In [ ]:
```python
# importing required libraries
import numpy as np
import pandas as pd
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.datasets import make_blobs
from sklearn.linear_model import LogisticRegression
```

In [ ]:
```python
# Sample data: observed frequencies in a contingency table
observed = [[10, 20, 30],
            [15, 25, 35]]

# Expected frequencies (assuming these are hypothetical values)
expected = [[18, 18, 36],
            [12, 12, 36]]

# Calculate Chi-square statistic, p-value, degrees of freedom, and expected table
chi2_statistic, pval, dof, expected_table = stats.chi2_contingency(observed, expect

# Print the results
print("Chi-square statistic:", chi2_statistic)
print("p-value:", pval)
print("Degrees of freedom:", dof)
print("Expected table:\n", expected_table)
```

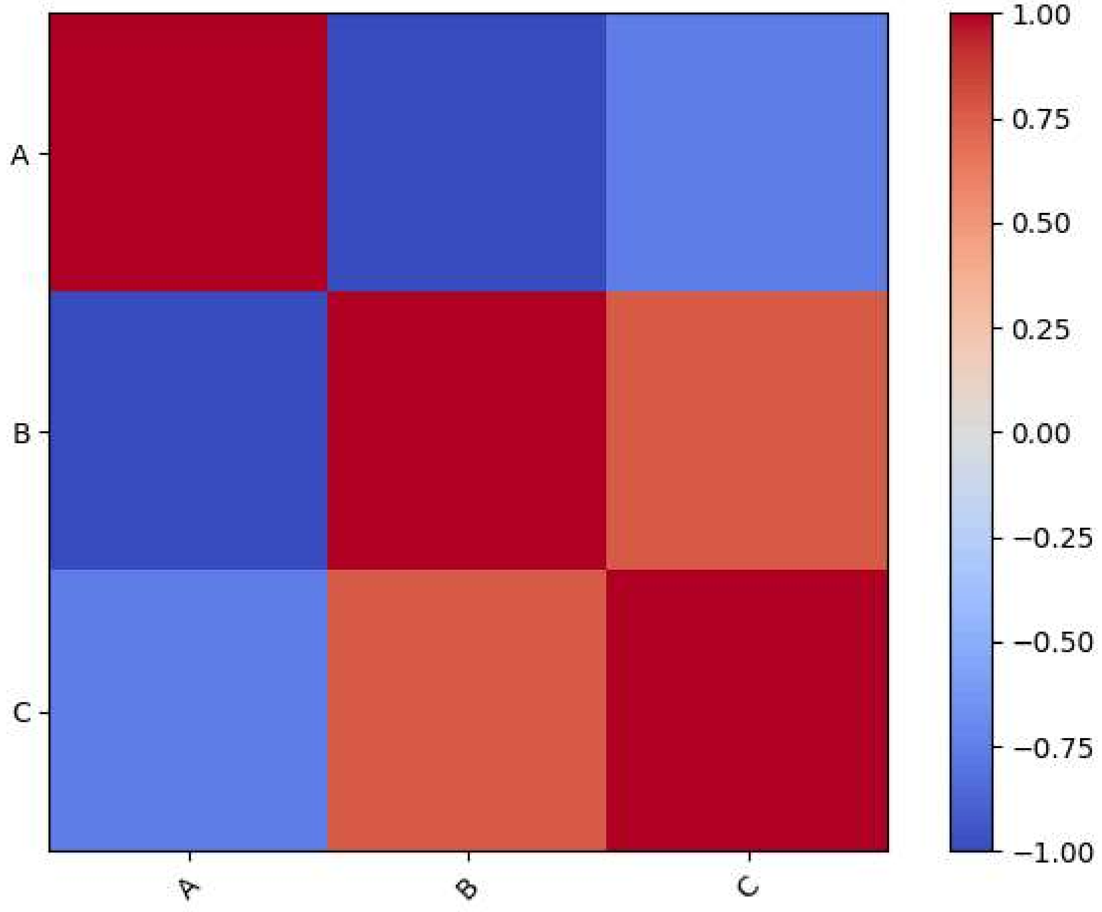
```
Chi-square statistic: 0.27692307692307694
p-value: 0.870696738961232
Degrees of freedom: 2
Expected table:
 [[11.11111111 20.         28.88888889]
 [13.88888889 25.         36.11111111]]
```

In [ ]:
```python
def calculate_chi_square(observed, expected):
    """Calculate Chi-square statistic given observed and expected frequencies."""
    return np.sum((observed - expected)**2 / expected)

def main():
    # Sample data
    observed = np.array([10, 20, 30, 40])
    expected = np.array([15, 15, 30, 40])  # Expected frequencies

    # Calculate Chi-square statistic
    chi_square = calculate_chi_square(observed, expected)
    print("Chi-square statistic:", chi_square)

if __name__ == "__main__":
    main()
```

```
Chi-square statistic: 3.3333333333333335
```

In [ ]:
```python
data = {'A': [1,2,3,4,5],
        'B': [6,5,4,3,2],
        'C': [4,6,5,2,1]}

# Create pandas dataframe
df = pd.DataFrame(data)
```

```
# Calculate correlation matrix
correlation = df.corr()

# Create heatmap
plt.imshow(correlation, cmap='coolwarm')
plt.colorbar()
plt.xticks(range(len(correlation.columns)), correlation.columns, rotation=45)
plt.yticks(range(len(correlation.columns)), correlation.columns)
plt.tight_layout()
plt.show()
```



