Why Logistic Regression?

"""

- Interpretability for healthcare applications
- Handles binary classification well
- Less prone to overfitting than complex models

"""

```
# Logistic regression model to predict

import pandas as pd

file_path = '/content/drive/MyDrive/ML related datasets/diabetes.csv'
df = pd.read_csv(file_path)
print(df.head())
print('\n',df.shape)
print('\n',df.info())
```

Show hidden output

```
# checking if null values
print(df.isnull().sum())
```

Show hidden output

```
# if null exist handling it using mode() method
for col in df.select_dtypes(include=['object']):
    df[col].fillna(df[col].mode()[0], inplace=True)

# dropping outcome column from dataframe and storing it as y
X = df.drop(columns=['Outcome'])
y = df['Outcome']

# applyiung z_score scalling(standard scaling technique)
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
X = scaler.fit_transform(X)

# splitting data into training and testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,
                                                    test_size=0.2,
                                                    random_state=42)



# training model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
```

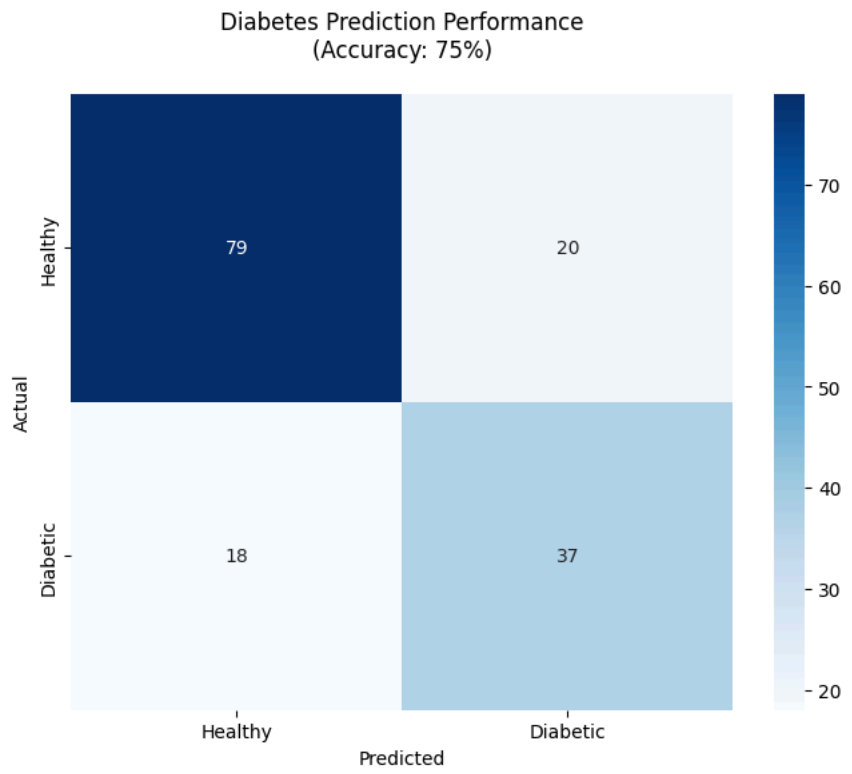
```
▾ LogisticRegression
LogisticRegression()
```

```
# Model Evaluation (lets check our model how it performs)

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
y_pred = model.predict(X_test)

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy : .2f}")

# classification report
print("\n classification report")
print(classification_report(y_test, y_pred))

# confusion matrix
print("\n confusion matrix")
print(confusion_matrix(y_test, y_pred))
```

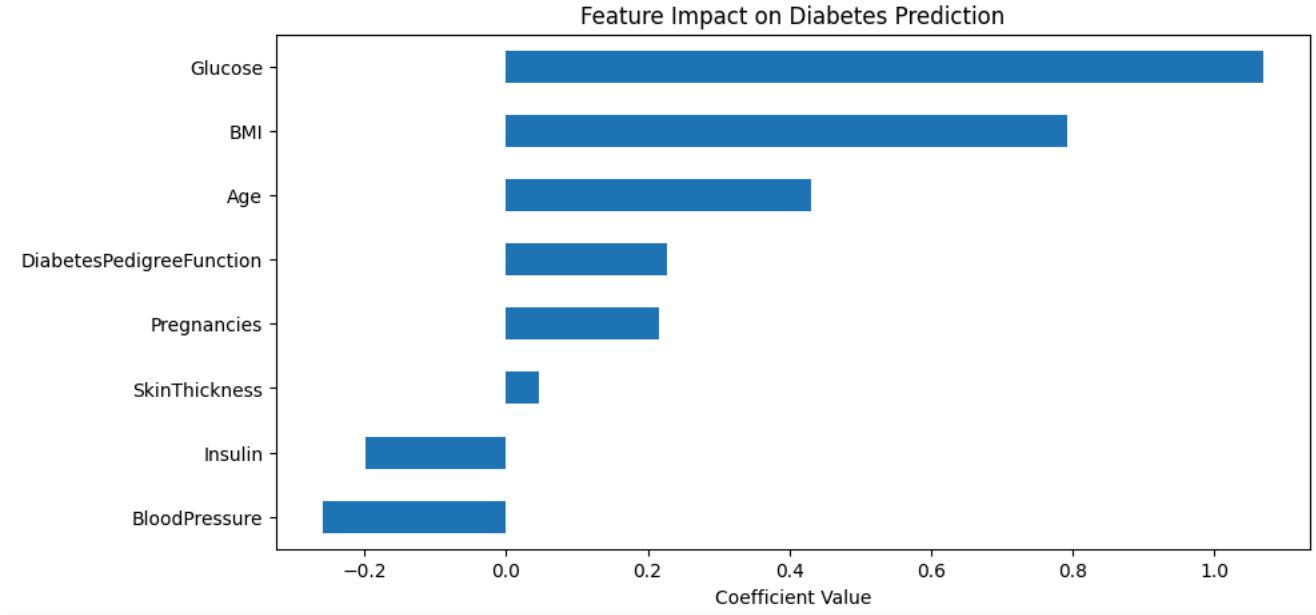
```
Accuracy:  0.75

 classification report
              precision    recall  f1-score   support

           0       0.81      0.80      0.81        99
           1       0.65      0.67      0.66        55

    accuracy                           0.75       154
   macro avg       0.73      0.74      0.73       154
weighted avg       0.76      0.75      0.75       154


 confusion matrix
[[79 20]
 [18 37]]
```

```
# visualizing its heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred),
            annot=True, fmt='d',
            cmap='Blues',
            xticklabels=['Healthy', 'Diabetic'],
            yticklabels=['Healthy', 'Diabetic'])
plt.title("Diabetes Prediction Performance\n(Accuracy: 75%)", pad=20)
```

```
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.savefig('confusion_matrix_clean.png')  # Save for portfolio
```



