Exp 8: Implement KNN model to classify the target in given dataset.

The K-Nearest Neighbors (KNN) algorithm is a simple, versatile, and powerful method used in classification and regression tasks. It is a supervised learning algorithm that relies on the proximity of data points to classify or predict the target variable.

KNN works on the principle that similar data points tend to be in close proximity to each other in the feature space.

When predicting the class of a new data point, KNN identifies the K nearest neighbors (the K closest data points) from the training dataset and assigns the most frequent class (for classification) or the average (for regression) of these neighbors as the prediction.

For Classification (KNN classification):

Select a value for K: The first step is to choose the number of neighbors (K) that will be considered when making the prediction. A smaller K makes the model sensitive to noise, while a larger K makes the model more generalized.

Calculate the distance: KNN computes the distance between the new data point and all other points in the dataset. Common distance metrics include:

- **Euclidean Distance**:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

- **Manhattan Distance**:

$$d = |x_1 - x_2| + |y_1 - y_2|$$

Find the K nearest neighbors: After computing the distance, the algorithm identifies the K nearest neighbors of the data point based on the smallest distances.

Assign the class: In classification, the class label of the new data point is determined by the majority class among the K neighbors. If there's a tie, some methods use a distance-weighted voting mechanism to break the tie.

For example, if K=5 and the 5 nearest neighbors have the classes: [0, 0, 1, 1, 1], the class of the new point will be 1 (majority vote).

Step 1: Import necessary libraries

```
# Step 1: Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

Step 2: Load the Wine dataset

Load the Wine Dataset:

The load_wine() function from sklearn.datasets loads the Wine dataset, which contains 178 samples with 13 features each (such as alcohol content, color intensity, flavonoids, etc.) and a target variable (y), which indicates the class of the wine (three classes: 0, 1, or 2).

```
# Step 2: Load the Wine dataset
wine = load_wine()
X = wine.data  # Features (13 features about wines)
y = wine.target  # Target (wine class: 0, 1, or 2)
```

Step 3: Convert the dataset into a DataFrame for easier manipulation (optional)

Step 4: Understand the data

Convert to DataFrame (Optional):

We convert the dataset into a pandas DataFrame for easier inspection and manipulation.

```
# Step 3: Convert the dataset into a DataFrame for easier manipulation (optional)
data = pd.DataFrame(X, columns=wine.feature_names)
data['target'] = y

# Step 4: Understand the data
print(data.info())  # Check the structure of the dataset
print(data.head())  # Preview the first few rows
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   alcohol                       178 non-null    float64
 1   malic_acid                    178 non-null    float64
 2   ash                           178 non-null    float64
```

```
 3   alcalinity_of_ash             178 non-null    float64
 4   magnesium                     178 non-null    float64
 5   total_phenols                 178 non-null    float64
 6   flavanoids                    178 non-null    float64
 7   nonflavanoid_phenols          178 non-null    float64
 8   proanthocyanins               178 non-null    float64
 9   color_intensity               178 non-null    float64
 10  hue                           178 non-null    float64
 11  od280/od315_of_diluted_wines  178 non-null    float64
 12  proline                       178 non-null    float64
 13  target                        178 non-null    int64
dtypes: float64(13), int64(1)
memory usage: 19.6 KB
None
   alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0    14.23        1.71  2.43               15.6      127.0           2.80
1    13.20        1.78  2.14               11.2      100.0           2.65
2    13.16        2.36  2.67               18.6      101.0           2.80
3    14.37        1.95  2.50               16.8      113.0           3.85
4    13.24        2.59  2.87               21.0      118.0           2.80

   flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue  \
0        3.06                  0.28             2.29             5.64  1.04
1        2.76                  0.26             1.28             4.38  1.05
2        3.24                  0.30             2.81             5.68  1.03
3        3.49                  0.24             2.18             7.80  0.86
4        2.69                  0.39             1.82             4.32  1.04

   od280/od315_of_diluted_wines  proline  target
0                          3.92   1065.0       0
1                          3.40   1050.0       0
2                          3.17   1185.0       0
3                          3.45   1480.0       0
4                          2.93    735.0       0
```

Step 5: Split the dataset into training and testing sets (80% train, 20% test)

Splitting the Data:

The dataset is split into training and testing sets using train_test_split(). 80% of the data will be used for training, and 20% will be used for testing.

```
# Step 5: Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 6: Feature scaling (important for distance-based models like KNN)

Feature Scaling:

We apply StandardScaler to scale the features, ensuring they have a mean of 0 and a standard deviation of 1. This is important for distance-based algorithms like KNN.

```
# Step 6: Feature scaling (important for distance-based models like KNN)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Step 7: Train the KNN model

Training the KNN Model:

We create a KNN classifier object with n_neighbors=5 (this means that the classifier will consider the 5 nearest neighbors to classify a wine sample). The model is trained with knn.fit(X_train, y_train).

```
# Step 7: Train the KNN model
# Choosing K = 5 for this example
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier()
```

Step 8: Make predictions

Making Predictions:

The trained model makes predictions for the test set using knn.predict(X_test).

```
# Step 8: Make predictions
y_pred = knn.predict(X_test)
```

Step 9: Evaluate the model

Model Evaluation:

The accuracy of the model is computed using accuracy_score, and a confusion matrix is displayed using confusion_matrix, which shows the performance of the classifier in more detail.

```
# Step 9: Evaluate the model
# Accuracy score
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(cm)
```

```
Accuracy: 0.9444444444444444
Confusion Matrix:
[[14  0  0]
 [ 1 12  1]
 [ 0  0  8]]
```

Step 10: Visualize the Confusion Matrix

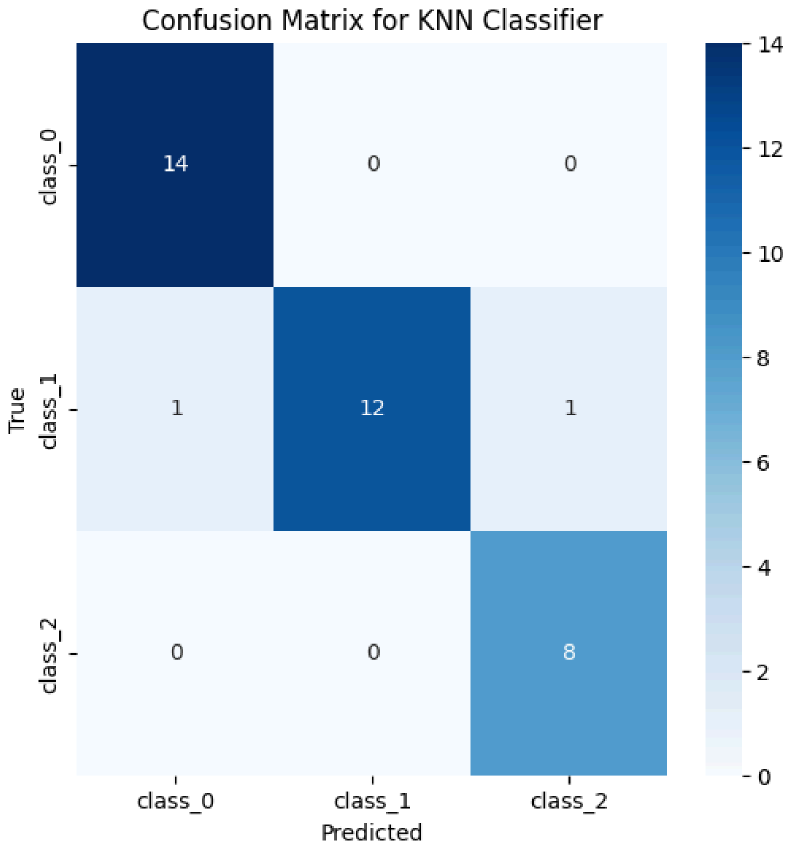
Confusion Matrix Visualization: You will see a heatmap, where:

The diagonal elements represent correct predictions for each class (e.g., 17 for class 0, 17 for class 1, and 15 for class 2).

Off-diagonal elements represent misclassifications.

```
# Step 10: Visualize the Confusion Matrix
plt.figure(figsize=(6,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=wine.target_names, yticklabels=wine.target_names)
plt.title('Confusion Matrix for KNN Classifier')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Step 11: Display some predictions
print("\nSample predictions:")
for i in range(10):
    print(f"True class: {y_test[i]}, Predicted class: {y_pred[i]}")
```




```
Sample predictions:
True class: 0, Predicted class: 0
True class: 0, Predicted class: 0
True class: 2, Predicted class: 2
True class: 0, Predicted class: 0
True class: 1, Predicted class: 1
True class: 0, Predicted class: 0
True class: 1, Predicted class: 1
True class: 2, Predicted class: 2
True class: 1, Predicted class: 1
True class: 2, Predicted class: 2
```