In [ ]:

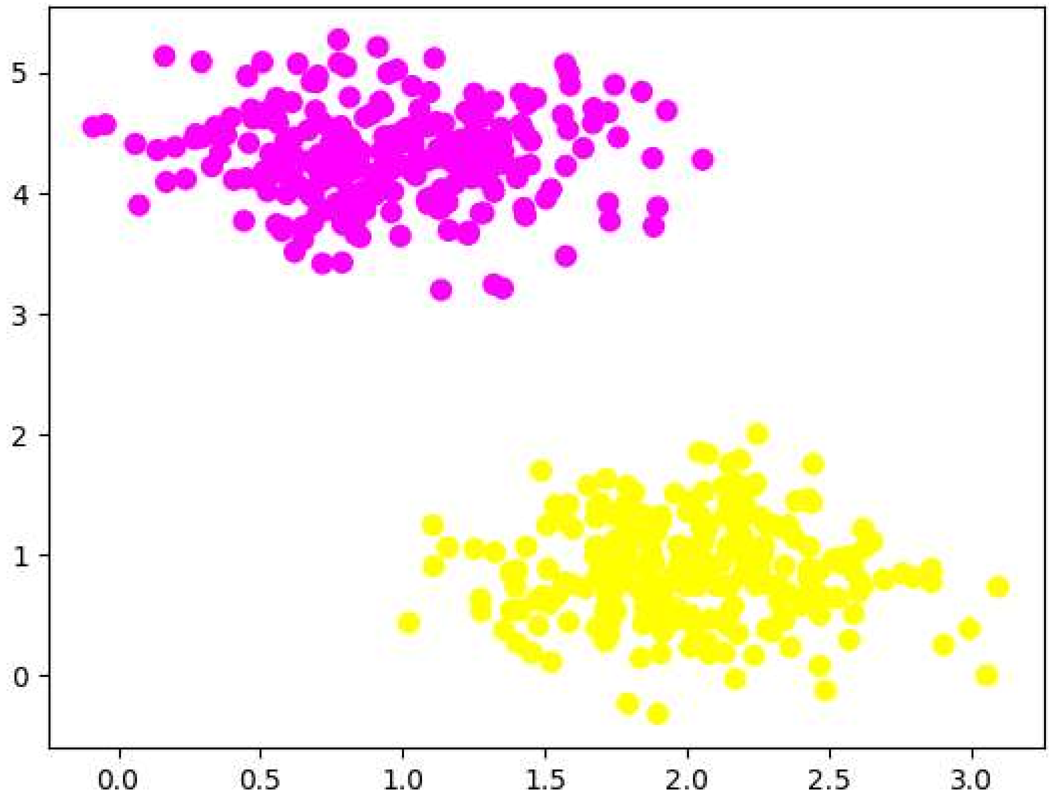
```
# Sample data (replace with your actual data)
X = np.array([[1, 2], [3, 4], [5, 1], [6, 0]])
y = np.array([0, 1, 1, 0])  # 0 for negative class, 1 for positive class
model = LogisticRegression()
# Train the model
model.fit(X, y)
new_data = np.array([[7, 3]])
predictions = model.predict(new_data)

# Print the predicted class labels
print("Predicted class labels:", predictions)

# Get probability estimates (optional)
probabilities = model.predict_proba(new_data)
print("Predicted probabilities:", probabilities)
```

```
Predicted class labels: [1]
Predicted probabilities: [[0.10139095 0.89860905]]
```

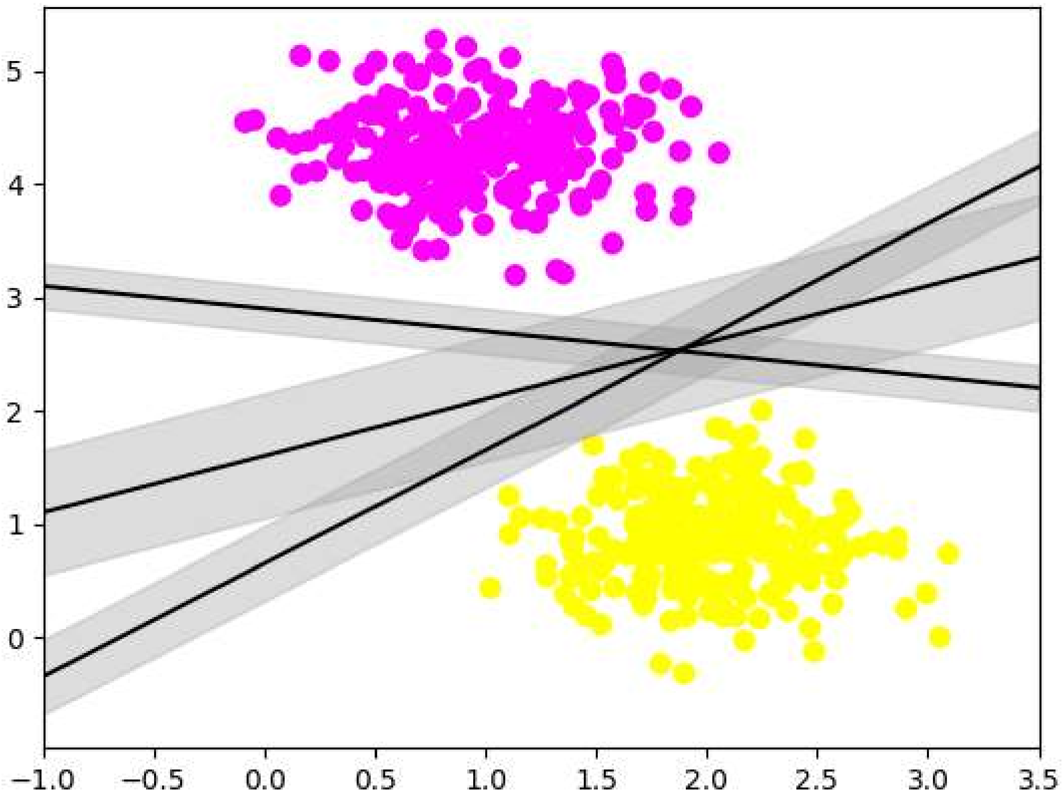
In [ ]:

```
X, Y = make_blobs(n_samples=500, centers=2,
                  random_state=0, cluster_std=0.40)
# plotting scatters
```

```
plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='spring');
plt.show()
```



In [ ]:
```
# creating linspace between -1 to 3.5
xfit = np.linspace(-1, 3.5)
# plotting scatter
plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='spring')
# plot a line between the different sets of data
for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2, 2.9, 0.2)]:
        yfit = m * xfit + b
        plt.plot(xfit, yfit, '-k')
        plt.fill_between(xfit, yfit - d, yfit + d, edgecolor='none',
        color='#AAAAAA', alpha=0.4)

plt.xlim(-1, 3.5);
plt.show()
```



In [ ]:

```
# reading csv file and extracting class column to y.
x = pd.read_csv("/advertising.csv")
a = np.array(x)
y  = a[:,1] # classes having 0 and 1

# extracting two features
x = np.column_stack((x.Sales,x.Radio))

#  2 features
x.shape

print (x),(y)
```

```
[[22.1 37.8]
 [10.4 39.3]
 [12.  45.9]
 [16.5 41.3]
 [17.9 10.8]
 [ 7.2 48.9]
 [11.8 32.8]
 [13.2 19.6]
 [ 4.8  2.1]
 [15.6  2.6]
 [12.6  5.8]
 [17.4 24. ]
 [ 9.2 35.1]
 [13.7  7.6]
 [19.  32.9]
 [22.4 47.7]
 [12.5 36.6]
 [24.4 39.6]
 [11.3 20.5]
 [14.6 23.9]
 [18.  27.7]
 [17.5  5.1]
 [ 5.6 15.9]
 [20.5 16.9]
 [ 9.7 12.6]
 [17.   3.5]
 [15.  29.3]
 [20.9 16.7]
 [18.9 27.1]
 [10.5 16. ]
 [21.4 28.3]
 [11.9 17.4]
 [13.2  1.5]
 [17.4 20. ]
 [11.9  1.4]
 [17.8  4.1]
 [25.4 43.8]
 [14.7 49.4]
 [10.1 26.7]
 [21.5 37.7]
 [16.6 22.3]
 [17.1 33.4]
 [20.7 27.7]
 [17.9  8.4]
 [ 8.5 25.7]
 [16.1 22.5]
 [10.6  9.9]
 [23.2 41.5]
 [19.8 15.8]
 [ 9.7 11.7]
 [16.4  3.1]
 [10.7  9.6]
 [22.6 41.7]
 [21.2 46.2]
 [20.2 28.8]
 [23.7 49.4]
 [ 5.5 28.1]
 [13.2 19.2]
 [23.8 49.6]
 [18.4 29.5]
 [ 8.1  2. ]
 [24.2 42.7]
 [20.7 15.5]
 [14.  29.6]
```

```
[16.  42.8]
[11.3  9.3]
[11.  24.6]
[13.4 14.5]
[18.9 27.5]
[22.3 43.9]
[18.3 30.6]
[12.4 14.3]
[ 8.8 33. ]
[11.   5.7]
[17.  24.6]
[ 8.7 43.7]
[ 6.9  1.6]
[14.2 28.5]
[ 5.3 29.9]
[11.   7.7]
[11.8 26.7]
[17.3  4.1]
[11.3 20.3]
[13.6 44.5]
[21.7 43. ]
[20.2 18.4]
[12.  27.5]
[16.  40.6]
[12.9 25.5]
[16.7 47.8]
[14.   4.9]
[ 7.3  1.5]
[19.4 33.5]
[22.2 36.5]
[11.5 14. ]
[16.9 31.6]
[16.7  3.5]
[20.5 21. ]
[25.4 42.3]
[17.2 41.7]
[16.7  4.3]
[23.8 36.3]
[19.8 10.1]
[19.7 17.2]
[20.7 34.3]
[15.  46.4]
[ 7.2 11. ]
[12.   0.3]
[ 5.3  0.4]
[19.8 26.9]
[18.4  8.2]
[21.8 38. ]
[17.1 15.4]
[20.9 20.6]
[14.6 46.8]
[12.6 35. ]
[12.2 14.3]
[ 9.4  0.8]
[15.9 36.9]
[ 6.6 16. ]
[15.5 26.8]
[ 7.  21.7]
[16.6  2.4]
[15.2 34.6]
[19.7 32.3]
[10.6 11.8]
[ 6.6 38.9]
[11.9  0. ]
```

```
[24.7 49. ]
[ 9.7 12. ]
[ 1.6 39.6]
[17.7  2.9]
[ 5.7 27.2]
[19.6 33.5]
[10.8 38.6]
[11.6 47. ]
[ 9.5 39. ]
[20.8 28.9]
[ 9.6 25.9]
[20.7 43.9]
[10.9 17. ]
[19.2 35.4]
[20.1 33.2]
[10.4  5.7]
[12.3 14.8]
[10.3  1.9]
[18.2  7.3]
[25.4 49. ]
[10.9 40.3]
[10.1 25.8]
[16.1 13.9]
[11.6  8.4]
[16.6 23.3]
[16.  39.7]
[20.6 21.1]
[ 3.2 11.6]
[15.3 43.5]
[10.1  1.3]
[ 7.3 36.9]
[12.9 18.4]
[16.4 18.1]
[13.3 35.8]
[19.9 18.1]
[18.  36.8]
[11.9 14.7]
[16.9  3.4]
[ 8.  37.6]
[17.2  5.2]
[17.1 23.6]
[20.  10.6]
[ 8.4 11.6]
[17.5 20.9]
[ 7.6 20.1]
[16.7  7.1]
[16.5  3.4]
[27.  48.9]
[20.2 30.2]
[16.7  7.8]
[16.8  2.3]
[17.6 10. ]
[15.5  2.6]
[17.2  5.4]
[ 8.7  5.7]
[26.2 43. ]
[17.6 21.3]
[22.6 45.1]
[10.3  2.1]
[17.3 28.7]
[20.9 13.9]
[ 6.7 12.1]
[10.8 41.1]
[11.9 10.8]
```

```
 [ 5.9  4.1]
 [19.6 42. ]
 [17.3 35.6]
 [ 7.6  3.7]
 [14.   4.9]
 [14.8  9.3]
 [25.5 42. ]
 [18.4  8.6]]
```

Out[ ]:
```
(None,
 array([37.8, 39.3, 45.9, 41.3, 10.8, 48.9, 32.8, 19.6,  2.1,  2.6,  5.8,
        24. , 35.1,  7.6, 32.9, 47.7, 36.6, 39.6, 20.5, 23.9, 27.7,  5.1,
        15.9, 16.9, 12.6,  3.5, 29.3, 16.7, 27.1, 16. , 28.3, 17.4,  1.5,
        20. ,  1.4,  4.1, 43.8, 49.4, 26.7, 37.7, 22.3, 33.4, 27.7,  8.4,
        25.7, 22.5,  9.9, 41.5, 15.8, 11.7,  3.1,  9.6, 41.7, 46.2, 28.8,
        49.4, 28.1, 19.2, 49.6, 29.5,  2. , 42.7, 15.5, 29.6, 42.8,  9.3,
        24.6, 14.5, 27.5, 43.9, 30.6, 14.3, 33. ,  5.7, 24.6, 43.7,  1.6,
        28.5, 29.9,  7.7, 26.7,  4.1, 20.3, 44.5, 43. , 18.4, 27.5, 40.6,
        25.5, 47.8,  4.9,  1.5, 33.5, 36.5, 14. , 31.6,  3.5, 21. , 42.3,
        41.7,  4.3, 36.3, 10.1, 17.2, 34.3, 46.4, 11. ,  0.3,  0.4, 26.9,
         8.2, 38. , 15.4, 20.6, 46.8, 35. , 14.3,  0.8, 36.9, 16. , 26.8,
        21.7,  2.4, 34.6, 32.3, 11.8, 38.9,  0. , 49. , 12. , 39.6,  2.9,
        27.2, 33.5, 38.6, 47. , 39. , 28.9, 25.9, 43.9, 17. , 35.4, 33.2,
         5.7, 14.8,  1.9,  7.3, 49. , 40.3, 25.8, 13.9,  8.4, 23.3, 39.7,
        21.1, 11.6, 43.5,  1.3, 36.9, 18.4, 18.1, 35.8, 18.1, 36.8, 14.7,
         3.4, 37.6,  5.2, 23.6, 10.6, 11.6, 20.9, 20.1,  7.1,  3.4, 48.9,
        30.2,  7.8,  2.3, 10. ,  2.6,  5.4,  5.7, 43. , 21.3, 45.1,  2.1,
        28.7, 13.9, 12.1, 41.1, 10.8,  4.1, 42. , 35.6,  3.7,  4.9,  9.3,
        42. ,  8.6]))
```

