NAME: RUCHIR MAKARAND ADNAIK  
ROLL: 08  
SRN: 202200282  
DIV: B  
BATCH: B1

ASSIGNMENT 09

CODE:

# Step 1: Importing Libraries

import pandas as pd

import numpy as np

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

import seaborn as sns

import matplotlib.pyplot as plt

# Step 2: Data Collection

data = pd.read\_csv('gym\_membership.csv')  # Load your dataset here

data = data.fillna(0)  # Fill missing values with 0 or an appropriate method

# Step 3: Exploratory Data Analysis (EDA)

print(data.head())  # Show the first few rows of the dataset

print(data.info())  # Get info on the data types and missing values

print(data.describe())  # Get statistical summary of the dataset

# Step 4: Feature Engineering

# Convert categorical variables to numerical if needed

data['personal\_training'] = data['personal\_training'].astype(int)  # Convert 'personal\_training' to numeric (0, 1)

data['attend\_group\_lesson'] = data['attend\_group\_lesson'].apply(lambda x: 1 if x == 'TRUE' else 0)  # Convert to binary

data['avg\_time\_check\_in'] = pd.to\_timedelta(data['avg\_time\_check\_in']).dt.total\_seconds() / 60  # Convert time to minutes

data['avg\_time\_check\_out'] = pd.to\_timedelta(data['avg\_time\_check\_out']).dt.total\_seconds() / 60  # Convert time to minutes

data['avg\_time\_in\_gym'] = pd.to\_timedelta(data['avg\_time\_in\_gym']).dt.total\_seconds() / 60  # Convert time to minutes

# We will drop non-numerical columns that won't be used in prediction

data = data.drop(columns=['id', 'first\_name', 'gender', 'abonoment\_type', 'fav\_group\_lesson', 'visit\_per\_week', 'days\_per\_week'])

# Target variable: Let's assume 'personal\_training' indicates whether they are likely to show up

y = data['personal\_training']

X = data.drop('personal\_training', axis=1)  # All other columns as features

# Step 5: Splitting the dataset into training and testing sets (Train-Test Split)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  # 70% training, 30% testing

# Step 6: Model Training

model = GaussianNB()  # Create an instance of the Gaussian Naive Bayes model

model.fit(X\_train, y\_train)  # Fit the model on training data

# After the data processing steps and before the prediction

# Step 7: Model Evaluation

y\_pred = model.predict(X\_test)  # Make predictions on the test set

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

conf\_matrix = confusion\_matrix(y\_test, y\_pred)  # Get confusion matrix

class\_report = classification\_report(y\_test, y\_pred)  # Get classification report

# Print evaluation metrics

print(f'Accuracy: {accuracy:.2f}')

print('Confusion Matrix:')

print(conf\_matrix)

print('Classification Report:')

print(class\_report)

# Check the columns of X

print("Columns in feature set (X):", X.columns)

print("Number of columns in feature set (X):", len(X.columns))

# Visualization of the confusion matrix

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

            xticklabels=['Did not show up', 'Showed up'],

            yticklabels=['Did not show up', 'Showed up'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

# Step 8: Single User Input Testing

# Create a function for predicting a single user input

def predict\_user\_input(input\_data):

    # Ensure input data matches the expected format (DataFrame)

    input\_df = pd.DataFrame([input\_data], columns=X.columns)

    prediction = model.predict(input\_df)  # Predict using the trained model

    return "Showed Up" if prediction[0] == 1 else "Did Not Show Up"

# Example of user input

# Adjust the user input to match the number of features in X

user\_input = [30, 130, 160, 75, 1, 0]  # Example input; change as needed

result = predict\_user\_input(user\_input)

print(f'Prediction for user input {user\_input}: {result}')

OUTPUT:



