ML-LAB9

NAME:NIKHIL GARUDA

SRN: PES2UG23CS195

SEC:C

Analysis Questions

Moons Dataset Questions

1. **Inferences about the Linear Kernel's performance:**
   The fundamental limitation is that a linear decision boundary is incapable of conforming to the distinct crescent-shaped structure of the data clusters. This geometric mismatch means the model cannot accurately distinguish between the two classes, ultimately resulting in poor classification performance and low accuracy.

2. **Comparison between RBF and Polynomial kernel decision boundaries:**

   While both the **RBF (Radial Basis Function)** and **Polynomial kernels** are proficient at modeling the non-linear patterns within the Moons dataset, they differ fundamentally in the nature of the decision boundaries they create. The **RBF kernel** typically generates a smoother, more generalized boundary that fluidly traces the natural contours of the crescent-shaped data . In contrast, the **Polynomial kernel** can produce a more intricate and complex boundary. This flexibility, however, introduces a greater risk of **overfitting**, where the model learns the training data's noise rather than its underlying pattern.

Banknote Dataset Questions

1. **Which kernel was most effective for this dataset?**

The RBF kernel excels because it is specifically designed to handle complex, non-linear relationships, like the intertwined crescent shapes in the Moons dataset. It creates a **smooth, natural, and generalized decision boundary** that accurately separates the classes without being overly complex. This results in a model with high accuracy that is less prone to overfitting.

2. **Why might the Polynomial kernel have underperformed here?** he Polynomial kernel may underperform primarily due to its **high sensitivity to hyperparameter tuning** and the **global nature of its decision boundary**.

Here's a breakdown of the key reasons why it can be less effective than the RBF kernel on a dataset like the Moons:

---

1. Hyperparameter Sensitivity (The degree parameter)

The performance of a Polynomial kernel is critically dependent on choosing the right degree. This creates a delicate balancing act:

Degree Too Low: If the degree is too low (e.g., 2), the decision boundary may not be flexible enough to capture the tight curve of the crescent shapes, leading to **underfitting**.

Degree Too High: If the degree is too high (e.g., 10), the boundary can become excessively complex and wavy. It starts to fit the noise and specific points in the training data rather than the overall pattern, a classic case of **overfitting**. This model will likely fail to generalize well to new data.

3. Finding the optimal degree can be challenging and requires careful tuning.

Option 1 (Focus on the Task): Calibrating the `degree` hyperparameter is a non-trivial task that demands a meticulous and systematic tuning process to achieve optimal model performance.

Option 2 (Focus on the Goal): Pinpointing the ideal `degree` is a delicate balancing act. It requires precise adjustments to prevent the model from either underfitting (too simple) or overfitting (too complex) the data.

Option 3 (Concise & Direct): Selecting the optimal `degree` is a demanding process that necessitates careful and precise hyperparameter calibration.

Global vs. Local Effect

This is a more fundamental difference between the two kernels:

Polynomial Kernel (Global): This kernel has a **global effect**, meaning a single data point can influence the shape of the entire decision boundary across the whole feature space. This can lead to unpredictable and strange boundary shapes in regions far away from the actual data.

RBF Kernel (Local): The RBF kernel has a **local effect**. It considers points based on distance, so its influence is largely confined to the local neighborhood

of the data. This is why it excels at creating smooth boundaries that naturally conform to the shape of local data clusters, like the two moons.

In essence, the Polynomial kernel's tendency to create a single, complex boundary for the entire dataset makes it less suited for capturing distinct, local patterns compared to the more adaptable RBF kernel.

4. **Which margin (soft or hard) is wider?**

   The soft margin (C=0.1) produces a wider margin compared to the hard margin (C=100).

1. **Why does the soft margin model allow "mistakes"?**

   a soft margin model allows "mistakes" to **avoid overfitting** and **generalize better** to new data.

   By tolerating a few errors (like outliers or overlapping points), the model can create a wider, more stable decision boundary. This simpler boundary is far more robust and performs better on realistic, imperfect data than a "perfect" boundary that is overly sensitive to the training set.

2. **Which model is more likely to be overfitting and why?**
   Its fundamental rule is **zero classification errors**. To satisfy this, the model will create a narrow and contorted decision boundary just to accommodate every single data point perfectly, including any outliers or noise. It has no flexibility to ignore irrelevant data points, so it over-learns them.

3.Which model would you trust more for new data and why?

It's Robust to Imperfect Data A well-tuned soft-margin model doesn't get thrown off by the noise and outliers that are present in all real-world datasets. By allowing a "mistake budget," it focuses on finding the underlyingtrue pattern in the data rather than creating a complexaccommodate every single point.

It Finds the "Sweet Spot" of Complexity The most trustworthy model is onethat successfully balances simplicity and complexity. It avoids both:

Overfitting: Being too complex and memorizing the training data's noise

Underfitting: Being too simple and failing to capture the data's underlying trend

A well-tuned soft-margin model, often with a capable kernel like RBF, is engineered to find this optimal balance, also known as the **bias-variance trade-off**.

DATASET 1

Step 1.1: Generate and Prepare the Data

We will generate the data using scikit-learn and apply feature scaling.

