Step 1: Load the necessary libraries

```
# Step 1: Import the required libraries
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import numpy as np
```

Step 2: Generate sample data around three class centers:

```
centers = [[-3, 2], [7, -4], [5, 8]]
n_classes = len(centers)
data, labels = make_blobs(n_samples=150, centers=np.array(centers), random_state=1)
```

#Plot the dataset

```
plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis', s=50)
plt.title("Generated Data with 3 Clusters")
plt.show()
```

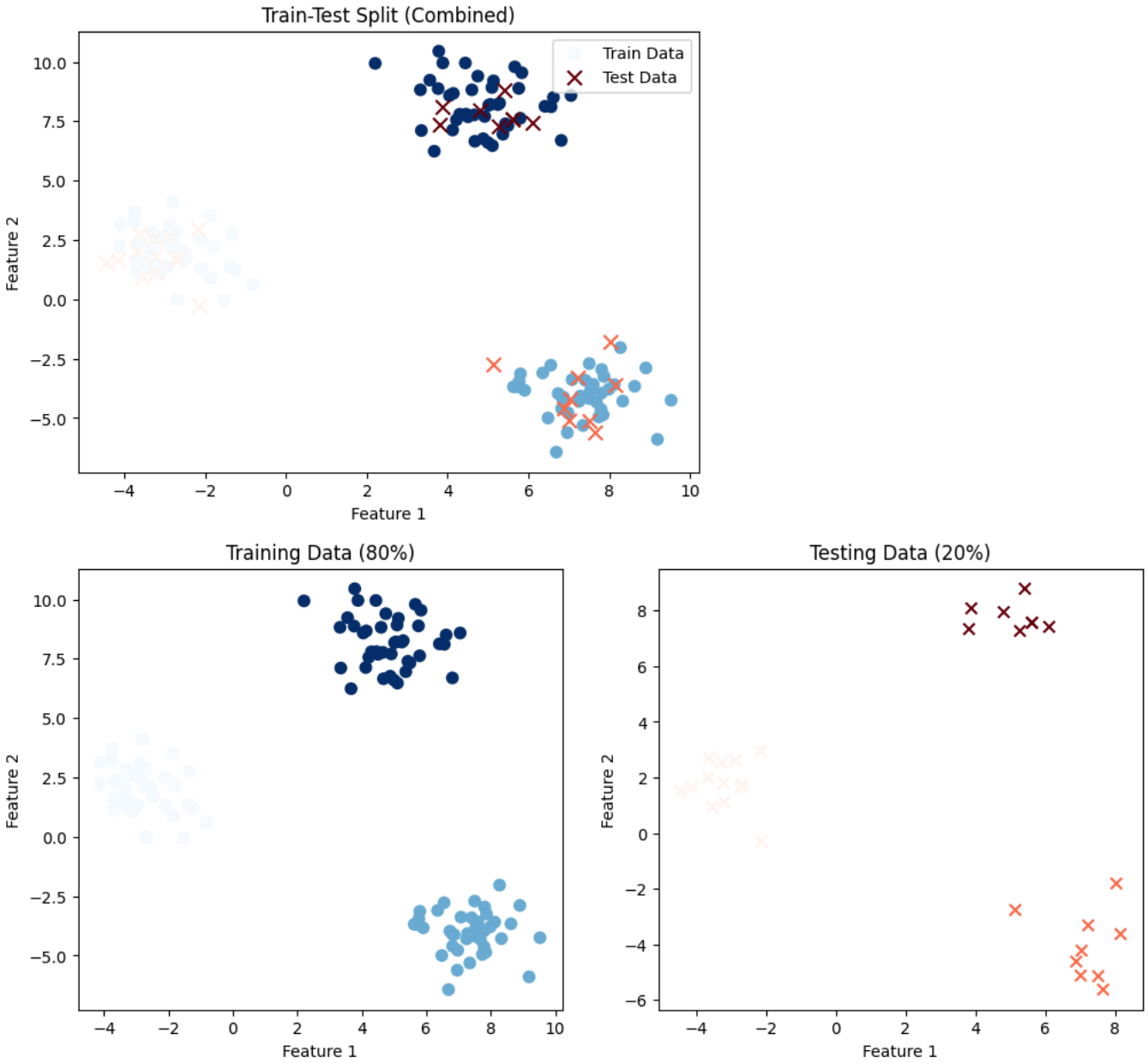
Interpretation: I created a fake data with 3 clusters (groups). Each cluster has a center point, and we plot it so we can see how the data looks.

