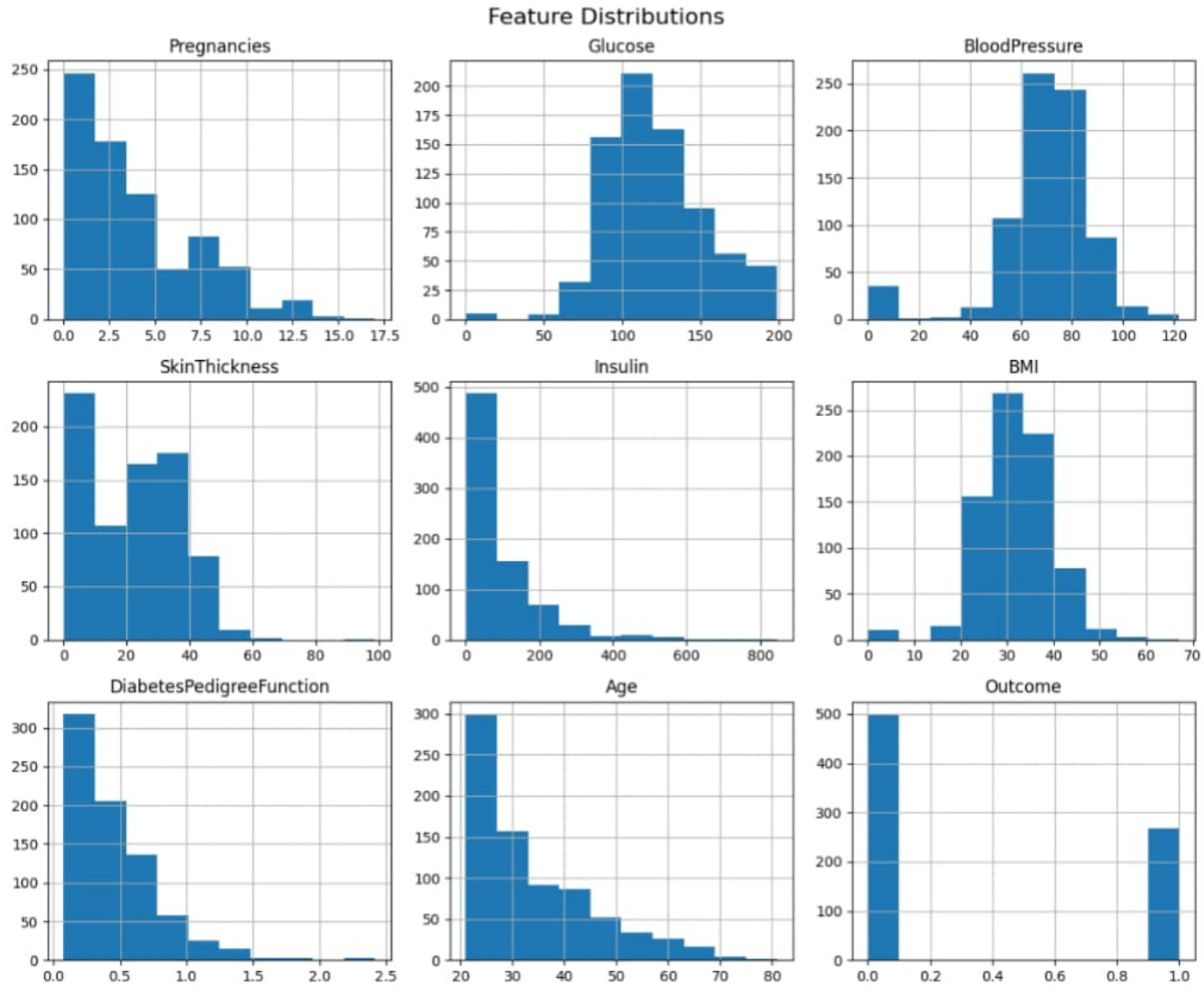
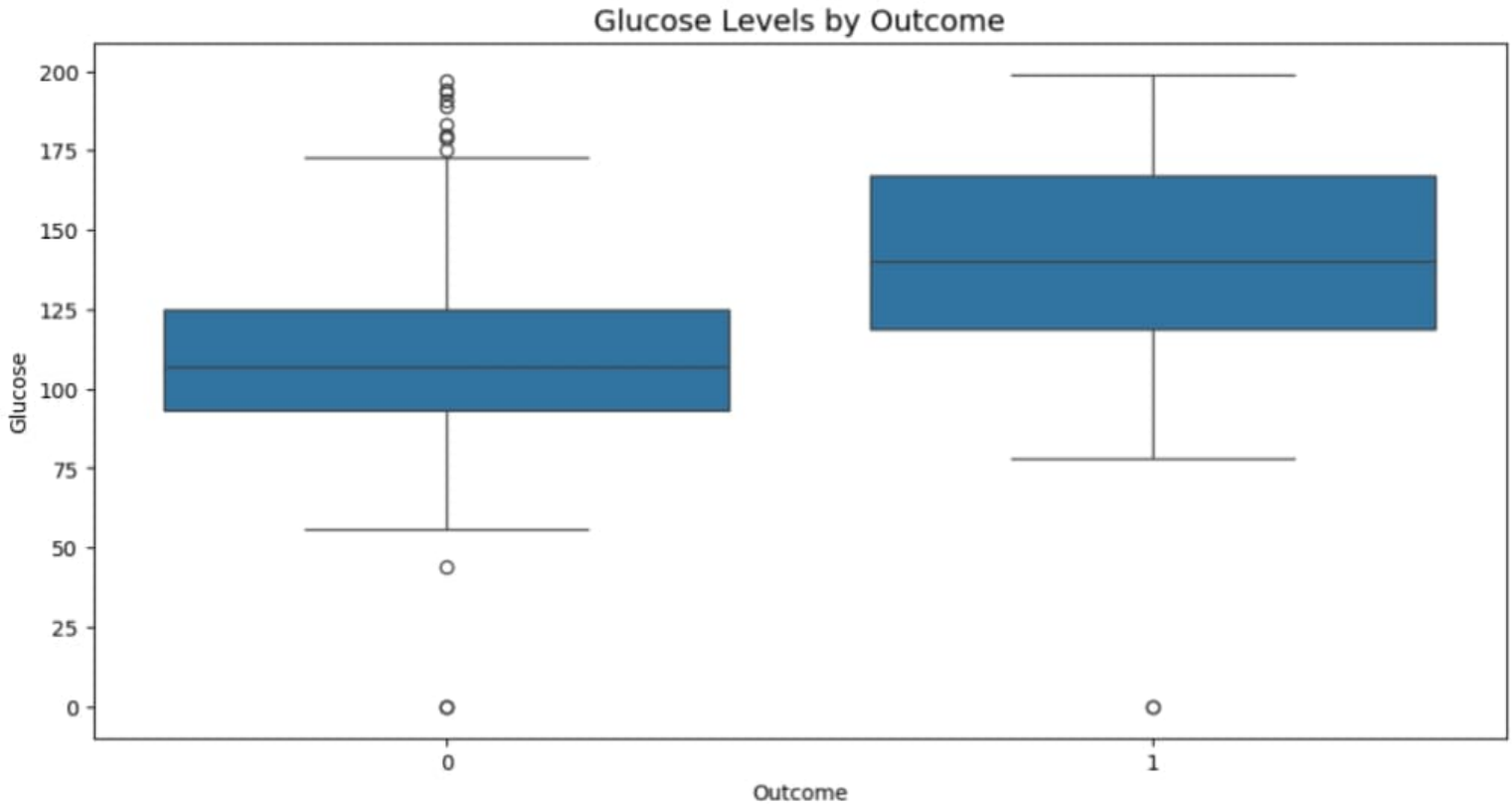
```python
# ... (your existing code) ...

# Before Preprocessing
print("\nBefore Preprocessing (Original Data):\n")
print(df.head())

