```
# Step 1: Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import pickle
import tkinter as tk
from tkinter import messagebox
import joblib

# Step 2: Load Dataset
data = pd.read_csv("diabetes_012_health_indicators_BRFSS2015.csv")
print(" ✅ Dataset Loaded Successfully!\n")
print(data.head())

# Step 3: Basic Info
print("\nDataset Information:")
print(data.info())
print("\nMissing Values:\n", data.isnull().sum())

# Step 4: Visualization
plt.figure(figsize=(8, 5))
sns.countplot(x='Diabetes_012', data=data, palette='Set2')
plt.title("Count of Diabetes Categories (0: No, 1: Prediabetes, 2: Diabetes)")
```

```
plt.show()

# Step 5: Feature Selection
selected_features = [
    'HighBP', 'HighChol', 'BMI', 'Smoker', 'PhysActivity',
    'Fruits', 'Veggies', 'HvyAlcoholConsump', 'GenHlth', 'Age'
]
X = data[selected_features]
y = data['Diabetes_012']

# Step 6: Split Data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
print(f"\nTraining Data Shape: {X_train.shape}")
print(f"Testing Data Shape: {X_test.shape}")

# Step 7: Standardize Data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 8: Train Model
model = LogisticRegression(max_iter=2000, multi_class='ovr')
model.fit(X_train_scaled, y_train)

# Step 9: Evaluate Model
y_pred = model.predict(X_test_scaled)
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"\n 🎯 Model Accuracy: {accuracy*100:.2f}%")
cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:\n", cm)
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# 🔥 Add Heatmap for Confusion Matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title("Confusion Matrix Heatmap")
plt.xlabel("Predicted Labels")
plt.ylabel("Actual Labels")
plt.show()

# Step 10: Save Model and Scaler
with open('diabetes_model.pkl', 'wb') as file:
    pickle.dump(model, file)

with open('scaler.pkl', 'wb') as file:
    pickle.dump(scaler, file)

print("\n 💾 Model and scaler saved successfully!")



# ---------- GUI Section ----------

import tkinter as tk
```

```
from tkinter import messagebox
import joblib

# Load trained model and scaler
model = joblib.load("diabetes_model.pkl")
scaler = joblib.load("scaler.pkl")

root = tk.Tk()
root.title(" 💧 Diabetes Prediction System")
root.configure(bg="#dbe9f4")

tk.Label(
    root,
    text=" 💧 Diabetes Prediction System",
    font=("Arial", 20, "bold"),
    bg="#dbe9f4"
).pack(pady=10)

frame = tk.Frame(root, bg="#dbe9f4")
frame.pack(padx=20, pady=10)

# Define fields
fields = {
    "HighBP (1=Yes, 0=No)": None,
    "HighChol (1=Yes, 0=No)": None,
    "BMI (e.g. 24.5)": None,
    "Smoker (1=Yes, 0=No)": None,
    "PhysActivity (1=Yes, 0=No)": None,
```

```
    "Fruits (1=Yes, 0=No)": None,
    "Veggies (1=Yes, 0=No)": None,
    "HvyAlcoholConsump (1=Yes, 0=No)": None,
    "GenHlth (1=Excellent → 5=Poor)": None,
    "Age (1=18-24 ... 13=80+)": None
}

entries = {}
for i, (label, _) in enumerate(fields.items()):
    tk.Label(frame, text=label, bg="#dbe9f4", font=("Arial", 11)).grid(row=i, column=0, sticky="w", pady=5)
    entry = tk.Entry(frame)
    entry.grid(row=i, column=1, pady=5)
    entries[label] = entry


def predict_diabetes():
    try:
        input_data = [float(entries[label].get()) for label in entries]
        input_data_scaled = scaler.transform([input_data])
        prediction = model.predict(input_data_scaled)[0]

        # Add Recommendation Section
        if prediction == 0:
            result = "✅ The person is NOT diabetic."
            recommendation = (
                "💡 Recommendation:\n"
                "• Maintain a balanced diet.\n"
```

```
            "• Exercise regularly.\n"
            "• Continue regular health checkups."
        )
    elif prediction == 1:
        result = "⚠️ The person is Prediabetic."
        recommendation = (
            "💡 Recommendation:\n"
            "• Monitor your blood sugar levels frequently.\n"
            "• Reduce sugar and carbohydrate intake.\n"
            "• Maintain a healthy weight through exercise."
        )
    else:
        result = "🚨 The person is Diabetic."
        recommendation = (
            "💡 Recommendation:\n"
            "• Consult your doctor for a proper treatment plan.\n"
            "• Follow a low-sugar diet and take prescribed medicines.\n"
            "• Avoid alcohol and smoking."
        )

    messagebox.showinfo("Prediction Result", f"{result}\n\n{recommendation}")

  except ValueError:
    messagebox.showerror("Error", "Please enter valid numeric values.")


def exit_app():
  root.destroy()
```

```
# Buttons
tk.Button(
    root, text="🔍 Predict", command=predict_diabetes,
    bg="#0078D7", fg="white", font=("Arial", 12, "bold"), width=15
).pack(pady=10)

tk.Button(
    root, text="❌ Exit", command=exit_app,
    bg="#d9534f", fg="white", font=("Arial", 12, "bold"), width=15
).pack(pady=5)

root.mainloop()
```