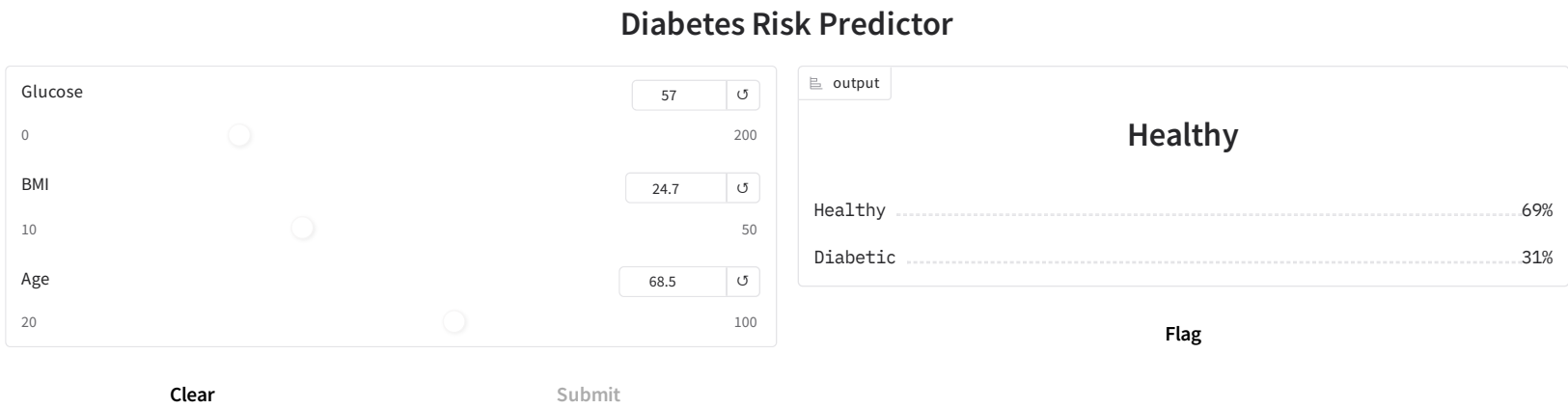

```
# Feature Importance Visualization (Add after model training)
import matplotlib.pyplot as plt
plt.figure(figsize=(10,5))
pd.Series(model.coef_[0], index=df.columns[:-1]).sort_values().plot(kind='barh')
plt.title("Feature Impact on Diabetes Prediction")
plt.xlabel("Coefficient Value")
plt.show()
```




```
!pip install gradio huggingface_hub --quiet

import gradio as gr
import joblib
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load model (replace with your actual model)
# model = joblib.load('model.pkl')
# scaler = joblib.load('scaler.pkl')

# Mock function (replace with your real predict function)
def predict(glucose, bmi, age):
    # Your prediction logic here
    risk_percentage = min(100, glucose * 0.2 + bmi * 0.5 + age * 0.1)  # Example calculation
    return {"Diabetic": risk_percentage/100, "Healthy": 1-risk_percentage/100}

# Gradio Interface
```

```
demo = gr.Interface(
    fn=predict,
    inputs=[
        gr.Slider(0, 200, label="Glucose"),
        gr.Slider(10, 50, label="BMI"),
        gr.Slider(20, 100, label="Age")
    ],
    outputs="label",
    title="Diabetes Risk Predictor"
)

demo.launch(debug=True)  # Test locally first
```

```
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio app to work, sharing must be enabled. Automatically setting `share=True` (you can turn this

Colab notebook detected. This cell will run indefinitely so that you can see errors and logs. To turn off, set debug=False in launch().
* Running on public URL: https://e0679426b4e747c278.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces
```




```
Keyboard interruption in main thread... closing server.
Killing tunnel 127.0.0.1:7860 <> https://e0679426b4e747c278.gradio.live
```

```
from huggingface_hub import notebook_login
notebook_login()  # Follow the link to get your HF token

!huggingface-cli login --token YOUR_HF_TOKEN  # Paste your token here
```

Show hidden output

```
# Save your app to a repo
!git config --global credential.helper store

# Create new Space (run only once)
!huggingface-cli repo create diabetes-risk-predictor --type space --space-sdk gradio

# Clone your Space
!git clone https://huggingface.co/spaces/AhmadSolail/diabetes-risk-predictor
%cd diabetes-risk-predictor

# Add your files
with open("app.py", "w") as f:
    f.write("""
import gradio as gr
def predict(glucose, bmi, age):
    return {"Risk": glucose*0.002 + bmi*0.01 + age*0.005}
demo = gr.Interface(fn=predict, inputs=["number", "number", "number"], outputs="label")
demo.launch()
""")

with open("requirements.txt", "w") as f:
```