In [1]:
```python
# Import required libraries
import numpy as np                    # For numerical computations
import matplotlib.pyplot as plt      # For plotting graphs
from sklearn.datasets import make_classification   # To create a synthetic classification dataset
from sklearn.neighbors import KNeighborsClassifier # KNN algorithm implementation from sklearn
from sklearn.model_selection import train_test_split  # To split the dataset into training and testing sets
from sklearn.metrics import accuracy_score        # To evaluate the model performance
```

In [2]:
```python
# Generate a synthetic dataset with 200 samples, 2 features, and 2 classes
X, y = make_classification(n_samples=200, n_features=2, n_classes=2, n_clusters_per_class=1,
                           n_informative=2, n_redundant=0, random_state=42)

# X contains the feature matrix (200 samples, 2 features)
# y contains the labels (0 or 1)
```

In [3]:
```python
# Split the dataset into 70% training and 30% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [4]:
```python
# Split the dataset into 70% training and 30% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [5]:
```python
# Create a KNN classifier with K=5 (consider 5 nearest neighbors)
knn = KNeighborsClassifier(n_neighbors=5)
```

In [6]:
```python
# Train the KNN model on the training data (X_train, y_train)
knn.fit(X_train, y_train)
```

Out[6]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

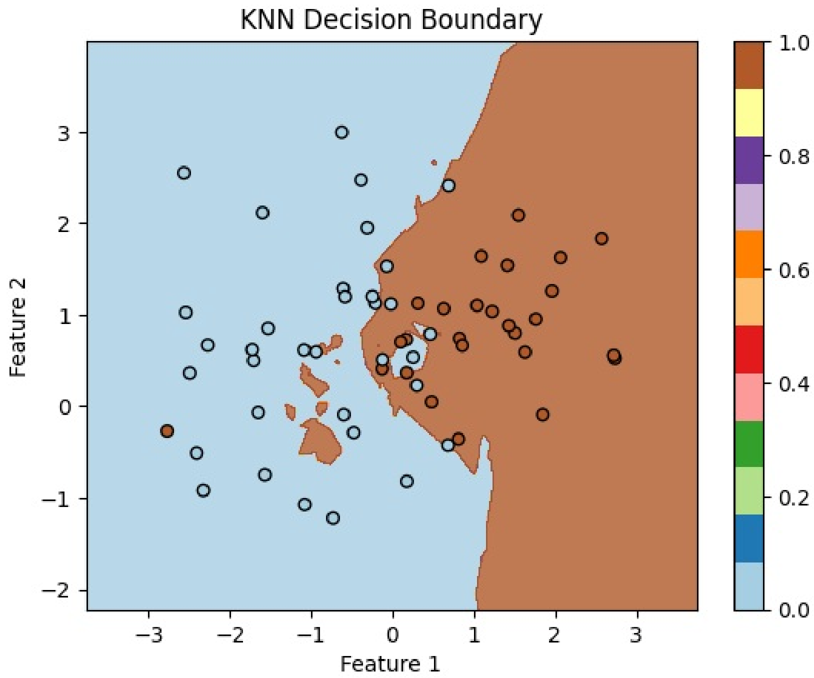
In [7]:
```python
# Make predictions on the test data (X_test)
y_pred = knn.predict(X_test)
```

In [9]:
```python
# Calculate the accuracy of the model on the test data by comparing predictions with actual labels
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")  # Print the accuracy as a percentage
```

```
Accuracy: 80.00%
```

In [20]:
```python
# Function to plot decision boundary for a 2D dataset
def plot_decision_boundary(X, y, model, title="KNN Decision Boundary"):
    # Define the range for the plot by taking min and max values of features and adding some margin
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    # Create a meshgrid over the feature space (to visualize the decision boundary)
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))

    # Use the trained model to predict the class for each point in the grid
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])  # Predict using flattened grid
    Z = Z.reshape(xx.shape)  # Reshape the predictions to match the grid shape

    # Plot the decision boundary using filled contours
    plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.Paired)

    # Plot the original test data points on top of the decision boundary
    scatter = plt.scatter(X[:, 0], X[:, 1], c=y, s=30, edgecolor='k', cmap=plt.cm.Paired)

    # Add a colorbar to indicate the classes in the scatter plot
    plt.colorbar(scatter)

    # Add title and axis labels
    plt.title(title)           # Add title to the plot
    plt.xlabel("Feature 1")    # Label for the x-axis (first feature)
    plt.ylabel("Feature 2")    # Label for the y-axis (second feature)

    # Display the plot
    plt.show()
```

In [21]:
```python
# Plot the decision boundary for the test data
plot_decision_boundary(X_test, y_test, knn)
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js