1) Apply SVM on IRIS data set from sklearn for classification:

In [9]:
```
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, classification_report
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

In [8]:
```
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target'] = iris.target
df.head()
```

Out[8]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
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 [10]:
```
X=iris.data
y=iris.target
```

In [11]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
```

In [12]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [13]:
```
svm = SVC(kernel='linear', random_state=42)
svm.fit(X_train, y_train)
```

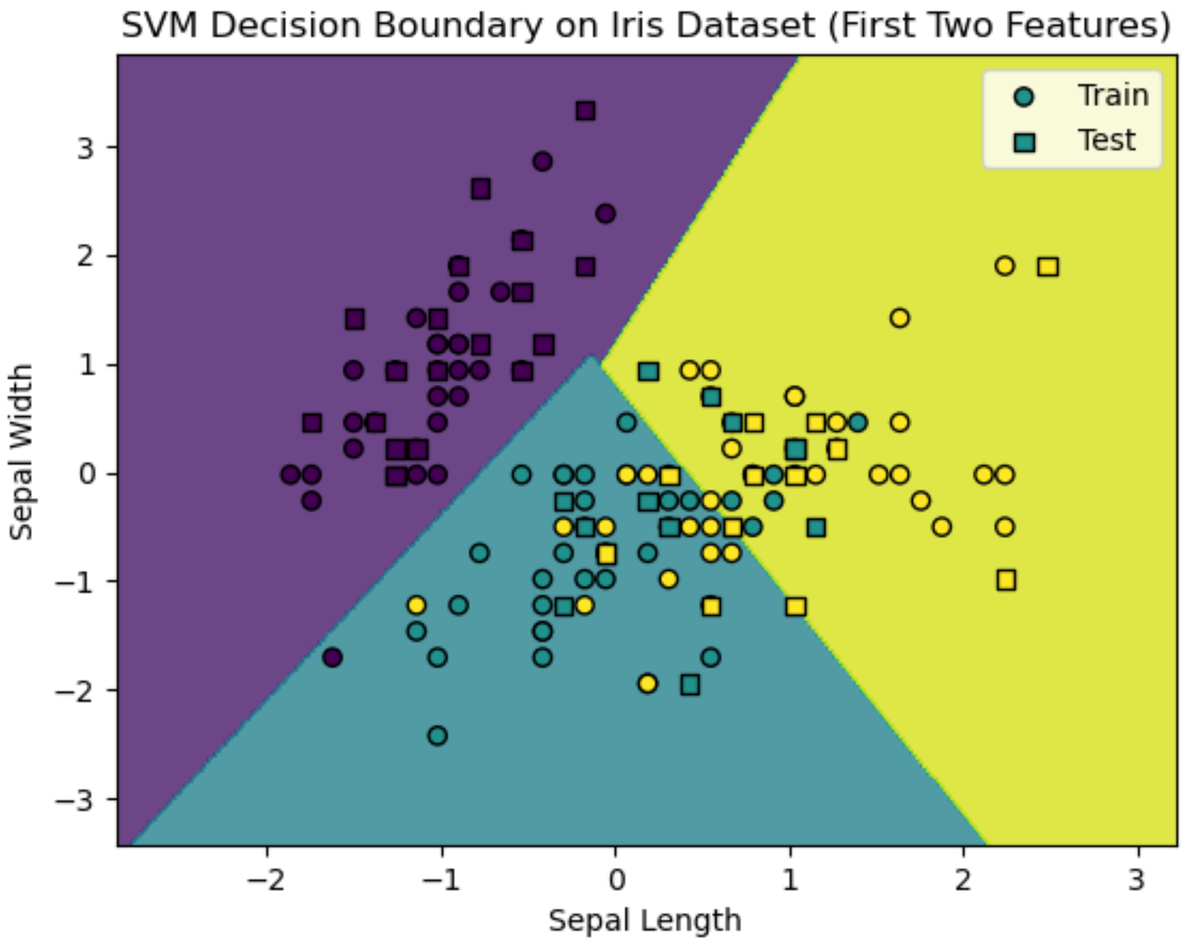
Out[13]:
```
▾ SVC
SVC(kernel='linear', random_state=42)
```

