assignmnet3-template

April 7, 2024

1 Assignment 3

[1]:

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
import matplotlib.pyplot as plt
```

[2]:

```
np.random.seed(156)
X, y = make_blobs(n_samples=300, centers=3, n_features=2, random_state=32)

# Mapping the numerical labels to colors
color_map = {0: 'orange', 1: 'green', 2: 'blue'}
colors = [color_map[label] for label in y]

# Creating a DataFrame
df = pd.DataFrame(X, columns=['feature1', 'feature2'])
df['color'] = colors

# Now df is your dataset with 'feature1', 'feature2', and 'color'
```

[3]:

```
# Setting random seed for reproducibility

def generate_clusters_color_them():
    # Generating synthetic data with 3 clusters
    X, y = make_blobs(n_samples=300, centers=3, n_features=2, random_state=32)

    # Mapping the numerical labels to colors
    color_map = {0: 'orange', 1: 'green', 2: 'blue'}
    colors = [color_map[label] for label in y]

    # Creating a DataFrame
    df = pd.DataFrame(X, columns=['feature1', 'feature2'])
    df['color'] = colors

    # Visualizing the data
    plt.scatter(df['feature1'], df['feature2'], c=df['color'])
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('Dataset with Three Clusters')
    plt.show()
```

[4]:

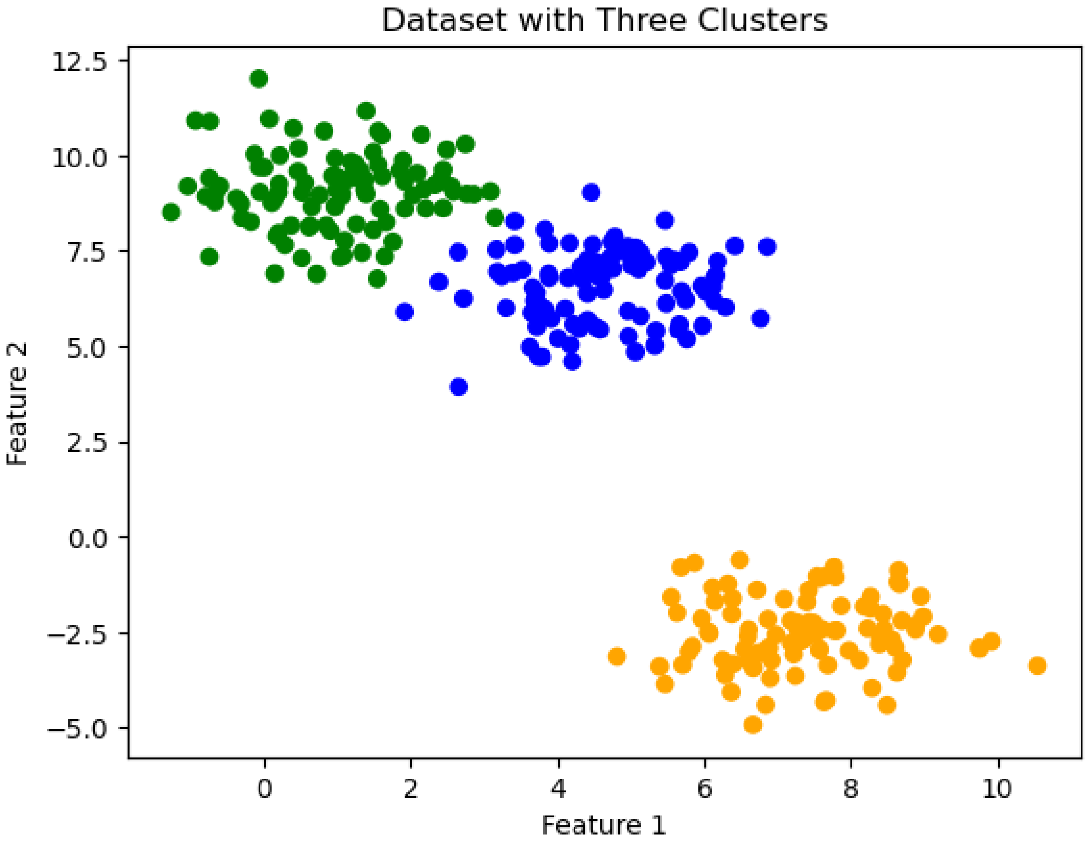
```
# Function to plot decision boundaries
def plot_decision_boundary(X, y, classifier, title):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                         np.arange(y_min, y_max, 0.1))
    Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.4)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor='k')
    plt.title(title)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()
```

1.1 Problem Description

You are provided with a synthetic dataset containing points categorized into three colors: 'orange', 'green', and 'blue'. These points are clustered and feature two distinct attributes. Your challenge is to create classification models that can accurately determine whether points are 'orange' or not and 'green' or not, considering the influence of the 'blue' points.

[5]:

```
# The input data and the 3 clusters
generate_clusters_color_them()
```



1.2 Q1

When a kernel other than "linear" is set, the SVC applies the kernel trick, which computes the similarity between pairs of data points using the kernel function without explicitly transforming the entire dataset. The kernel trick surpasses the otherwise necessary matrix transformation of the whole dataset by only considering the relations between all pairs of data points. The kernel function maps two vectors (each pair of observations) to their similarity using their dot product.

The hyperplane can then be calculated using the kernel function as if the dataset were represented in a higher-dimensional space. Using a kernel function instead of an explicit matrix transformation improves performance, as the kernel function has a time complexity of , whereas matrix transformation scales according to the specific transformation being applied.

In this example, we compare the most common kernel types of Support Vector Machines: the linear kernel ("linear"), the polynomial kernel ("poly"), the radial basis function kernel ("rbf")

1.2.1 Q1.1

```
[6]: df['is_blue'] = df['color'].apply(lambda x: 1 if x == 'blue' else 0)
```

1.2.2 Q1.2

[7]:

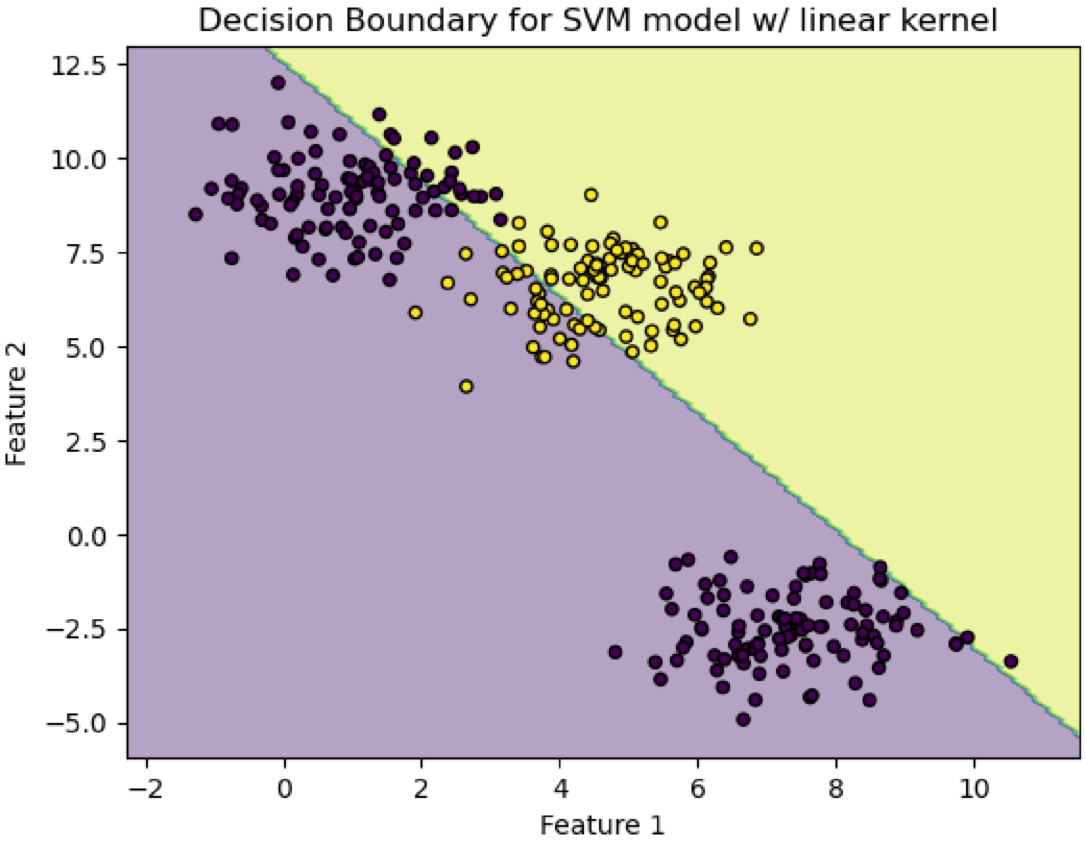
```
# Fit the SVM model with a linear kernel
X = df[['feature1', 'feature2']]
y_blue = df['is_blue']
X_train_blue, X_test_blue, y_train_blue, y_test_blue = train_test_split(X,␣
 ↪y_blue, test_size=0.2, random_state=156)

svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train_blue, y_train_blue)
predict = svm_linear.predict(X_test_blue)

print(accuracy_score(y_test_blue, predict))
print(f1_score(y_test_blue, predict))
```

```
0.85
0.7272727272727273
```

[8]:

```
# Plot the decision boundary
plot_decision_boundary(X.values, y_blue.values, svm_linear, "Decision Boundary␣
 ↪for SVM model w/ linear kernel ")
```

```
C:\Users\adeen_pn07n1p\anaconda3\Lib\site-packages\sklearn\base.py:493:
UserWarning: X does not have valid feature names, but SVC was fitted with
feature names
  warnings.warn(
```

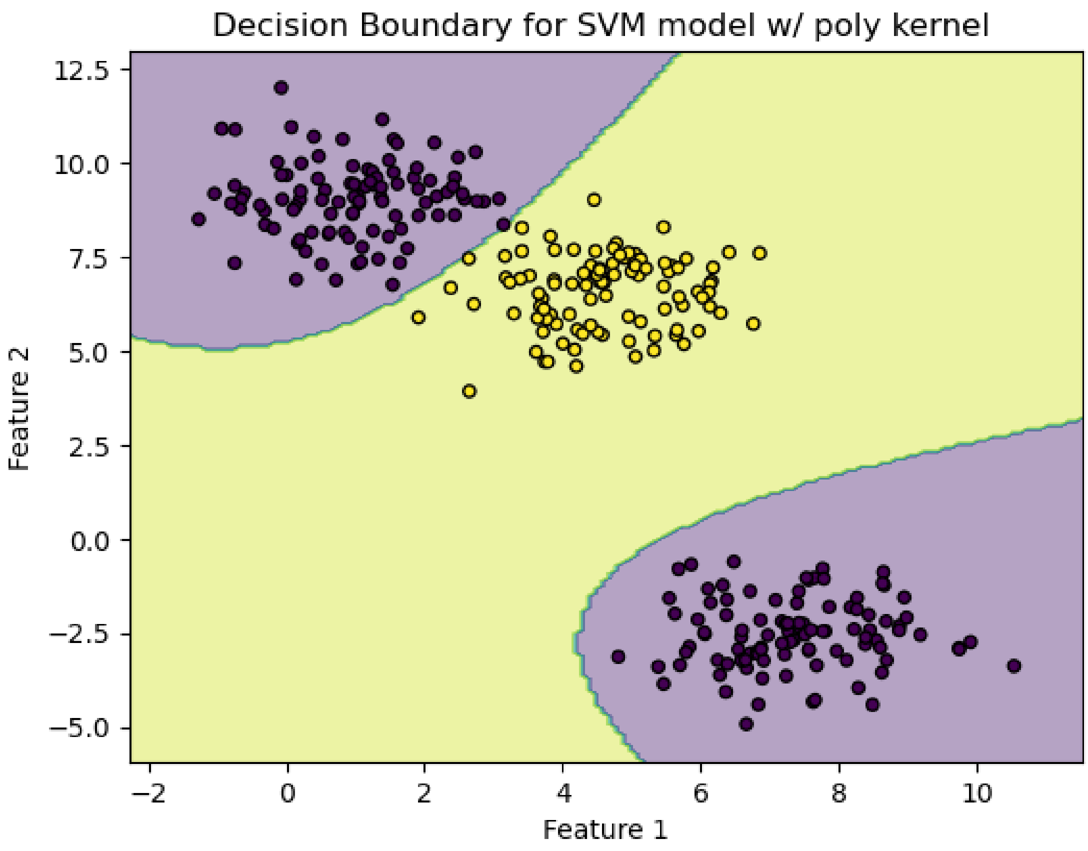


```
[9]: svm_poly = SVC(kernel='poly')
     svm_poly.fit(X_train_blue, y_train_blue)
     predict = svm_poly.predict(X_test_blue)

     print(accuracy_score(y_test_blue, predict))
     print(f1_score(y_test_blue, predict))
```

```
1.0
1.0
```

```
[10]: plot_decision_boundary(X.values, y_blue.values, svm_poly, "Decision Boundary␣
       ↪for SVM model w/ poly kernel ")
```

```
C:\Users\adeen_pn07n1p\anaconda3\Lib\site-packages\sklearn\base.py:493:
UserWarning: X does not have valid feature names, but SVC was fitted with
feature names
  warnings.warn(
```

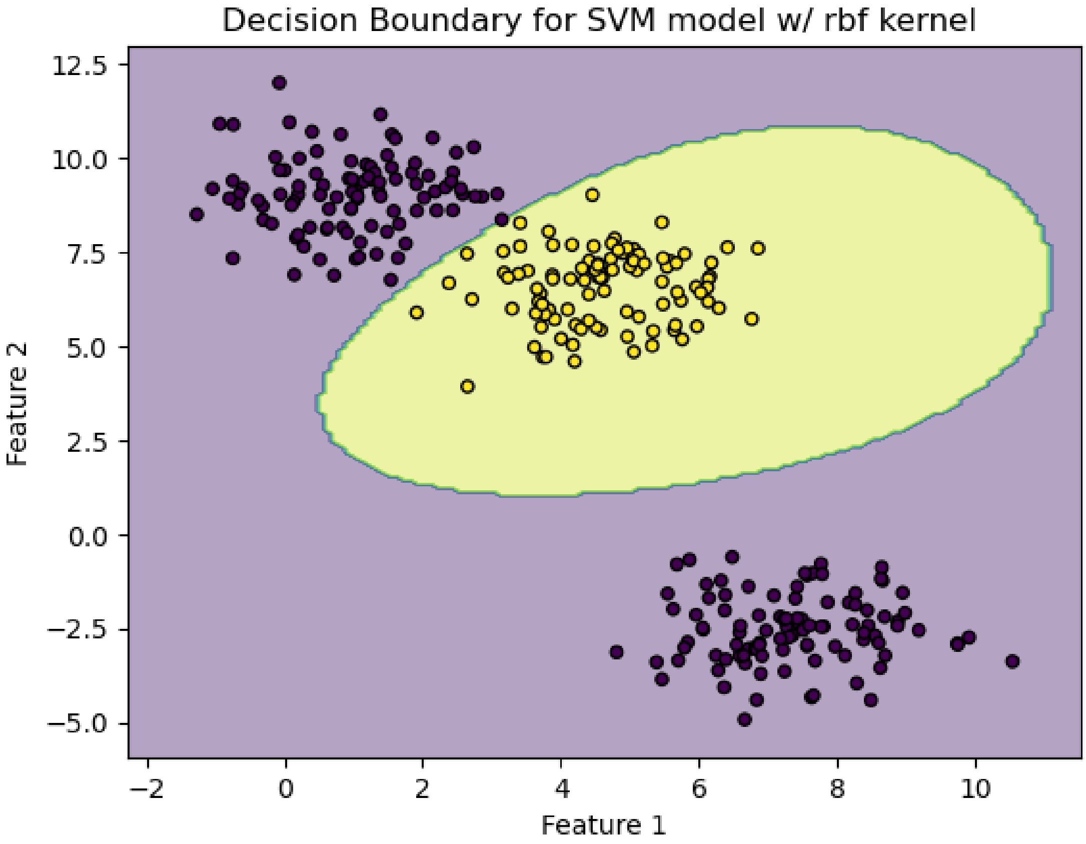


```
[11]: svm_rbf = SVC(kernel='rbf')
      svm_rbf.fit(X_train_blue, y_train_blue)
      predict = svm_rbf.predict(X_test_blue)

      print(accuracy_score(y_test_blue, predict))
      print(f1_score(y_test_blue, predict))
```

```
1.0
1.0
```

```
[12]: plot_decision_boundary(X.values, y_blue.values, svm_rbf, "Decision Boundary for␣
        ↪SVM model w/ rbf kernel ")
```

```
C:\Users\adeen_pn07n1p\anaconda3\Lib\site-packages\sklearn\base.py:493:
UserWarning: X does not have valid feature names, but SVC was fitted with
feature names
  warnings.warn(
```



1.2.3 Q1.3

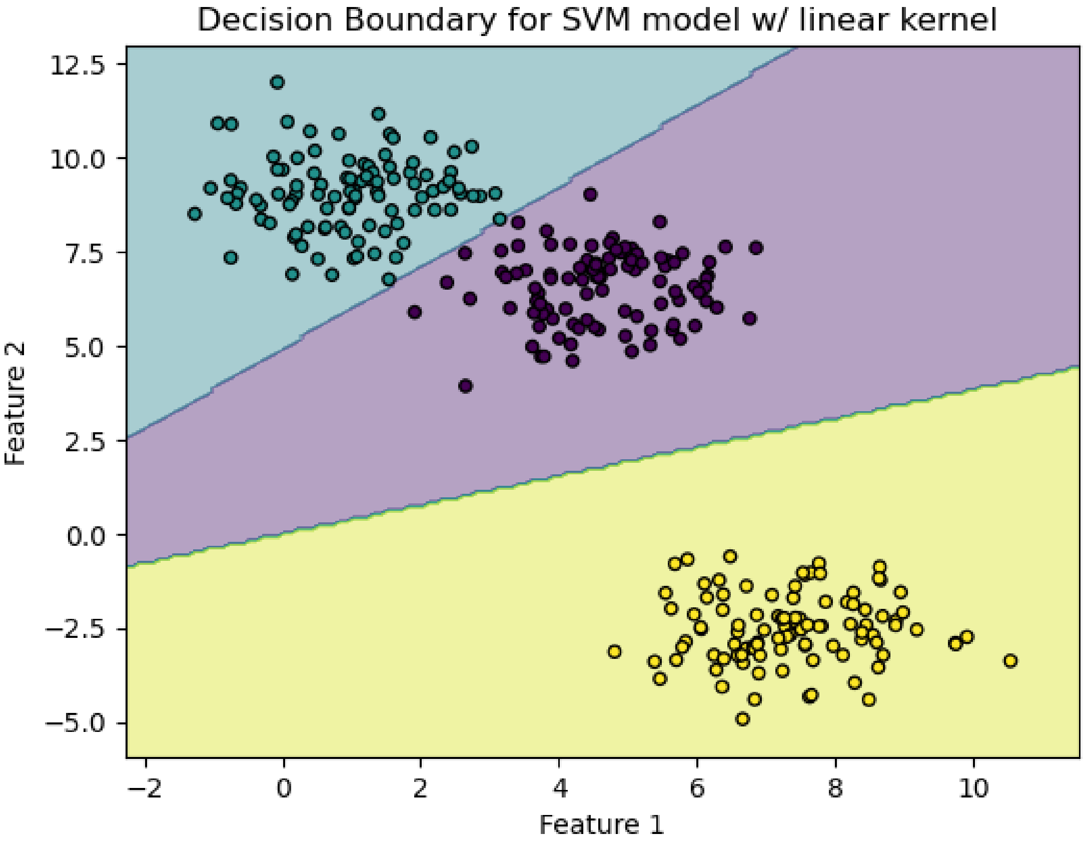
```
[13]: from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['color_encoded'] = le.fit_transform(df['color'])

X = df[['feature1', 'feature2']]
y = df['color_encoded']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=156)

svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train, y_train)
predict = svm_linear.predict(X_test)

print(accuracy_score(y_test, predict))
print(f1_score(y_test, predict, average='weighted'))
print(confusion_matrix(y_test, predict))
```

```
1.0
```

```
1.0
[[18  0  0]
 [ 0 25  0]
 [ 0  0 17]]
```

[14]:
```
plot_decision_boundary(X.values, y.values, svm_linear, "Decision Boundary for␣
  ↪SVM model w/ linear kernel ")
```

```
C:\Users\adeen_pn07n1p\anaconda3\Lib\site-packages\sklearn\base.py:493:
UserWarning: X does not have valid feature names, but SVC was fitted with
feature names
  warnings.warn(
```

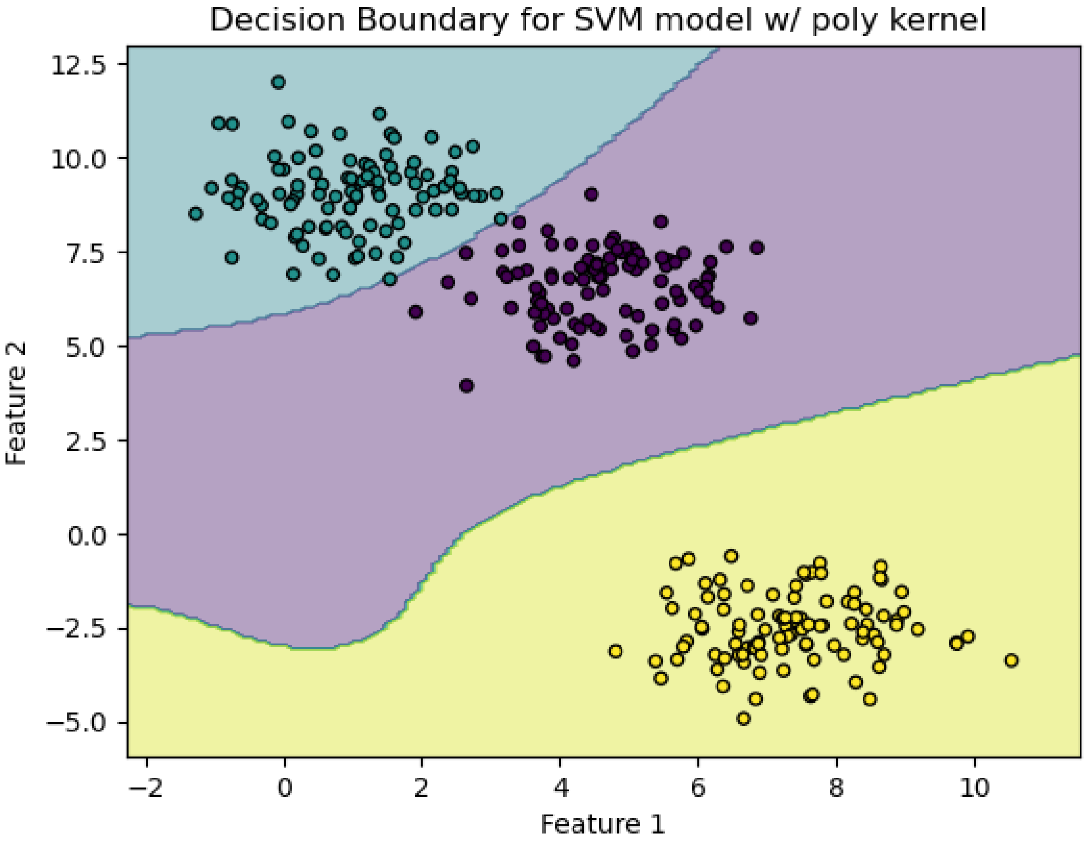


[15]:
```
svm_poly = SVC(kernel='poly')
svm_poly.fit(X_train, y_train)
predict = svm_poly.predict(X_test)

print(accuracy_score(y_test, predict))
print(f1_score(y_test, predict, average='weighted'))
print(confusion_matrix(y_test, predict))
```

```
0.9833333333333333
0.9833884905313478
[[18  0  0]
 [ 1 24  0]
 [ 0  0 17]]
```

[16]:
```
plot_decision_boundary(X.values, y.values, svm_poly, "Decision Boundary for SVM␣
↪model w/ poly kernel ")
```

```
C:\Users\adeen_pn07n1p\anaconda3\Lib\site-packages\sklearn\base.py:493:
UserWarning: X does not have valid feature names, but SVC was fitted with
feature names
  warnings.warn(
```

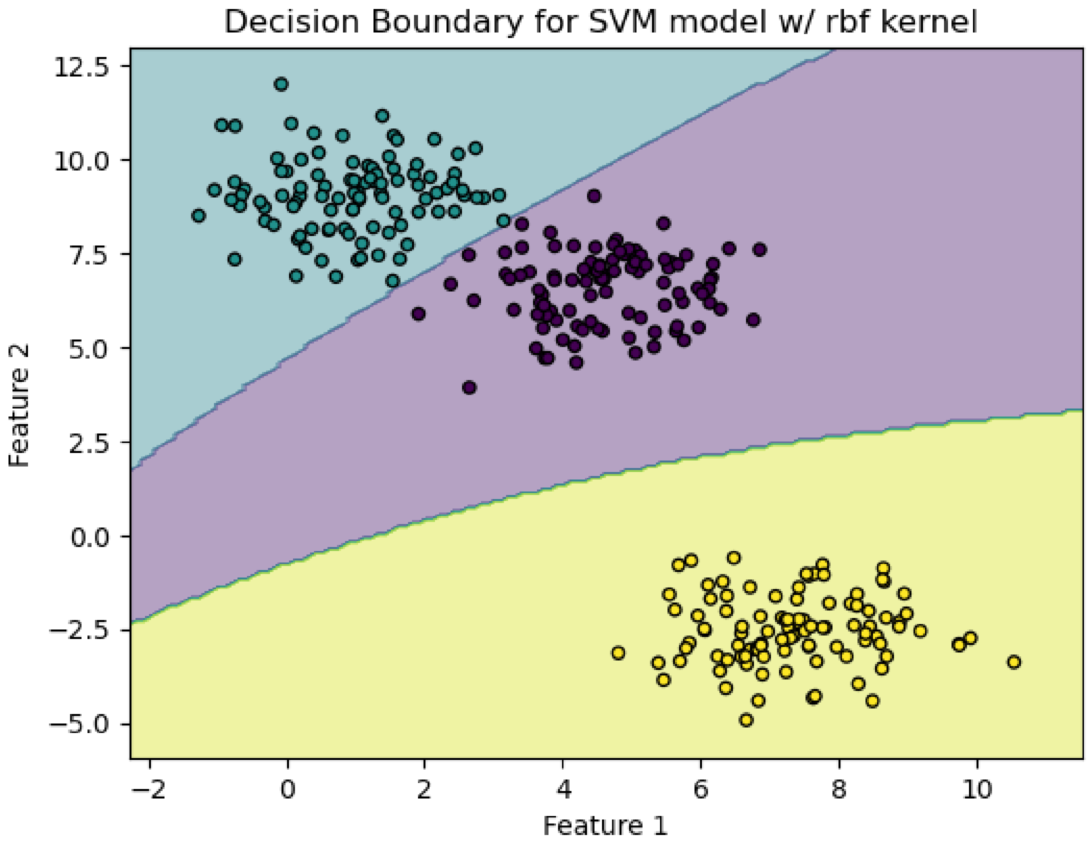


[17]:
```
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(X_train, y_train)
predict = svm_rbf.predict(X_test)

print(accuracy_score(y_test, predict))
print(f1_score(y_test, predict, average='weighted'))
print(confusion_matrix(y_test, predict))
```

```
1.0
1.0
[[18  0  0]
 [ 0 25  0]
 [ 0  0 17]]
```

[18]:
```
plot_decision_boundary(X.values, y.values, svm_rbf, "Decision Boundary for SVM␣
↪model w/ rbf kernel ")
```

```
C:\Users\adeen_pn07n1p\anaconda3\Lib\site-packages\sklearn\base.py:493:
UserWarning: X does not have valid feature names, but SVC was fitted with
feature names
  warnings.warn(
```



1.2.4 Q1.4

1.3 Q2

[19]:

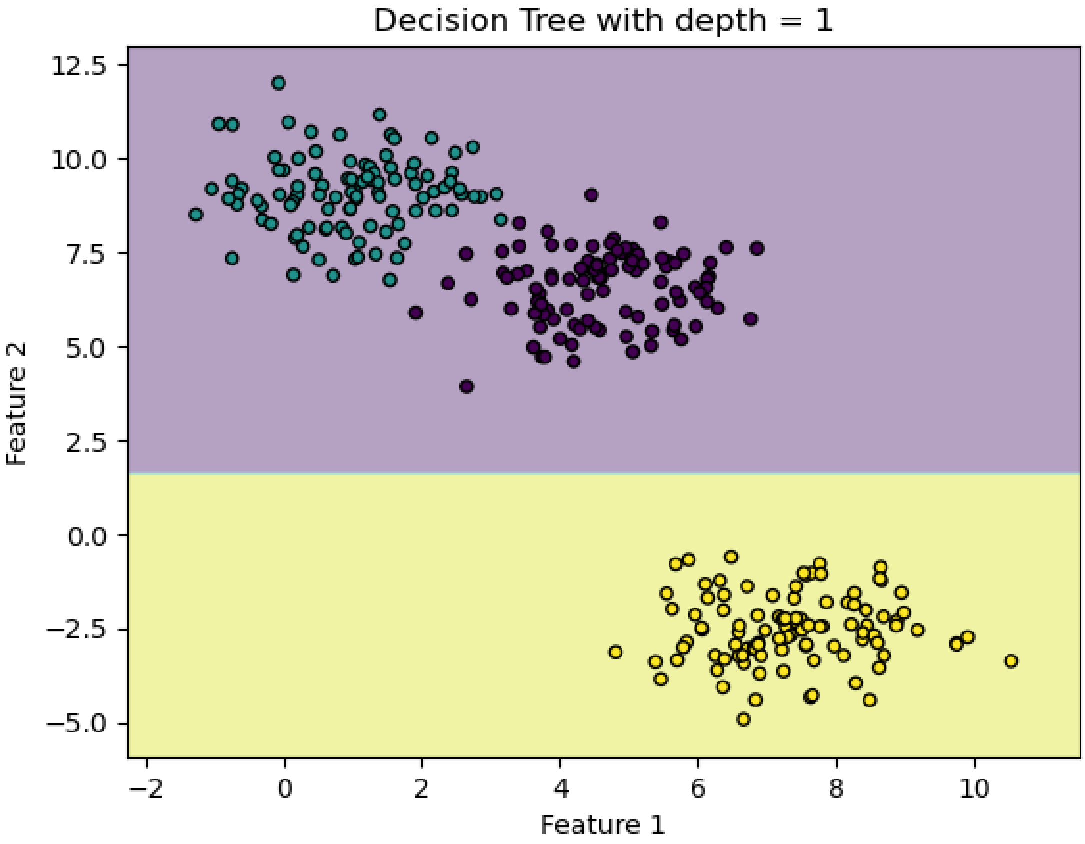
```
le = LabelEncoder()
df['color_encoded'] = le.fit_transform(df['color'])

X = df[['feature1', 'feature2']]
y = df['color_encoded']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=156)
dt_classifiers= []
depths = [1, 2, 3, 4, 5, 10, None]
for depth in depths:
    model = DecisionTreeClassifier(max_depth=depth)
    model = model.fit(X_train, y_train)
    predict = model.predict(X_test)
    dt_classifiers.append(model)
    accuracy = accuracy_score(y_test, predict)
    f1 = f1_score(y_test, predict, average='macro')

    print(f"Depth: {depth}, Accuracy Score: {accuracy}, F1 Score: {f1}")
```

```
Depth: 1, Accuracy Score: 0.5833333333333334, F1 Score: 0.5300546448087432
Depth: 2, Accuracy Score: 0.9833333333333333, F1 Score: 0.9839402427637722
Depth: 3, Accuracy Score: 0.9666666666666667, F1 Score: 0.9681481481481482
Depth: 4, Accuracy Score: 0.9666666666666667, F1 Score: 0.9681481481481482
Depth: 5, Accuracy Score: 0.9666666666666667, F1 Score: 0.9681481481481482
Depth: 10, Accuracy Score: 0.9666666666666667, F1 Score: 0.9681481481481482
Depth: None, Accuracy Score: 0.9666666666666667, F1 Score: 0.9681481481481482
```

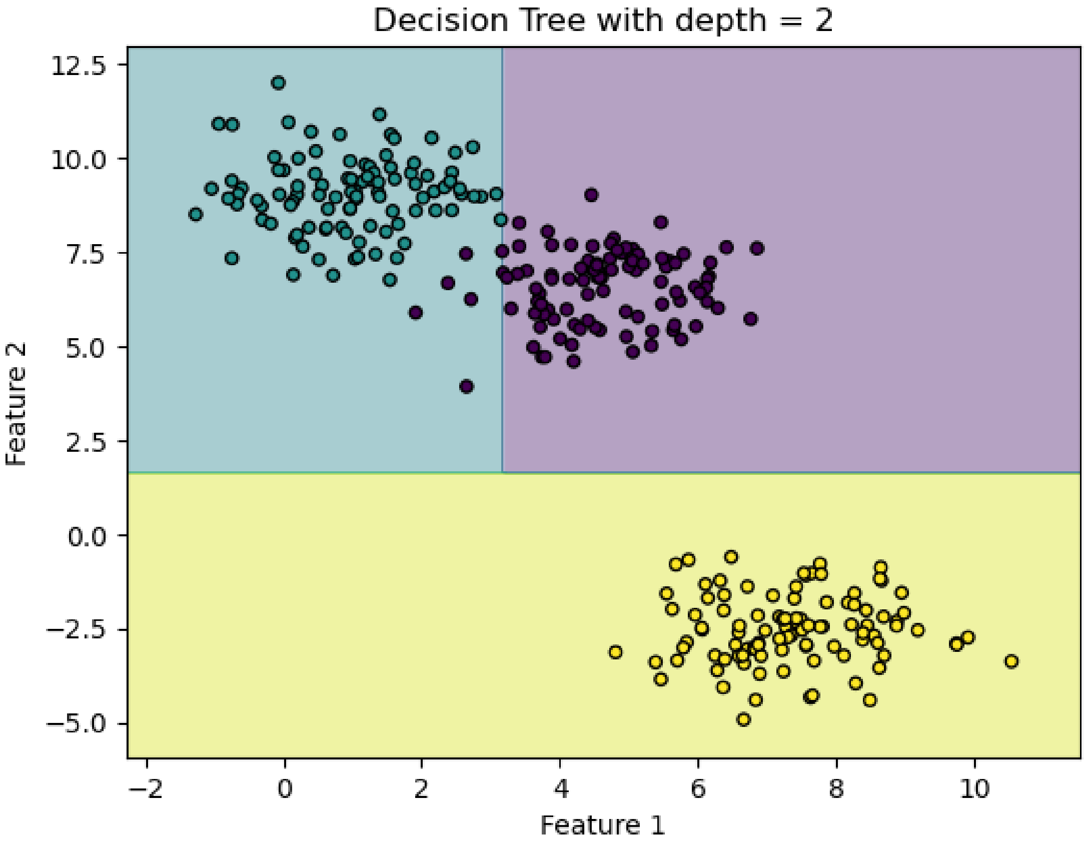
[20]:

```
for i in range(len(dt_classifiers)):
    plot_decision_boundary(X.values, y.values, dt_classifiers[i], f"Decision␣
 ↪Tree with depth = {depths[i]}")
```

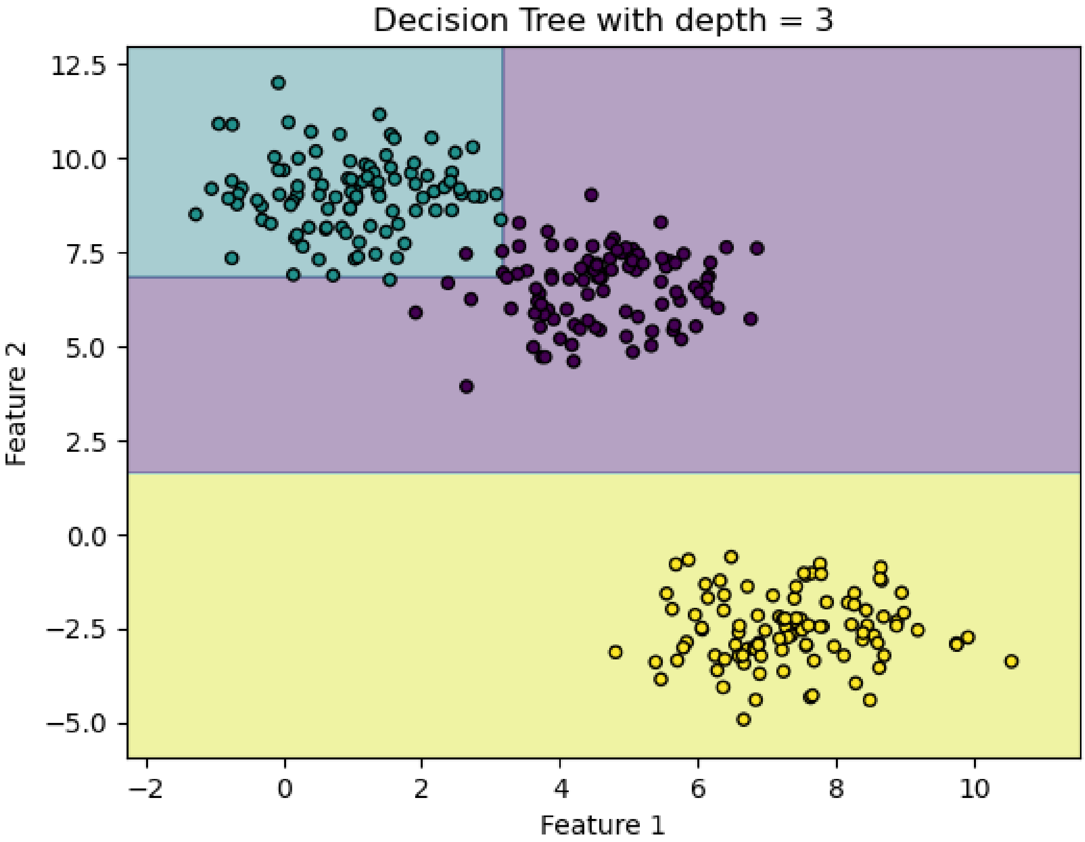
```
C:\Users\adeen_pn07n1p\anaconda3\Lib\site-packages\sklearn\base.py:493:
UserWarning: X does not have valid feature names, but DecisionTreeClassifier was
fitted with feature names
  warnings.warn(
```



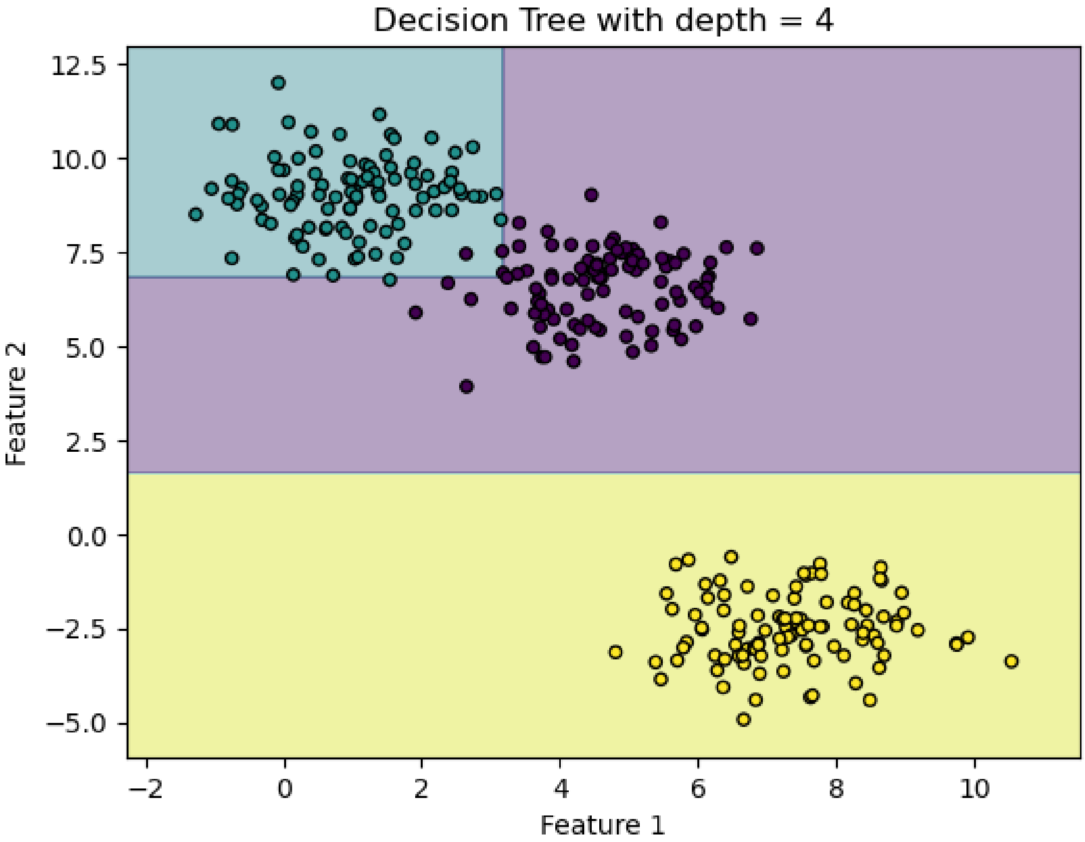
```
C:\Users\adeen_pn07n1p\anaconda3\Lib\site-packages\sklearn\base.py:493:
UserWarning: X does not have valid feature names, but DecisionTreeClassifier was
fitted with feature names
  warnings.warn(
```



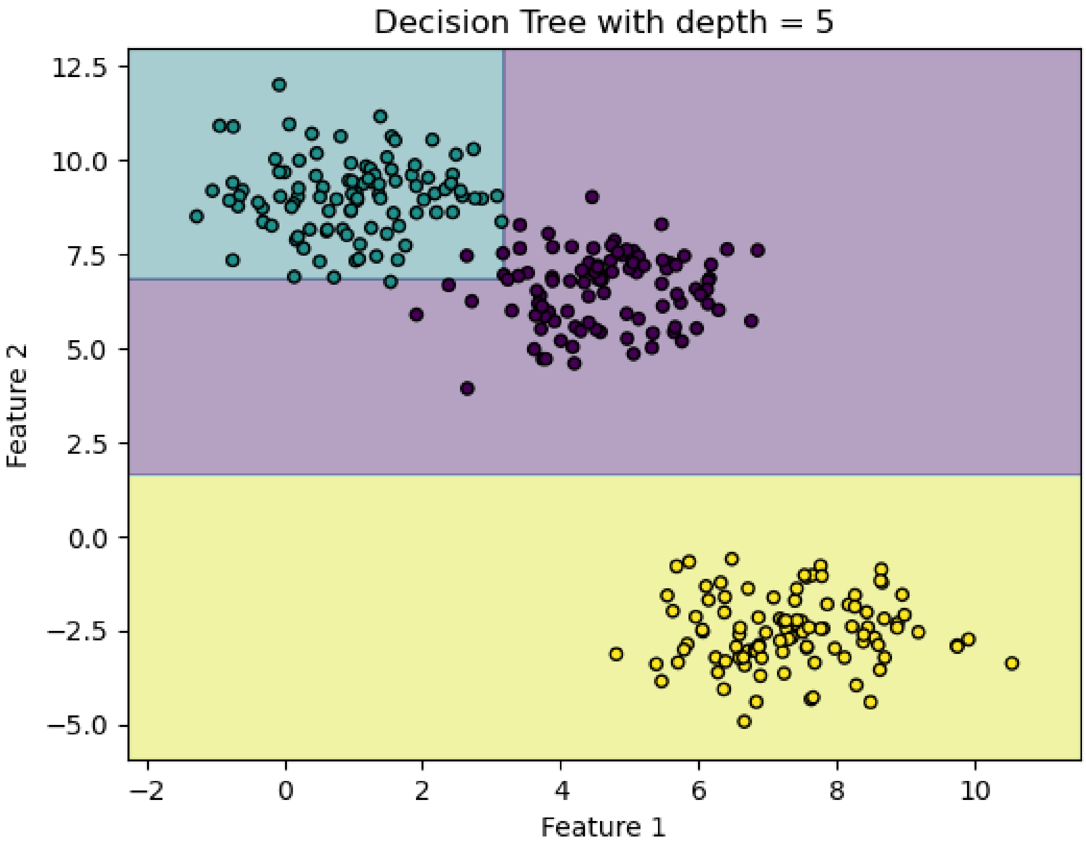
```
C:\Users\adeen_pn07n1p\anaconda3\Lib\site-packages\sklearn\base.py:493:
UserWarning: X does not have valid feature names, but DecisionTreeClassifier was
fitted with feature names
  warnings.warn(
```



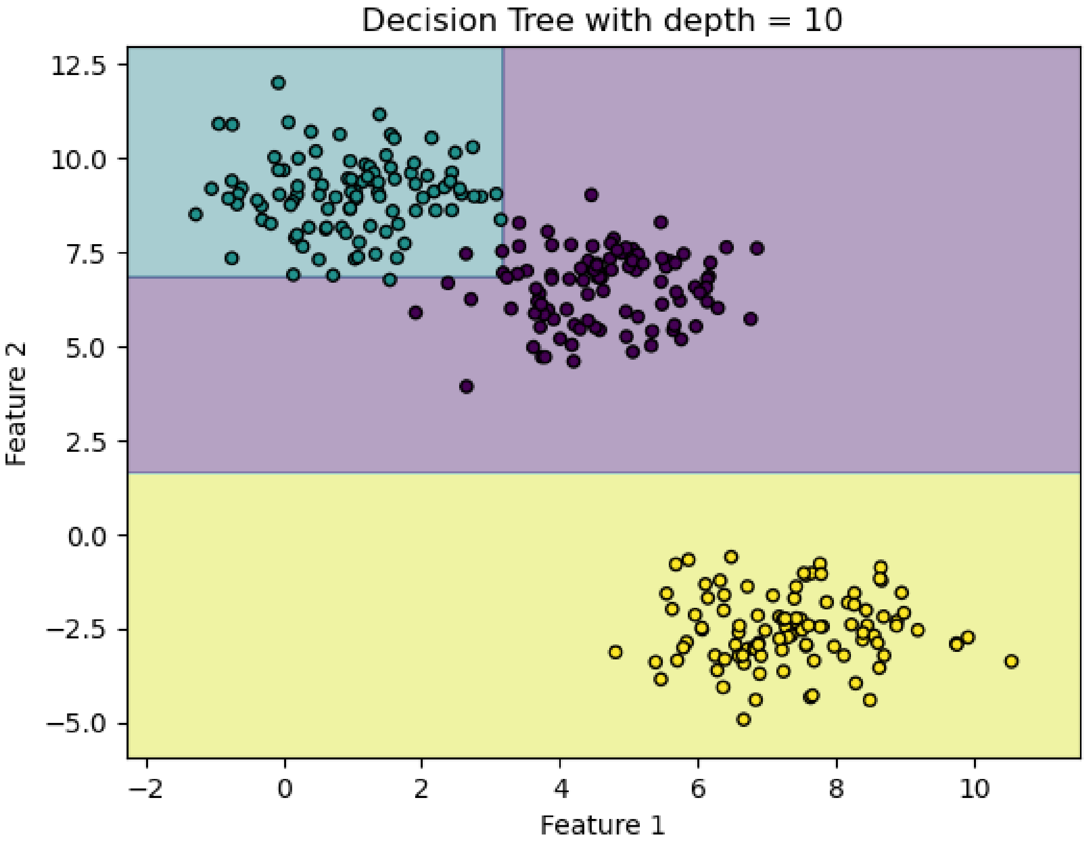
```
C:\Users\adeen_pn07n1p\anaconda3\Lib\site-packages\sklearn\base.py:493:
UserWarning: X does not have valid feature names, but DecisionTreeClassifier was
fitted with feature names
  warnings.warn(
```



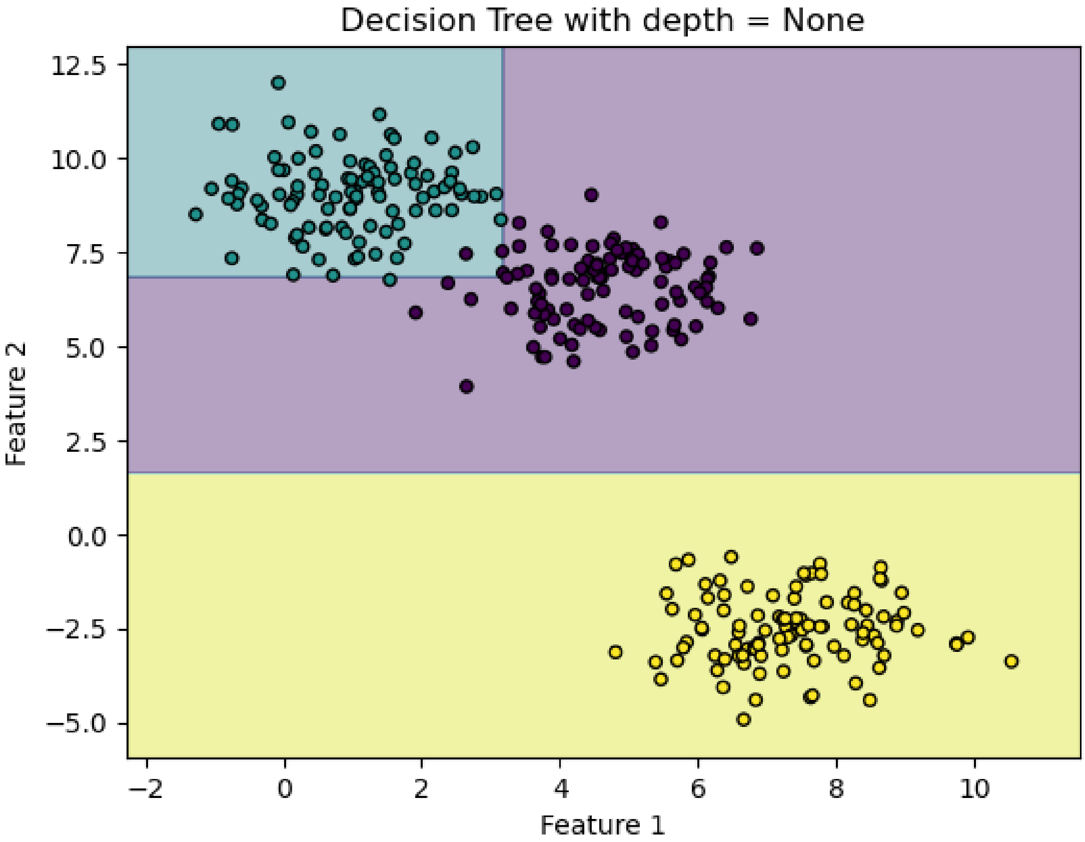
```
C:\Users\adeen_pn07n1p\anaconda3\Lib\site-packages\sklearn\base.py:493:
UserWarning: X does not have valid feature names, but DecisionTreeClassifier was
fitted with feature names
  warnings.warn(
```



```
C:\Users\adeen_pn07n1p\anaconda3\Lib\site-packages\sklearn\base.py:493:
UserWarning: X does not have valid feature names, but DecisionTreeClassifier was
fitted with feature names
  warnings.warn(
```



```
C:\Users\adeen_pn07n1p\anaconda3\Lib\site-packages\sklearn\base.py:493:
UserWarning: X does not have valid feature names, but DecisionTreeClassifier was
fitted with feature names
  warnings.warn(
```



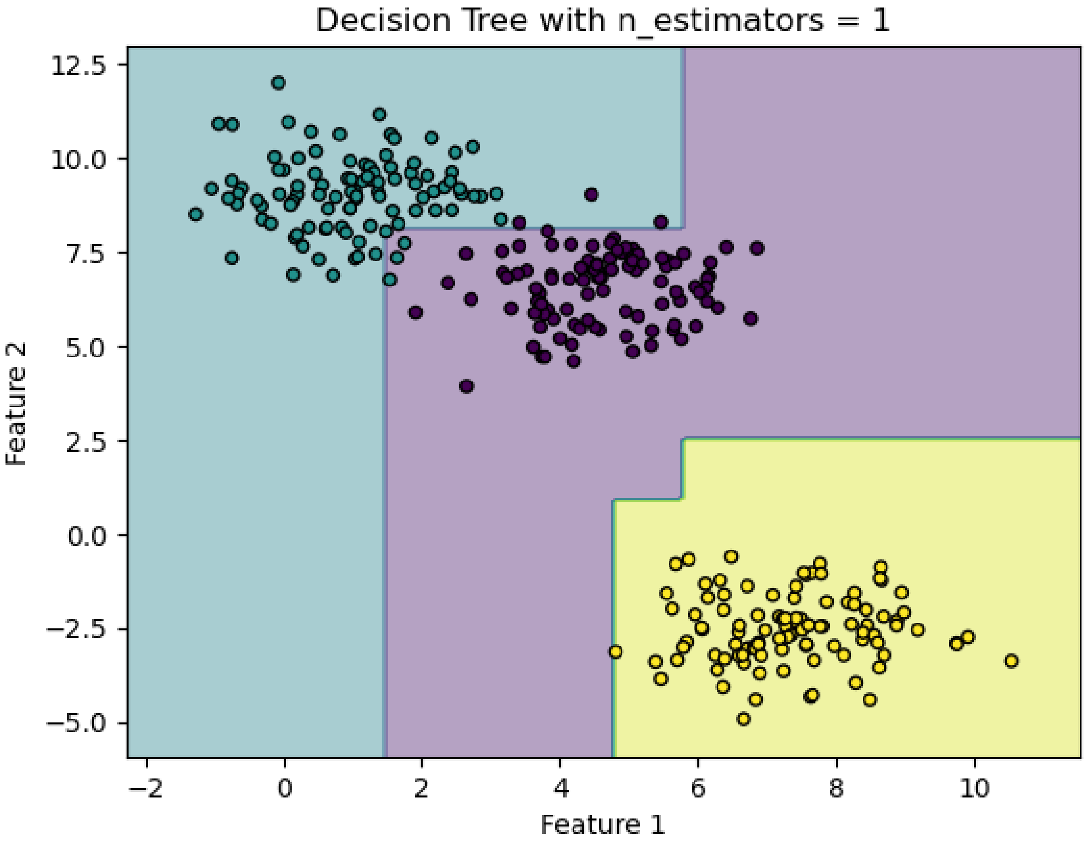
[22]:
```
rf_classifiers = []
num_estimators = [1, 5, 10, 50, 100]
for n in num_estimators:
    model = RandomForestClassifier(n_estimators=n)
    model = model.fit(X_train, y_train)
    predict = model.predict(X_test)
    rf_classifiers.append(model)
    accuracy = accuracy_score(y_test, predict)
    f1 = f1_score(y_test, predict, average='macro')

    print(f"Number of Estimators: {n}, Accuracy Score: {accuracy}, F1 Score:␣
 ↪{f1}")
```

```
Number of Estimators: 1, Accuracy Score: 0.95, F1 Score: 0.9525648097076669
Number of Estimators: 5, Accuracy Score: 0.9833333333333333, F1 Score:
0.9841882699025556
Number of Estimators: 10, Accuracy Score: 1.0, F1 Score: 1.0
Number of Estimators: 50, Accuracy Score: 1.0, F1 Score: 1.0
Number of Estimators: 100, Accuracy Score: 1.0, F1 Score: 1.0
```

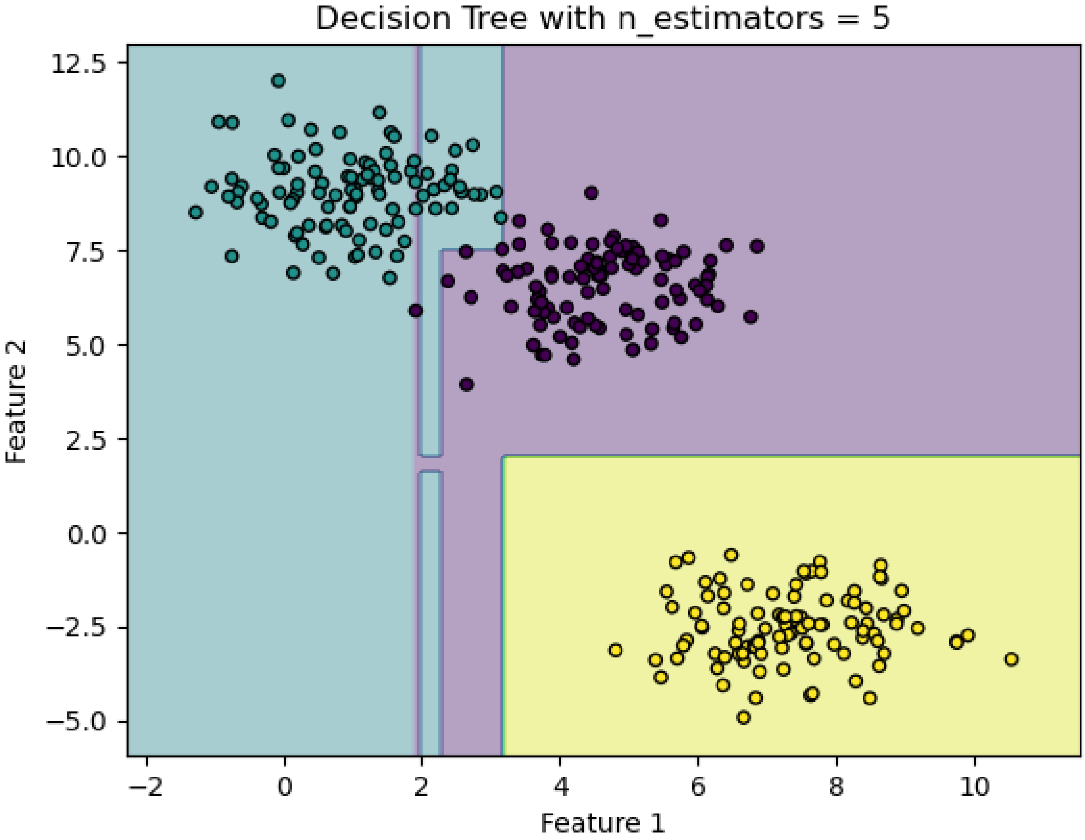
[23]:

```
for i in range(len(num_estimators)):
    plot_decision_boundary(X.values, y.values, rf_classifiers[i], f"Decision␣
  ↪Tree with n_estimators = {num_estimators[i]}")
```

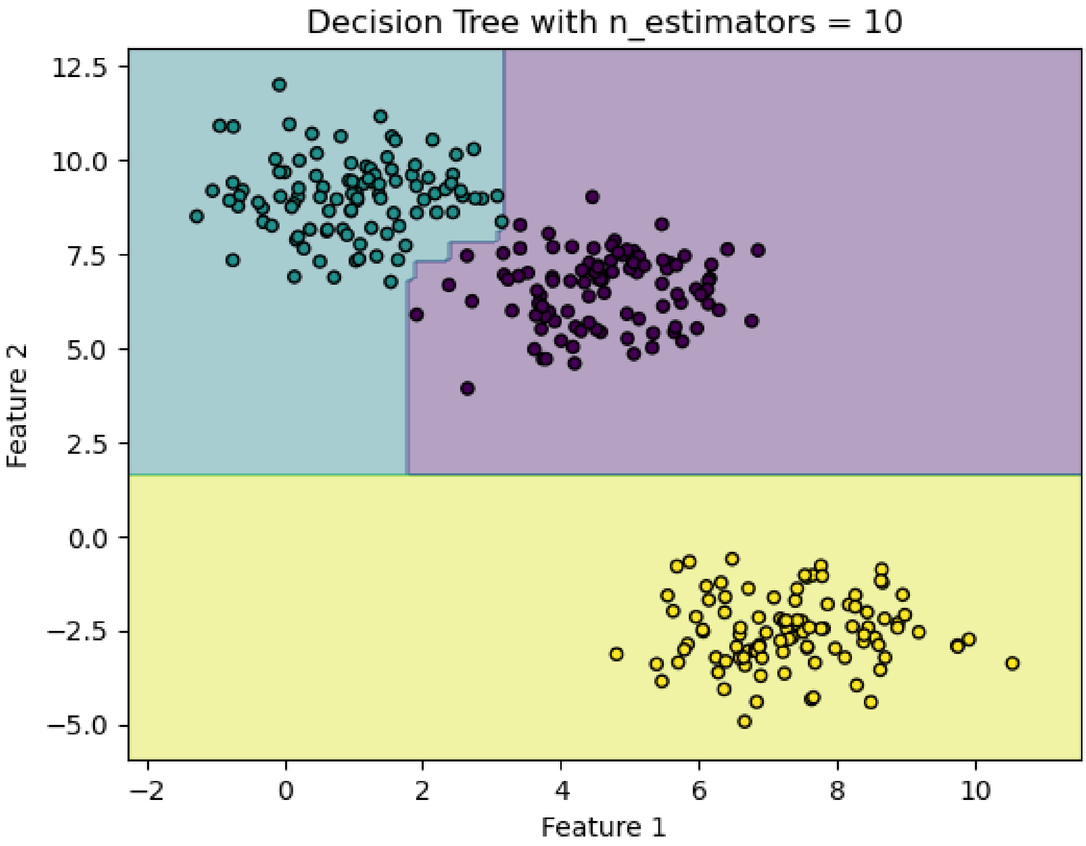
```
C:\Users\adeen_pn07n1p\anaconda3\Lib\site-packages\sklearn\base.py:493:
UserWarning: X does not have valid feature names, but RandomForestClassifier was
fitted with feature names
  warnings.warn(
```



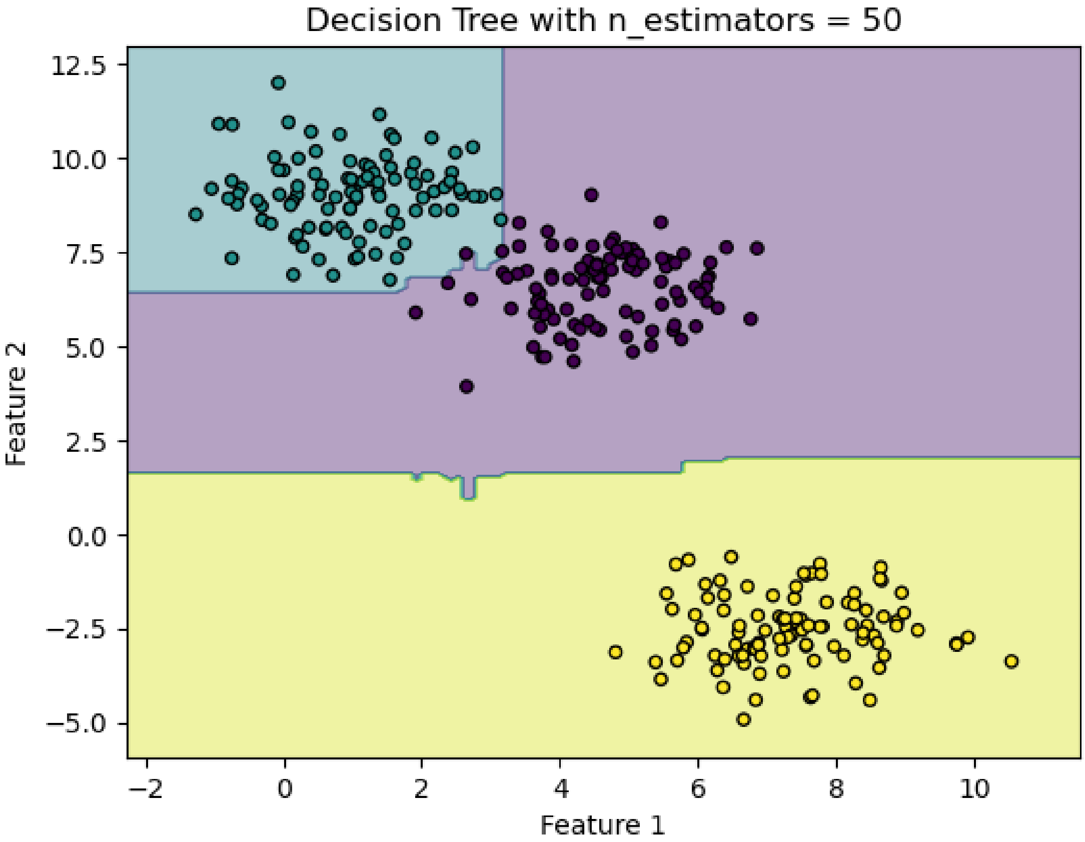
```
C:\Users\adeen_pn07n1p\anaconda3\Lib\site-packages\sklearn\base.py:493:
UserWarning: X does not have valid feature names, but RandomForestClassifier was
fitted with feature names
  warnings.warn(
```



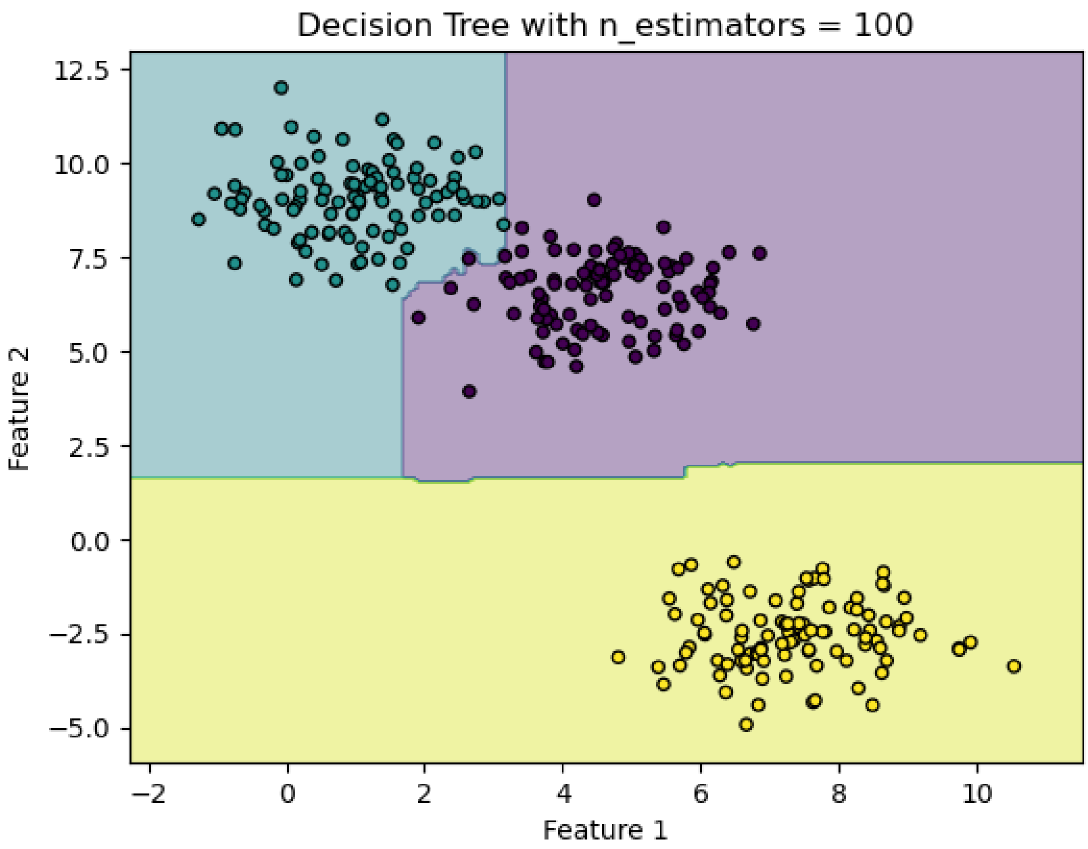
```
C:\Users\adeen_pn07n1p\anaconda3\Lib\site-packages\sklearn\base.py:493:
UserWarning: X does not have valid feature names, but RandomForestClassifier was
fitted with feature names
  warnings.warn(
```



```
C:\Users\adeen_pn07n1p\anaconda3\Lib\site-packages\sklearn\base.py:493:
UserWarning: X does not have valid feature names, but RandomForestClassifier was
fitted with feature names
  warnings.warn(
```



```
C:\Users\adeen_pn07n1p\anaconda3\Lib\site-packages\sklearn\base.py:493:
UserWarning: X does not have valid feature names, but RandomForestClassifier was
fitted with feature names
  warnings.warn(
```



1.3.1 Q2.3

…

2 Q3

[24]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

[25]:

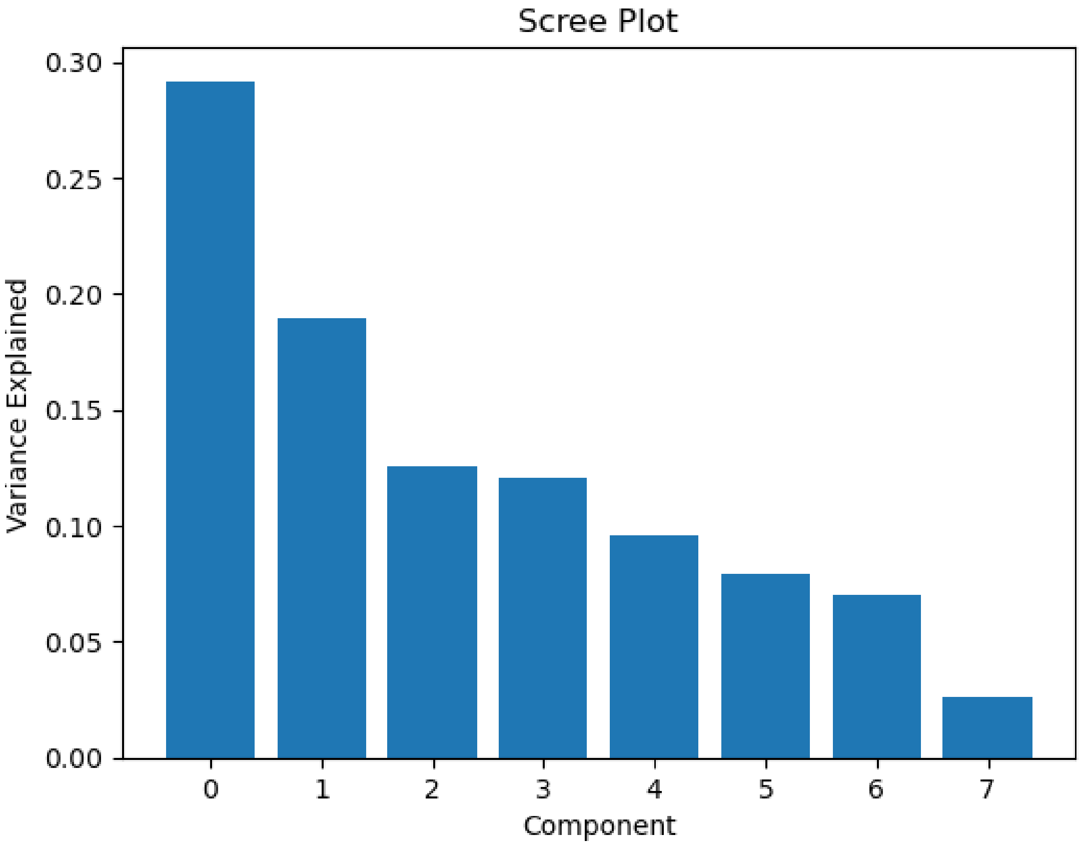
```
spotify_df = pd.read_csv("spotify_songs.csv")
spotify_df = spotify_df[['playlist_genre', 'danceability', 'energy',␣
  ↪'loudness', 'speechiness', 'acousticness', 'instrumentalness', 'liveness',␣
  ↪'valence']]
spotify_df = spotify_df[(spotify_df['playlist_genre'] == 'rock') |␣
  ↪(spotify_df['playlist_genre'] == 'rap')]
spotify_df = spotify_df.sample(n=5000)
```

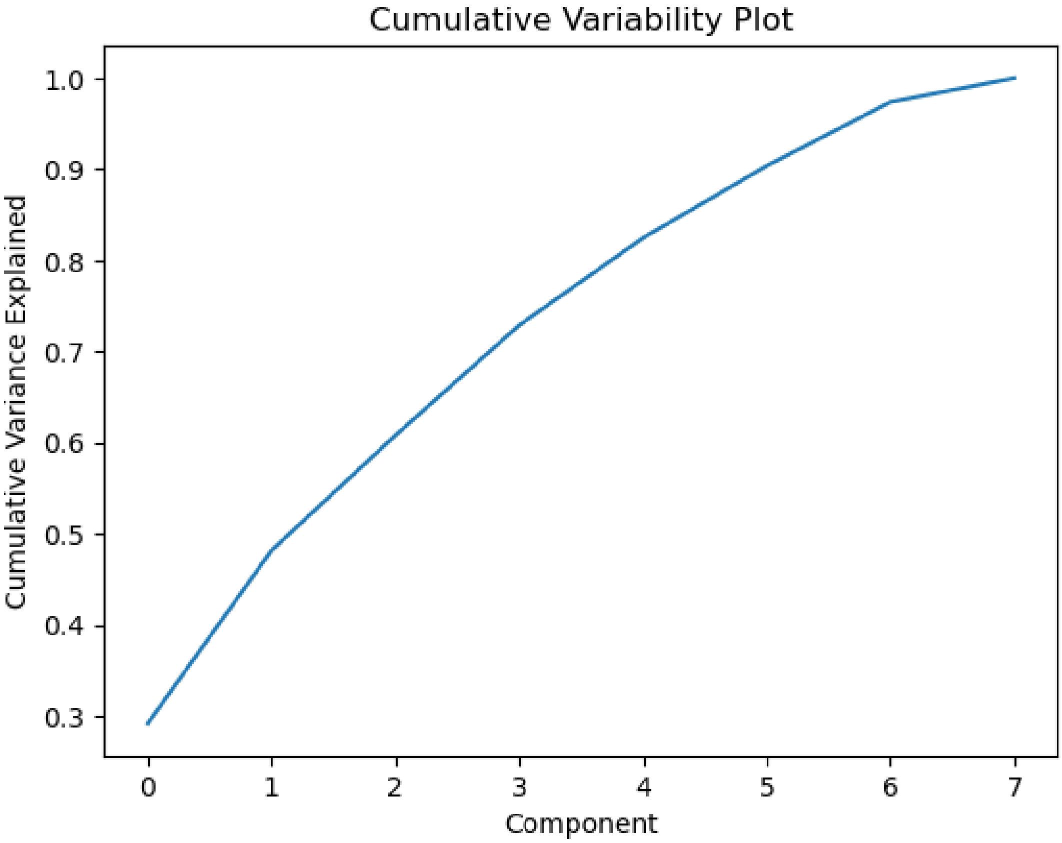
[26]:

```
scaler = StandardScaler()
scaled_features = scaler.fit_transform(spotify_df.drop('playlist_genre',␣
 ↪axis=1))
```

[27]:

```
pca = PCA()
pca_features = pca.fit_transform(scaled_features)

plt.bar(x=range(len(pca.explained_variance_ratio_)), height=pca.
 ↪explained_variance_ratio_)
plt.title('Scree Plot')
plt.xlabel('Component')
plt.ylabel('Variance Explained')
plt.show()

plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.title('Cumulative Variability Plot')
plt.xlabel('Component')
plt.ylabel('Cumulative Variance Explained')
plt.show()

print(f"Variability explained by the first two principal axes: {sum(pca.
 ↪explained_variance_ratio_[:2])*100}%")
```



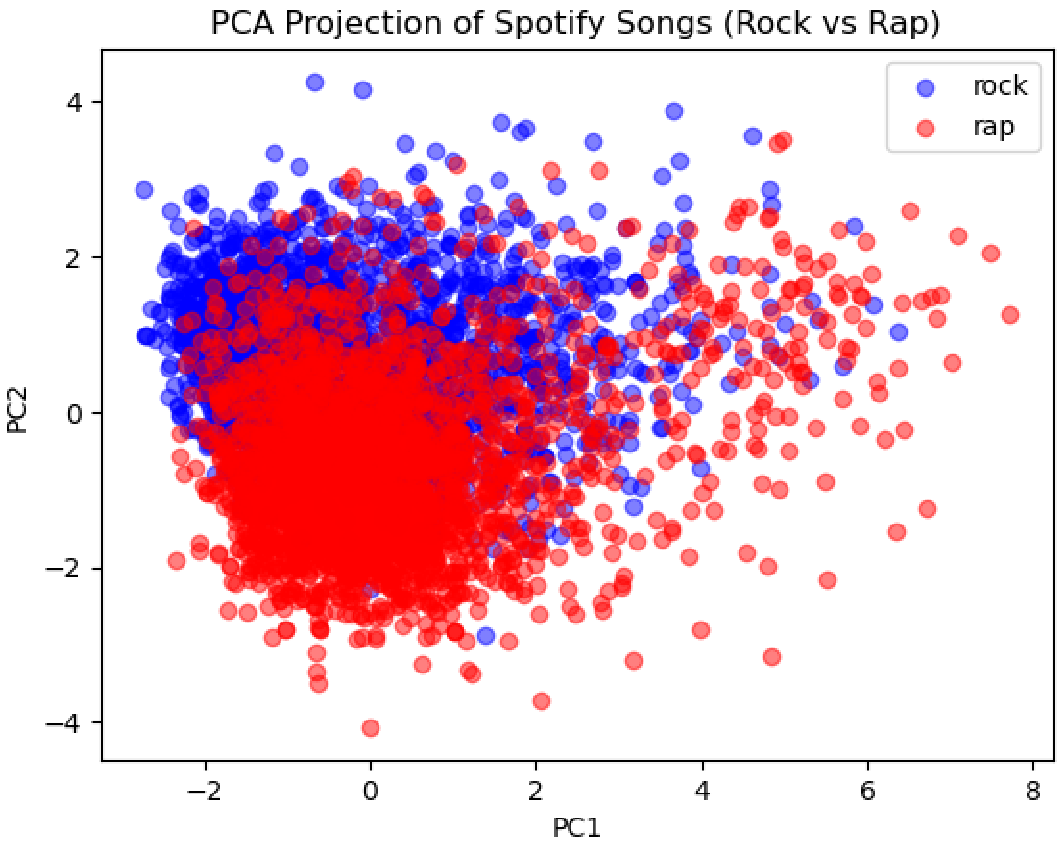


```
Variability explained by the first two principal axes: 48.16269527220485%
```

[28]:
```
features = spotify_df.drop('playlist_genre', axis=1).columns
coefficients = pca.components_[0]
pc1_formula = " + ".join([f"({coefficient:.4f} * {feature})" for coefficient,␣
  ↪feature in zip(coefficients, features)])
print(f"PC1 = {pc1_formula}")
```

```
PC1 = (0.1195 * danceability) + (-0.5798 * energy) + (-0.5284 * loudness) +
(-0.0044 * speechiness) + (0.4694 * acousticness) + (0.3442 * instrumentalness)
+ (-0.1332 * liveness) + (-0.1170 * valence)
```

[29]:
```
for genre, color in [('rock', 'blue'), ('rap', 'red')]:
    index = spotify_df['playlist_genre'] == genre
    plt.scatter(pca_features[index, 0], pca_features[index, 1], c=color,␣
  ↪label=genre, alpha=0.5)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('PCA Projection of Spotify Songs (Rock vs Rap)')
plt.legend()
plt.show()
```



3 Q4

[31]:

```
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix
```

[32]:

```
spotify_df = pd.read_csv("spotify_songs.csv")
spotify_df = spotify_df[['playlist_genre', 'danceability', 'energy',␣
  ↪'loudness', 'speechiness', 'acousticness', 'instrumentalness', 'liveness',␣
  ↪'valence']]
spotify_df = spotify_df[(spotify_df['playlist_genre'] == 'rock') |␣
  ↪(spotify_df['playlist_genre'] == 'rap')]
spotify_df = spotify_df.sample(n=5000)

scaler = StandardScaler()
```

```
scaled_features = scaler.fit_transform(spotify_df.drop('playlist_genre',␣
 ↪axis=1))
```

[33]:
```
spotify_df['playlist_genre'] = spotify_df['playlist_genre'].map({'rap': 0,␣
 ↪'rock': 1})
```

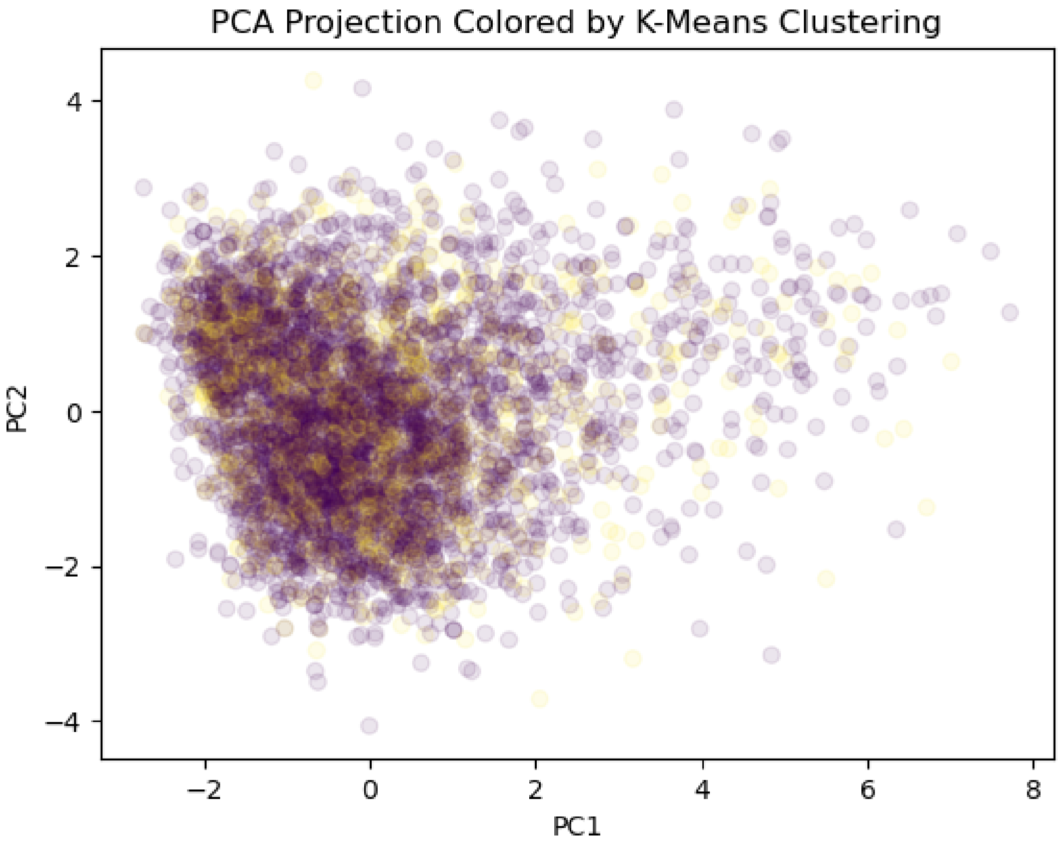
[34]:
```
model = KMeans(n_clusters=2)
model.fit(scaled_features)
```

[34]:
```
KMeans(n_clusters=2)
```

[41]:
```
print(confusion_matrix(spotify_df.playlist_genre, model.labels_))
```

```
[[2116  578]
 [1675  631]]
```

[43]:
```
plt.scatter(pca_features[:, 0], pca_features[:, 1], c=model.labels_, alpha=0.1)

plt.title('PCA Projection Colored by K-Means Clustering')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```



[37]:

```
features = ['danceability', 'energy', 'loudness', 'speechiness',␣
 ↪'acousticness', 'instrumentalness', 'liveness', 'valence']

for i, center in enumerate(model.cluster_centers_):
    print(f"Center of Cluster {i}:")
    for feature, value in zip(features, center):
        print(f"{feature}: {value:.2f}")
    print()
```

```
Center of Cluster 0:
danceability: 0.02
energy: 0.38
loudness: 0.37
speechiness: 0.09
acousticness: -0.32
instrumentalness: -0.19
liveness: 0.08
valence: 0.11

Center of Cluster 1:
danceability: -0.06
energy: -1.20
loudness: -1.16
speechiness: -0.28
acousticness: 1.00
instrumentalness: 0.60
liveness: -0.26
valence: -0.35
```