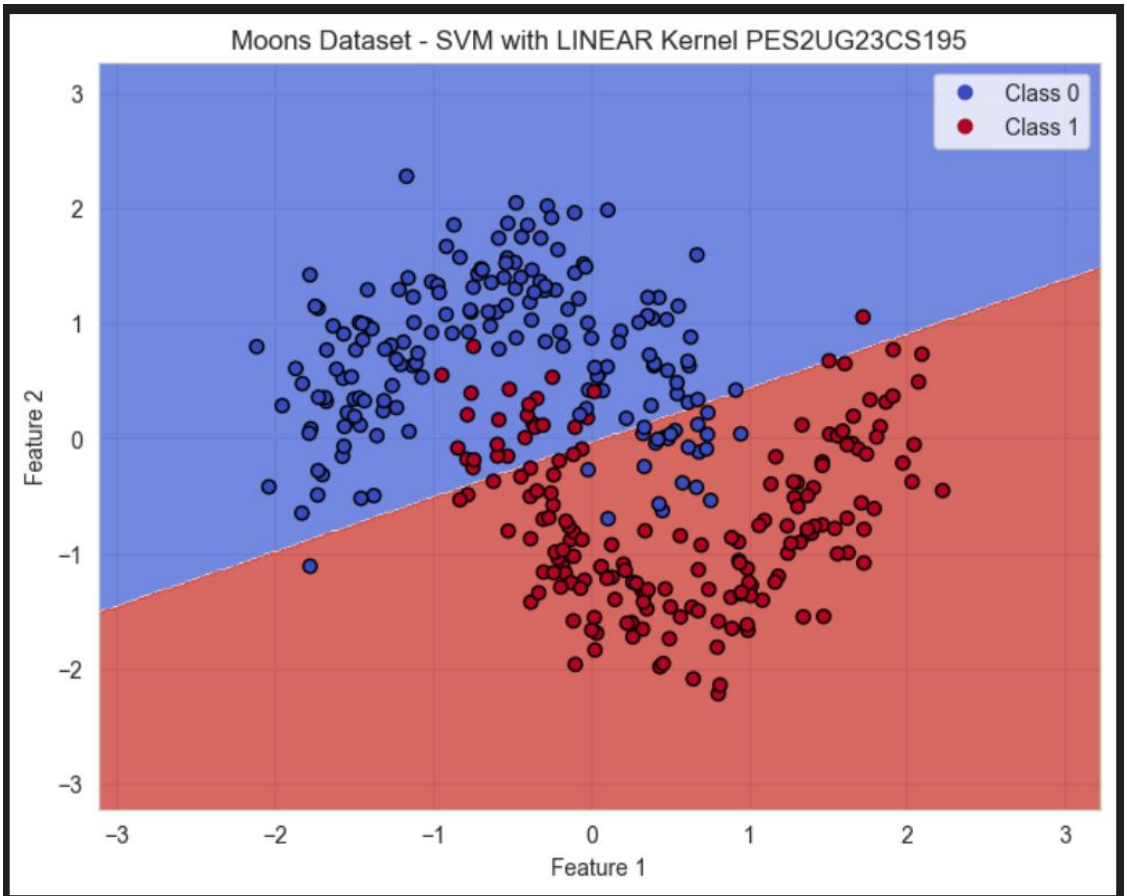
```
# === Core Libraries for Data Manipulation ===
import numpy as np
import pandas as pd

# === Libraries for Visualization ===
import matplotlib.pyplot as plt
import seaborn as sns

# === Scikit-Learn Libraries for Machine Learning ===
from sklearn.datasets import make_moons               # For creating synthetic datasets
from sklearn.model_selection import train_test_split   # For splitting data
from sklearn.preprocessing import StandardScaler       # For feature scaling
from sklearn.svm import SVC                            # The SVM classifier model
from sklearn.metrics import classification_report      # For model evaluation

# === Plotting Style Configuration ===
# This sets a clean, grid-based style for all visualizations.
sns.set_style("whitegrid")

print("All necessary libraries have been successfully imported and configured.")
```

