Handwritten Digits Dataset

```
# Standard library
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Scikit-learn
from sklearn import datasets
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,
ConfusionMatrixDisplay, roc_curve, auc
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPClassifier

# Statistics
from scipy import stats

# Visualization
import seaborn as sns

# Dataset Fetching
from sklearn.datasets import load_wine

def plot_roc_auc(model, X_test, y_test, class_labels=None, title="ROC
Curve", figsize=(10, 7)):

    # Binarize labels for multiclass
    y_test_bin = label_binarize(y_test, classes=np.unique(y_test))
    n_classes = y_test_bin.shape[1]

    # Use predict_proba or decision_function
    if hasattr(model, "predict_proba"):
        y_score = model.predict_proba(X_test)
    elif hasattr(model, "decision_function"):
        y_score = model.decision_function(X_test)
    else:
        raise ValueError("Model must implement either predict_proba or
decision_function.")

    # Compute ROC and AUC per class
    fpr, tpr, roc_auc = dict(), dict(), dict()

    for i in range(n_classes):
```

```
        fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Macro-average AUC
    all_fpr = np.unique(np.concatenate([fpr[i] for i in
range(n_classes)]))
    mean_tpr = np.zeros_like(all_fpr)

    for i in range(n_classes):
        mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])

    mean_tpr /= n_classes
    fpr["macro"] = all_fpr
    tpr["macro"] = mean_tpr
    roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

    # Plot
    plt.figure(figsize=figsize)
    for i in range(n_classes):
        label = f"{class_labels[i] if class_labels else i} (AUC =
{roc_auc[i]:.2f})"
        plt.plot(fpr[i], tpr[i], lw=2, label=label)

    plt.plot([0, 1], [0, 1], 'k--', lw=1)
    plt.plot(fpr["macro"], tpr["macro"], linestyle='--', color='navy',
             label=f"Macro Avg (AUC = {roc_auc['macro']:.2f})", lw=2)

    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(title)
    plt.legend(loc="lower right", fontsize=9)
    plt.grid(True)
    plt.tight_layout()
    plt.show()

digits = datasets.load_digits()
digits
```

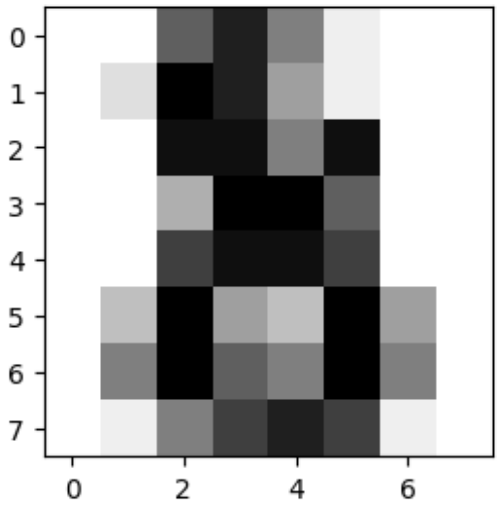
```
{'data': array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
        [ 0.,  0.,  0., ..., 10.,  0.,  0.],
        [ 0.,  0.,  0., ..., 16.,  9.,  0.],
        ...,
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
        [ 0.,  0.,  2., ..., 12.,  0.,  0.],
        [ 0.,  0., 10., ..., 12.,  1.,  0.]]),
 'target': array([0, 1, 2, ..., 8, 9, 8]),
 'frame': None,
 'feature_names': ['pixel_0_0',
```

```
  'pixel_0_1',
  'pixel_0_2',
  'pixel_0_3',
  'pixel_0_4',
  'pixel_0_5',
  'pixel_0_6',
  'pixel_0_7',
  'pixel_1_0',
  'pixel_1_1',
  'pixel_1_2',
  'pixel_1_3',
  'pixel_1_4',
  'pixel_1_5',
  'pixel_1_6',
  'pixel_1_7',
  'pixel_2_0',
  'pixel_2_1',
  'pixel_2_2',
  'pixel_2_3',
  'pixel_2_4',
  'pixel_2_5',
  'pixel_2_6',
  'pixel_2_7',
  'pixel_3_0',
  'pixel_3_1',
  'pixel_3_2',
  'pixel_3_3',
  'pixel_3_4',
  'pixel_3_5',
  'pixel_3_6',
  'pixel_3_7',
  'pixel_4_0',
  'pixel_4_1',
  'pixel_4_2',
  'pixel_4_3',
  'pixel_4_4',
  'pixel_4_5',
  'pixel_4_6',
  'pixel_4_7',
  'pixel_5_0',
  'pixel_5_1',
  'pixel_5_2',
  'pixel_5_3',
  'pixel_5_4',
  'pixel_5_5',
  'pixel_5_6',
  'pixel_5_7',
  'pixel_6_0',
  'pixel_6_1',
```

```
  'pixel_6_2',
  'pixel_6_3',
  'pixel_6_4',
  'pixel_6_5',
  'pixel_6_6',
  'pixel_6_7',
  'pixel_7_0',
  'pixel_7_1',
  'pixel_7_2',
  'pixel_7_3',
  'pixel_7_4',
  'pixel_7_5',
  'pixel_7_6',
  'pixel_7_7'],
 'target_names': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 'images': array([[[ 0.,  0.,  5., ...,  1.,  0.,  0.],
         [ 0.,  0., 13., ..., 15.,  5.,  0.],
         [ 0.,  3., 15., ..., 11.,  8.,  0.],
         ...,
         [ 0.,  4., 11., ..., 12.,  7.,  0.],
         [ 0.,  2., 14., ..., 12.,  0.,  0.],
         [ 0.,  0.,  6., ...,  0.,  0.,  0.]],

        [[ 0.,  0.,  0., ...,  5.,  0.,  0.],
         [ 0.,  0.,  0., ...,  9.,  0.,  0.],
         [ 0.,  0.,  3., ...,  6.,  0.,  0.],
         ...,
         [ 0.,  0.,  1., ...,  6.,  0.,  0.],
         [ 0.,  0.,  1., ...,  6.,  0.,  0.],
         [ 0.,  0.,  0., ..., 10.,  0.,  0.]],

        [[ 0.,  0.,  0., ..., 12.,  0.,  0.],
         [ 0.,  0.,  3., ..., 14.,  0.,  0.],
         [ 0.,  0.,  8., ..., 16.,  0.,  0.],
         ...,
         [ 0.,  9., 16., ...,  0.,  0.,  0.],
         [ 0.,  3., 13., ..., 11.,  5.,  0.],
         [ 0.,  0.,  0., ..., 16.,  9.,  0.]],

        ...,

        [[ 0.,  0.,  1., ...,  1.,  0.,  0.],
         [ 0.,  0., 13., ...,  2.,  1.,  0.],
         [ 0.,  0., 16., ..., 16.,  5.,  0.],
         ...,
         [ 0.,  0., 16., ..., 15.,  0.,  0.],
         [ 0.,  0., 15., ..., 16.,  0.,  0.],
         [ 0.,  0.,  2., ...,  6.,  0.,  0.]],
```

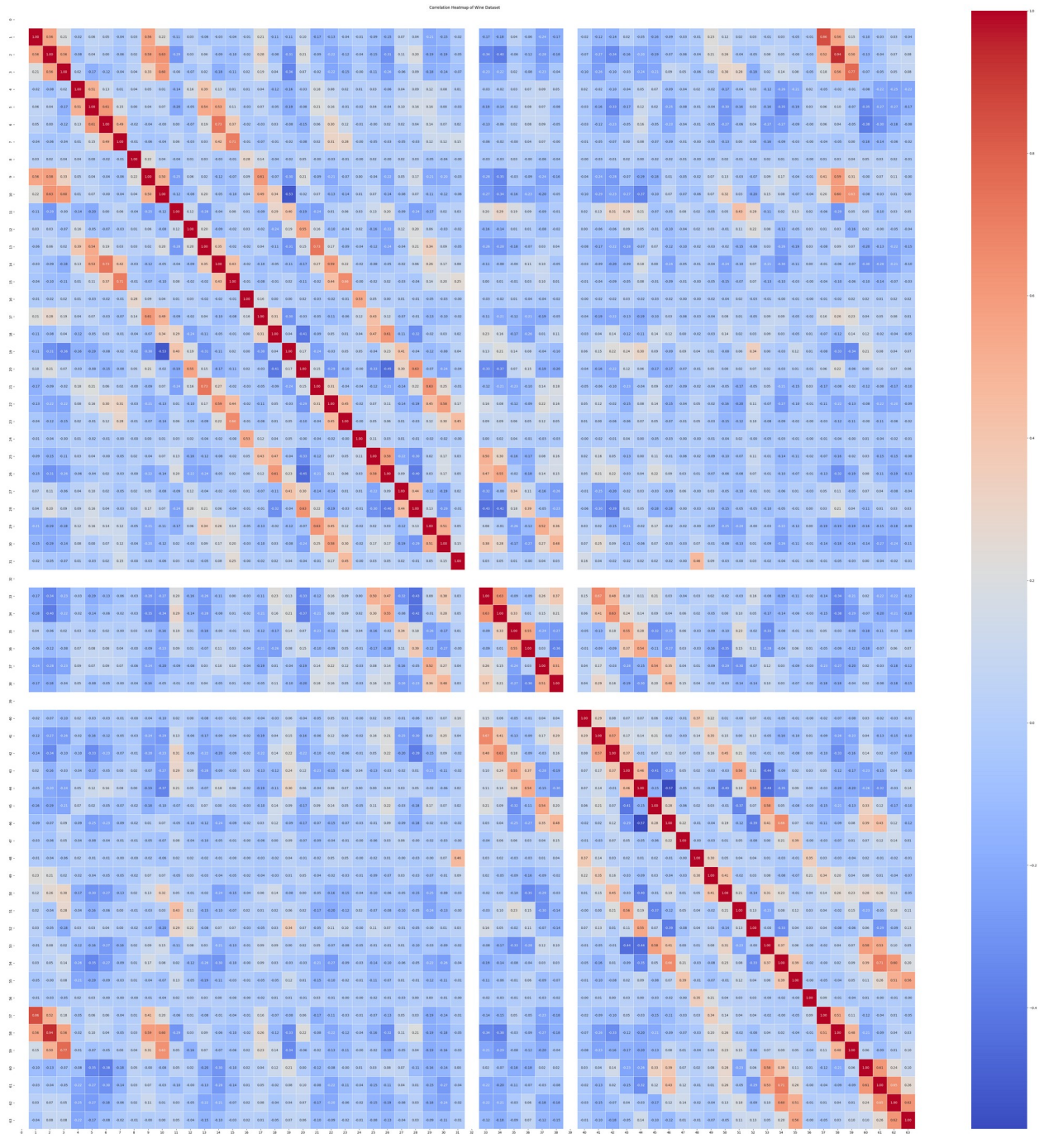
```
        [[ 0.,  0.,  2., ...,  0.,  0.,  0.],
         [ 0.,  0., 14., ..., 15.,  1.,  0.],
         [ 0.,  4., 16., ..., 16.,  7.,  0.],
         ...,
         [ 0.,  0.,  0., ..., 16.,  2.,  0.],
         [ 0.,  0.,  4., ..., 16.,  2.,  0.],
         [ 0.,  0.,  5., ..., 12.,  0.,  0.]],

        [[ 0.,  0., 10., ...,  1.,  0.,  0.],
         [ 0.,  2., 16., ...,  1.,  0.,  0.],
         [ 0.,  0., 15., ..., 15.,  0.,  0.],
         ...,
         [ 0.,  4., 16., ..., 16.,  6.,  0.],
         [ 0.,  8., 16., ..., 16.,  8.,  0.],
         [ 0.,  1.,  8., ..., 12.,  1.,  0.]]]),
 'DESCR': ".. _digits_dataset:\n\nOptical recognition of handwritten
digits dataset\n--------------------------------------------------\n\
n**Data Set Characteristics:**\n\n:Number of Instances: 1797\n:Number
of Attributes: 64\n:Attribute Information: 8x8 image of integer pixels
in the range 0..16.\n:Missing Attribute Values: None\n:Creator: E.
Alpaydin (alpaydin '@' boun.edu.tr)\n:Date: July; 1998\n\nThis is a
copy of the test set of the UCI ML hand-written digits datasets\
nhttps://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwr
itten+Digits\n\nThe data set contains images of hand-written digits:
10 classes where\neach class refers to a digit.\n\nPreprocessing
programs made available by NIST were used to extract\nnormalized
bitmaps of handwritten digits from a preprinted form. From a\ntotal of
43 people, 30 contributed to the training set and different 13\nto the
test set. 32x32 bitmaps are divided into nonoverlapping blocks of\n4x4
and the number of on pixels are counted in each block. This generates\
nan input matrix of 8x8 where each element is an integer in the range\
n0..16. This reduces dimensionality and gives invariance to small\
ndistortions.\n\nFor info on NIST preprocessing routines, see M. D.
Garris, J. L. Blue, G.\nT. Candela, D. L. Dimmick, J. Geist, P. J.
Grother, S. A. Janet, and C.\nL. Wilson, NIST Form-Based Handprint
Recognition System, NISTIR 5469,\n1994.\n\n.. dropdown:: References\n\
n  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and
Their\n    Applications to Handwritten Digit Recognition, MSc Thesis,
Institute of\n    Graduate Studies in Science and Engineering,
Bogazici University.\n  - E. Alpaydin, C. Kaynak (1998) Cascading
Classifiers, Kybernetika.\n  - Ken Tang and Ponnuthurai N. Suganthan
and Xi Yao and A. Kai Qin.\n    Linear dimensionalityreduction using
relevance weighted LDA. School of\n    Electrical and Electronic
Engineering Nanyang Technological University.\n    2005.\n  - Claudio
Gentile. A New Approximate Maximal Margin Classification\n
Algorithm. NIPS. 2000.\n"}

feature_names = digits.feature_names
scaler = StandardScaler()
```

```
X = digits.data
X = scaler.fit_transform(X)

y = digits.target

plt.figure(1, figsize=(3, 3))
plt.imshow(digits.images[-1], cmap=plt.cm.gray_r,
interpolation="nearest")
plt.show()
```