In [14]:
```
y_pred=svm.predict(X_test)
```

In [15]:
```
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(report)
```

```
Accuracy: 0.98
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.92      0.96        13
           2       0.93      1.00      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45
```

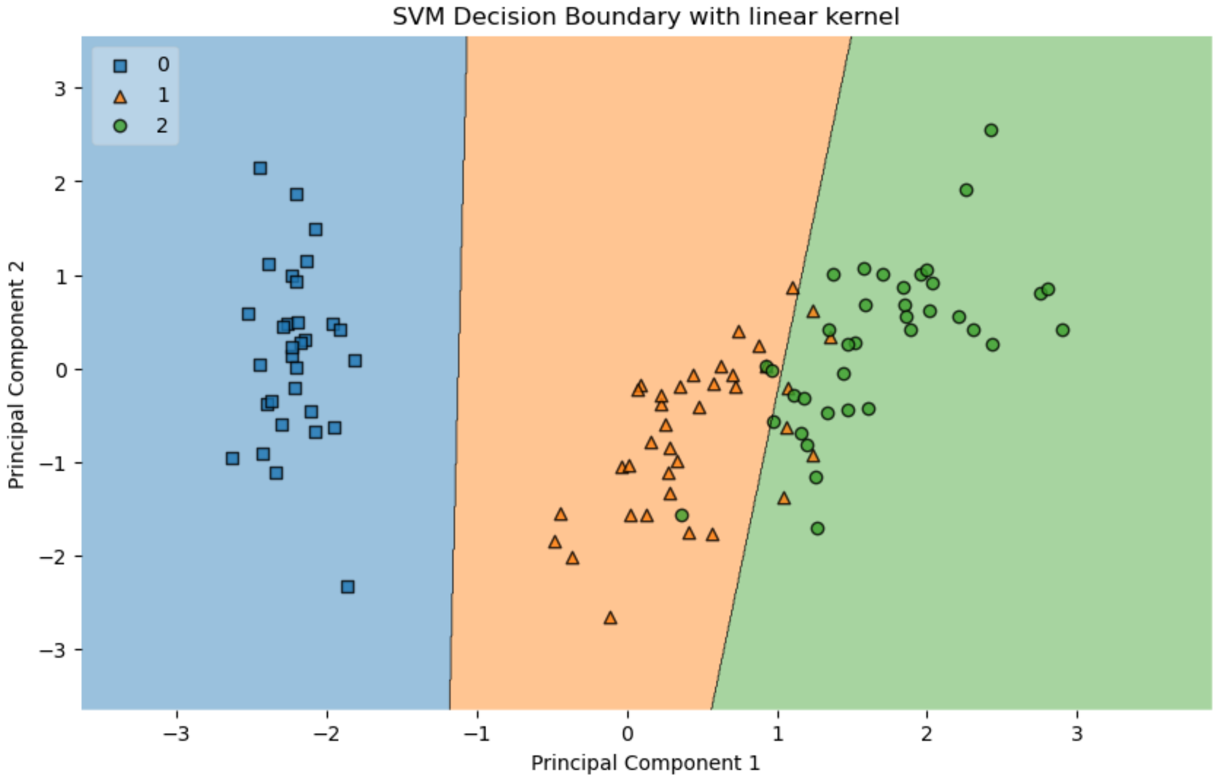
In [18]:
```
X_train_2d = X_train[:, :2]
X_test_2d = X_test[:, :2]

svm_2d = SVC(kernel='linear', random_state=42)
svm_2d.fit(X_train_2d, y_train)

x_min, x_max = X_train_2d[:, 0].min() - 1, X_train_2d[:, 0].max() + 1
y_min, y_max = X_train_2d[:, 1].min() - 1, X_train_2d[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_ma

Z = svm_2d.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundary and the scatter plot
plt.contourf(xx, yy, Z, alpha=0.8)
plt.scatter(X_train_2d[:, 0], X_train_2d[:, 1], c=y_train, marker='o', ed
plt.scatter(X_test_2d[:, 0], X_test_2d[:, 1], c=y_test, marker='s', edgec

plt.title('SVM Decision Boundary on Iris Dataset (First Two Features)')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.legend()
plt.show()
```