# ... (your existing preprocessing code: train_test_split, etc.) ...

# After Preprocessing
print("\nAfter Preprocessing:\n")
print(f"  X_train shape: {X_train.shape}")
print(f"  X_test shape: {X_test.shape}")
print(f"  y_train shape: {y_train.shape}")
print(f"  y_test shape: {y_test.shape}")

# ... (rest of your code) ...
```

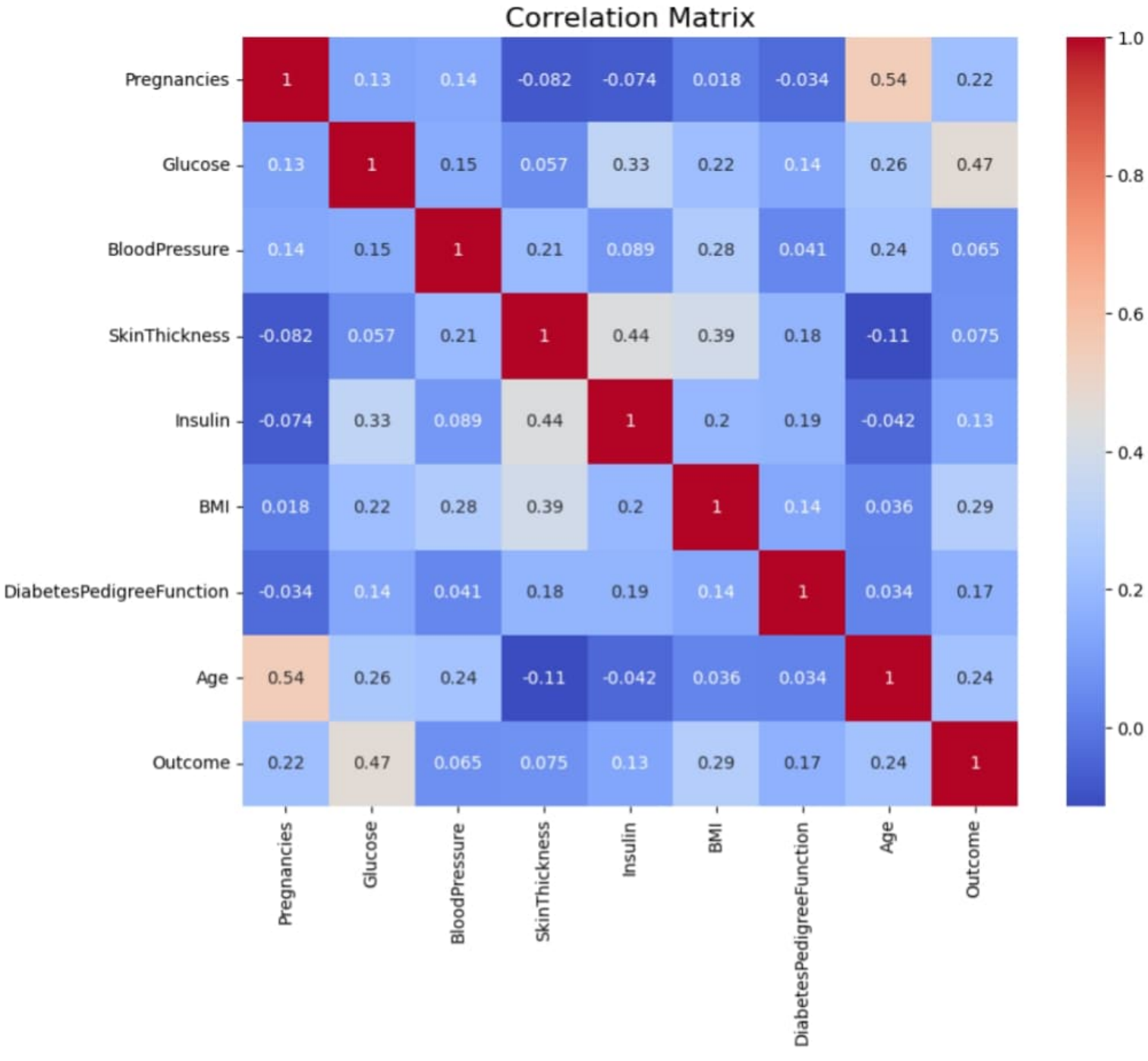
[26]
```
import matplotlib.pyplot as plt
import pandas as pd

# Step 1: Reload the dataframe (df)
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv'
columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
           'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
df = pd.read_csv(url, names=columns)

# Step 2: Now create the histograms
df.hist(figsize=(12, 10))
plt.suptitle("Feature Distributions", fontsize=16)
plt.tight_layout()
plt.show()
```



```
[27] import seaborn as sns

     # Create box plots for features grouped by outcome
     plt.figure(figsize=(12, 6))
     sns.boxplot(x="Outcome", y="Glucose", data=df)
     plt.title("Glucose Levels by Outcome", fontsize=14)
     plt.show()
```



```
[29] import seaborn as sns

     correlation_matrix = df.corr()
     plt.figure(figsize=(10, 8))
     sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
     plt.title("Correlation Matrix", fontsize=16)
     plt.show()
```



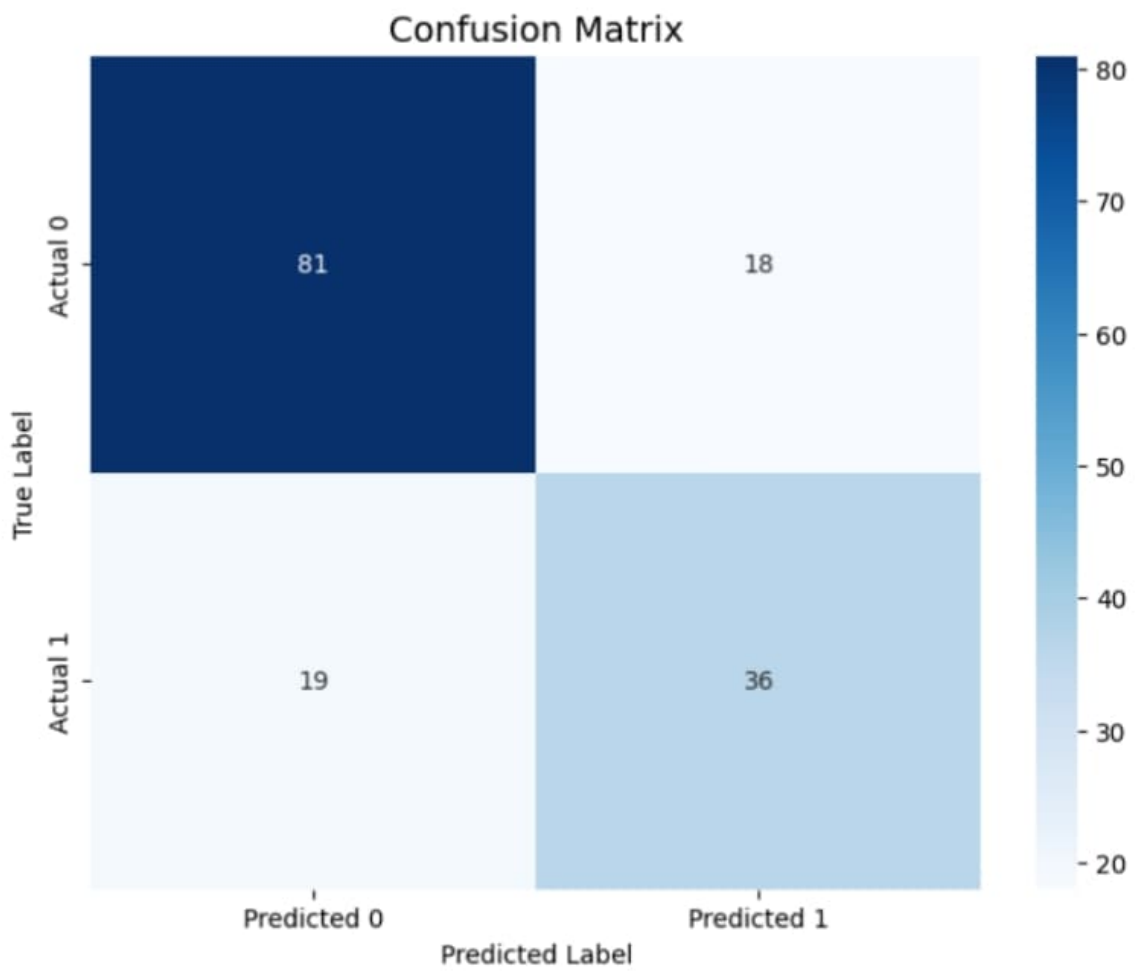
```python
# Step 4: Evaluate model
accuracy = accuracy_score(y_test, model.predict(X_test))
print(f"Model Accuracy: {accuracy * 100:.2f}%")
```

```
Model Accuracy: 72.73%
```

```
[30] from sklearn.metrics import f1_score
     import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.ensemble import RandomForestClassifier
     import joblib # Make sure joblib is imported