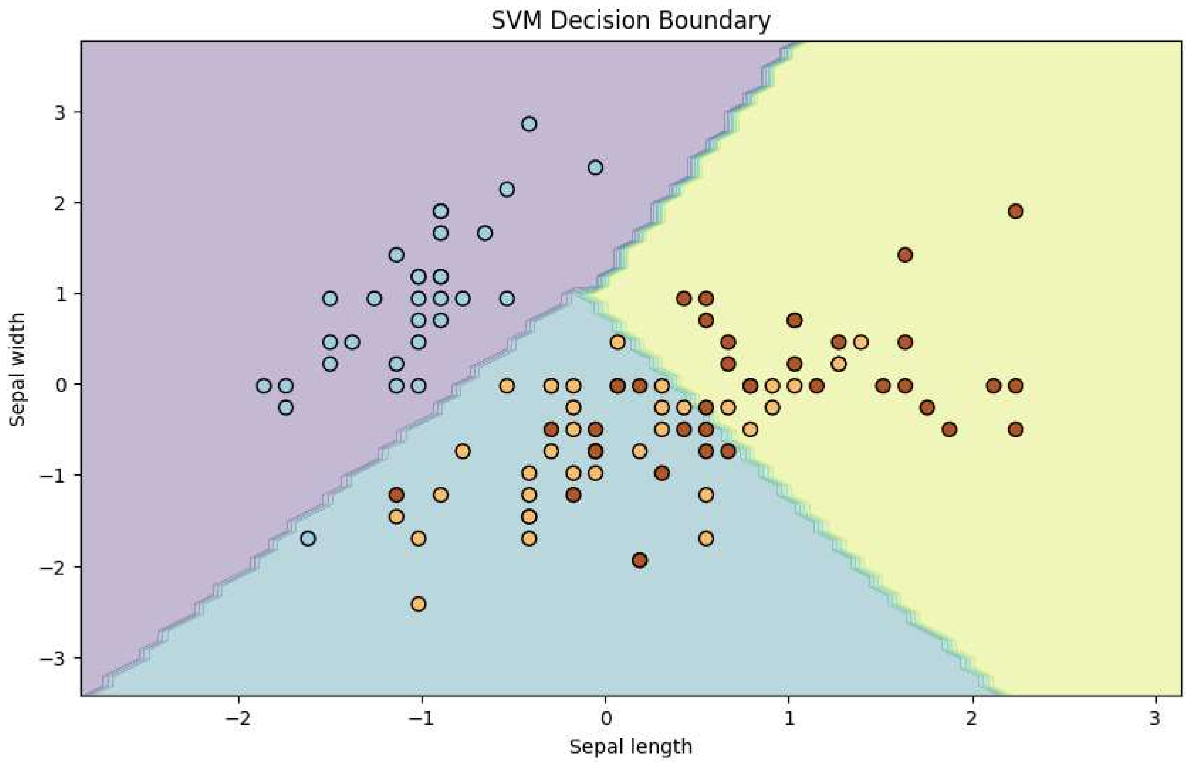
In [ ]:
```python
from sklearn.svm import SVC
# Create sample data
X = [[0], [1], [2], [3]]
y = [0, 1, 2, 3]

# Create a linear Support Vector Classifier
clf = SVC(kernel='linear')

# Train the classifier
clf.fit(X, y)
```

Out[ ]:
```
▾ SVC
SVC(kernel='linear')
```

In [ ]:
```python
# Load the iris dataset
iris = datasets.load_iris()
X = iris.data[:, :2]  # Considering only the first two features for simplicity
y = iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train the SVM model
svm_model = SVC(kernel='linear', C=1, random_state=42)
svm_model.fit(X_train_scaled, y_train)

# Make predictions
y_pred = svm_model.predict(X_test_scaled)
```

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Visualize decision boundary
def plot_decision_boundary(X, y, model):
    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, 0.1),
                         np.arange(y_min, y_max, 0.1))
    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, s=50, edgecolors='k', cmap=plt.cm.Paired)
    plt.xlabel('Sepal length')
    plt.ylabel('Sepal width')
    plt.title('SVM Decision Boundary')

# Plot decision boundary
plt.figure(figsize=(10, 6))
plot_decision_boundary(X_train_scaled, y_train, svm_model)
plt.show()
```

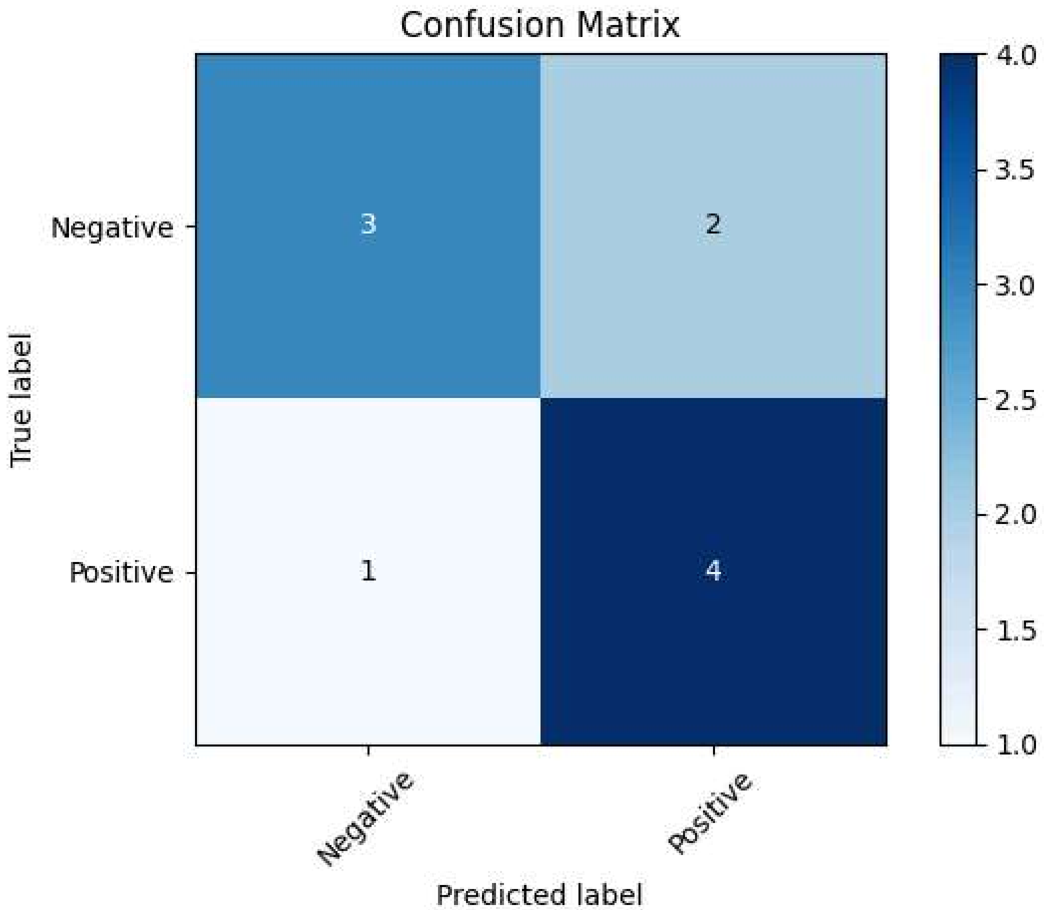
```
Accuracy: 0.7333333333333333
```



In [ ]:

```
import numpy as np
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

# Sample data
true_labels = np.array([1, 0, 1, 1, 0, 1, 0, 0, 1, 0])
predicted_labels = np.array([1, 1, 0, 1, 0, 1, 0, 1, 1, 0])

# Compute confusion matrix
cm = confusion_matrix(true_labels, predicted_labels)

# Display confusion matrix
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
```

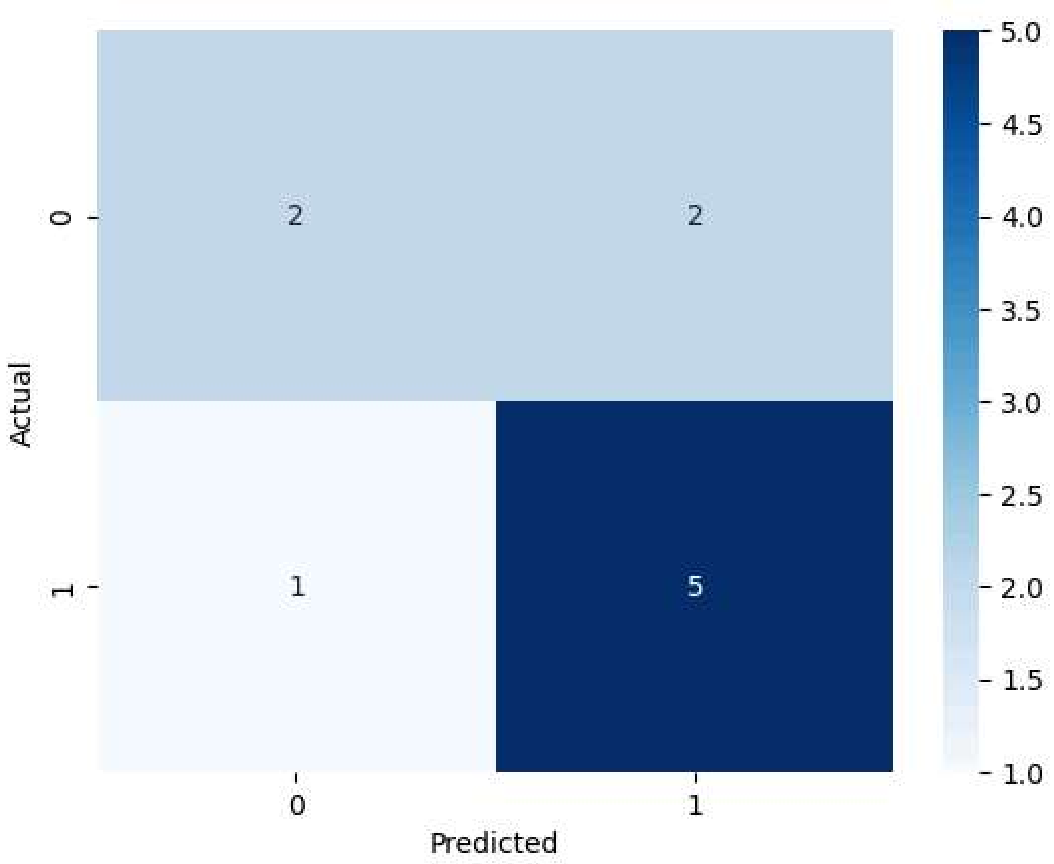
```python
plt.colorbar()
classes = ['Negative', 'Positive']
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)

fmt = 'd'
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], fmt),
                 ha="center", va="center",
                 color="white" if cm[i, j] > thresh else "black")

plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.tight_layout()
plt.show()
```



In [ ]:
```python
import numpy as np
from sklearn.metrics import confusion_matrix

