**Healthcare Data Analysis and Medical Condition Predictor**

**Overview**

The Healthcare Data Analysis and Medical Condition Predictor is a Python-based project that aims to analyse healthcare data, identify correlations between various factors such as blood group, gender, and medical condition, and develop a web application to predict the medical condition a person is most at risk of based on their input data.

**Project Structure**

The project consists of the following components:

1.Data Analysis and Visualization: Python scripts to analyse the healthcare data and create visualizations to identify correlations and insights.

2.Model Development: Python scripts to develop machine learning models using the healthcare data to predict medical conditions.

3.Web Application: A Flask-based web application that allows users to input their name, gender, and blood group, and displays the predicted medical condition they are at most risk of.

**Features**

1.Data Analysis and Visualization:

* Analysis of correlations between blood group, gender, and medical condition.
* Visualization of data using plots and graphs.

2.Model Development:

* Training machine learning models to predict medical conditions based on input data.
* Evaluation of model performance.

3.Web Application:

* User-friendly interface for entering name, gender, and blood group.
* Prediction of the medical condition a person is most at risk of.
* Storage of user-entered data in a separate CSV/JSON file.

**Architecture**

**1.Data Analysis:** Utilizes Python libraries such as Pandas and Matplotlib for data analysis and visualization.

**Importing Libraries:**

pandas as pd: Pandas is a powerful data manipulation library in Python.

matplotlib.pyplot as plt: Matplotlib is a plotting library for Python.

seaborn as sns: Seaborn is a statistical data visualization library based on Matplotlib.

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

Loading the Dataset:

pd.read\_csv("healthcare\_dataset.csv"): This line loads the dataset from a CSV file named "healthcare\_dataset.csv" into a DataFrame called df.

# Load the dataset from CSV

df = pd.read\_csv("healthcare\_dataset.csv")

Plotting Correlation:

sns.heatmap(): This function creates a heatmap of the correlation between different variables.

pd.crosstab(): This function computes a cross-tabulation of two or more factors. In this case, it's used to create a table of counts between "Blood Group Type", "Gender", and "Medical Condition".

annot=True: This parameter adds annotations (numbers) to each cell in the heatmap.

fmt="d": This parameter specifies the format of the annotations as integers.

cmap="YlGnBu": This parameter sets the color map for the heatmap.

# Plot correlation between Blood Group Type, Gender, and Medical Condition

plt.figure(figsize=(10, 6))

sns.heatmap(pd.crosstab([df["Blood Group Type"], df["Gender"]], df["Medical Condition"]),

annot=True, fmt="d", cmap="YlGnBu")

plt.title("Correlation between Blood Group Type, Gender, and Medical Condition")

plt.xlabel("Medical Condition")

plt.ylabel("Blood Group Type, Gender")

plt.xticks(rotation=45)

plt.yticks(rotation=0)

plt.show()

Calculating Average Billing Amount:

df.groupby("Medical Condition")["Billing Amount"].mean(): This line groups the data by "Medical Condition" and calculates the mean of the "Billing Amount" for each group. The result is stored in the avg\_billing DataFrame.

# Calculate average billing amount per medical condition

avg\_billing = df.groupby("Medical Condition")["Billing Amount"].mean().reset\_index()

Plotting Average Billing Amount:

sns.barplot(): This function creates a bar plot.

x="Medical Condition", y="Billing Amount": This specifies the variables to plot on the x-axis and y-axis respectively.

# Plot average billing amount per medical condition

plt.figure(figsize=(10, 6))

sns.barplot(x="Medical Condition", y="Billing Amount", data=avg\_billing)

plt.title("Average Billing Amount per Medical Condition")

plt.xlabel("Medical Condition")

plt.ylabel("Average Billing Amount")

plt.xticks(rotation=45)

plt.show()

Overall, this code loads a healthcare dataset, visualizes the correlation between blood group type, gender, and medical condition using a heatmap, and then plots the average billing amount per medical condition using a bar plot.