     # Step 1: Reload the dataframe (if needed) and train model
     # (This assumes you've run the cell that loaded and trained the model previously)
     # Otherwise, you need to reload your dataset and train your model again here:
     # url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv'
     # ... (load and train the model as in your original code) ...

     try:
         model = joblib.load('diabetes_model.pkl')  # Attempt to load existing model
     except FileNotFoundError:
         print("Model file not found. Please re-train the model.")
         # Add code to load dataset and train the model
         # This prevents crashes if the model file is missing
         url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv
         columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
                    'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
         df = pd.read_csv(url, names=columns)
         X = df.drop('Outcome', axis=1)
         y = df['Outcome']
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
         model = RandomForestClassifier()
         model.fit(X_train, y_train)

     # Generate predictions using the model
     predictions = model.predict(X_test)

     # Now calculate the F1 score
     f1 = f1_score(y_test, predictions)
     print(f"\n{'F1 Score:':<15} {f1:.2f}")
```

```
F1 Score:       0.66
```

[31]
```python
# prompt: code for confusion matrix

from sklearn.metrics import confusion_matrix
import seaborn as sns

# Assuming 'y_test' and 'predictions' are defined from previous code

cm = confusion_matrix(y_test, predictions)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title("Confusion Matrix", fontsize=14)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