Step 3: Split the data into training (80%) and testing (20%)

```
X_train, X_test, y_train, y_test = train_test_split(
    data, labels, test_size=0.2, random_state=1
)
```

```
# Combined plot: Train vs Test
plt.figure(figsize=(7, 5))
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='Blues', s=50, label="Train Data")
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='Reds', marker='x', s=80, label="Test Data")
plt.title("Train-Test Split (Combined)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()

# Side-by-side plots
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Plot training data only
axes[0].scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='Blues', s=50)
```

```
axes[0].set_title("Training Data (80%)")
axes[0].set_xlabel("Feature 1")
axes[0].set_ylabel("Feature 2")

# Plot testing data only
axes[1].scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='Reds', s=50, marker='x')
axes[1].set_title("Testing Data (20%)")
axes[1].set_xlabel("Feature 1")
axes[1].set_ylabel("Feature 2")

plt.show()
```



Interpretation: This plot shows how the dataset is divided. Blue dots represent training data (80%) used to build the model, while red crosses represent testing data (20%) used to check accuracy. Separate plots make the split clearer for understanding.

Step 4: KNN model with default hyperparameter (k=5)

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

Step 5: Predict and check accuracy

```
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy with k=5: {accuracy:.2f}")
```

```
Accuracy with k=5: 1.00
```

Interpretation: I tested the model by checking how well it predicts. Accuracy shows the percentage of predictions that are correct. A higher accuracy means the model makes fewer mistakes and performs better at solving the problem.

Step 6: Try KNN with another hyperparameter (k=3)

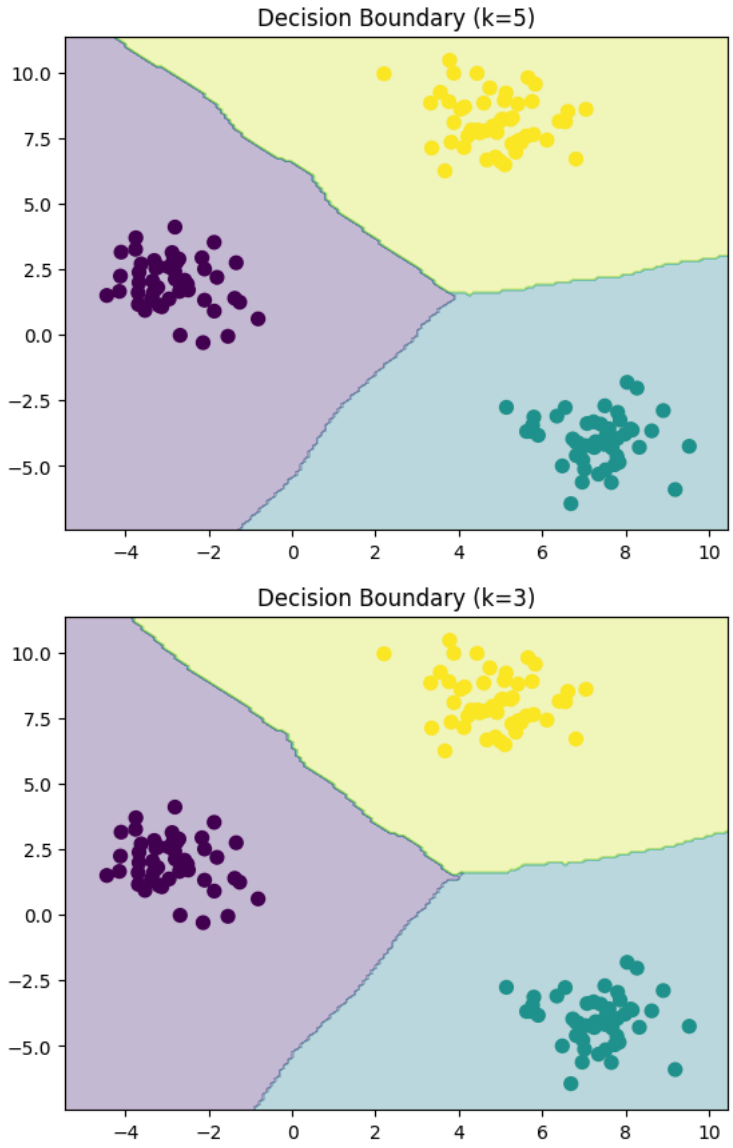
```
knn2 = KNeighborsClassifier(n_neighbors=3)
knn2.fit(X_train, y_train)

y_pred2 = knn2.predict(X_test)
accuracy2 = accuracy_score(y_test, y_pred2)
print(f"Accuracy with k=3: {accuracy2:.2f}")
```

```
Accuracy with k=3: 1.00
```

Interpretation: We run KNN again using k=3. This helps us see how accuracy changes when we use more neighbors. Comparing results shows the effect of hyperparameters and helps choose the best model for predictions.

Step 7: Plot decision boundaries for both models

```
def plot_decision_boundary(model, X, y, title):
    h = 0.1  # step size in the mesh
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.3)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', s=50)
    plt.title(title)
    plt.show()

    # Plot for k=5
plot_decision_boundary(knn, data, labels, "Decision Boundary (k=5)")

# Plot for k=3
plot_decision_boundary(knn2, data, labels, "Decision Boundary (k=3)")
```



Interpretation: Decision boundaries are the dividing lines the model creates to separate different classes. By drawing them, we can see