Heatmap

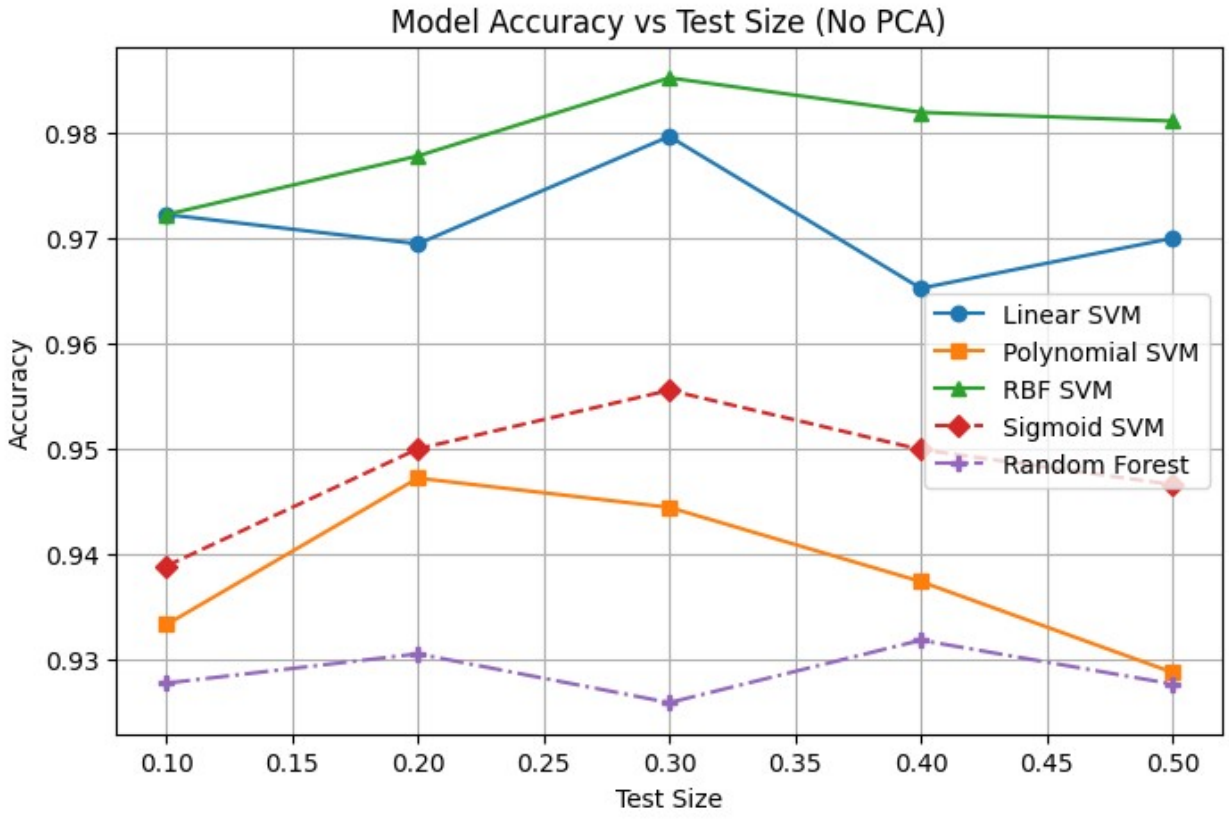
```
corr = pd.DataFrame(X).corr()

# Plot heatmap
plt.figure(figsize=(64,64))
sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f",
linewidths=0.5)
plt.title("Correlation Heatmap of Wine Dataset")
plt.show()
```




```
def analyse_train_test_split(X, y, pca_n=None, random_state=10):
    test_sizes = np.linspace(0.1, 0.5, 5)

    linear_acc = []
    poly_acc = []
    rbf_acc = []
    sigmoid_acc = []
    random_forest_acc = []
```

```
    for t in test_sizes:
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=t, shuffle=True,
random_state=random_state, stratify=y
        )

        scaler = StandardScaler()
        X_train_s = scaler.fit_transform(X_train)
        X_test_s  = scaler.transform(X_test)

        if pca_n is not None:
            pca = PCA(n_components=pca_n, svd_solver="full")
            X_train_s = pca.fit_transform(X_train_s)
            X_test_s  = pca.transform(X_test_s)

        svm_linear  = SVC(kernel="linear", C=1)
        svm_poly    = SVC(kernel="poly", degree=3, C=1, gamma="scale")
        svm_rbf     = SVC(kernel="rbf", C=1, gamma="scale")
        svm_sigmoid = SVC(kernel="sigmoid", C=1, gamma="scale")
        forest      = RandomForestClassifier(n_estimators=200,
criterion="entropy",
                                             max_depth=4,
random_state=random_state)

        svm_linear.fit(X_train_s, y_train)
        linear_acc.append(accuracy_score(y_test,
svm_linear.predict(X_test_s)))

        svm_poly.fit(X_train_s, y_train)
        poly_acc.append(accuracy_score(y_test,
svm_poly.predict(X_test_s)))

        svm_rbf.fit(X_train_s, y_train)
        rbf_acc.append(accuracy_score(y_test,
svm_rbf.predict(X_test_s)))

        svm_sigmoid.fit(X_train_s, y_train)
        sigmoid_acc.append(accuracy_score(y_test,
svm_sigmoid.predict(X_test_s)))

        forest.fit(X_train_s, y_train)
        random_forest_acc.append(accuracy_score(y_test,
forest.predict(X_test_s)))

    plt.figure(figsize=(8, 5))
    plt.plot(test_sizes, linear_acc, marker='o', linestyle='-',
label='Linear SVM')
    plt.plot(test_sizes, poly_acc,   marker='s',
label='Polynomial SVM')
    plt.plot(test_sizes, rbf_acc,    marker='^',
```

```
label='RBF SVM')
    plt.plot(test_sizes, sigmoid_acc,marker='D',
linestyle='--',label='Sigmoid SVM')
    plt.plot(test_sizes, random_forest_acc, marker='P',
linestyle='-.', label='Random Forest')
    plt.xlabel("Test Size")
    plt.ylabel("Accuracy")
    title_tag = f"(PCA={pca_n})" if pca_n is not None else "(No PCA)"
    plt.title(f"Model Accuracy vs Test Size {title_tag}")
    plt.legend()
    plt.grid(True)
    plt.show()

analyse_train_test_split(X, y)
```



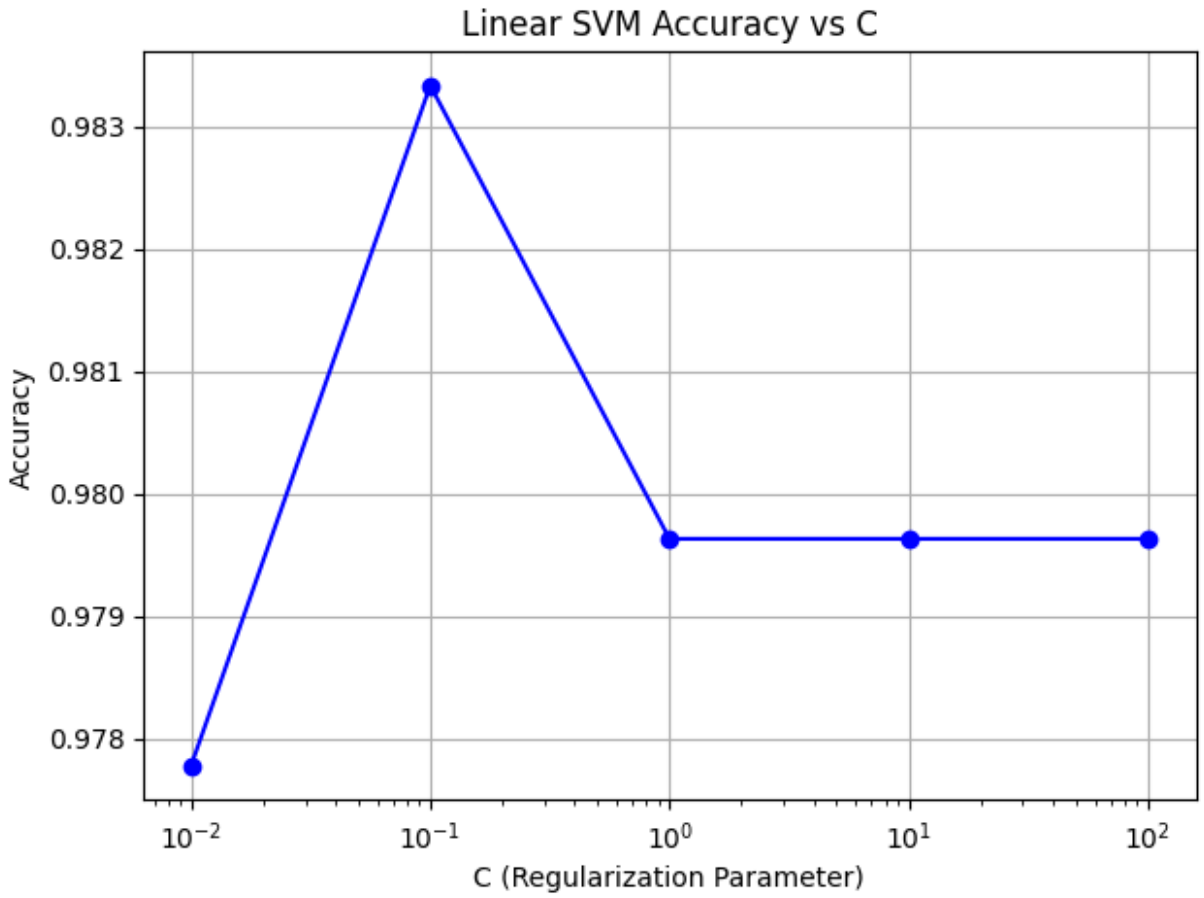
We conclude that, for all the different kernels of SVM, we will use 30% of the dataset for testing and the rest for training. And we are using 40% split for Random Forest Classifier.

Linear SVM

```
def train_linear_svm(X, y, split=0.30, shuffle=True, random_state=10):
    # Train/test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=split, shuffle=shuffle,
random_state=random_state, stratify=y
    )

    # Scale features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # C values
    c_params = [0.01, 0.1, 1, 10, 100]
    linear_svm_acc = []

    # Train SVM for each C
    for c in c_params:
        svm_clf = SVC(kernel="linear", C=c)
        svm_clf.fit(X_train, y_train)
        y_pred = svm_clf.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        linear_svm_acc.append(acc)
        print(f"C={c}: Accuracy={acc:.4f}")

    # Plot results
    plt.figure(figsize=(7,5))
    plt.plot(c_params, linear_svm_acc, marker="o", linestyle="-",
color="b")
    plt.xscale("log")
    plt.xlabel("C (Regularization Parameter)")
    plt.ylabel("Accuracy")
    plt.title("Linear SVM Accuracy vs C")
    plt.grid(True)
    plt.show()

    return c_params, linear_svm_acc

train_linear_svm(X, y)
```

```
C=0.01: Accuracy=0.9778
C=0.1: Accuracy=0.9833
```

```
C=1: Accuracy=0.9796
C=10: Accuracy=0.9796
C=100: Accuracy=0.9796
```




```
([0.01, 0.1, 1, 10, 100],
 [0.9777777777777777,
  0.9833333333333333,
  0.9796296296296296,
  0.9796296296296296,
  0.9796296296296296])
```

Output without hyperparameter tuning

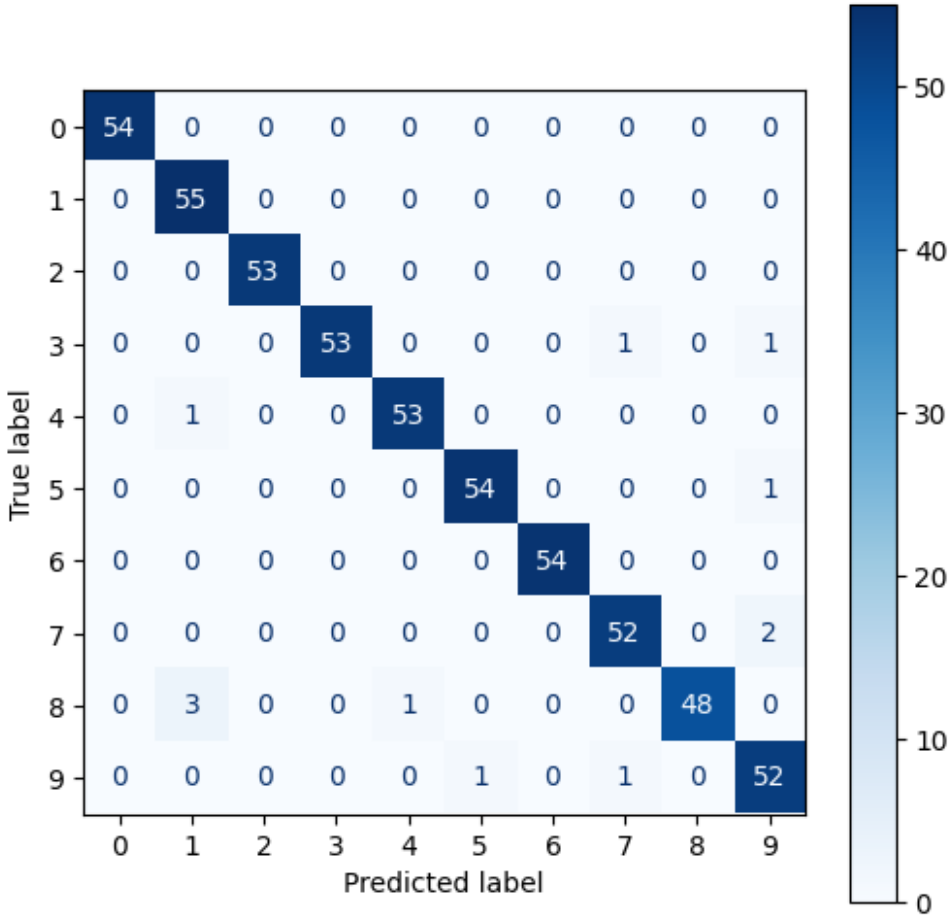
```
svm_clf = SVC(kernel="linear")

X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.30, shuffle=True, random_state=10,
stratify=y
    )

svm_clf.fit(X_train, y_train)
```

```
y_pred = svm_clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        54
           1       0.93      1.00      0.96        55
           2       1.00      1.00      1.00        53
           3       1.00      0.96      0.98        55
           4       0.98      0.98      0.98        54
           5       0.98      0.98      0.98        55
           6       1.00      1.00      1.00        54
           7       0.96      0.96      0.96        54
           8       1.00      0.92      0.96        52
           9       0.93      0.96      0.95        54

    accuracy                           0.98       540
   macro avg       0.98      0.98      0.98       540
weighted avg       0.98      0.98      0.98       540
```

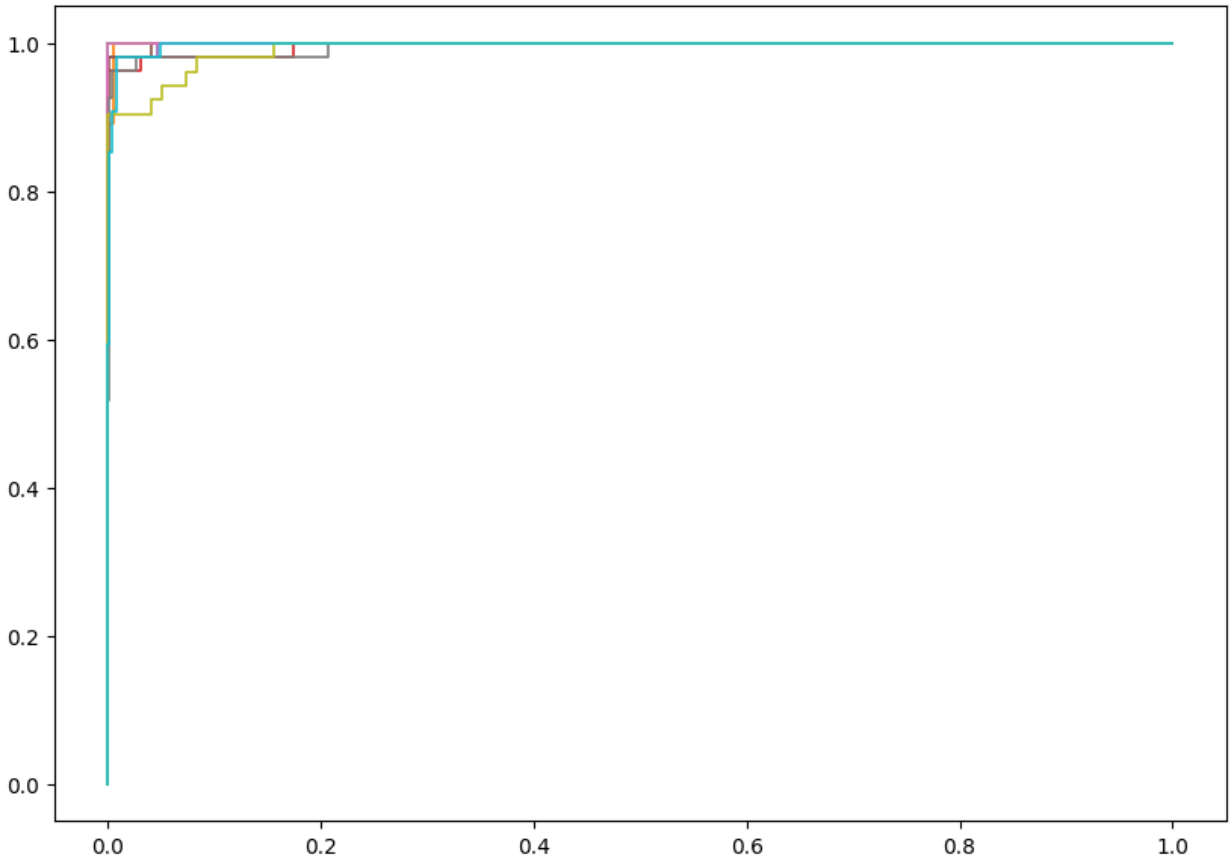
```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(svm_clf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7a18541954f0>
```



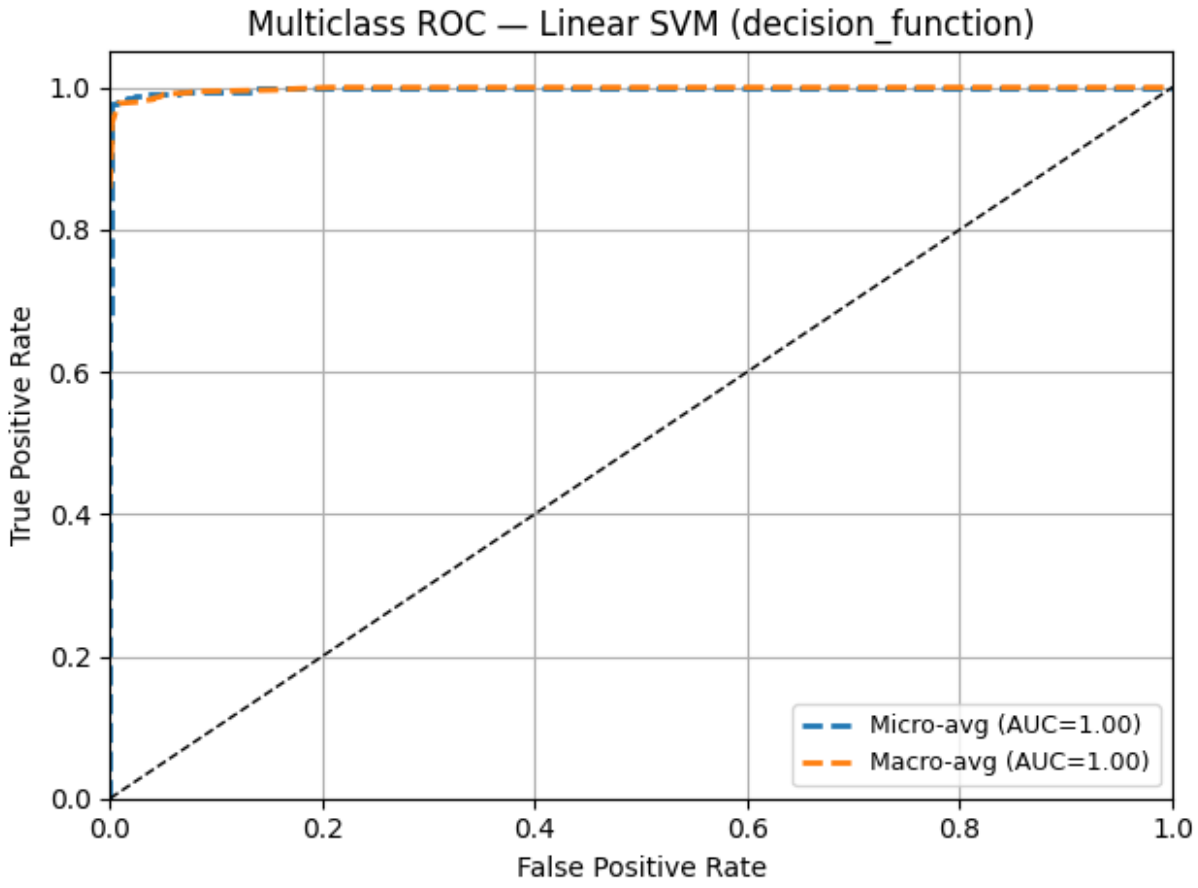

```
y_score = svm_clf.decision_function(X_test)

classes = np.unique(y)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(),
y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

all_fpr = np.unique(np.concatenate([fpr[i] for i in
range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
```

```
fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

plt.figure(figsize=(10, 7))

for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], lw=1.2, label=f"Class {classes[i]}
(AUC={roc_auc[i]:.2f})")
```




```
plt.plot(fpr["micro"], tpr["micro"], linestyle="--", lw=2,
label=f"Micro-avg (AUC={roc_auc['micro']:.2f})")
plt.plot(fpr["macro"], tpr["macro"], linestyle="--", lw=2,
label=f"Macro-avg (AUC={roc_auc['macro']:.2f})")

plt.plot([0,1],[0,1],'k--',lw=1)
plt.xlim([0.0, 1.0]); plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
plt.title("Multiclass ROC – Linear SVM (decision_function)")
plt.legend(loc="lower right", fontsize=9)
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
print(f"Micro-average AUC: {roc_auc['micro']:.4f}")
print(f"Macro-average AUC: {roc_auc['macro']:.4f}")
```




```
Micro-average AUC: 0.9979
Macro-average AUC: 0.9981
```

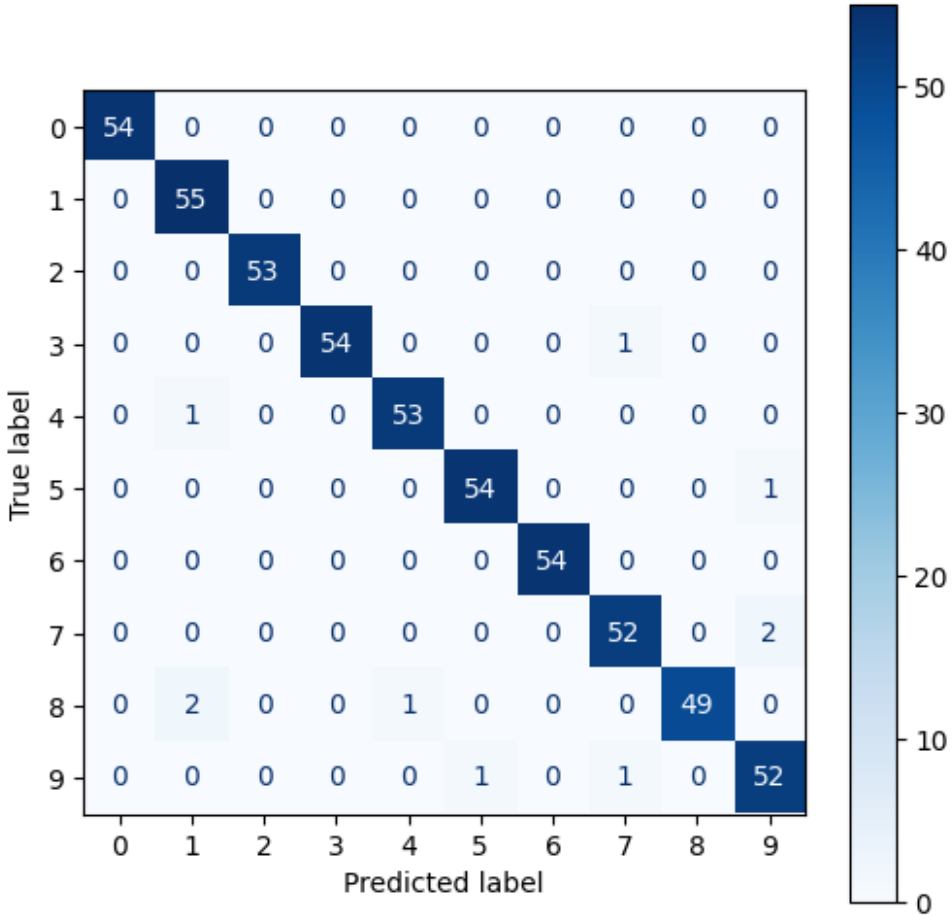
Output with Hyperparameter Tuning

```
svm_clf = SVC(kernel="linear", C=0.1)

X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.30, shuffle=True, random_state=10,
stratify=y
    )

svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        54
           1       0.95      1.00      0.97        55
           2       1.00      1.00      1.00        53
           3       1.00      0.98      0.99        55
           4       0.98      0.98      0.98        54
           5       0.98      0.98      0.98        55
           6       1.00      1.00      1.00        54
           7       0.96      0.96      0.96        54
           8       1.00      0.94      0.97        52
           9       0.95      0.96      0.95        54

    accuracy                           0.98       540
   macro avg       0.98      0.98      0.98       540
weighted avg       0.98      0.98      0.98       540
```

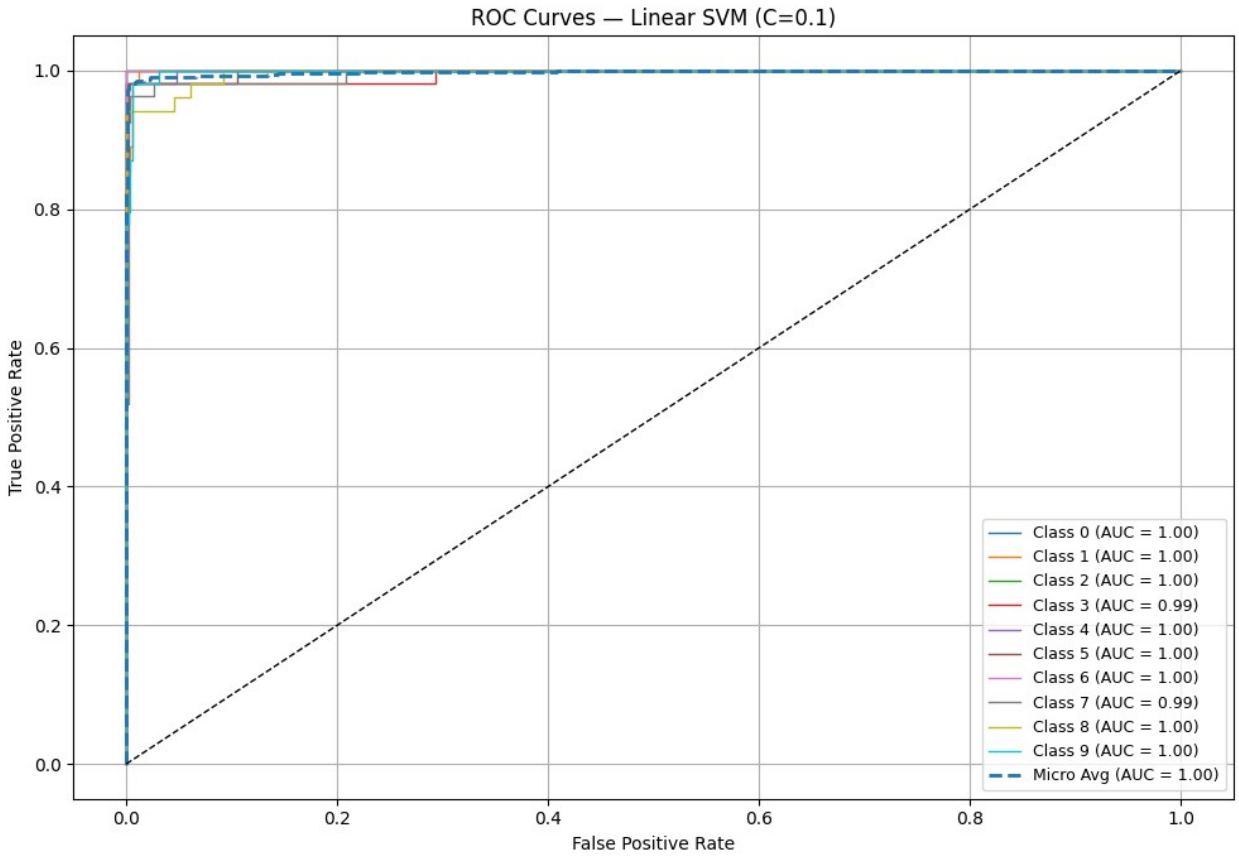
```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(svm_clf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7a185827f650>
```




```
# Binarize the labels
y_test_bin = label_binarize(y_test, classes=np.unique(y))
n_classes = y_test_bin.shape[1]

y_score = svm_clf.decision_function(X_test)

fpr, tpr, roc_auc = {}, {}, {}

# Per-class ROC
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Micro-average
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(),
y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

plt.figure(figsize=(10, 7))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f"Class {i} (AUC =
```

```
{roc_auc[i]:.2f})", lw=1)

plt.plot(fpr["micro"], tpr["micro"], linestyle='--', lw=2,
label=f"Micro Avg (AUC = {roc_auc['micro']:.2f})")
plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves — Linear SVM (C=0.1)")
plt.legend(loc="lower right", fontsize=9)
plt.grid(True)
plt.tight_layout()
plt.show()
```



Polynomial SVM

```
def _get_scores(model, X):

    if hasattr(model, "predict_proba"):
        scores = model.predict_proba(X)
        # Some estimators return shape (n_samples,) for binary → make
it (n_samples, 2)
        if scores.ndim == 1:
            scores = np.vstack([1 - scores, scores]).T
```

```
        return scores
    if hasattr(model, "decision_function"):
        scores = model.decision_function(X)
        # Some binary estimators return shape (n_samples,) → make it
(n_samples, 2)
        if scores.ndim == 1:
            scores = np.vstack([-scores, scores]).T
        return scores
    raise ValueError("Model must implement predict_proba or
decision_function for ROC/AUC.")

def roc_auc_report(model, X_test, y_test, classes=None, title="ROC",
plot=True, per_class=True):

    # Normalize class list and binarize ground truth
    if classes is None:
        classes = np.unique(y_test)
    classes = np.array(classes)
    y_test_bin = label_binarize(y_test, classes=classes)
    n_classes = y_test_bin.shape[1]

    # Scores matrix: shape (n_samples, n_classes)
    y_score = _get_scores(model, X_test)

    # If estimator is binary and returned a single column/proba,
expand to two
    if y_score.ndim == 1:
        y_score = np.vstack([1 - y_score, y_score]).T

    # Per-class ROC and AUC
    fpr, tpr, roc_auc = {}, {}, {}
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Micro-average
    fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(),
y_score.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

    # Macro-average (interpolate all FPRs, then average TPRs)
    all_fpr = np.unique(np.concatenate([fpr[i] for i in
range(n_classes)]))
    mean_tpr = np.zeros_like(all_fpr)
    for i in range(n_classes):
        mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
    mean_tpr /= n_classes
    fpr["macro"] = all_fpr
    tpr["macro"] = mean_tpr
    roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])
```

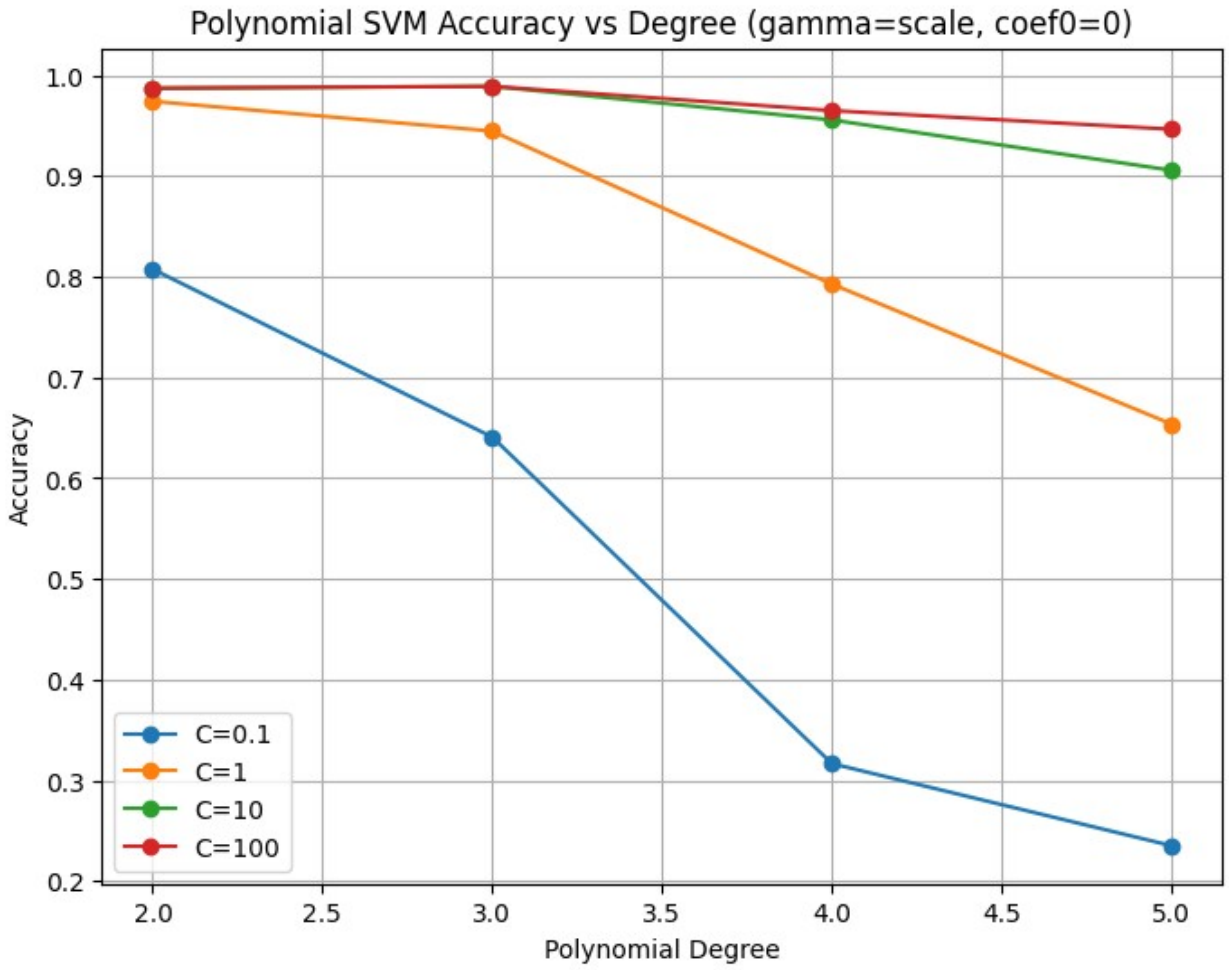
```
    # ----- Print summary -----
    print(f"\n=== ROC-AUC Report: {title} ===")
    print(f"Micro-average AUC: {roc_auc['micro']:.4f}")
    print(f"Macro-average AUC: {roc_auc['macro']:.4f}")
    if per_class:
        for i, c in enumerate(classes):
            print(f"Class {c} AUC: {roc_auc[i]:.4f}")

    # ----- Plot -----
    if plot:
        plt.figure(figsize=(10, 7))
        if per_class:
            for i, c in enumerate(classes):
                plt.plot(fpr[i], tpr[i], lw=1.2, label=f"Class {c}
(AUC={roc_auc[i]:.2f})")
        plt.plot(fpr["micro"], tpr["micro"], linestyle="--", lw=2,
                 label=f"Micro-avg (AUC={roc_auc['micro']:.2f})")
        plt.plot(fpr["macro"], tpr["macro"], linestyle="--", lw=2,
                 label=f"Macro-avg (AUC={roc_auc['macro']:.2f})")
        plt.plot([0, 1], [0, 1], "k--", lw=1)
        plt.xlim([0.0, 1.0]); plt.ylim([0.0, 1.05])
        plt.xlabel("False Positive Rate"); plt.ylabel("True Positive
Rate")
        plt.title(f"{title} – ROC Curves")
        plt.legend(loc="lower right", fontsize=9)
        plt.grid(True)
        plt.tight_layout()
        plt.show()

    return {"micro": roc_auc["micro"], "macro": roc_auc["macro"],
            "per_class": {c: roc_auc[i] for i, c in
enumerate(classes)}}

param_grid = {
    "C": [0.1, 1, 10, 100],
    "degree": [2, 3, 4, 5],
    "gamma": ["scale", "auto"],
    "coef0": [0, 1, 2]
}

def train_polynomial_svm(X, y, split=0.30, shuffle=True,
random_state=10):
    # Train/test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=split, shuffle=shuffle,
random_state=random_state, stratify=y
    )

    # Scale features
```

```
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    results = []

    for c in param_grid["C"]:
        for degree in param_grid["degree"]:
            for gamma in param_grid["gamma"]:
                for coef in param_grid["coef0"]:

                    svm_clf = SVC(kernel="poly", C=c, degree=degree,
gamma=gamma, coef0=coef)
                    svm_clf.fit(X_train, y_train)
                    y_pred = svm_clf.predict(X_test)
                    acc = accuracy_score(y_test, y_pred)

                    results.append({
                        "C": c,
                        "degree": degree,
                        "gamma": gamma,
                        "coef0": coef,
                        "accuracy": acc
                    })

                    print(f"C={c}, degree={degree}, gamma={gamma},
coef0={coef} → Accuracy={acc:.4f}")

    df = pd.DataFrame(results)

    best = df.loc[df["accuracy"].idxmax()]
    print("\n Best Parameters:")
    print(best)

    plt.figure(figsize=(8,6))
    for c in param_grid["C"]:
        subset = df[(df["C"]==c) & (df["gamma"]=="scale") &
(df["coef0"]==0)]
        plt.plot(subset["degree"], subset["accuracy"], marker="o",
label=f"C={c}")

    plt.xlabel("Polynomial Degree")
    plt.ylabel("Accuracy")
    plt.title("Polynomial SVM Accuracy vs Degree (gamma=scale,
coef0=0)")
    plt.legend()
    plt.grid(True)
    plt.show()

    return df, best
```

```
best = train_polynomial_svm(X, y)

C=0.1, degree=2, gamma=scale, coef0=0 → Accuracy=0.8074
C=0.1, degree=2, gamma=scale, coef0=1 → Accuracy=0.9685
C=0.1, degree=2, gamma=scale, coef0=2 → Accuracy=0.9796
C=0.1, degree=2, gamma=auto, coef0=0 → Accuracy=0.8130
C=0.1, degree=2, gamma=auto, coef0=1 → Accuracy=0.9630
C=0.1, degree=2, gamma=auto, coef0=2 → Accuracy=0.9796
C=0.1, degree=3, gamma=scale, coef0=0 → Accuracy=0.6407
C=0.1, degree=3, gamma=scale, coef0=1 → Accuracy=0.9815
C=0.1, degree=3, gamma=scale, coef0=2 → Accuracy=0.9852
C=0.1, degree=3, gamma=auto, coef0=0 → Accuracy=0.5519
C=0.1, degree=3, gamma=auto, coef0=1 → Accuracy=0.9778
C=0.1, degree=3, gamma=auto, coef0=2 → Accuracy=0.9852
C=0.1, degree=4, gamma=scale, coef0=0 → Accuracy=0.3167
C=0.1, degree=4, gamma=scale, coef0=1 → Accuracy=0.9907
C=0.1, degree=4, gamma=scale, coef0=2 → Accuracy=0.9907
C=0.1, degree=4, gamma=auto, coef0=0 → Accuracy=0.2593
C=0.1, degree=4, gamma=auto, coef0=1 → Accuracy=0.9852
C=0.1, degree=4, gamma=auto, coef0=2 → Accuracy=0.9889
C=0.1, degree=5, gamma=scale, coef0=0 → Accuracy=0.2352
C=0.1, degree=5, gamma=scale, coef0=1 → Accuracy=0.9926
C=0.1, degree=5, gamma=scale, coef0=2 → Accuracy=0.9889
C=0.1, degree=5, gamma=auto, coef0=0 → Accuracy=0.2074
C=0.1, degree=5, gamma=auto, coef0=1 → Accuracy=0.9926
C=0.1, degree=5, gamma=auto, coef0=2 → Accuracy=0.9907
C=1, degree=2, gamma=scale, coef0=0 → Accuracy=0.9741
C=1, degree=2, gamma=scale, coef0=1 → Accuracy=0.9852
C=1, degree=2, gamma=scale, coef0=2 → Accuracy=0.9870
C=1, degree=2, gamma=auto, coef0=0 → Accuracy=0.9722
C=1, degree=2, gamma=auto, coef0=1 → Accuracy=0.9852
C=1, degree=2, gamma=auto, coef0=2 → Accuracy=0.9852
C=1, degree=3, gamma=scale, coef0=0 → Accuracy=0.9444
C=1, degree=3, gamma=scale, coef0=1 → Accuracy=0.9926
C=1, degree=3, gamma=scale, coef0=2 → Accuracy=0.9889
C=1, degree=3, gamma=auto, coef0=0 → Accuracy=0.9333
C=1, degree=3, gamma=auto, coef0=1 → Accuracy=0.9907
C=1, degree=3, gamma=auto, coef0=2 → Accuracy=0.9889
C=1, degree=4, gamma=scale, coef0=0 → Accuracy=0.7926
C=1, degree=4, gamma=scale, coef0=1 → Accuracy=0.9907
C=1, degree=4, gamma=scale, coef0=2 → Accuracy=0.9870
C=1, degree=4, gamma=auto, coef0=0 → Accuracy=0.7574
C=1, degree=4, gamma=auto, coef0=1 → Accuracy=0.9907
C=1, degree=4, gamma=auto, coef0=2 → Accuracy=0.9889
C=1, degree=5, gamma=scale, coef0=0 → Accuracy=0.6537
C=1, degree=5, gamma=scale, coef0=1 → Accuracy=0.9907
C=1, degree=5, gamma=scale, coef0=2 → Accuracy=0.9889
C=1, degree=5, gamma=auto, coef0=0 → Accuracy=0.5981
C=1, degree=5, gamma=auto, coef0=1 → Accuracy=0.9907
C=1, degree=5, gamma=auto, coef0=2 → Accuracy=0.9907
```

```
C=10, degree=2, gamma=scale, coef0=0 → Accuracy=0.9870
C=10, degree=2, gamma=scale, coef0=1 → Accuracy=0.9889
C=10, degree=2, gamma=scale, coef0=2 → Accuracy=0.9889
C=10, degree=2, gamma=auto, coef0=0 → Accuracy=0.9870
C=10, degree=2, gamma=auto, coef0=1 → Accuracy=0.9889
C=10, degree=2, gamma=auto, coef0=2 → Accuracy=0.9889
C=10, degree=3, gamma=scale, coef0=0 → Accuracy=0.9889
C=10, degree=3, gamma=scale, coef0=1 → Accuracy=0.9907
C=10, degree=3, gamma=scale, coef0=2 → Accuracy=0.9889
C=10, degree=3, gamma=auto, coef0=0 → Accuracy=0.9889
C=10, degree=3, gamma=auto, coef0=1 → Accuracy=0.9889
C=10, degree=3, gamma=auto, coef0=2 → Accuracy=0.9889
C=10, degree=4, gamma=scale, coef0=0 → Accuracy=0.9556
C=10, degree=4, gamma=scale, coef0=1 → Accuracy=0.9907
C=10, degree=4, gamma=scale, coef0=2 → Accuracy=0.9870
C=10, degree=4, gamma=auto, coef0=0 → Accuracy=0.9537
C=10, degree=4, gamma=auto, coef0=1 → Accuracy=0.9907
C=10, degree=4, gamma=auto, coef0=2 → Accuracy=0.9889
C=10, degree=5, gamma=scale, coef0=0 → Accuracy=0.9056
C=10, degree=5, gamma=scale, coef0=1 → Accuracy=0.9907
C=10, degree=5, gamma=scale, coef0=2 → Accuracy=0.9889
C=10, degree=5, gamma=auto, coef0=0 → Accuracy=0.8778
C=10, degree=5, gamma=auto, coef0=1 → Accuracy=0.9907
C=10, degree=5, gamma=auto, coef0=2 → Accuracy=0.9907
C=100, degree=2, gamma=scale, coef0=0 → Accuracy=0.9870
C=100, degree=2, gamma=scale, coef0=1 → Accuracy=0.9889
C=100, degree=2, gamma=scale, coef0=2 → Accuracy=0.9889
C=100, degree=2, gamma=auto, coef0=0 → Accuracy=0.9870
C=100, degree=2, gamma=auto, coef0=1 → Accuracy=0.9889
C=100, degree=2, gamma=auto, coef0=2 → Accuracy=0.9889
C=100, degree=3, gamma=scale, coef0=0 → Accuracy=0.9889
C=100, degree=3, gamma=scale, coef0=1 → Accuracy=0.9907
C=100, degree=3, gamma=scale, coef0=2 → Accuracy=0.9889
C=100, degree=3, gamma=auto, coef0=0 → Accuracy=0.9889
C=100, degree=3, gamma=auto, coef0=1 → Accuracy=0.9889
C=100, degree=3, gamma=auto, coef0=2 → Accuracy=0.9889
C=100, degree=4, gamma=scale, coef0=0 → Accuracy=0.9648
C=100, degree=4, gamma=scale, coef0=1 → Accuracy=0.9907
C=100, degree=4, gamma=scale, coef0=2 → Accuracy=0.9870
C=100, degree=4, gamma=auto, coef0=0 → Accuracy=0.9648
C=100, degree=4, gamma=auto, coef0=1 → Accuracy=0.9907
C=100, degree=4, gamma=auto, coef0=2 → Accuracy=0.9889
C=100, degree=5, gamma=scale, coef0=0 → Accuracy=0.9463
C=100, degree=5, gamma=scale, coef0=1 → Accuracy=0.9907
C=100, degree=5, gamma=scale, coef0=2 → Accuracy=0.9889
C=100, degree=5, gamma=auto, coef0=0 → Accuracy=0.9444
C=100, degree=5, gamma=auto, coef0=1 → Accuracy=0.9907
C=100, degree=5, gamma=auto, coef0=2 → Accuracy=0.9907
```

```
 Best Parameters:
C                0.1
degree             5
gamma          scale
coef0              1
accuracy    0.992593
Name: 19, dtype: object
```




```
best
```

	C	degree	gamma	coef0	accuracy
0	0.1	2	scale	0	0.807407
1	0.1	2	scale	1	0.968519
2	0.1	2	scale	2	0.979630
3	0.1	2	auto	0	0.812963
4	0.1	2	auto	1	0.962963
..	...	...	...	...	...
91	100.0	5	scale	1	0.990741
92	100.0	5	scale	2	0.988889

```
93  100.0       5   auto      0  0.944444
94  100.0       5   auto      1  0.990741
95  100.0       5   auto      2  0.990741

[96 rows x 5 columns],
C                 0.1
degree              5
gamma           scale
coef0               1
accuracy     0.992593
Name: 19, dtype: object)
```

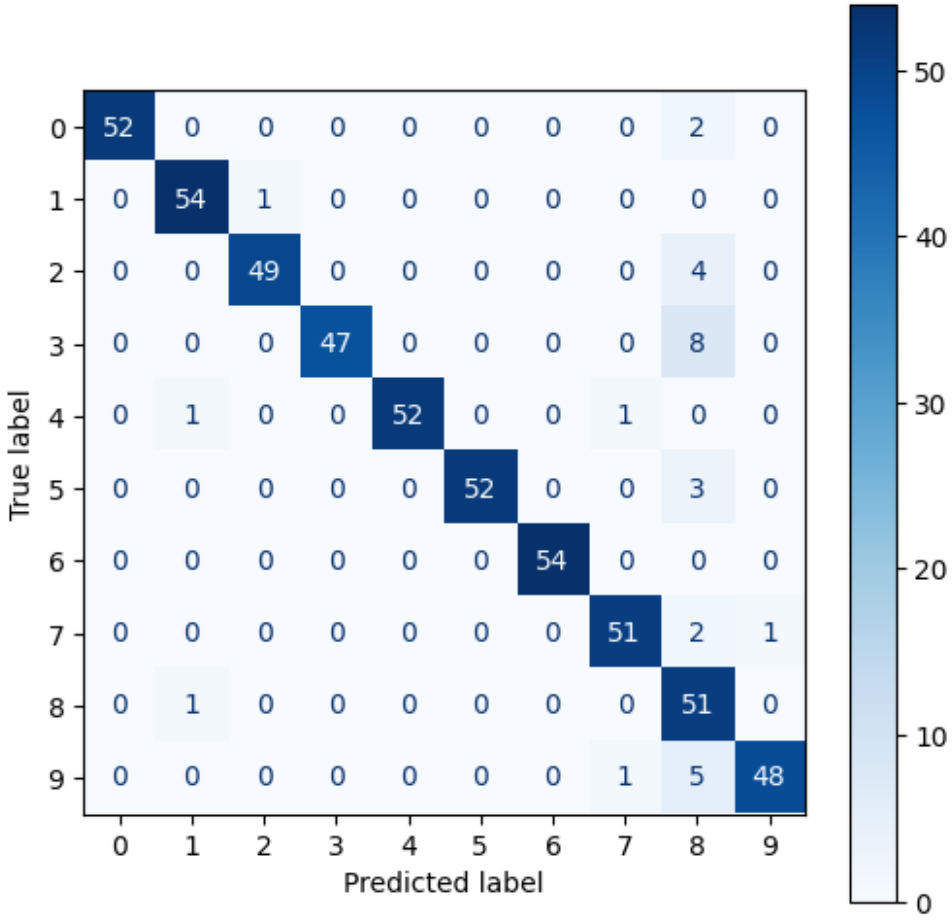
Output without Hyperparameter Tuning

```
svm_clf = SVC(kernel="poly")

X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.30, shuffle=True, random_state=10,
stratify=y
    )

svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.96      0.98        54
           1       0.96      0.98      0.97        55
           2       0.98      0.92      0.95        53
           3       1.00      0.85      0.92        55
           4       1.00      0.96      0.98        54
           5       1.00      0.95      0.97        55
           6       1.00      1.00      1.00        54
           7       0.96      0.94      0.95        54
           8       0.68      0.98      0.80        52
           9       0.98      0.89      0.93        54

    accuracy                           0.94       540
   macro avg       0.96      0.94      0.95       540
weighted avg       0.96      0.94      0.95       540
```

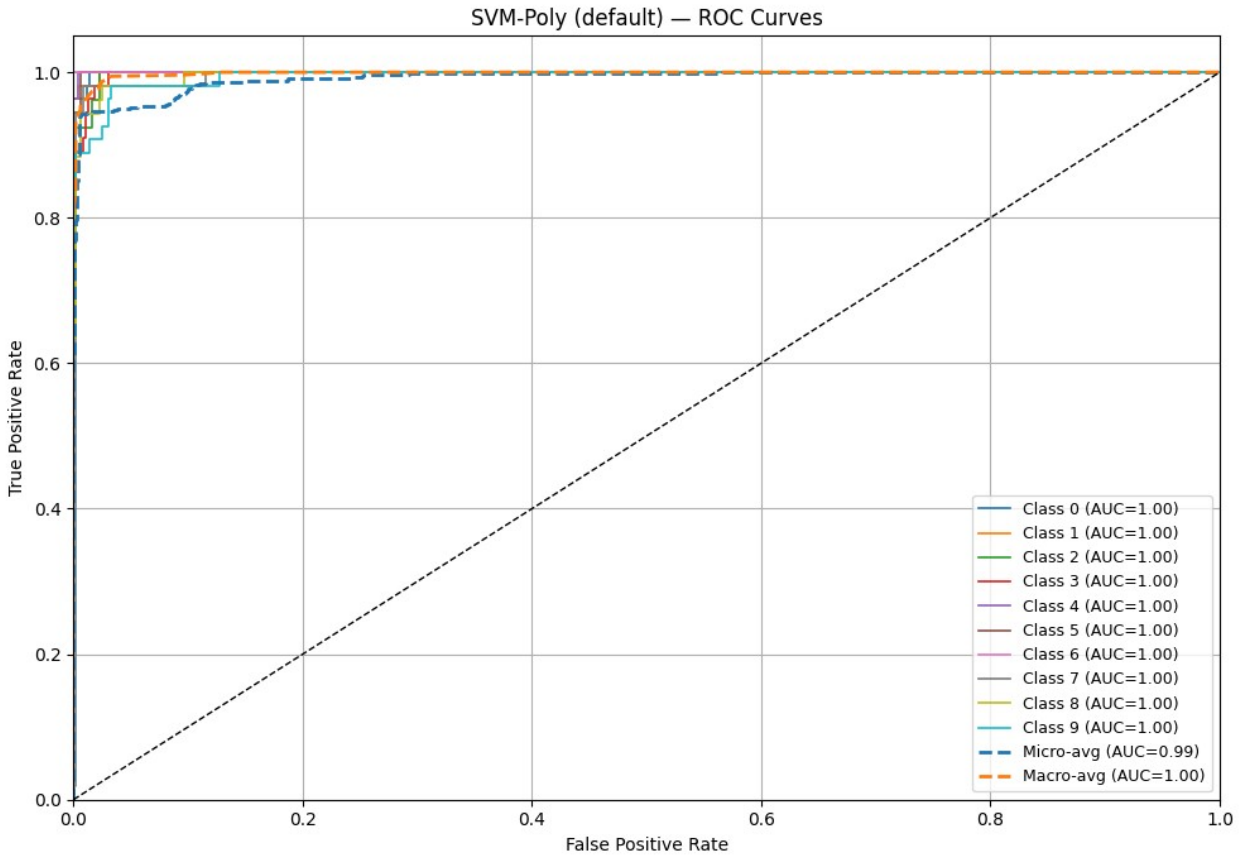
```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(svm_clf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7a181de029c0>
```




```
svm_poly_default = SVC(kernel="poly")
svm_poly_default.fit(X_train, y_train)
_ = roc_auc_report(svm_poly_default, X_test, y_test,
classes=np.unique(y),
                   title="SVM-Poly (default)")
```

```
=== ROC-AUC Report: SVM-Poly (default) ===
Micro-average AUC: 0.9919
Macro-average AUC: 0.9984
Class 0 AUC: 0.9995
Class 1 AUC: 0.9995
Class 2 AUC: 0.9985
Class 3 AUC: 0.9983
Class 4 AUC: 0.9998
Class 5 AUC: 0.9997
Class 6 AUC: 1.0000
Class 7 AUC: 0.9957
Class 8 AUC: 0.9964
Class 9 AUC: 0.9951
```



Output with Hyperparameter Tuning

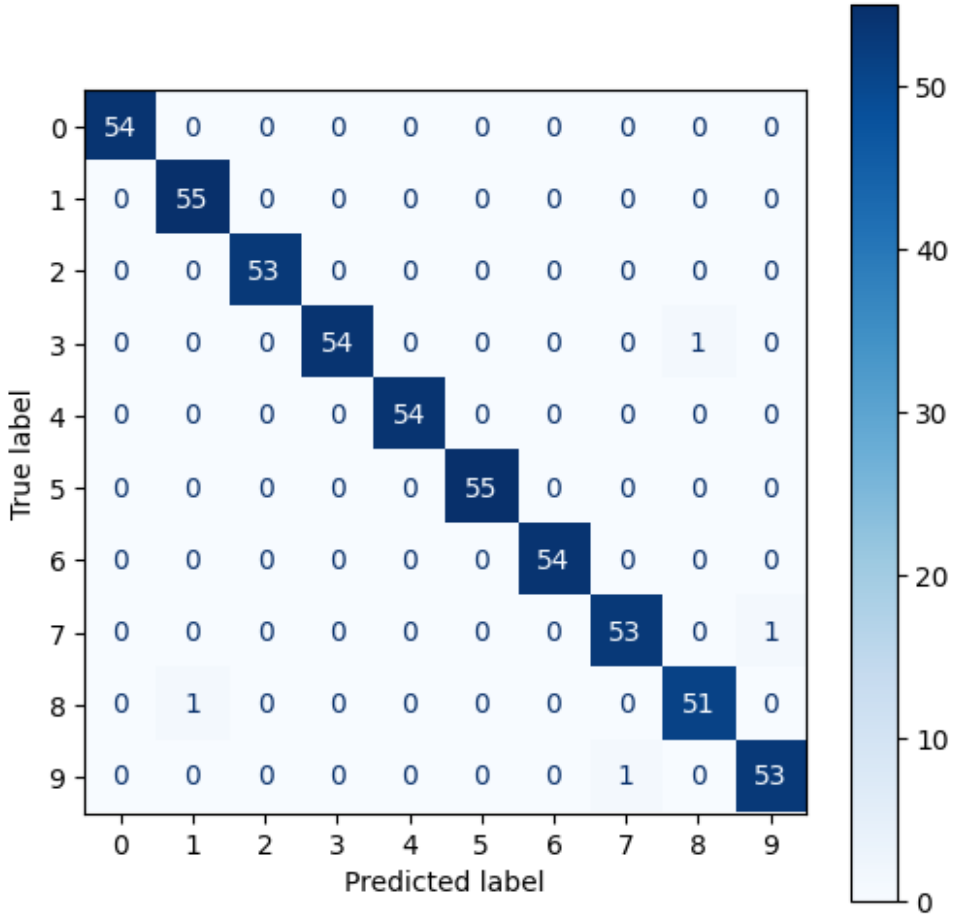
```
svm_clf = SVC(kernel="poly", C=0.1, degree=5, gamma="scale", coef0=1)

X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.30, shuffle=True, random_state=10,
stratify=y
    )

svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	54
1	0.98	1.00	0.99	55
2	1.00	1.00	1.00	53
3	1.00	0.98	0.99	55
4	1.00	1.00	1.00	54
5	1.00	1.00	1.00	55
6	1.00	1.00	1.00	54
7	0.98	0.98	0.98	54

```
           8       0.98      0.98      0.98        52
           9       0.98      0.98      0.98        54

    accuracy                           0.99       540
   macro avg       0.99      0.99      0.99       540
weighted avg       0.99      0.99      0.99       540
```

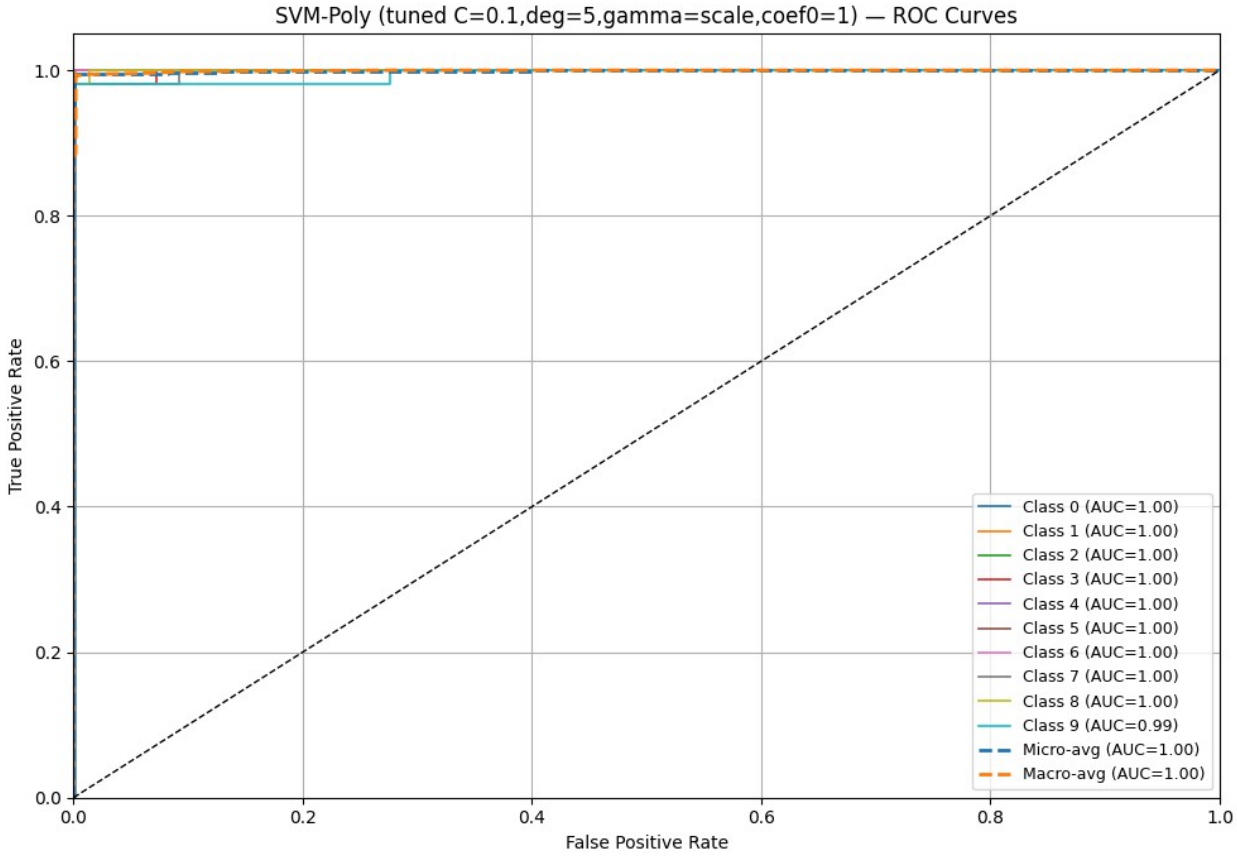
```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(svm_clf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7a18156b7230>
```




```
svm_poly_tuned = SVC(kernel="poly", C=0.1, degree=5, gamma="scale",
coef0=1)
svm_poly_tuned.fit(X_train, y_train)
_ = roc_auc_report(svm_poly_tuned, X_test, y_test,
classes=np.unique(y),
```

```
                    title="SVM-Poly (tuned
C=0.1,deg=5,gamma=scale,coef0=1)")


=== ROC–AUC Report: SVM-Poly (tuned C=0.1,deg=5,gamma=scale,coef0=1)
===
Micro-average AUC: 0.9986
Macro-average AUC: 0.9992
Class 0 AUC: 1.0000
Class 1 AUC: 0.9999
Class 2 AUC: 1.0000
Class 3 AUC: 0.9987
Class 4 AUC: 1.0000
Class 5 AUC: 1.0000
Class 6 AUC: 1.0000
Class 7 AUC: 0.9963
Class 8 AUC: 0.9996
Class 9 AUC: 0.9948
```



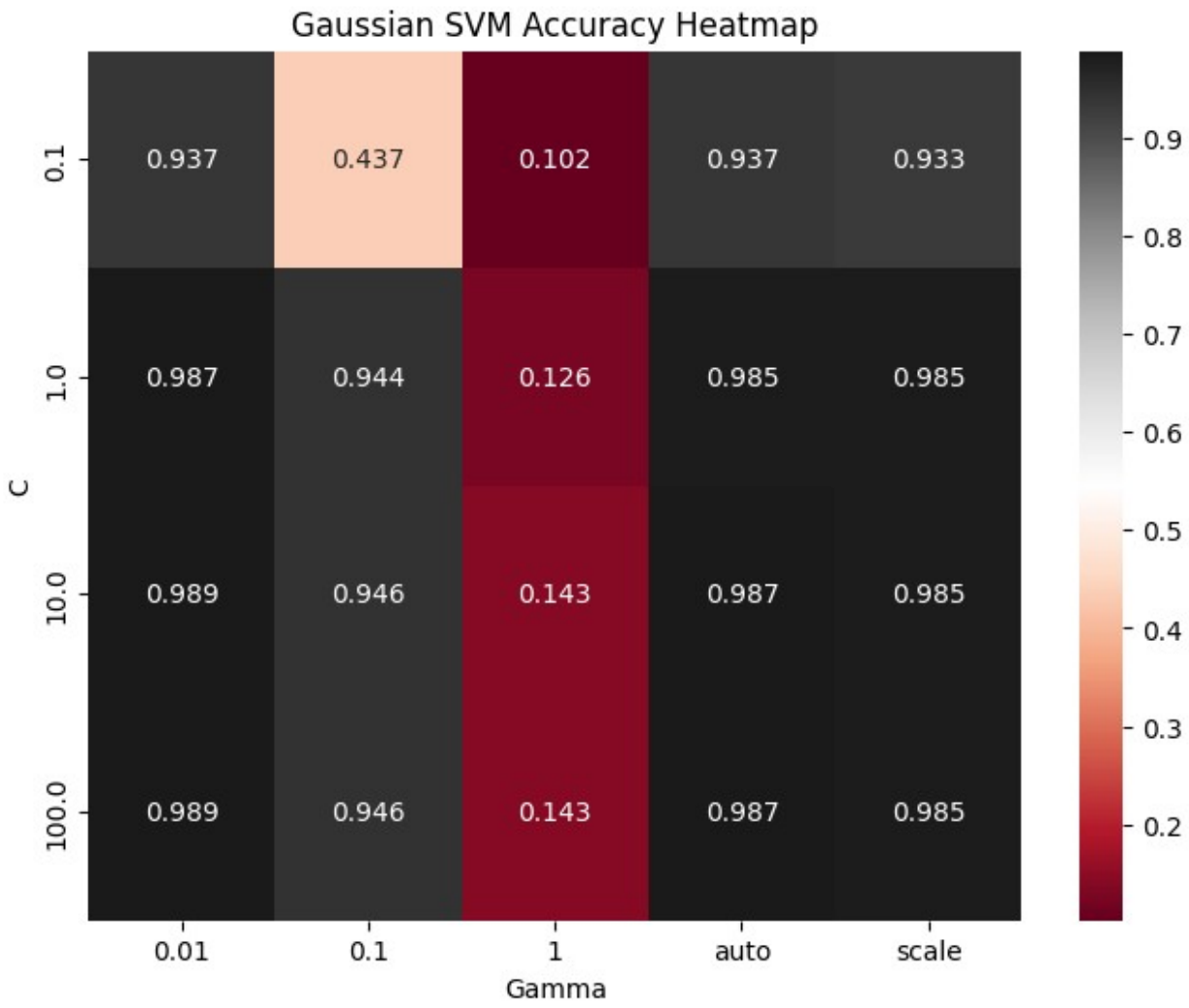
Gausian SVM

```
param_grid_rbf = {
    "C": [0.1, 1, 10, 100],
```

```
    "gamma": ["scale", "auto", 0.01, 0.1, 1]  # can also try numeric
values
}

def train_rbf_svm(X, y, split=0.30, shuffle=True, random_state=10):
    # Train/test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=split, shuffle=shuffle,
        random_state=random_state, stratify=y
    )

    # Scale features (important for SVMs)
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    results = []

    for c in param_grid_rbf["C"]:
        for gamma in param_grid_rbf["gamma"]:
            svm_clf = SVC(kernel="rbf", C=c, gamma=gamma)
            svm_clf.fit(X_train, y_train)
            y_pred = svm_clf.predict(X_test)
            acc = accuracy_score(y_test, y_pred)

            results.append({
                "C": c,
                "gamma": gamma,
                "accuracy": acc
            })

            print(f"C={c}, gamma={gamma} → Accuracy={acc:.4f}")

    df = pd.DataFrame(results)

    # pick best row
    best = df.loc[df["accuracy"].idxmax()]
    print("\n Best Parameters:")
    print(best)

    # Pivot for heatmap
    pivot = df.pivot(index="C", columns="gamma", values="accuracy")

    plt.figure(figsize=(8,6))
    sns.heatmap(pivot, annot=True, cmap="RdGy", fmt=".3f")
    plt.title("Gaussian SVM Accuracy Heatmap")
    plt.ylabel("C")
    plt.xlabel("Gamma")
    plt.show()
```

```
    return df, best

train_rbf_svm(X, y)
```

```
C=0.1, gamma=scale → Accuracy=0.9333
C=0.1, gamma=auto → Accuracy=0.9370
C=0.1, gamma=0.01 → Accuracy=0.9370
C=0.1, gamma=0.1 → Accuracy=0.4370
C=0.1, gamma=1 → Accuracy=0.1019
C=1, gamma=scale → Accuracy=0.9852
C=1, gamma=auto → Accuracy=0.9852
C=1, gamma=0.01 → Accuracy=0.9870
C=1, gamma=0.1 → Accuracy=0.9444
C=1, gamma=1 → Accuracy=0.1259
C=10, gamma=scale → Accuracy=0.9852
C=10, gamma=auto → Accuracy=0.9870
C=10, gamma=0.01 → Accuracy=0.9889
C=10, gamma=0.1 → Accuracy=0.9463
C=10, gamma=1 → Accuracy=0.1426
C=100, gamma=scale → Accuracy=0.9852
C=100, gamma=auto → Accuracy=0.9870
C=100, gamma=0.01 → Accuracy=0.9889
C=100, gamma=0.1 → Accuracy=0.9463
C=100, gamma=1 → Accuracy=0.1426

 Best Parameters:
C               10.0
gamma           0.01
accuracy    0.988889
Name: 12, dtype: object
```




```
(        C  gamma  accuracy
 0     0.1  scale  0.933333
 1     0.1   auto  0.937037
 2     0.1   0.01  0.937037
 3     0.1    0.1  0.437037
 4     0.1      1  0.101852
 5     1.0  scale  0.985185
 6     1.0   auto  0.985185
 7     1.0   0.01  0.987037
 8     1.0    0.1  0.944444
 9     1.0      1  0.125926
 10   10.0  scale  0.985185
 11   10.0   auto  0.987037
 12   10.0   0.01  0.988889
 13   10.0    0.1  0.946296
 14   10.0      1  0.142593
 15  100.0  scale  0.985185
 16  100.0   auto  0.987037
```

```
 17  100.0   0.01  0.988889
 18  100.0    0.1  0.946296
 19  100.0      1  0.142593,
 C               10.0
 gamma           0.01
 accuracy    0.988889
 Name: 12, dtype: object)
```

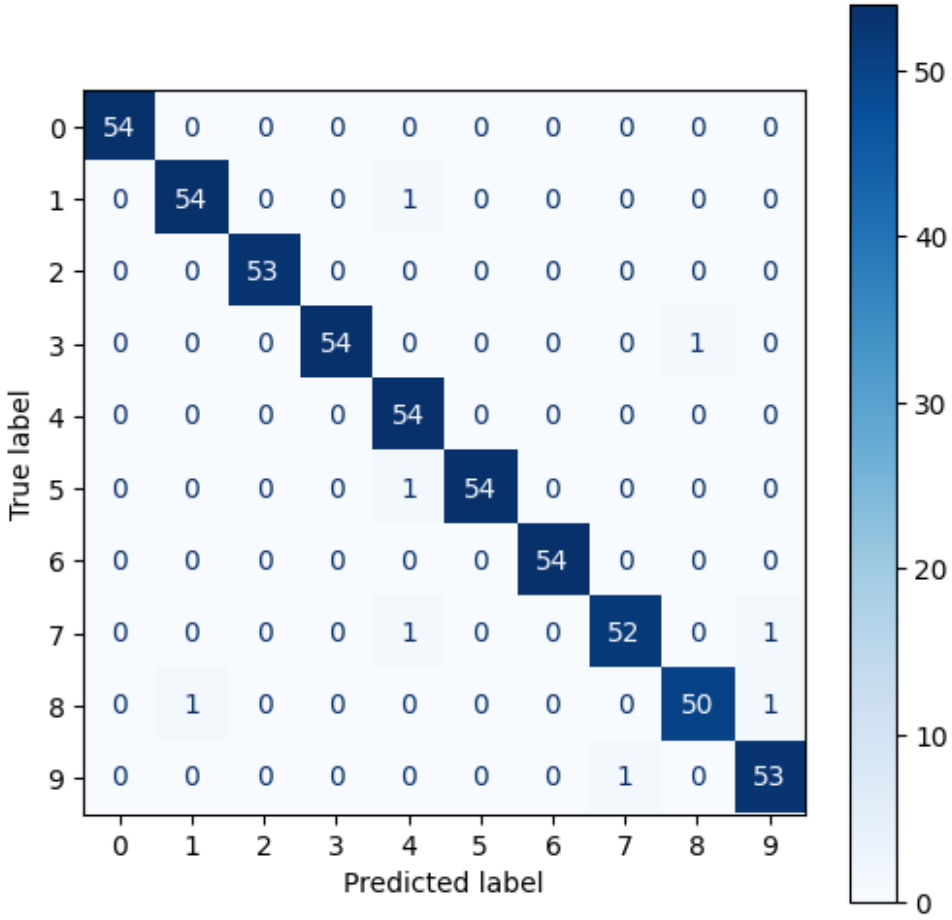
Output without Hyperparameter Tuning