# Sample actual and predicted values
actual = [1, 0, 1, 1, 0, 0, 1, 1, 0, 1]
predicted = [1, 1, 1, 1, 0, 1, 0, 1, 0, 1]

# Create the confusion matrix
cm = confusion_matrix(actual, predicted)
print(cm)

# Visualize the confusion matrix using a heatmap (optional)
import matplotlib.pyplot as plt
import seaborn as sns
```

```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

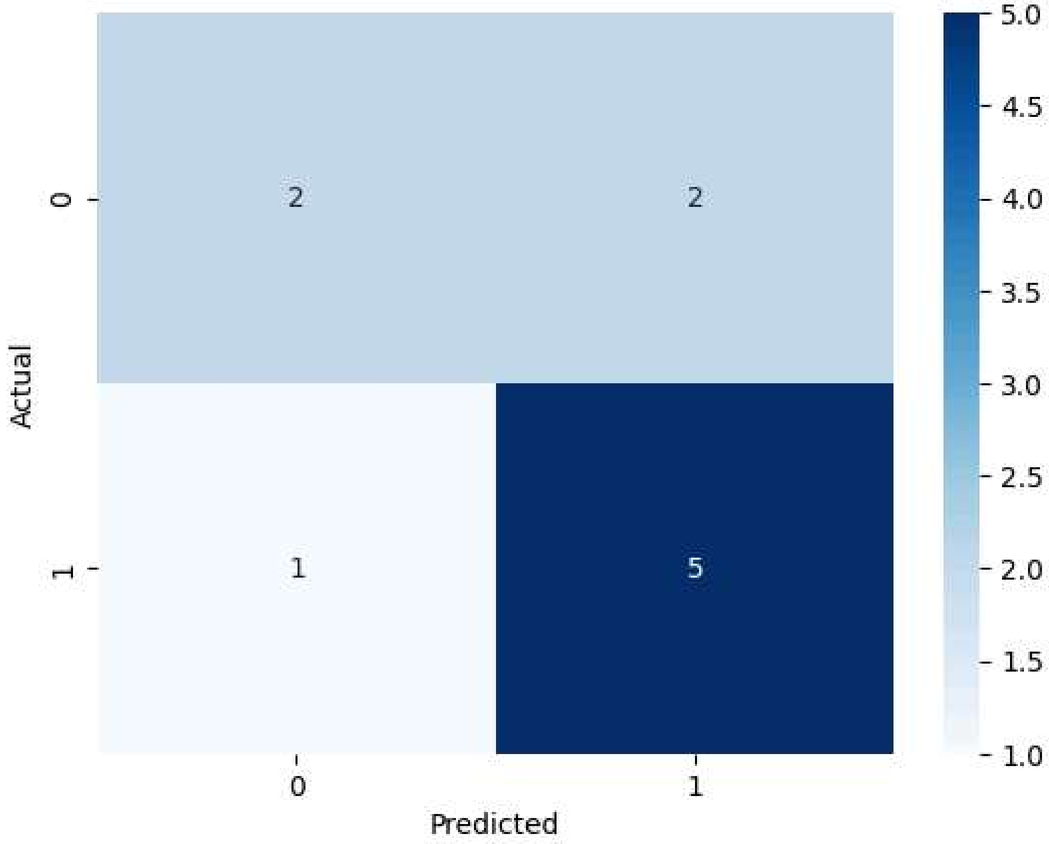
```
[[2 2]
 [1 5]]
```



In [ ]:

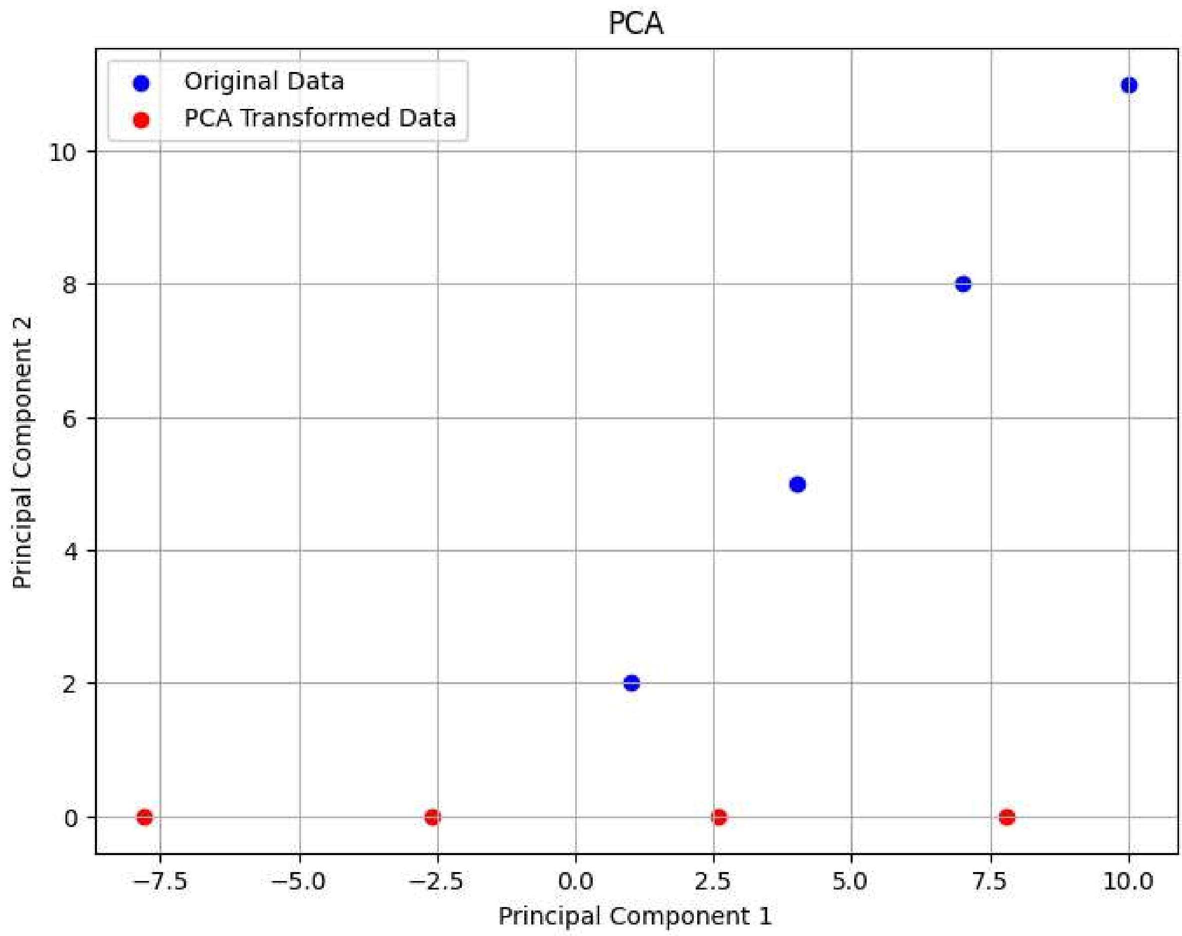
```
import numpy as np
from sklearn.metrics import confusion_matrix

# Sample actual and predicted values
actual = [1, 0, 1, 1, 0, 0, 1, 1, 0, 1]
predicted = [1, 1, 1, 1, 0, 1, 0, 1, 0, 1]

# Create the confusion matrix
cm = confusion_matrix(actual, predicted)
print(cm)

# Visualize the confusion matrix using a heatmap (optional)
import matplotlib.pyplot as plt
import seaborn as sns

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
[[2 2]
 [1 5]]
```



In [ ]:

```
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Sample data
data = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])

# Apply PCA
pca = PCA(n_components=2)  # Reduce to 2 principal components
principal_components = pca.fit_transform(data)

# Plotting original data
plt.figure(figsize=(8, 6))
plt.scatter(data[:, 0], data[:, 1], color='blue', label='Original Data')

# Plotting data after PCA
plt.scatter(principal_components[:, 0], principal_components[:, 1], color='red', la
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA')
plt.legend()
plt.grid(True)
plt.show()
```



In [ ]:

```
import numpy as np
from sklearn.decomposition import PCA

# Sample data (replace with your actual data)
data = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
])

# Standardize the data (optional but recommended)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# Perform PCA with 2 components (you can adjust n_components)
pca = PCA(n_components=2)
pca.fit(data_scaled)

# Transform the data to the principal components
data_reduced = pca.transform(data_scaled)

# Print the explained variance ratio (percentage of variance explained by each comp
print("Explained variance ratio:", pca.explained_variance_ratio_)

# Print the first two principal components (if using n_components=2)
print("First principal component:", data_reduced[:, 0])
print("Second principal component:", data_reduced[:, 1])

# (Optional) Project the data back to the original space
data_original_projected = pca.inverse_transform(data_reduced)
print("Original data projected back:", data_original_projected)
```

```
Explained variance ratio: [1. 0.]
First principal component: [ 2.12132034  0.          -2.12132034]
Second principal component: [-4.36708632e-16  0.00000000e+00  4.36708632e-16]
Original data projected back: [[-1.22474487 -1.22474487 -1.22474487]
 [ 0.          0.          0.        ]
 [ 1.22474487  1.22474487  1.22474487]]
```

In [ ]:

```
# Sample data (replace with your own data)
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=200, n_features=4, n_classes=2, random_state=4

# Split data into training and testing sets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Import and train the Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf.predict(X_test)

# Evaluate the model performance (you can choose other metrics here)
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.875
```

In [ ]:

```
# Importing necessary libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Splitting 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_sta

# Creating a decision tree classifier
clf = DecisionTreeClassifier()

# Training the classifier
clf.fit(X_train, y_train)

# Making predictions on the testing set
y_pred = clf.predict(X_test)

# Evaluating the model
accuracy = clf.score(X_test, y_test)
print("Accuracy:", accuracy)

# Visualizing the decision tree
plt.figure(figsize=(12, 8))
```

```
plot_tree(clf, filled=True, feature_names=iris.feature_names, class_names=iris.targ
plt.show()
```

```
Accuracy: 1.0
```