The Diabetes Dataset: Logistic Regression

1. Importing necessary libraries

First, we need to import the libraries required for our analysis: `numpy` and `pandas` for data manipulation, `sklearn` for building the logistic regression model and evaluating its performance, and `matplotlib` and `seaborn` for creating a confusion matrix heatmap.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_diabetes
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, classification_report, confusion_matrix
```

2. Loading the dataset

Here, we use the Diabetes dataset available under the `sklearn` module in Python. The continuous target variable is converted into a binary format (0 or 1) where values are classified based on whether they are above or below the median.

```
diabetes = load_diabetes()
X, y = diabetes.data, diabetes.target

# Binarizing the target variable
y_bin = (y > np.median(y)).astype(int)
```

3. Splitting the data into training and testing sets

Next, we split the dataset into training and testing sets using the `train_test_split` function. We specify `test_size=0.2` to reserve 20% of the data for testing, and `random_state=42` to ensure the data is split consistently every time we run the code.

```
X_train, X_test, y_train, y_test = train_test_split(X, y_bin, test_size=0.2, random_state=42)
```

4. Standardising the features

This section standardises the feature variables to improve model performance. The features are scaled so they have a mean of 0 and a standard deviation of 1. The scaler is fitted to the training data and then applied to both training and testing sets.

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

5. Creating and training the model

Now, we create an instance of the logistic regression model and train it. The model learns the relationship between the features (`X_train`) and the target labels (`y_train`).

```
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

6. Making predictions

Once the model is trained, we use it to make predictions on the test data (`X_test`). The model generates predicted labels (`y_pred`) for the test set, which we will compare with the actual labels (`y_test`) to evaluate its performance.

```
y_pred = model.predict(X_test)
```

7. Evaluating the model

Now, we calculate the accuracy of the model and print a detailed classification report, which includes precision, recall, and F1-score for each class, providing insight into the model's performance.

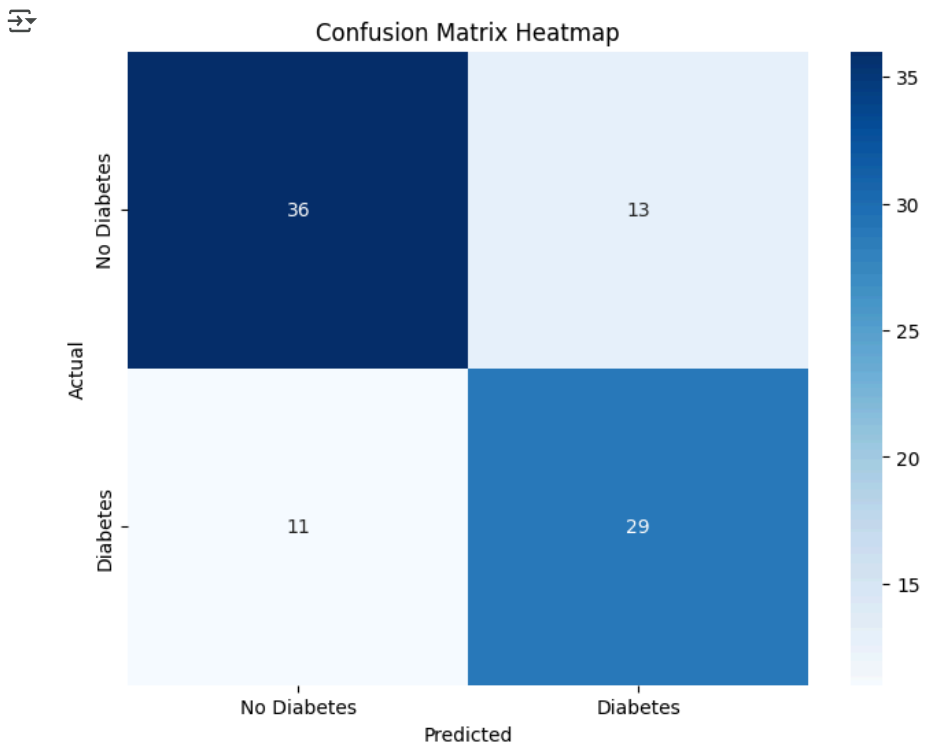
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 73.03%

Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.73      0.75        49
           1       0.69      0.72      0.71        40

    accuracy                           0.73        89
   macro avg       0.73      0.73      0.73        89
weighted avg       0.73      0.73      0.73        89
```

8. Generating the confusion matrix

We also visualise the confusion matrix, which shows how many instances of each class were correctly or incorrectly classified. This helps us understand specific areas where the model may be confusing certain classes.

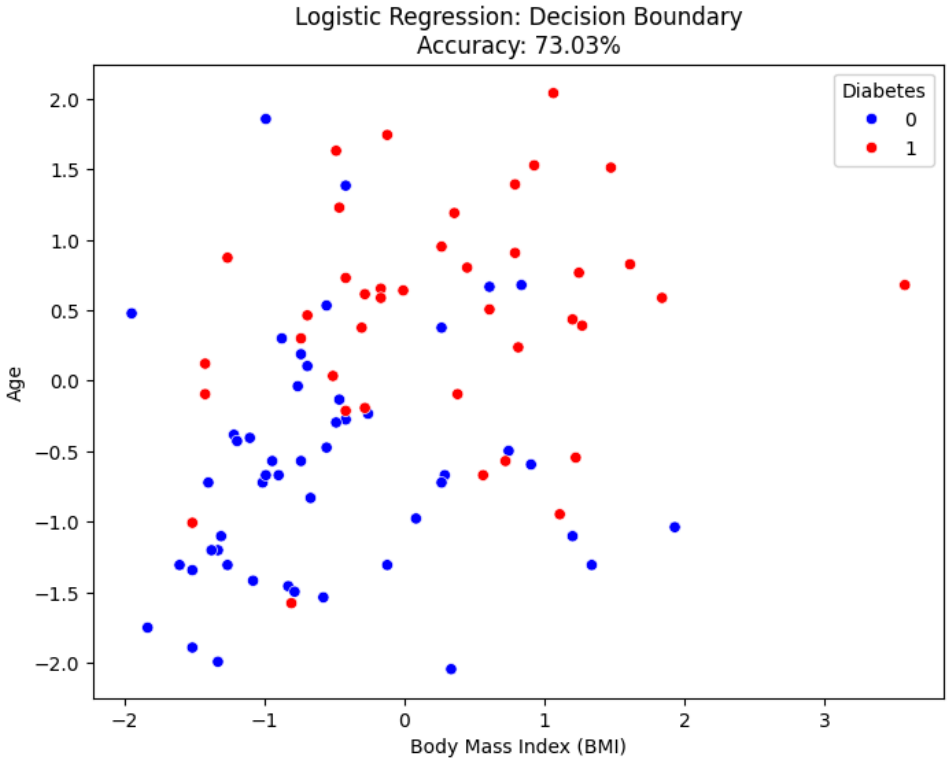
```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Diabetes', 'Diabetes'], yticklabels=['No Diabetes', 'Diabetes'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix Heatmap')
plt.show()
```



9. Visualising the results

Finally, we create a scatterplot to visualise the decision boundary of the logistic regression model.

```
plt.figure(figsize=(8,6))
sns.scatterplot(x=X_test[:, 2], y=X_test[:, 8], hue=y_test, palette={0: 'blue', 1: 'red'}, marker='o')
plt.xlabel("Body Mass Index (BMI)")
plt.ylabel("Age")
plt.title("Logistic Regression: Decision Boundary\nAccuracy: {:.2f}%".format(accuracy * 100))
plt.legend(title="Diabetes", loc="upper right")
plt.show()
```


Logistic Regression: Decision Boundary
Accuracy: 73.03%
Diabetes
0
1
Age
Body Mass Index (BMI)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2
−1
0
1
2
3