```
svm_clf = SVC(kernel="rbf")

X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.30, shuffle=True, random_state=10,
stratify=y
    )

svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        54
           1       0.98      0.98      0.98        55
           2       1.00      1.00      1.00        53
           3       1.00      0.98      0.99        55
           4       0.95      1.00      0.97        54
           5       1.00      0.98      0.99        55
           6       1.00      1.00      1.00        54
           7       0.98      0.96      0.97        54
           8       0.98      0.96      0.97        52
           9       0.96      0.98      0.97        54

    accuracy                           0.99       540
   macro avg       0.99      0.99      0.99       540
weighted avg       0.99      0.99      0.99       540
```

```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(svm_clf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7a1821c99550>
```



Output with Hyperparamter Tuning

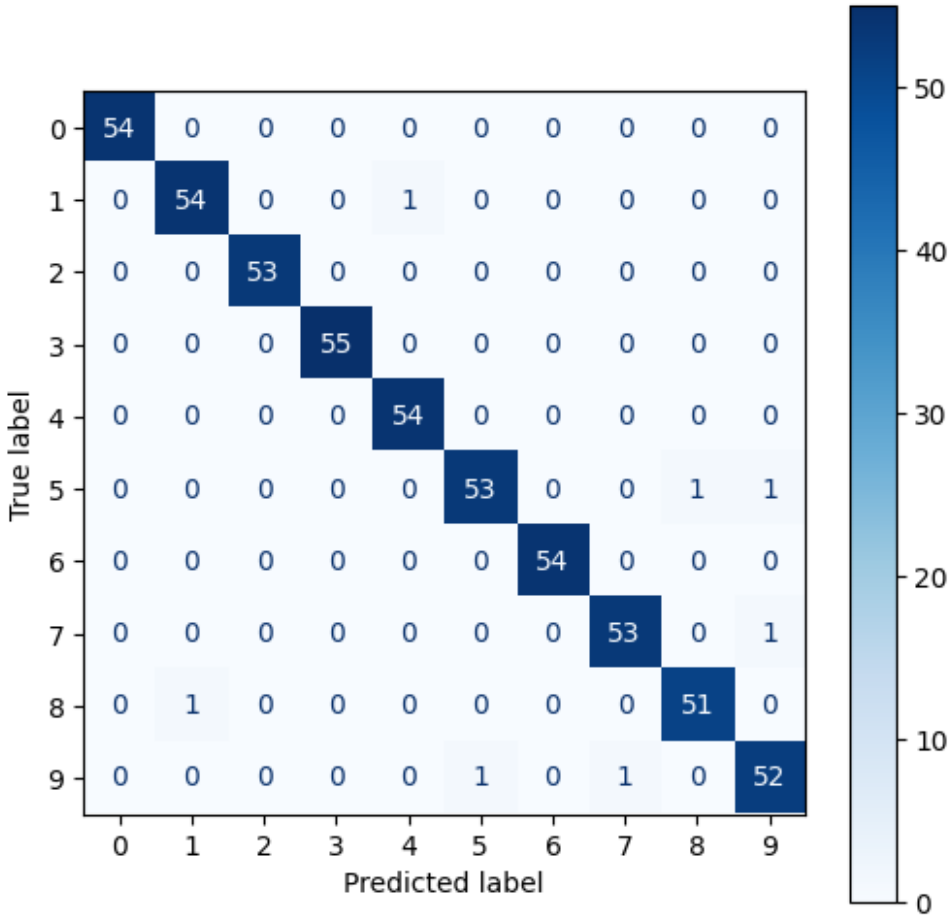
```
svm_clf = SVC(kernel="rbf", C=10, gamma=0.01)

X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.30, shuffle=True, random_state=10,
stratify=y
    )

svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        54
           1       0.98      0.98      0.98        55
           2       1.00      1.00      1.00        53
           3       1.00      1.00      1.00        55
           4       0.98      1.00      0.99        54
           5       0.98      0.96      0.97        55
```

```
           6       1.00      1.00      1.00        54
           7       0.98      0.98      0.98        54
           8       0.98      0.98      0.98        52
           9       0.96      0.96      0.96        54

    accuracy                           0.99       540
   macro avg       0.99      0.99      0.99       540
weighted avg       0.99      0.99      0.99       540
```

```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(svm_clf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7a1821cee240>
```



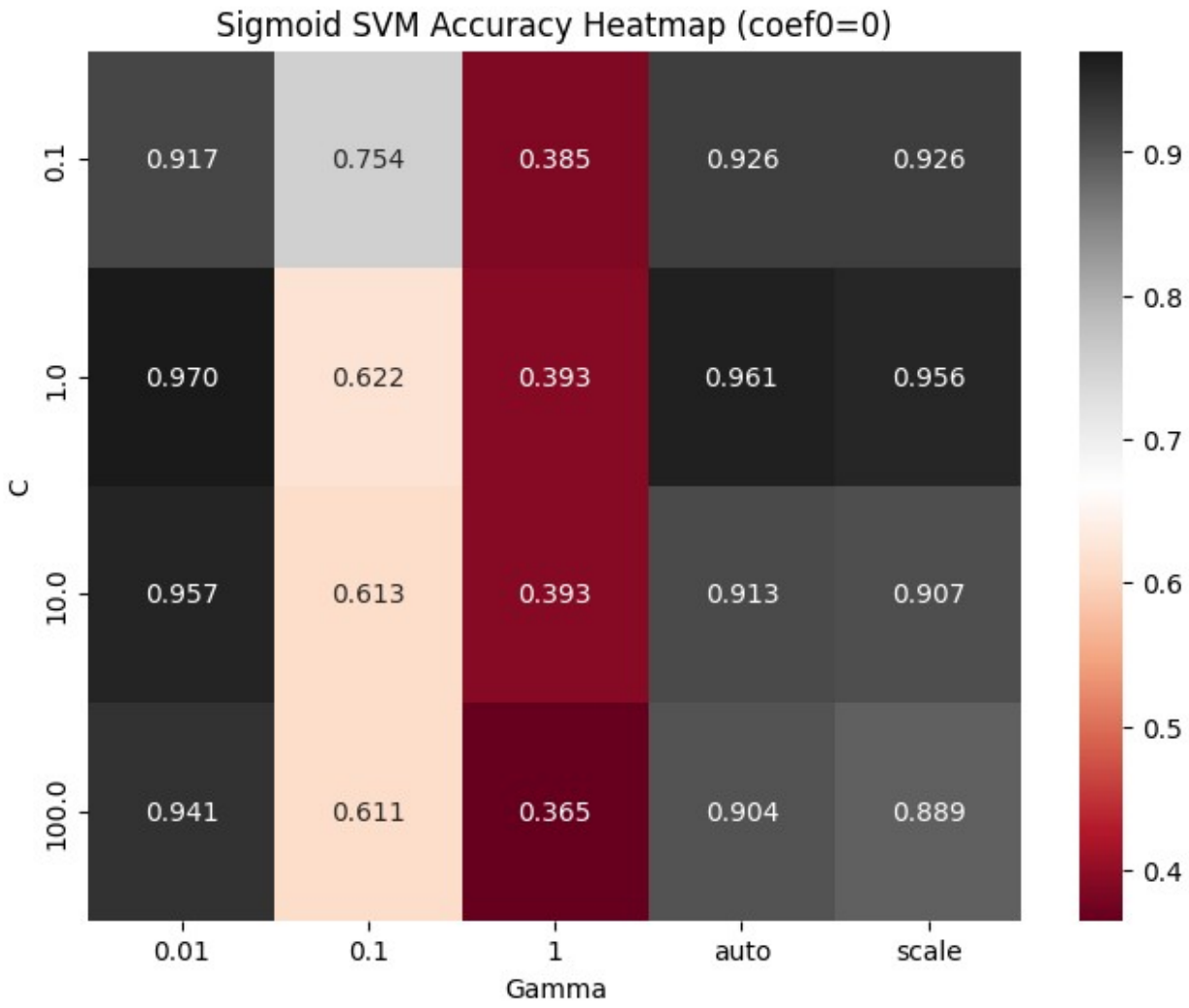
Sigmoid SVM

```
param_grid_sigmoid = {
    "C": [0.1, 1, 10, 100],
    "gamma": ["scale", "auto", 0.01, 0.1, 1],
    "coef0": [0, 1, 5]
}

def train_sigmoid_svm(X, y, split=0.30, shuffle=True,
random_state=10):
    # Train/test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=split, shuffle=shuffle,
        random_state=random_state, stratify=y
    )

    # Scale features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    results = []

    # Loop through params
    for c in param_grid_sigmoid["C"]:
        for gamma in param_grid_sigmoid["gamma"]:
            for coef in param_grid_sigmoid["coef0"]:
                svm_clf = SVC(kernel="sigmoid", C=c, gamma=gamma,
coef0=coef)
                svm_clf.fit(X_train, y_train)
                y_pred = svm_clf.predict(X_test)
                acc = accuracy_score(y_test, y_pred)

                results.append({
                    "C": c,
                    "gamma": gamma,
                    "coef0": coef,
                    "accuracy": acc
                })

                print(f"C={c}, gamma={gamma}, coef0={coef} →
Accuracy={acc:.4f}")

    df = pd.DataFrame(results)

    # Best params
    best = df.loc[df["accuracy"].idxmax()]
    print("\n Best Parameters:")
    print(best)
```

```
    # Pivot table (for fixed coef0=0 just for visualization)
    subset = df[df["coef0"]==0]
    pivot = subset.pivot(index="C", columns="gamma",
values="accuracy")

    plt.figure(figsize=(8,6))
    sns.heatmap(pivot, annot=True, cmap="RdGy", fmt=".3f")
    plt.title("Sigmoid SVM Accuracy Heatmap (coef0=0)")
    plt.ylabel("C")
    plt.xlabel("Gamma")
    plt.show()

    return df, best

train_sigmoid_svm(X, y)
```

```
C=0.1, gamma=scale, coef0=0 → Accuracy=0.9259
C=0.1, gamma=scale, coef0=1 → Accuracy=0.8778
C=0.1, gamma=scale, coef0=5 → Accuracy=0.1019
C=0.1, gamma=auto, coef0=0 → Accuracy=0.9259
C=0.1, gamma=auto, coef0=1 → Accuracy=0.8759
C=0.1, gamma=auto, coef0=5 → Accuracy=0.1019
C=0.1, gamma=0.01, coef0=0 → Accuracy=0.9167
C=0.1, gamma=0.01, coef0=1 → Accuracy=0.8111
C=0.1, gamma=0.01, coef0=5 → Accuracy=0.1019
C=0.1, gamma=0.1, coef0=0 → Accuracy=0.7537
C=0.1, gamma=0.1, coef0=1 → Accuracy=0.6296
C=0.1, gamma=0.1, coef0=5 → Accuracy=0.1019
C=0.1, gamma=1, coef0=0 → Accuracy=0.3852
C=0.1, gamma=1, coef0=1 → Accuracy=0.3389
C=0.1, gamma=1, coef0=5 → Accuracy=0.2463
C=1, gamma=scale, coef0=0 → Accuracy=0.9556
C=1, gamma=scale, coef0=1 → Accuracy=0.8926
C=1, gamma=scale, coef0=5 → Accuracy=0.1019
C=1, gamma=auto, coef0=0 → Accuracy=0.9611
C=1, gamma=auto, coef0=1 → Accuracy=0.8981
C=1, gamma=auto, coef0=5 → Accuracy=0.1019
C=1, gamma=0.01, coef0=0 → Accuracy=0.9704
C=1, gamma=0.01, coef0=1 → Accuracy=0.9278
C=1, gamma=0.01, coef0=5 → Accuracy=0.1019
C=1, gamma=0.1, coef0=0 → Accuracy=0.6222
C=1, gamma=0.1, coef0=1 → Accuracy=0.4833
C=1, gamma=0.1, coef0=5 → Accuracy=0.2111
C=1, gamma=1, coef0=0 → Accuracy=0.3926
C=1, gamma=1, coef0=1 → Accuracy=0.3056
C=1, gamma=1, coef0=5 → Accuracy=0.2648
C=10, gamma=scale, coef0=0 → Accuracy=0.9074
C=10, gamma=scale, coef0=1 → Accuracy=0.8056
C=10, gamma=scale, coef0=5 → Accuracy=0.1019
C=10, gamma=auto, coef0=0 → Accuracy=0.9130
```

```
C=10, gamma=auto, coef0=1 → Accuracy=0.8296
C=10, gamma=auto, coef0=5 → Accuracy=0.1019
C=10, gamma=0.01, coef0=0 → Accuracy=0.9574
C=10, gamma=0.01, coef0=1 → Accuracy=0.8944
C=10, gamma=0.01, coef0=5 → