Assignment No 1

Name: Ashutosh Shivthare

Roll No: C43364

Batch :B16

Recursive Program:

```
def fibonacci_Recursive(n):
    sequence = []
    for i in range(n):
        if i == 0:
            sequence.append(0)
        elif i == 1:
            sequence.append(1)
        else:
            sequence.append(sequence[i - 1] + sequence[i - 2])
    return sequence

# Example usage:
n = int(input("Enter the Number:"))  # Change this value for a different length of the
sequence
fib_sequence = fibonacci_Recursive(n)
for num in fib_sequence:
    print(num)
```

Output:

```
Enter the Number:10
0
1
1
2
3
5
8
13
21
34
```

Iterative Program:

```
def fibonacci_iterative(n):
    if n <= 0:
        return []
    elif n == 1:
        return [0]

    sequence = [0, 1]
    for _ in range(2, n):
        sequence.append(sequence[-1] + sequence[-2])
    return sequence
```

Output:

```
Enter the Number:10
Iterative Fibonacci sequence: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Assignment No 2

Name: Ashutosh Shivthare

Roll No: C43364

Batch :B16

Program:

```
import heapq
from collections import defaultdict, namedtuple

HuffmanNode = namedtuple('HuffmanNode', ['freq', 'char', 'left', 'right'])

def build_huffman_tree(frequency):
    heap = [HuffmanNode(freq, char, None, None) for char, freq in frequency.items()]
    heapq.heapify(heap)

    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        merged = HuffmanNode(left.freq + right.freq, None, left, right)
        heapq.heappush(heap, merged)

    return heap[0]

def build_codes(node, prefix="", codebook=None):
    if codebook is None: codebook = {}
    if node.char is not None:
        codebook[node.char] = prefix
    else:
        build_codes(node.left, prefix + "0", codebook)
        build_codes(node.right, prefix + "1", codebook)
    return codebook

def huffman_encoding(data):
    frequency = defaultdict(int, {char: data.count(char) for char in set(data)})
    root = build_huffman_tree(frequency)
    huffman_codes = build_codes(root)
    return ''.join(huffman_codes[char] for char in data), huffman_codes

def huffman_decoding(encoded_data, huffman_codes):
    reverse_codes = {v: k for k, v in huffman_codes.items()}
    current_code, decoded_data = "", []
    for bit in encoded_data:
        current_code += bit
        if current_code in reverse_codes:
            decoded_data.append(reverse_codes[current_code])
            current_code = ""
    return ''.join(decoded_data)
```

```
# Get user input
data = input("Enter a string to encode: ")
encoded_data, huffman_codes =
huffman_encoding(data

print("Original Data:", data)
print("Encoded Data:", encoded_data)
print("Huffman Codes:", huffman_codes)
print("Decoded Data:", huffman_decoding(encoded_data, huffman_codes))
```

Output:

```
Enter a string to encode: python
Original Data: python
Encoded Data: 1110100100110101
Huffman Codes: {'t': '00', 'y': '01', 'h': '100', 'n': '101', 'o': '110', 'p': '111'}
Decoded Data: python
```

Assignment No 3

Name: Ashutosh Shivthare
Roll No: C43364
Batch :B16

Program:

```
class Item:
    def init (self, value, weight):
        self.value = value
        self.weight = weight
        self.cost = value / weight  # Cost per weight unit

def fractional_knapsack(capacity, items):
    # Sort items by cost (value/weight) in descending order
    items.sort(key=lambda item: item.cost, reverse=True)

    total_value = 0
    for item in items:
        if capacity == 0:
            break
        if item.weight <= capacity:
            # Take the whole item
            total_value += item.value
            capacity -= item.weight
        else:
            # Take the fraction of the remaining item
            total_value += item.cost * capacity = 0  #
            Knapsack is now full

    return total_value

# Example usage
if__name__== "_main__":
    n = int(input("Enter the number of items: "))
    items = []

    for _ in range(n):
        value = float(input("Enter the value of the item: "))
        weight = float(input("Enter the weight of the item: "))
        items.append(Item(value, weight))

    capacity = float(input("Enter the capacity of the knapsack: "))

    max_value = fractional_knapsack(capacity, items)
    print("Maximum value in the knapsack:", max_value)
```

Output:

Enter the number of items: 3
Enter the value of the item: 24
Enter the weight of the item: 15
Enter the value of the item: 25
Enter the weight of the item: 15
Enter the value of the item: 10
Enter the weight of the item: 18
Enter the capacity of the knapsack: 20
Maximum value in the knapsack: 33.0

Assignment No 4

Name: Ashutosh Shivthare

Roll No: C43364

Batch :B16

Program:

```
def is_safe(board, row, col):
    # Check this column on upper side
    for i in range(row):
        if board[i][col] == 1:
            return False

    # Check upper diagonal on left side
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if j < 0:
            break
        if board[i][j] == 1:
            return False

    # Check upper diagonal on right side
    for i, j in zip(range(row, -1, -1), range(col, len(board))):
        if j >= len(board):
            break
        if board[i][j] == 1:
            return False

    return True

def solve_n_queens(board, row):
    if row >= len(board):
        print_board(board)
        return True  # Change this to return all solutions instead of stopping at first

    for col in range(len(board)):
        if is_safe(board, row, col):
            board[row][col] = 1 # Place queen
            solve_n_queens(board, row + 1)  # Recur to place the rest
            board[row][col] = 0  # Backtrack

    return False

def print_board(board):
    for row in board:
        print(" ".join('Q' if x == 1 else '.' for x in row))
    print()
```

```
if__name__== "__main__":
    N = int(input("Enter the size of the board:"))
    board = [[0 for _ in range(N)] for _ in range(N)]


    # Place the first queen at (0, 0)
    board[0][0] = 1

    # Solve for the remaining queens
    solve_n_queens(board, 1)
```

Output:

```
PS C:\Users\ Ashutosh Shivthare \OneDrive\Documents\Academics\BE> & "C:/Python
3.11.3/python.exe" "c:/Users/Ashutosh Shivthare
OneDrive/Documents/Academics/BE/Lp3/Assignment No 4.py"
Enter the size of the board:8
Q . . . . . . .
. . . . Q . . .
..............Q
. . . . . Q . .
. . Q . . . . .
............Q .
. Q . . . . . .
. . . Q . . . .

Q . . . . . . .
. . . . . Q . .
..............Q
. . Q . . . . .
............Q .
. . . Q . . . .
. Q . . . . . .
. . . . Q . . .

Q . . . . . . .
............Q .
. . . Q . . . .
. . . . . Q . .
..............Q
. Q . . . . . .
. . . . Q . . .
. . Q . . . . .

Q . . . . . . .
............Q .
. . . . Q . . .
..............Q
. Q . . . . . .
. . . Q . . . .
. . . . . Q . .
. . Q . . . . .
```

Assignment No 6

Name: Ashutosh Shivthare

Roll No: C43364

Batch :B16

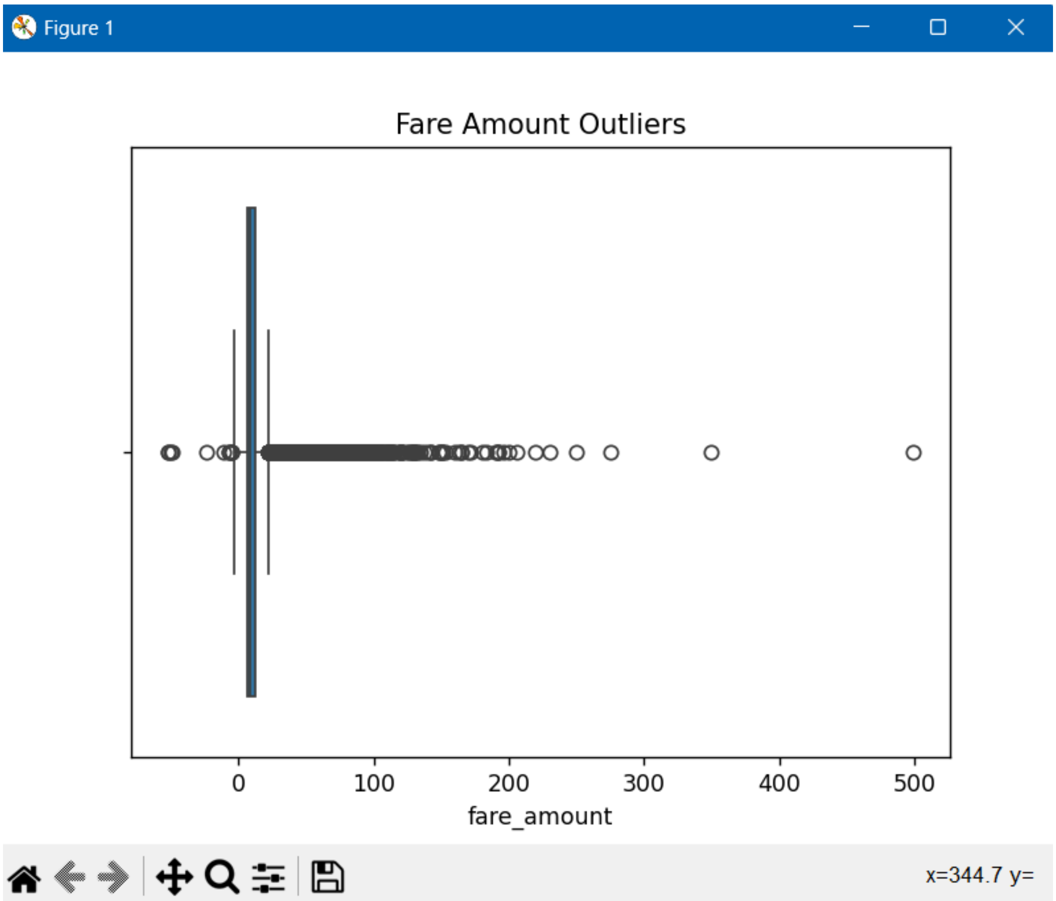
Program:

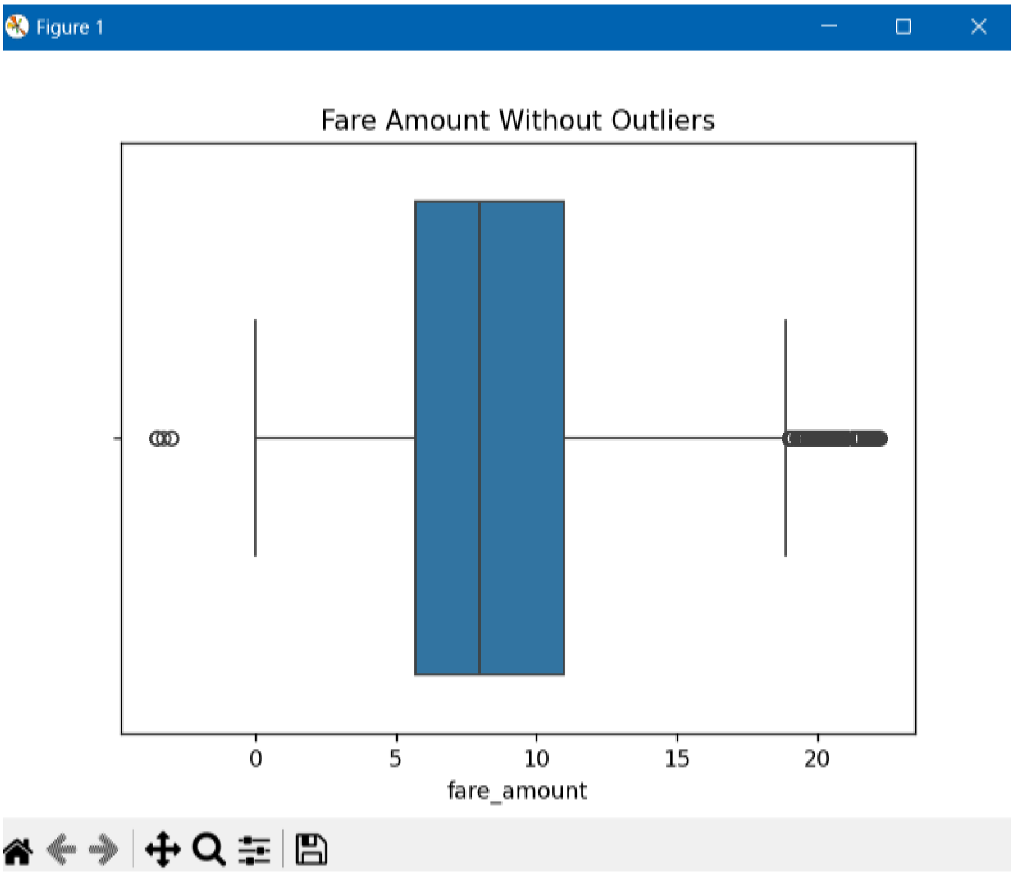
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error

# Load the dataset
data = pd.read_csv("uber.csv")

# 1. Pre-process the dataset
data["pickup_datetime"] = pd.to_datetime(data["pickup_datetime"])

# Check for missing values
missing_values = data.isnull().sum()
print("Missing values in the dataset:")
print(missing_values)

# Handle missing values
data.dropna(inplace=True)

# Ensure there are no more missing values
missing_values = data.isnull().sum()
print("Missing values after handling:")
print(missing_values)

# 2. Identify outliers
sns.boxplot(x=data["fare_amount"])
plt.title('Fare Amount Outliers')
plt.show()

# Calculate the IQR for the 'fare_amount' column
Q1 = data["fare_amount"].quantile(0.25)
Q3 = data["fare_amount"].quantile(0.75)
IQR = Q3 - Q1

# Define thresholds for outlier detection
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Remove outliers
data_no_outliers = data[(data["fare_amount"] >= lower_bound) & (data["fare_amount"] <=
upper_bound)]
```

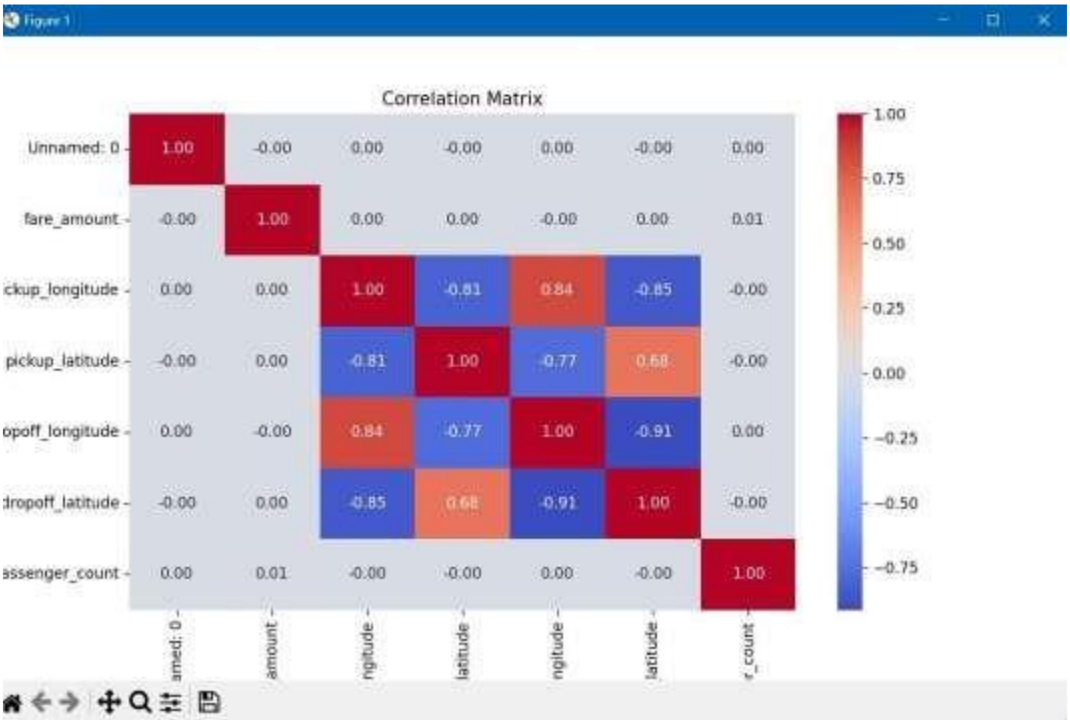
```
# Visualize the 'fare_amount' distribution without outliers
sns.boxplot(x=data_no_outliers["fare_amount"])
plt.title('Fare Amount Without Outliers')
plt.show()

# 3. Check the correlation
# Exclude non-numeric columns before calculating correlation
correlation_matrix = data_no_outliers.select_dtypes(include=[np.number]).corr()
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

# 4. Implement linear regression and random forest regression models
X = data_no_outliers[['pickup_longitude', 'pickup_latitude',
                      'dropoff_longitude', 'dropoff_latitude',
                      'passenger_count']]
y = data_no_outliers['fare_amount']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the linear regression model
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Create and train the random forest regression model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# 5. Evaluate the models
y_pred_lr = lr_model.predict(X_test)
y_pred_rf = rf_model.predict(X_test)

# Calculate R-squared (R2) and Root Mean Squared Error (RMSE) for both models
r2_lr = r2_score(y_test, y_pred_lr)
rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))

r2_rf = r2_score(y_test, y_pred_rf)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))

# Display results
print("Linear Regression - R2:", r2_lr)
print("Linear Regression - RMSE:", rmse_lr)

print("Random Forest Regression - R2:", r2_rf)
print("Random Forest Regression - RMSE:", rmse_rf)
```

Output:

```
Missing values in the dataset:
Unnamed: 0           0
key                  0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    1
dropoff_latitude     1
passenger_count      0
dtype: int64
Missing values after handling:
Unnamed: 0           0
key                  0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64
```







Linear Regression – R2: 8.29713376748753e-05
Linear Regression – RMSE: 4.136624287486402
Random Forest Regression – R2: 0.7052136223044838
Random Forest Regression – RMSE: 2.2460416246528774

Assignment No 7

Name: Ashutosh Shivthare

Roll No: C43364

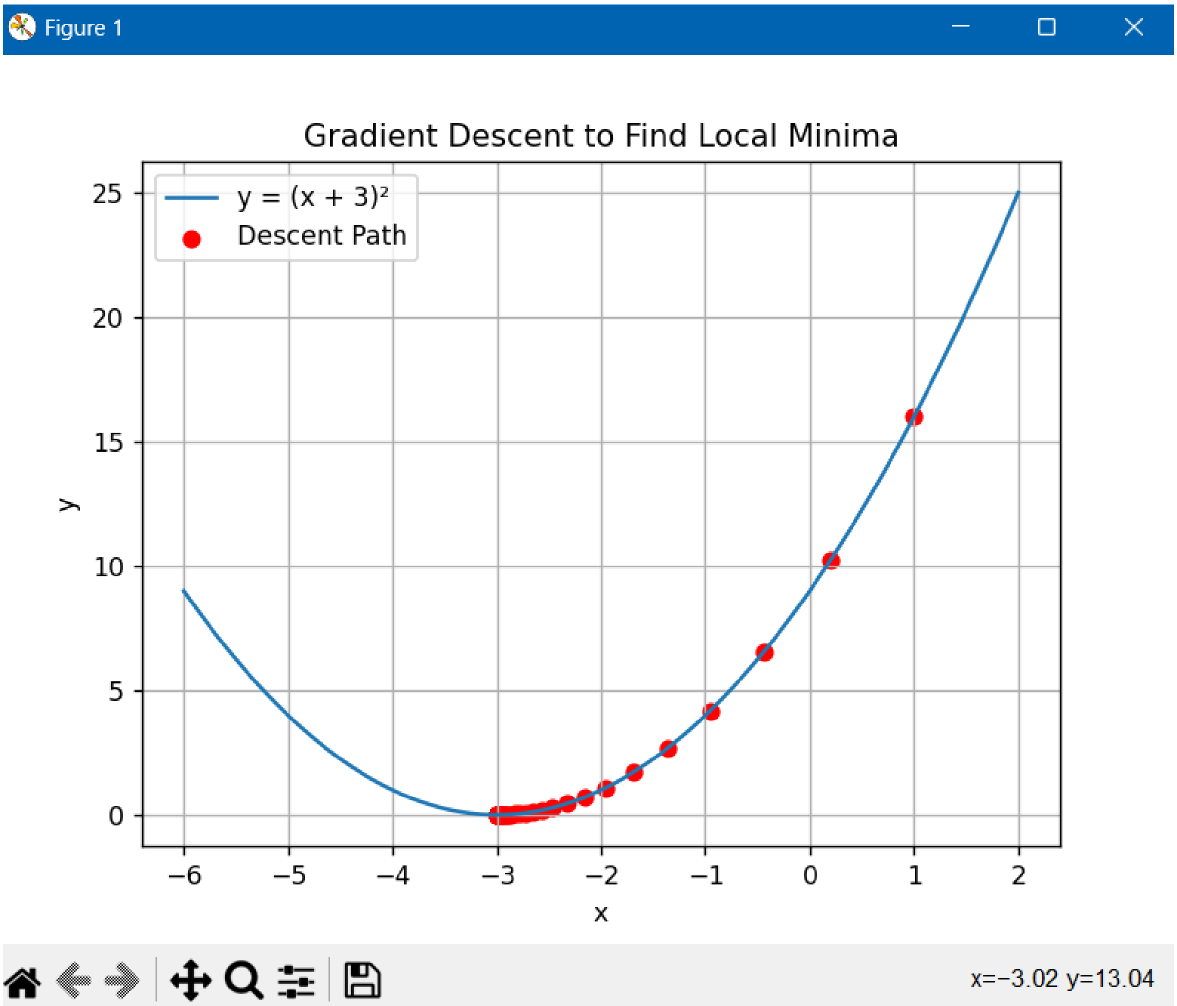
Batch :B16

Program:

```
import numpy as np
import matplotlib.pyplot as plt

# Define the function and its derivative
def function(x):
    return (x + 3) ** 2

def derivative(x):
    return 2 * (x + 3)

# Gradient Descent Algorithm
def gradient_descent(starting_point, learning_rate, num_iterations):
    x = starting_point
    history = []  # To store the values for plotting

    for _ in range(num_iterations):
        x -= learning_rate * derivative(x)
        history.append(x)

    return x, history

# Parameters
starting_point = 2
learning_rate = 0.1
num_iterations = 100

# Run Gradient Descent
minima, history = gradient_descent(starting_point, learning_rate, num_iterations)

# Output results
print(f"Local minima found at x = {minima:.4f}, y = {function(minima):.4f}")

# Plotting the function and the descent path
x_values = np.linspace(-6, 2, 100)
y_values = function(x_values)

plt.plot(x_values, y_values, label='y = (x + 3)²')
plt.scatter(history, function(np.array(history)), color='red', label='Descent Path')
plt.title('Gradient Descent to Find Local Minima')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid()
plt.show()
```

Output:

PS C:\Users\ Ashutosh Shivthare OneDrive\Documents\Academics\BE> & "C:/Python 3.11.3/python.exe" "c:/Users/Ashutosh Shivthare /OneDrive/Documents/Academics/BE/Lp3/Assignment No B2.py"
Local minima found at x = -3.0000, y = 0.0000



Assignment No 8

Name: Ashutosh Shivthare
Roll No: C43364
Batch :B16

Program:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('diabetes.csv')

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

# Check for missing values
print("Missing values in the dataset:")
print(data.isnull().sum())

# Split the data into features and target variable
X = data.drop('Outcome', axis=1)  # Features
y = data['Outcome']               # Target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create the KNN model
k = 5  # You can adjust this value
knn_model = KNeighborsClassifier(n_neighbors=k)

# Train the model
knn_model.fit(X_train, y_train)

# Make predictions
y_pred = knn_model.predict(X_test)

# Evaluate the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
```

```
print(classification_report(y_test, y_pred))

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

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

Output:

```
= RESTART: C:\Users\ Ashutosh Shivthare
\OneDrive\Documents\Academics\BE\Lp3\Assignment No B3\Assignment No B3.py
   Pregnancies  Glucose  BloodPressure  ...  DiabetesPedigreeFunction  Age  Outcome
0            6      148             72  ...                     0.627   50        1
1            1       85             66  ...                     0.351   31        0
2            8      183             64  ...                     0.672   32        1
3            1       89             66  ...                     0.167   21        0
4            0      137             40  ...                     2.288   33        1

[5 rows x 9 columns]
Missing values in the dataset:
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
Confusion Matrix:
[[79 20]
 [27 28]]

Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.80      0.77        99
           1       0.58      0.51      0.54        55

    accuracy                           0.69       154
   macro avg       0.66      0.65      0.66       154
weighted avg       0.69      0.69      0.69       154
```

Accuracy: 0.69