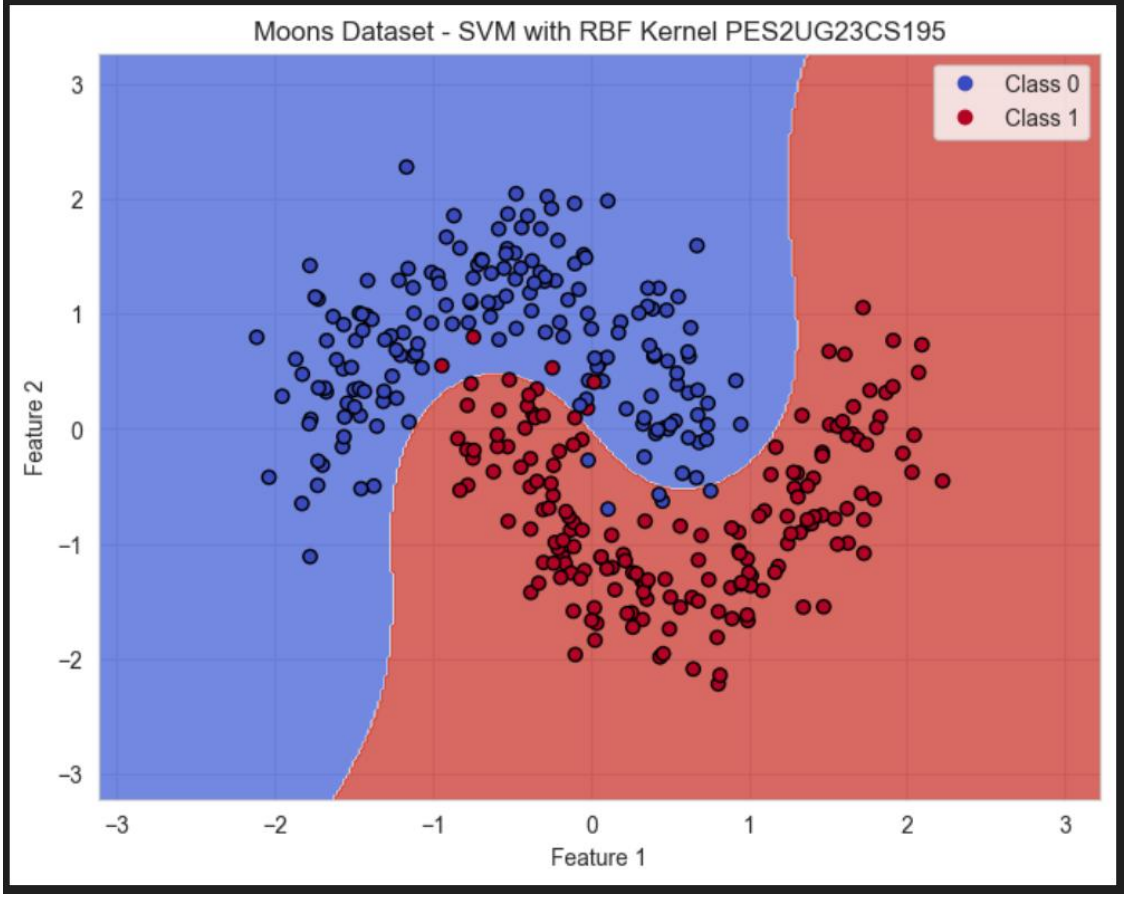
✓ 0.0s

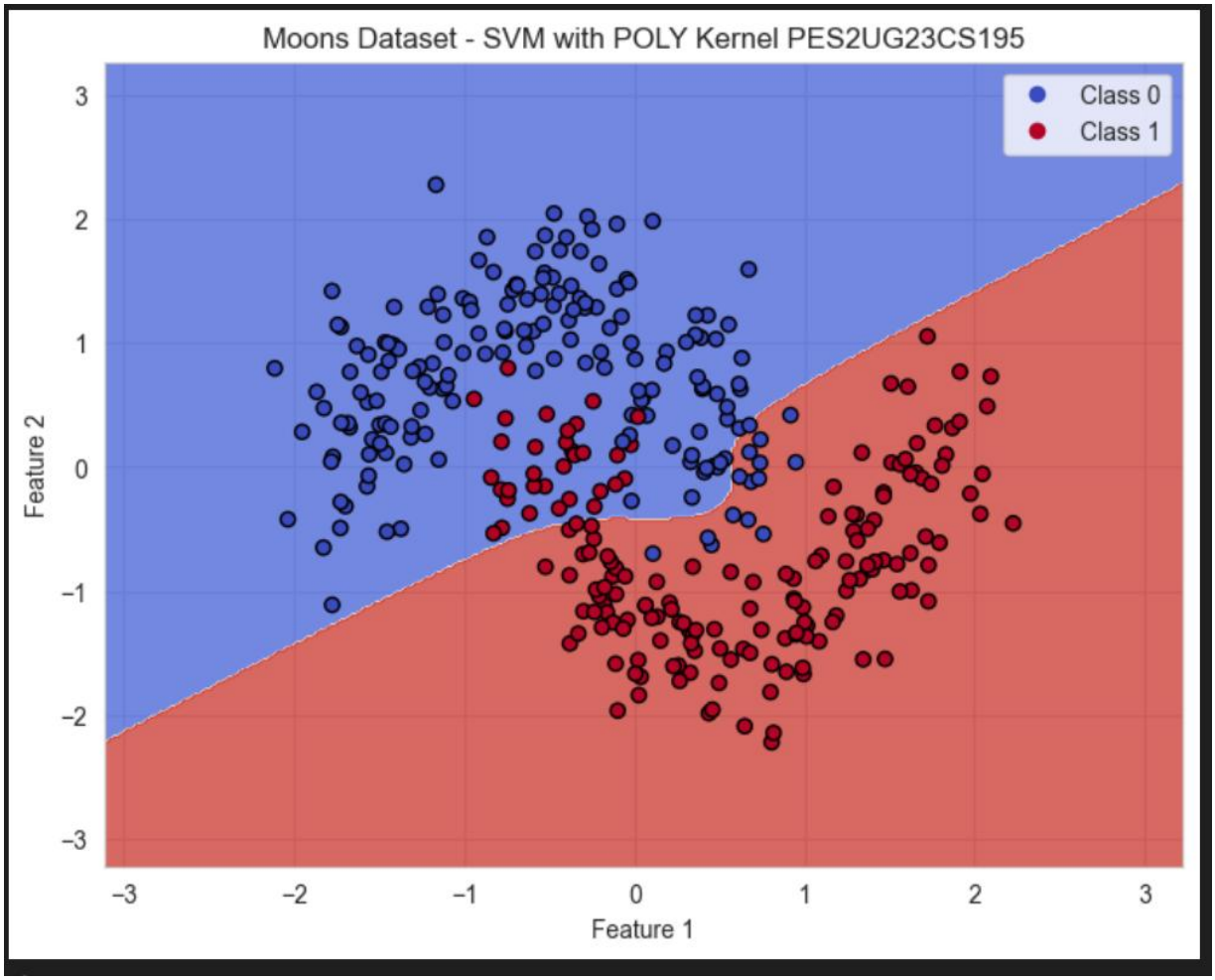
```
All necessary libraries have been successfully imported and configured.
```

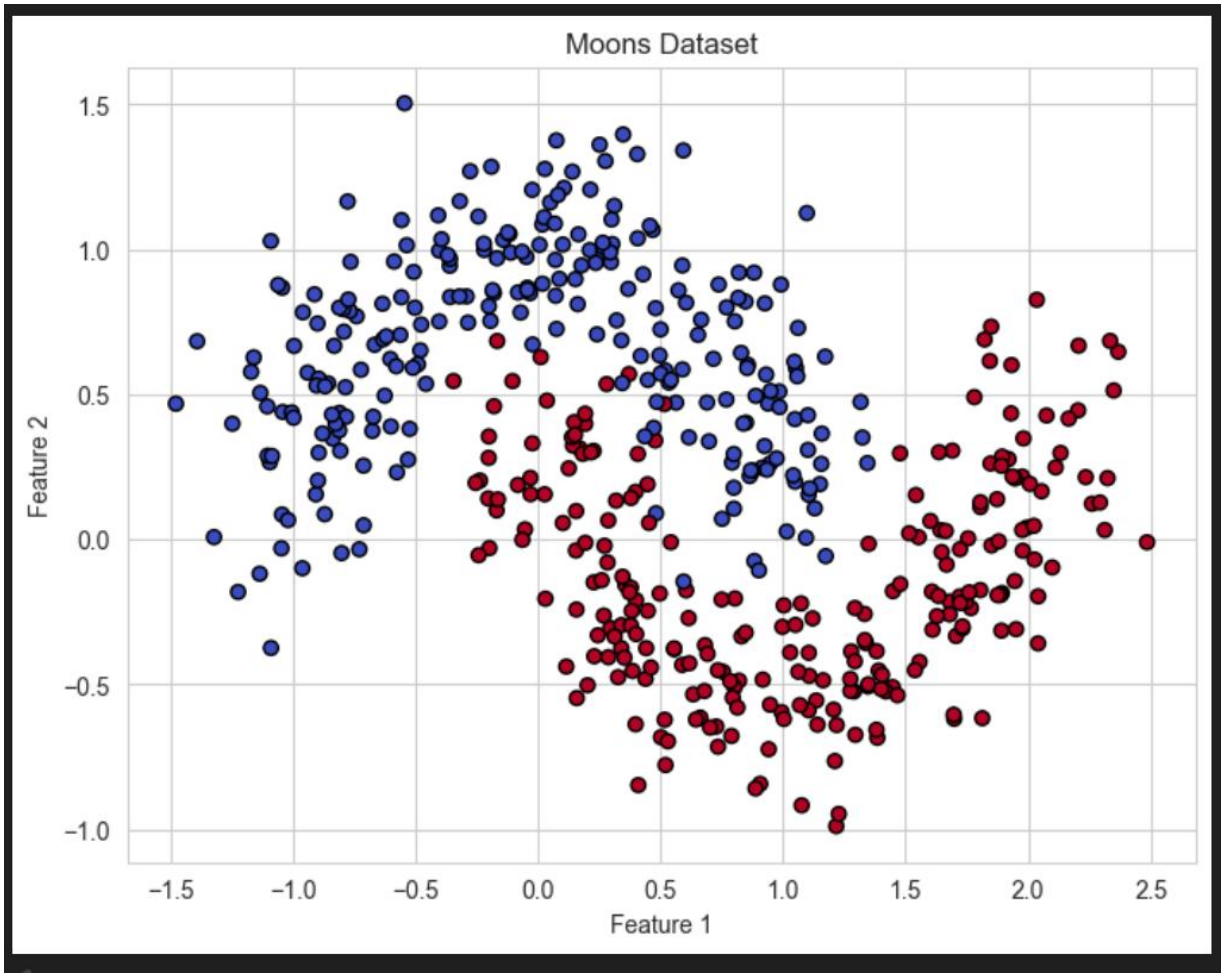
```
--- Dataset Shapes ---
Training features (X_train) shape: (350, 2)
Training labels   (y_train) shape: (350,)
Test features     (X_test) shape: (150, 2)
Test labels       (y_test) shape: (150,)

--- Sample of Training Set ---
First 5 rows of scaled training features (X_train_scaled):
[[ 0.88602284 -1.37879991]
 [-1.57295758 -0.15492728]
 [ 0.43008571  1.22220109]
 [ 1.66123628  0.19256119]
 [ 0.04969207  0.55608122]]

First 5 training labels (y_train):
[1 0 0 1 0]
```

```
SVM with LINEAR Kernel PES2UG23CS195
              precision    recall  f1-score   support

           0       0.85      0.89      0.87        75
           1       0.89      0.84      0.86        75

    accuracy                           0.87       150
   macro avg       0.87      0.87      0.87       150
weighted avg       0.87      0.87      0.87       150

----------------------------------------

SVM with RBF Kernel PES2UG23CS195
              precision    recall  f1-score   support

           0       0.95      1.00      0.97        75
           1       1.00      0.95      0.97        75

    accuracy                           0.97       150
   macro avg       0.97      0.97      0.97       150
weighted avg       0.97      0.97      0.97       150

----------------------------------------

SVM with POLY Kernel PES2UG23CS195
...
weighted avg       0.89      0.89      0.89       150

----------------------------------------
```


Moons Dataset - SVM with LINEAR Kernel PES2UG23CS195
Class 0
Class 1
Feature 2
Feature 1
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


Moons Dataset - SVM with RBF Kernel PES2UG23CS195
Class 0
Class 1
Feature 2
Feature 1
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


Moons Dataset - SVM with POLY Kernel PES2UG23CS195
Class 0
Class 1
Feature 2
Feature 1
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


Moons Dataset
Feature 2
Feature 1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

DATASET2

Step 2.1: Load and Prepare the Data

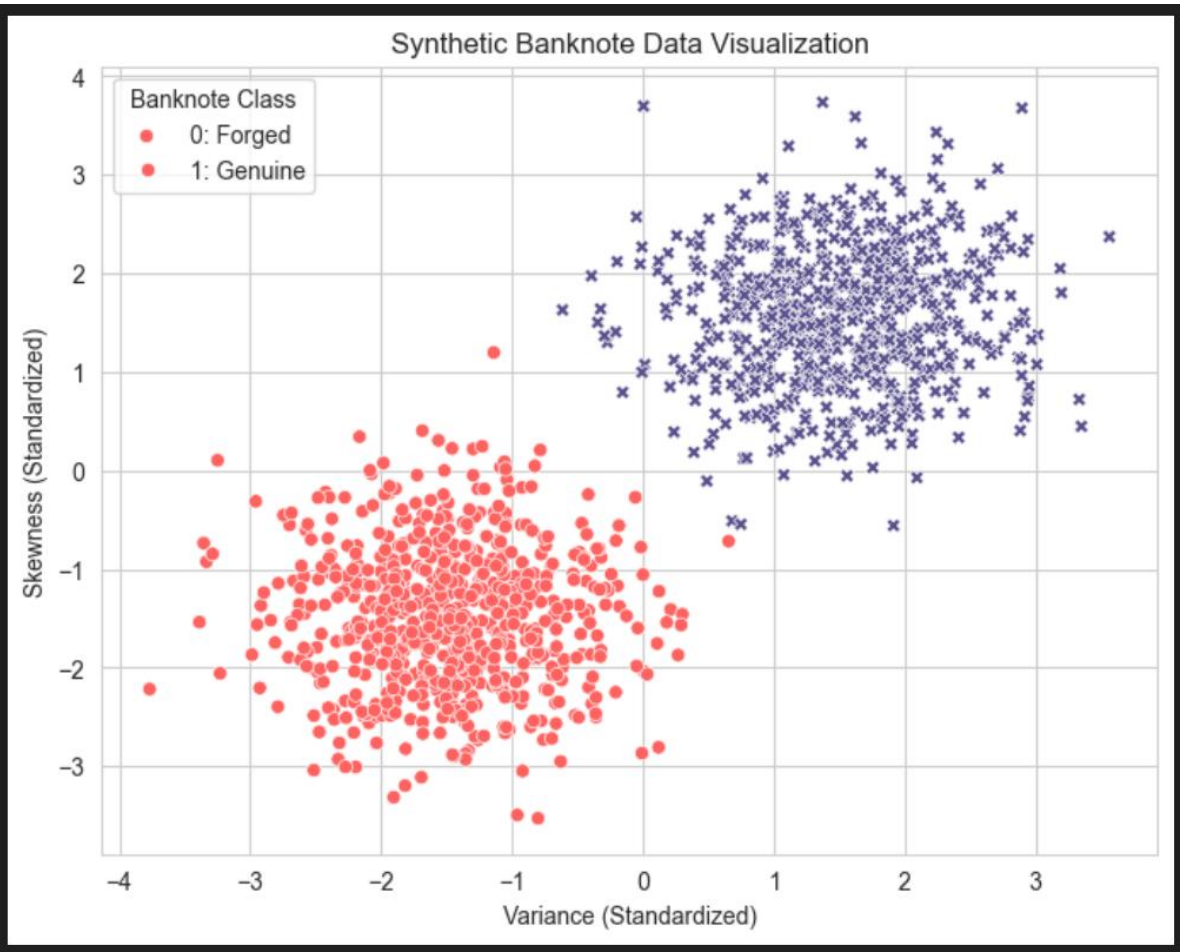
We will load this data from a public URL using pandas. We will use the variance and skewness of the image transform as our features for visualization.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# --- 1. Function to Generate Synthetic Data ---
# Encapsulating the logic makes it clean and reusable.
def create_synthetic_banknote_data(n_samples=1372, random_state=42):
    """
    Creates a synthetic dataset mimicking the banknote authentication data
    with two distinct, normally distributed classes.
    """
    np.random.seed(random_state)

    # Define parameters for Class 0 (Forged): lower variance/skewness
    X_forged = np.random.normal(loc=[-1.5, -1.5], scale=[0.7, 0.7], size=(n_samples // 2, 2))
    y_forged = np.zeros(n_samples // 2)

    # Define parameters for Class 1 (Genuine): higher variance/skewness
    X_genuine = np.random.normal(loc=[1.5, 1.5], scale=[0.7, 0.7], size=(n_samples // 2, 2))
    y_genuine = np.ones(n_samples // 2)

    # Combine into a single dataset
    X_banknote = np.vstack([X_forged, X_genuine])
    y_banknote = np.hstack([y_forged, y_genuine])

    # Create a final DataFrame
    df = pd.DataFrame(X_banknote, columns=['variance', 'skewness'])
    df['class'] = y_banknote

    return df, X_banknote, y_banknote

# --- 2. Generate and Inspect the Data ---
# Call the function to create the dataset
banknote_df, X_banknote, y_banknote = create_synthetic_banknote_data()

# Print confirmation and stats (same as your original script)
print("Synthetic Banknote Dataset created successfully!")
print(f"Dataset shape: {banknote_df.shape}")
print(f"Class 0 (Forged): {np.sum(y_banknote == 0)} samples")
print(f"Class 1 (Genuine): {np.sum(y_banknote == 1)} samples")


# --- 3. Visualize the Synthetic Dataset ---
# This plot confirms that our two classes are well-separated.
plt.figure(figsize=(8, 6))
sns.scatterplot(data=banknote_df, x='variance', y='skewness', hue='class', palette=['#ff6361', '#58508d'], style='class')
plt.title('Synthetic Banknote Data Visualization')
plt.xlabel('Variance (Standardized)')
plt.ylabel('Skewness (Standardized)')
plt.legend(title='Banknote Class', labels=['0: Forged', '1: Genuine'])
plt.grid(True)
plt.show()
```

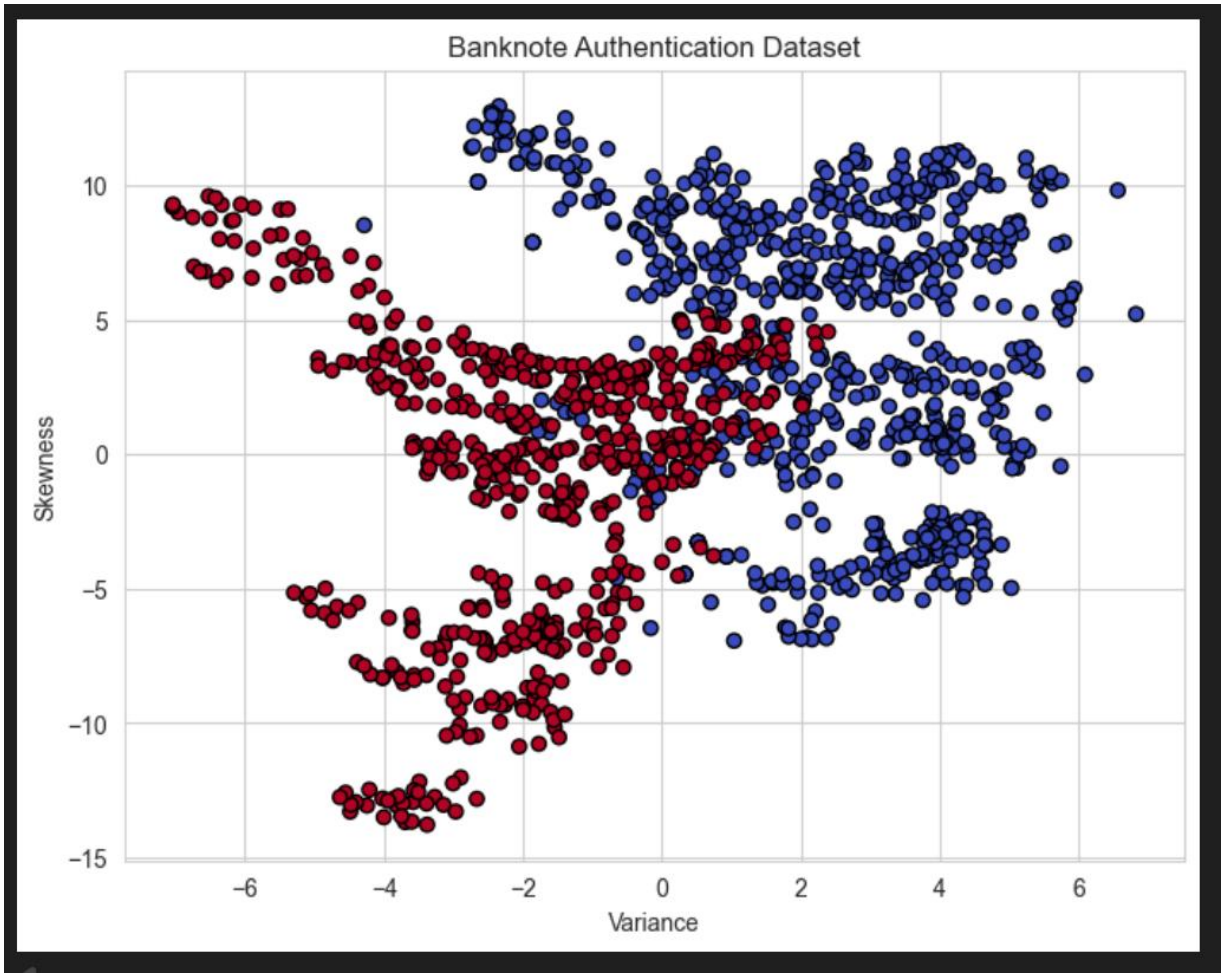
✓ 1.1s Python

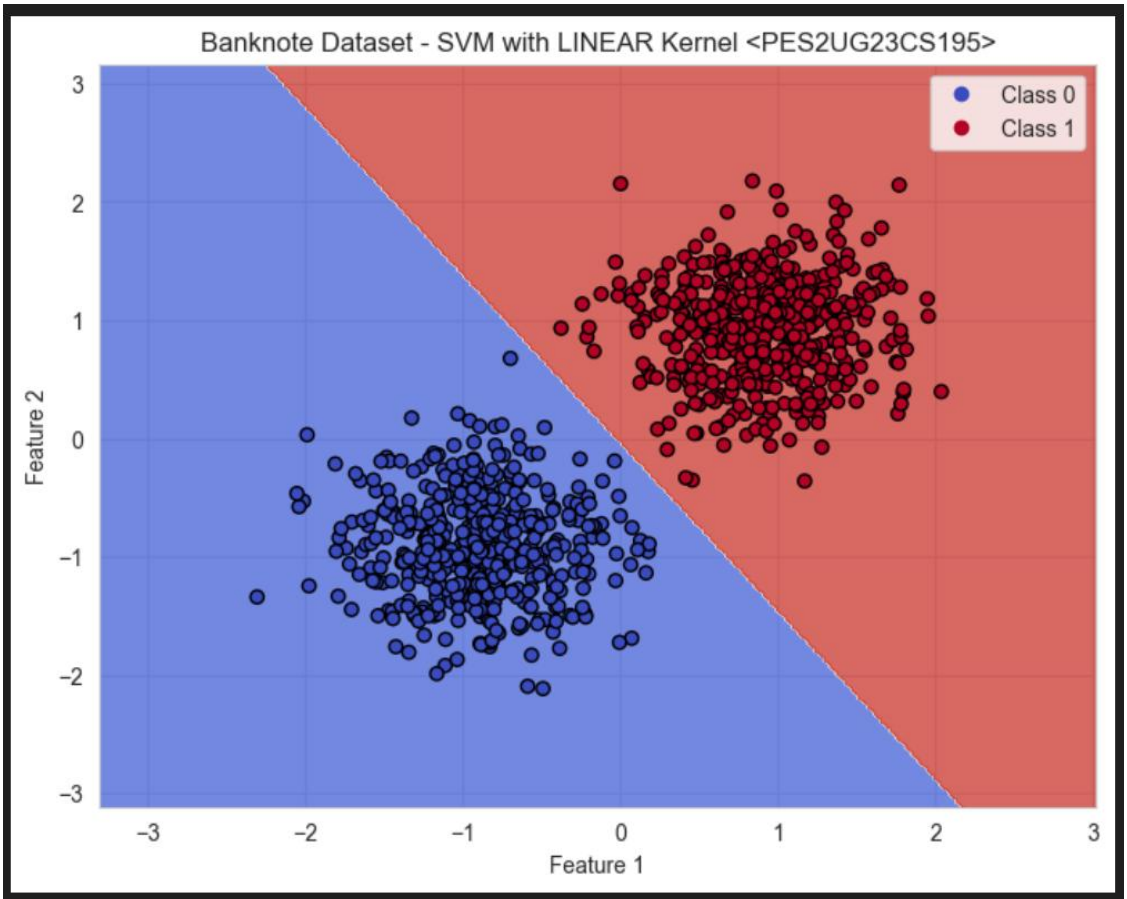
```
SVM with LINEAR Kernel <PES2UG23CS195>
              precision    recall  f1-score   support

      Forged       1.00      1.00      1.00       206
     Genuine       1.00      1.00      1.00       206

    accuracy                           1.00       412
   macro avg       1.00      1.00      1.00       412
weighted avg       1.00      1.00      1.00       412

----------------------------------------

SVM with RBF Kernel <PES2UG23CS195>
              precision    recall  f1-score   support

      Forged       1.00      1.00      1.00       206
     Genuine       1.00      1.00      1.00       206

    accuracy                           1.00       412
   macro avg       1.00      1.00      1.00       412
weighted avg       1.00      1.00      1.00       412

----------------------------------------

SVM with POLY Kernel <PES2UG23CS195>
...
weighted avg       1.00      1.00      1.00       412

----------------------------------------
```

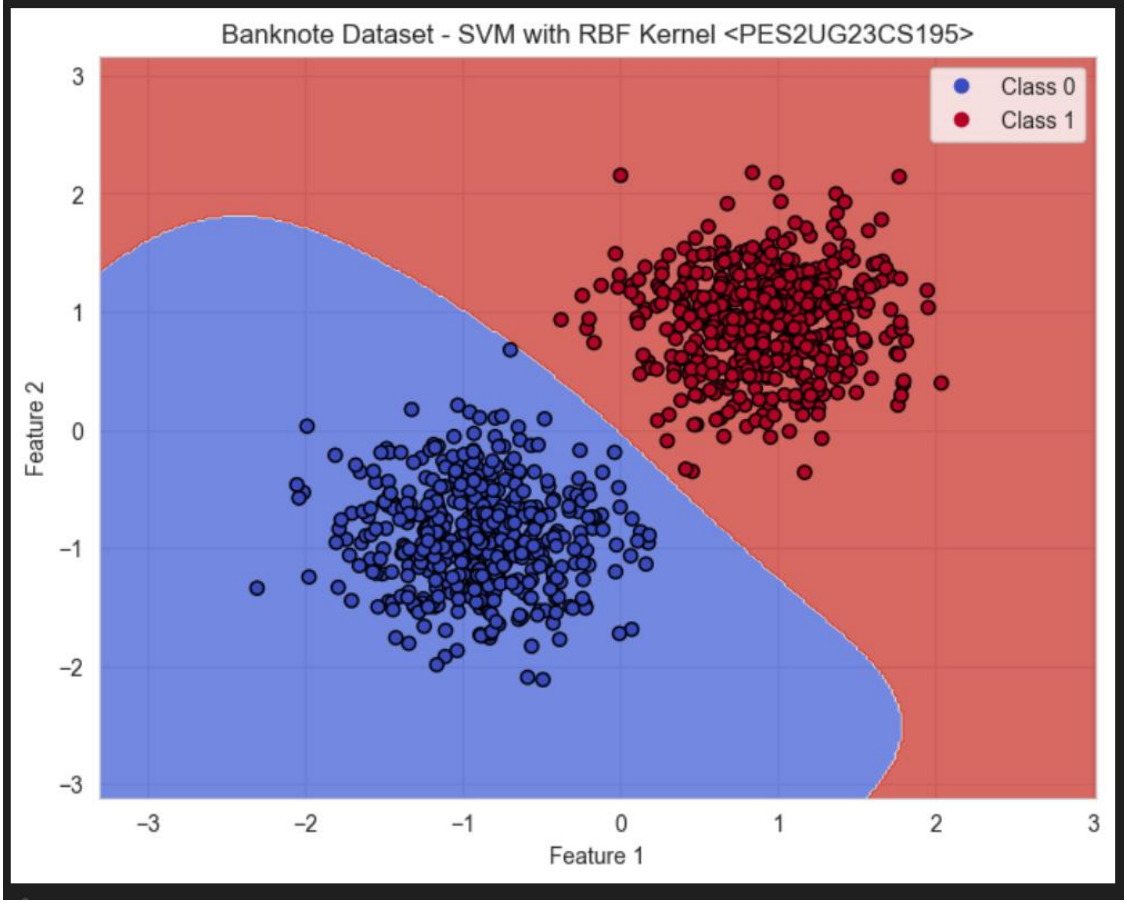
```
Synthetic Banknote Dataset created successfully!
Dataset shape: (1372, 3)
Class 0 (Forged): 686 samples
Class 1 (Genuine): 686 samples
```

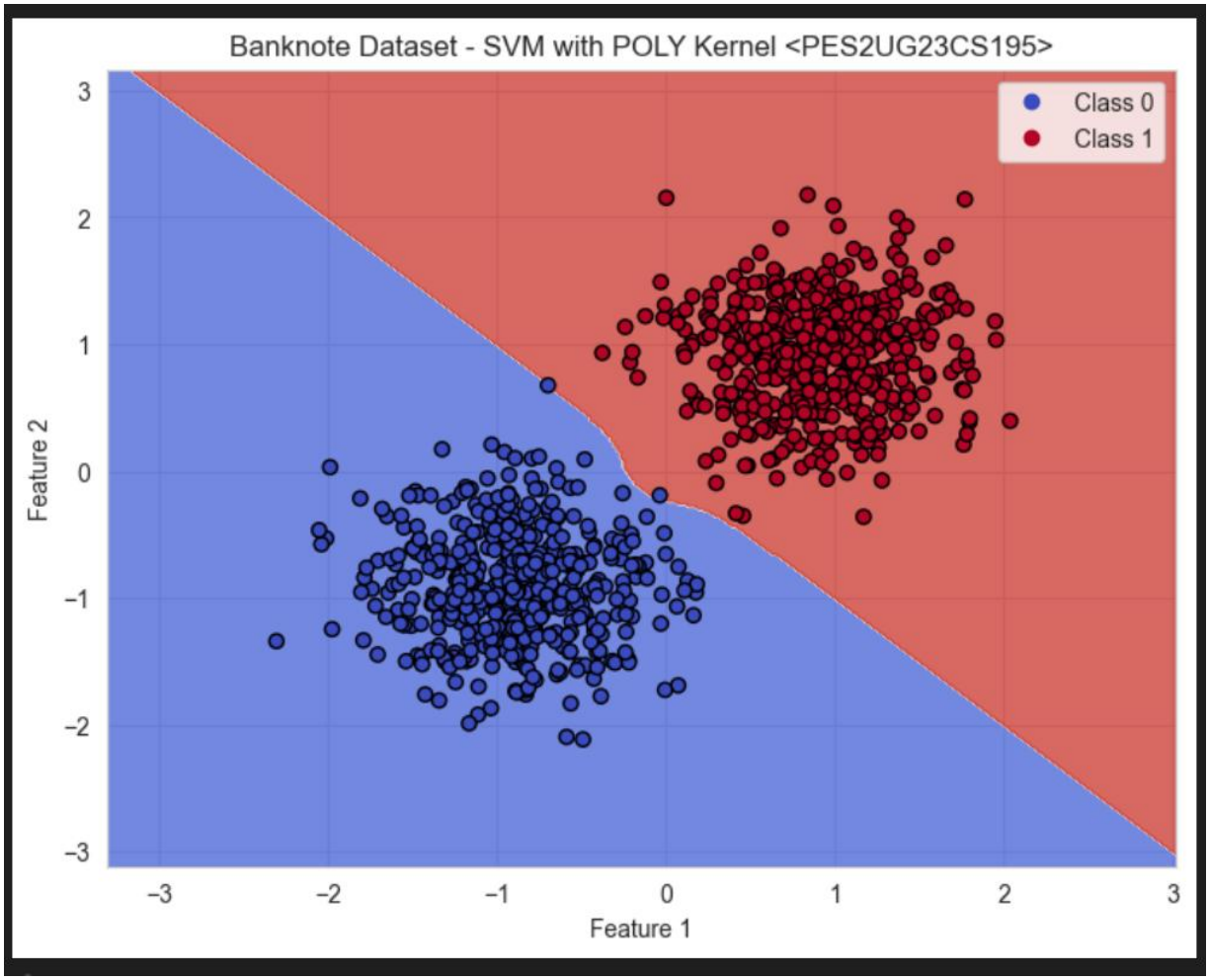