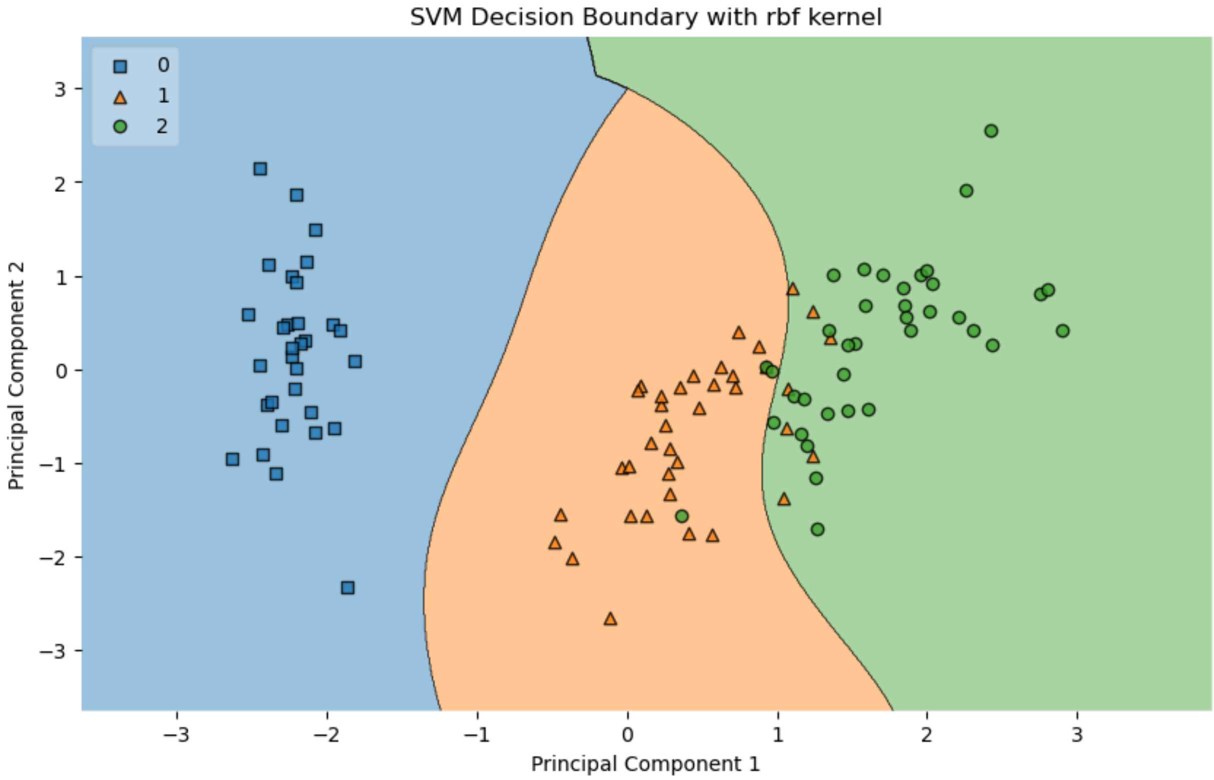


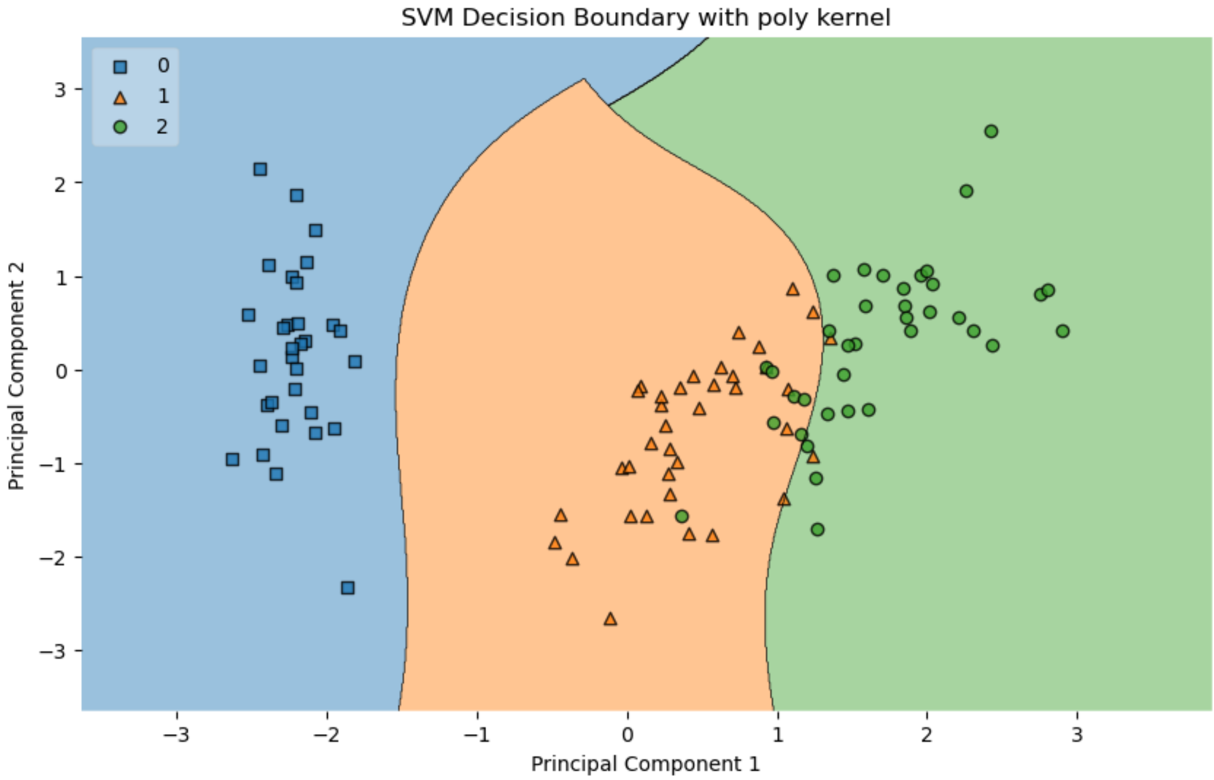
2) Visualize data classification with different kernels as explained in the Google Colab shared on the theory classroom:

In [24]:
```
from mlxtend.plotting import plot_decision_regions
from sklearn.decomposition import PCA

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0

def plot_svm_decision_boundary(kernel_name):
    svm = SVC(kernel=kernel_name, random_state=42)
    svm.fit(X_train, y_train)
    plt.figure(figsize=(10, 6))
    plot_decision_regions(X_train, y_train, clf=svm, legend=2)
    plt.title(f'SVM Decision Boundary with {kernel_name} kernel')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.show()
```

In [25]:
```
# Visualize decision boundaries for linear, rbf, and polynomial kernels
plot_svm_decision_boundary('linear')
plot_svm_decision_boundary('rbf')
plot_svm_decision_boundary('poly')
```







3) Explain the different types of kernels, and also explain the process to choose the appropriate kernel for a dataset classification:

SVM Kernels and Their Types:

1. Linear Kernel: The linear kernel is the simplest kernel and works by computing the dot product between two feature vectors. It's effective when the data is linearly separable, i.e., classes can be divided by a straight line (or hyperplane in higher dimensions).

2. Polynomial Kernel: This kernel computes the dot product and raises it to the power of n, introducing non-linearity. Suitable for data where the relationship between features and classes is polynomial.

3. Radial Basis Function (RBF) or Gaussian Kernel: The RBF kernel measures similarity between two points based on the distance. It transforms the data into an infinite-dimensional space, making it possible to classify very complex patterns. Works well when the boundary between classes is highly non-linear.

4. Sigmoid Kernel: Inspired by neural networks, the sigmoid kernel computes a non-linear transformation similar to how an activation function works in neural networks. Often used in situations similar to RBF, though less commonly because RBF generally performs better.

Choosing the Appropriate Kernel for Dataset Classification:

1. Understand Your Data: If your data is linearly separable, a linear kernel should suffice. You can check linear separability using dimensionality reduction techniques like PCA to visualize the data. If the classes are not linearly separable, try polynomial or RBF kernels. These can handle non-linear decision boundaries.

2. Commence with Simplicity: Initiate with a linear kernel to establish a foundational benchmark.

3. Account for Complexity: Assess computational demands, particularly with expansive datasets.