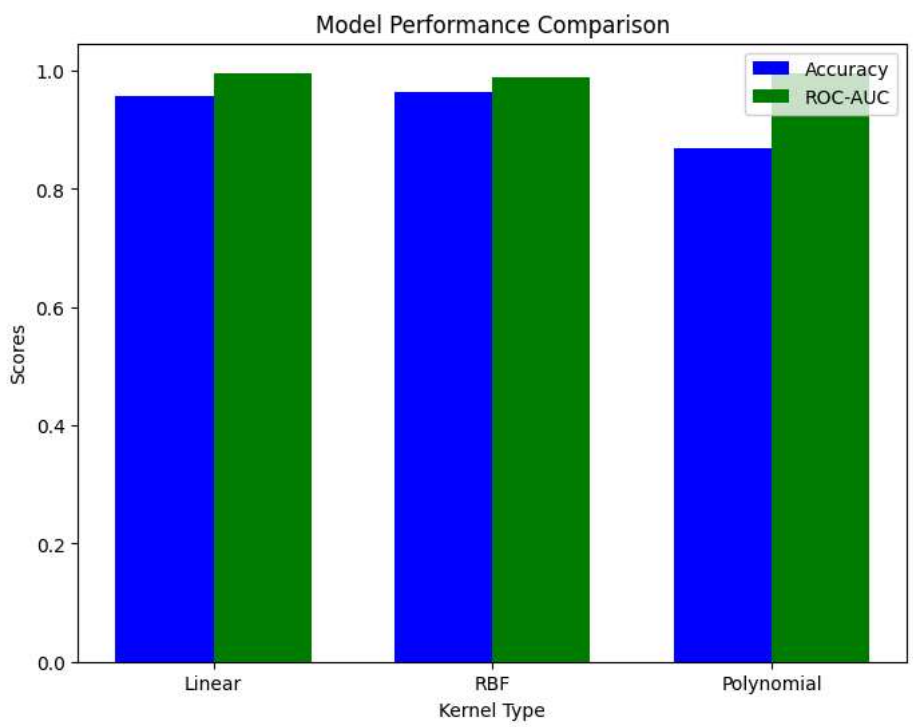
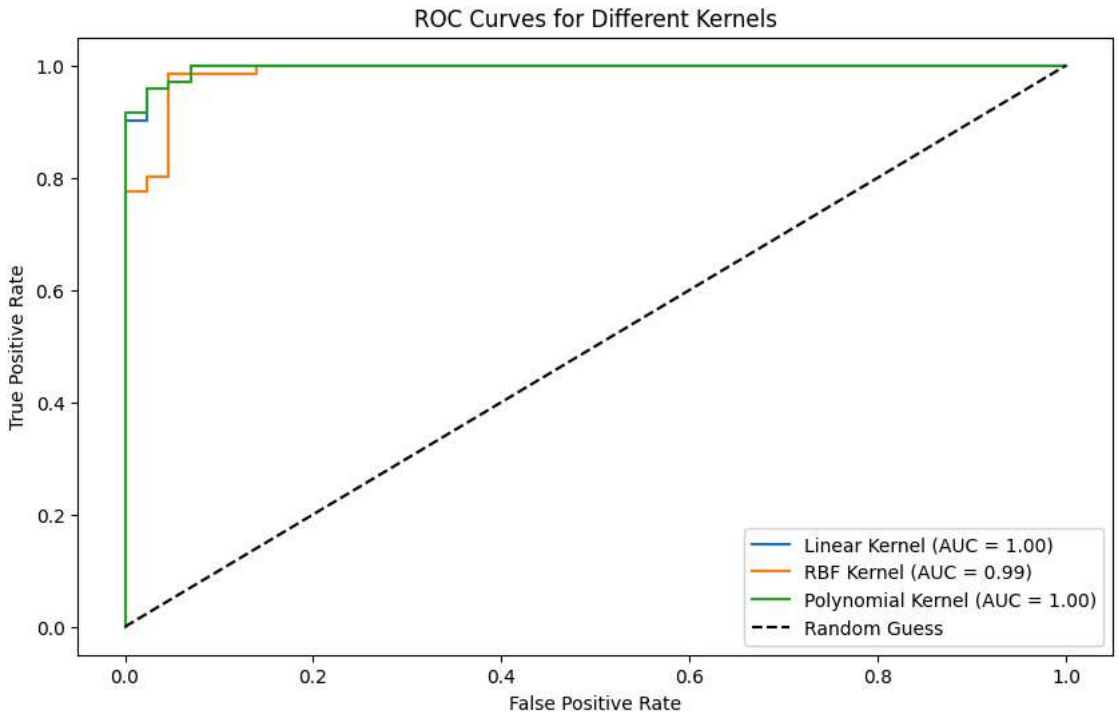
```
# Import Libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, roc_auc_score, roc_curve, accuracy_score
from sklearn.decomposition import PCA



# Load Dataset
data = load_breast_cancer()
X, y = data.data, data.target

# Normalize Features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)



# Load Dataset
data = load_breast_cancer()
X, y = data.data, data.target

# Normalize Features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)




models = {
      'Linear': SVC(kernel='linear', C=1, probability=True),
          'RBF': SVC(kernel='rbf', C=1, gamma=0.1, probability=True),
              'Polynomial': SVC(kernel='poly', degree=3, C=1, probability=True)
              }

              # Fit the models
for name, model in models.items(): # Removed extra indentation here
    model.fit(X_train, y_train)



# Evaluate Models
results = {}
for name, model in models.items():
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1]
    results[name] = {
                     'accuracy': accuracy_score(y_test, y_pred),
                     'roc_auc': roc_auc_score(y_test, y_proba),
                     'report': classification_report(y_test, y_pred, output_dict=True)
                                    }

                                    # Metrics Comparison (Bar Chart)
accuracy = [results[name]['accuracy'] for name in results]
roc_auc = [results[name]['roc_auc'] for name in results]

                                    # Bar plot for accuracy and ROC-AUC
labels = list(results.keys())
x = np.arange(len(labels))
width = 0.35

fig, ax = plt.subplots(figsize=(8, 6))
bars1 = ax.bar(x - width/2, accuracy, width, label='Accuracy', color='blue')
```

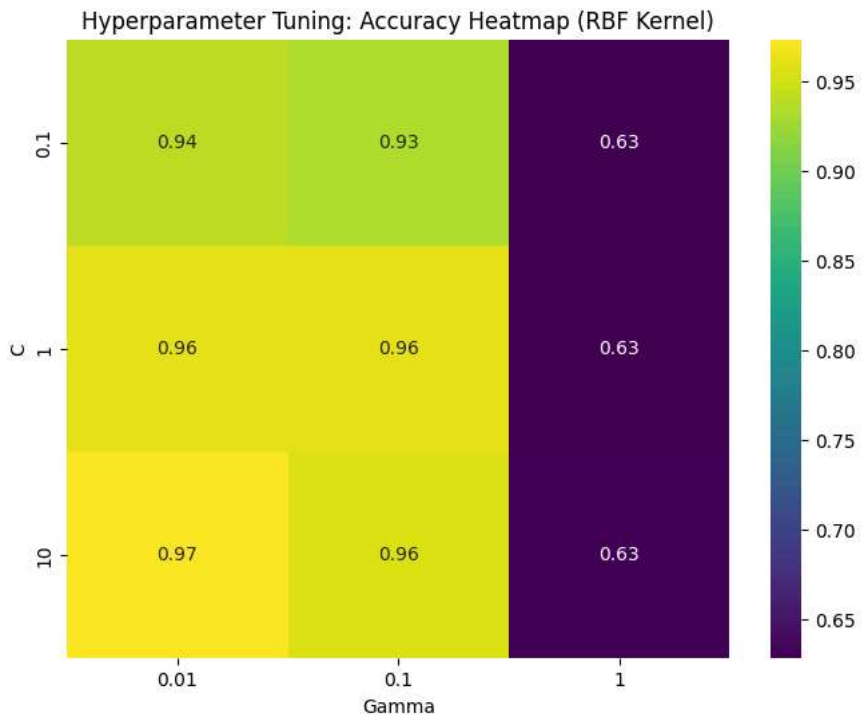
```
bars2 = ax.bar(x + width/2, roc_auc, width, label='ROC-AUC', color='green')

ax.set_xlabel('Kernel Type')
ax.set_ylabel('Scores')
ax.set_title('Model Performance Comparison')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()
plt.show()
```



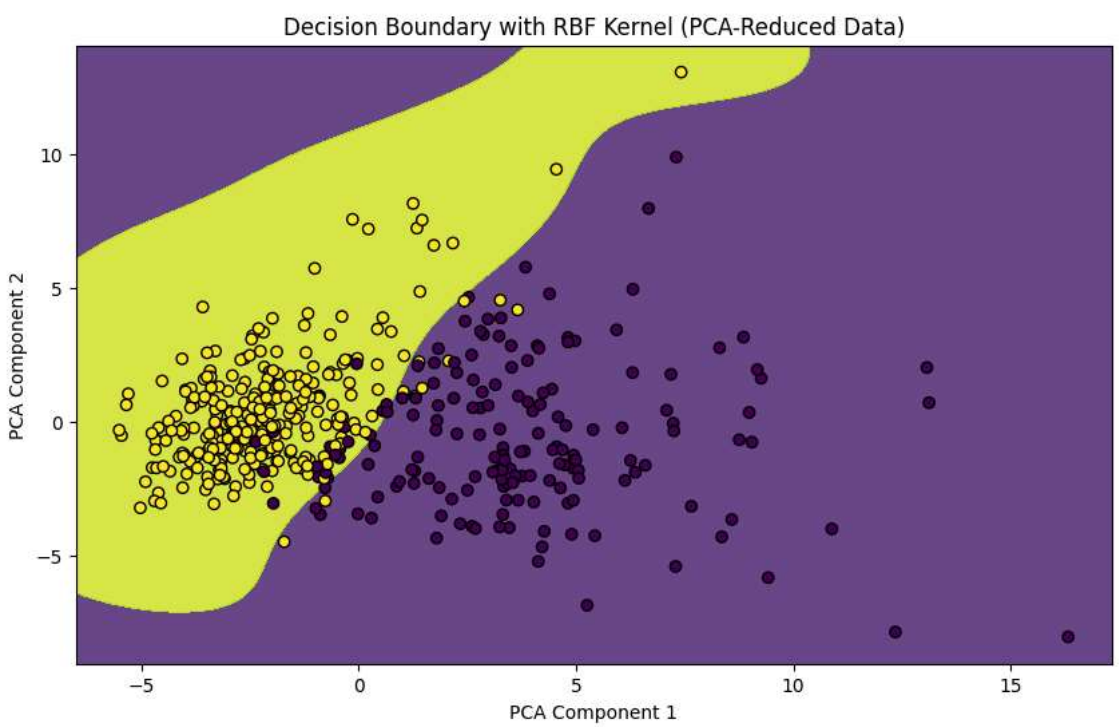

```
# ROC Curve for Different Kernels
plt.figure(figsize=(10, 6))
for name, model in models.items():
    y_proba = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_proba)
    auc = roc_auc_score(y_test, y_proba)
    plt.plot(fpr, tpr, label=f'{name} Kernel (AUC = {auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for Different Kernels')
plt.legend()
plt.show()
```




```
# Hyperparameter Tuning for RBF Kernel
param_grid = {'C': [0.1, 1, 10], 'gamma': [0.01, 0.1, 1]}
grid = GridSearchCV(SVC(kernel='rbf', probability=True), param_grid, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)

# Grid Search Results Visualization
results_grid = grid.cv_results_
scores = np.array(results_grid['mean_test_score']).reshape(len(param_grid['C']), len(param_grid['gamma']))

plt.figure(figsize=(8, 6))
sns.heatmap(scores, annot=True, xticklabels=param_grid['gamma'], yticklabels=param_grid['C'], cmap='viridis')
plt.xlabel('Gamma')
plt.ylabel('C')
plt.title('Hyperparameter Tuning: Accuracy Heatmap (RBF Kernel)')
plt.show()
```



```
# Decision Boundary Visualization with PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Retrain RBF model on PCA-reduced data
rbf_model_pca = SVC(kernel='rbf', C=1, gamma=0.1, probability=True)
rbf_model_pca.fit(X_train_pca, y_train)

# Mesh grid for decision boundary
x_min, x_max = X_train_pca[:, 0].min() - 1, X_train_pca[:, 0].max() + 1
y_min, y_max = X_train_pca[:, 1].min() - 1, X_train_pca[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))

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

# Plot decision boundary
plt.figure(figsize=(10, 6))
plt.contourf(xx, yy, Z, alpha=0.8, cmap='viridis')
plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train, edgecolors='k', cmap='viridis')
plt.title('Decision Boundary with RBF Kernel (PCA-Reduced Data)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.show()
```



Start coding or generate with AI.