```
[32] !pip install gradio
     import gradio as gr
     import pandas as pd
     import joblib

     # Load the trained model
     model = joblib.load('diabetes_model.pkl')

     # Define the prediction function
     def predict_diabetes(Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigr
         user_data = pd.DataFrame({
             'Pregnancies': [Pregnancies],
             'Glucose': [Glucose],
             'BloodPressure': [BloodPressure],
             'SkinThickness': [SkinThickness],
             'Insulin': [Insulin],
             'BMI': [BMI],
             'DiabetesPedigreeFunction': [DiabetesPedigreeFunction],
             'Age': [Age]
         })
         prediction = model.predict(user_data)[0]
         return "Likely Diabetic" if prediction == 1 else "Unlikely Diabetic"

     # Create the Gradio interface
     iface = gr.Interface(
         fn=predict_diabetes,
         inputs=[
             gr.Number(label="Pregnancies"),
             gr.Number(label="Glucose"),
             gr.Number(label="Blood Pressure"),
             gr.Number(label="Skin Thickness"),
             gr.Number(label="Insulin"),
             gr.Number(label="BMI"),
             gr.Number(label="Diabetes Pedigree Function"),
             gr.Number(label="Age")
         ],
         outputs="text",
         title="Diabetes Prediction App"
     )

     # Launch the interface
     iface.launch()
```

Diabetes Prediction App

Pregnancies

6

Glucose

147

Blood Pressure

72

Skin Thickness

35

Insulin

0

output

Likely Diabetic

Flag

35

Insulin

0

BMI

33.6

Diabetes Pedigree Function

0.627

Age

50

Clear

Submit

Use via API · Built with Gradio · Settings