**2. Model Development:** Built a machine learning model using a Random Forest Classifier on a healthcare dataset. Let's break it down step by step:

Importing Libraries:

pandas as pd: Pandas is used for data manipulation and analysis.

train\_test\_split from sklearn.model\_selection: This function splits the dataset into random train and test subsets.

RandomForestClassifier from sklearn.ensemble: This is an implementation of a Random Forest classifier, which is an ensemble learning method.

accuracy\_score from sklearn.metrics: This function computes the accuracy classification score.

python code

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

import joblib

Loading the Dataset:

df = pd.read\_csv("healthcare\_dataset.csv"): This line loads the healthcare dataset from a CSV file into a DataFrame named df.

Data Preprocessing:

Irrelevant columns are dropped using df.drop(columns=...).

Rows with missing values are dropped using df.dropna().

Categorical variables are encoded using one-hot encoding with pd.get\_dummies().

Feature Selection:

Relevant features are selected based on some prior analysis or domain knowledge. Here, features related to age, gender, and blood group type are selected.

Model Training:

The dataset is split into training and testing sets using train\_test\_split().

A Random Forest Classifier is initialized and trained on the training data using clf.fit().

Model Evaluation:

The trained model is used to make predictions on the testing data (X\_test) using clf.predict().

The accuracy of the model is evaluated by comparing the predicted labels with the actual labels using accuracy\_score().

Python code

# Model Training

# Split data 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)

# Train a Random Forest Classifier

clf = RandomForestClassifier(random\_state=42)

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

# Model Evaluation

# Predict on the testing data

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

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

joblib.dump(clf, "random\_forest\_model.pkl")

Saving the Trained Model:

joblib.dump(clf, "random\_forest\_model.pkl"): This line saves the trained Random Forest Classifier (clf) to a file named "random\_forest\_model.pkl" using the joblib.dump() function.

So, in summary, this code loads a healthcare dataset, preprocesses it, selects relevant features, trains a Random Forest Classifier, evaluates its accuracy, and finally saves the trained model to a file.

**3.Web Application:** Built using the Flask web framework for the backend and HTML/CSS for the frontend.

Importing Libraries:

Flask: Flask is a web framework for Python.

render\_template: This function renders HTML templates.

request: This module handles incoming HTTP requests.

joblib: This library loads the trained machine learning model.

pandas as pd: Pandas is used for data manipulation and analysis.

Python code

from flask import Flask, render\_template, request

import joblib

import pandas as pd

Initializing Flask App:

app = Flask(\_\_name\_\_): This creates a Flask application instance.

Loading the Trained Model:

model = joblib.load('random\_forest\_model.pkl'): This loads the trained machine learning model (Random Forest Classifier) from the file "random\_forest\_model.pkl" using joblib.load().

Defining Routes:

'/' route: This route renders the home page (index.html).

'/predict' route: This route is for predicting medical conditions based on user input. It handles POST requests from the form submission.

Handling Form Submission and Prediction:

Inside the /predict route, if the request method is POST:

User input (name, age, blood group, gender) is retrieved from the form.

Blood group and gender are mapped to one-hot encoded format.

Input data is structured into a DataFrame.

Feature names used during model training are retrieved.

Input data columns are reordered to match the order of feature names.

Prediction is made using the loaded model (model.predict()).

The predicted medical condition is extracted from the prediction.

User input along with the predicted condition is saved to a CSV file.

The predicted condition is passed to the result page (result.html).

Running the Application:

app.run(debug=True): This starts the Flask application in debug mode.

Python code

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

app.run(debug=True)

This Flask application allows users to input their data, predicts their medical condition using the trained machine learning model, and displays the result on a web page.

Installation

1.Clone the repository from GitHub: GitHub Repository Link

2.Install the required Python libraries using pip:

pip install flask scikit-learn pandas matplotlib

3.Run the Flask application:

python app.py

4.Access the web application through a web browser at http://localhost:5000.

Contributors

• Shreyansh Parashar