In [7]:
```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import urllib.request
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix,
precision_score, recall_score
import seaborn as sns
```

In [4]:
```
# URL of the Iris dataset from UCI Repository
url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data"

# Column names based on dataset description
column_names = ["Sepal_Length", "Sepal_Width", "Petal_Length", "Petal_Width",
"Class"]

# Load the dataset into a Pandas DataFrame
iris = pd.read_csv(url, names=column_names)
```

In [6]:
```
# Initialize the data frame
iris.head()
```

Out[6]:

| | Sepal_Length | Sepal_Width | Petal_Length | Petal_Width | Class |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |

Perform Data Preprocessing

In [9]:
```
# Convert categorical target variable to numerical
label_encoder = LabelEncoder()
iris["Class"] = label_encoder.fit_transform(iris["Class"])
```

In [10]:
```
iris.head()
```

Out[10]:

| | Sepal_Length | Sepal_Width | Petal_Length | Petal_Width | Class |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | 0 |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | 0 |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | 0 |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | 0 |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | 0 |

In [11]:
```
# Check for null values
iris.isnull().sum()
```

Out[11]:
```
Sepal_Length    0
Sepal_Width     0
Petal_Length    0
Petal_Width     0
Class           0
dtype: int64
```

In [13]:
```
# Divide dataset into Independent (X) and Dependent (Y) variables
X = iris.drop(columns=["Class"])  # Features
Y = iris["Class"]  # Target variable
```

In [15]:
```
# Split dataset into training and testing datasets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

In [17]:
```
# Feature Scaling
from sklearn import preprocessing
# Apply Min-Max Scaling only on numerical columns
scaler = preprocessing.MinMaxScaler()
features_scaled = scaler.fit_transform(X)

# Create a new DataFrame with scaled features and original labels
iris_normalized = pd.DataFrame(features_scaled, columns=X.columns)
iris_normalized["Class"] = Y  # Add the class column back

# Display first few rows
iris_normalized.head()
```

Out[17]:

| | Sepal_Length | Sepal_Width | Petal_Length | Petal_Width | Class |
|---|---|---|---|---|---|
| **0** | 0.222222 | 0.625000 | 0.067797 | 0.041667 | 0 |
| **1** | 0.166667 | 0.416667 | 0.067797 | 0.041667 | 0 |
| **2** | 0.111111 | 0.500000 | 0.050847 | 0.041667 | 0 |
| **3** | 0.083333 | 0.458333 | 0.084746 | 0.041667 | 0 |
| **4** | 0.194444 | 0.666667 | 0.067797 | 0.041667 | 0 |

In [18]:
```
# Train the Model using Naïve Bayes
naive_bayes = GaussianNB()
naive_bayes.fit(X_train, y_train)
```

Out[18]:
```
GaussianNB()
```

In [19]:
```
# Make Predictions
y_pred_train = naive_bayes.predict(X_train)
y_pred_test = naive_bayes.predict(X_test)
```

In [20]:
```
# Evaluate Model Performance
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)
```

In [21]:
```
# Evaluate Model Performance
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

precision = precision_score(y_test, y_pred_test, average="micro")
recall = recall_score(y_test, y_pred_test, average="micro")
```

In [23]:
```
# Compute Confusion Matrix
cm = confusion_matrix(y_test, y_pred_test)
```

Out[23]:
```
array([[10,  0,  0],
       [ 0,  9,  0],
       [ 0,  0, 11]], dtype=int64)
```

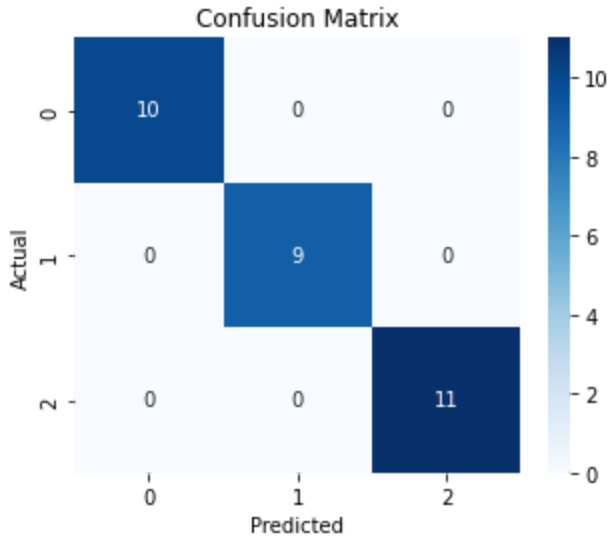
In [24]:
```
# Display results
print("\nTraining Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("\nConfusion Matrix:\n", cm)
```

```
Training Accuracy: 0.95
Testing Accuracy: 1.0
Precision: 1.0
Recall: 1.0

Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

In [25]:

```
# Visualizing Confusion Matrix
plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```




In [ ]: