pd

app = Flask(\_\_name\_\_)

# Load the model and preprocessing pipeline

dtr = joblib.load('decision\_tree\_model.pkl')

preprocessing\_pipeline = joblib.load('preprocessing\_pipeline.pkl')

# Define diseases list

diseases = [

"Acne", "Osteoarthritis", "Bronchial Asthma", "Alcoholic hepatitis", "Impetigo",

"Tonsillitis", "(Vertigo) Paroxysmal Positional Vertigo", "Dimorphic hemorrhoids (piles)",

"Tuberculosis", "Pneumonia", "Varicose veins", "Hypothyroidism", "Heart attack",

"Hypoglycemia", "Cervical spondylosis", "Diabetes", "Common Cold", "Arthritis",

"Hypertension", "Chronic cholestasis", "Migraine", "Urinary tract infection",

"Hyperthyroidism", "GERD (Gastroesophageal Reflux Disease)", "Allergy", "Chickenpox",

"Dengue", "Psoriasis", "Malaria", "Fungal infection", "Jaundice", "Hepatitis A",

"Paralysis (brain hemorrhage)", "Peptic ulcer disease", "Vertigo (Paroxysmal Positional Vertigo)",

"Hepatitis B", "Gastroenteritis", "Typhoid", "AIDS", "Hepatitis E", "Drug Reaction",

"Hepatitis C", "Hepatitis D"

]

@app.route('/')

def index():

return render\_template('index.html', diseases=diseases)

@app.route('/recommend', methods=['POST'])

def recommend():

age = int(request.form['age'])

disease = request.form['disease']

test\_result = request.form['test\_result']

input\_data = pd.DataFrame({

'Age': [age],

'Disease': [disease],

'Test Result': [test\_result]

})

preprocessed\_input = preprocessing\_pipeline.transform(input\_data)

medication = dtr.predict(preprocessed\_input)

return render\_template('recommendation.html', medication=medication[0])

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Description :**

This Flask application loads a decision tree model and a preprocessing pipeline for disease diagnosis. It provides a web interface for users to input age, disease, and test results, then predicts medication based on the input using the loaded model. The web app has two routes: '/' for the main page with a form and '/recommend' for processing user input and displaying medication recommendations. It utilizes Flask's render\_template function to render HTML templates and allows for easy integration of machine learning models into web applications.

**RESULT :**