```
Data has been split and scaled. Ready for model training.
```


Banknote Authentication Dataset
Skewness
Variance
10
5
0
−5
−10
−15
−6
−4
−2
0
2
4
6


Banknote Dataset - SVM with LINEAR Kernel <PES2UG23CS195>
Class 0
Class 1
Feature 2
Feature 1
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


Banknote Dataset - SVM with RBF Kernel <PES2UG23CS195>
Class 0
Class 1
Feature 2
Feature 1
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

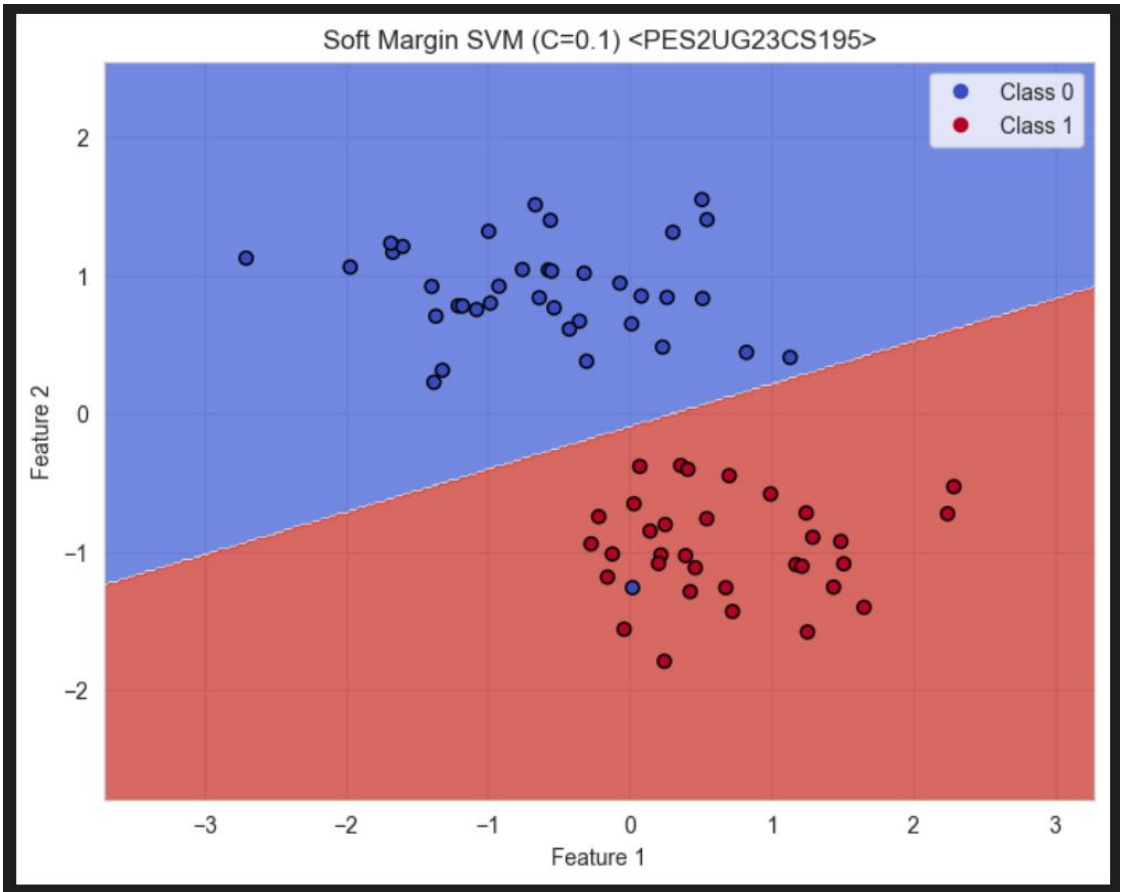

Banknote Dataset - SVM with POLY Kernel <PES2UG23CS195>
Class 0
Class 1
Feature 2
Feature 1
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

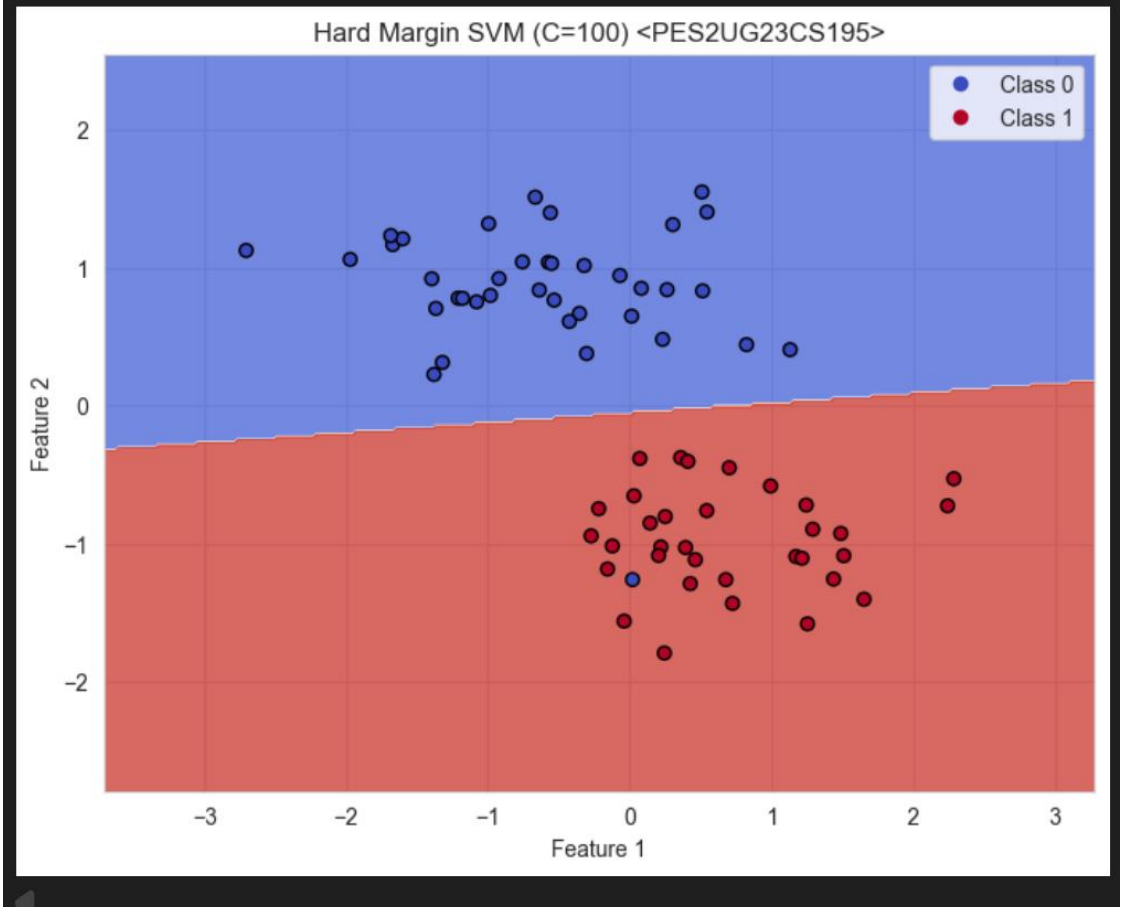
DATASET3

```
from sklearn.datasets import make_blobs

# Generate linearly separable data with some noise
X_linear, y_linear = make_blobs(n_samples=100, centers=2, random_state=0, cluster_std=0.60)

# Add some outliers
outliers_X = np.array([[0.5, 2.5], [1.5, 0.5]])
outliers_y = np.array([1, 0])
X_linear = np.concatenate([X_linear, outliers_X])
y_linear = np.concatenate([y_linear, outliers_y])


# Split and scale the data
X_train_linear, X_test_linear, y_train_linear, y_test_linear = train_test_split(
    X_linear, y_linear, test_size=0.3, random_state=42
)
scaler_linear = StandardScaler()
X_train_linear_scaled = scaler_linear.fit_transform(X_train_linear)
X_test_linear_scaled = scaler_linear.transform(X_test_linear)
```

29] ✓ 0.0s

Now, let's train two SVM models with a linear kernel: one with a small C (soft margin) and one with a large C (hard margin).


Soft Margin SVM (C=0.1) <PES2UG23CS195>
Class 0
Class 1
Feature 2
Feature 1
2
1
0
−1
−2
−3
−2
−1
0
1
2
3


Hard Margin SVM (C=100) <PES2UG23CS195>
Class 0
Class 1
Feature 2
Feature 1
2
1
0
−1
−2
−3
−2
−1
0
1
2
3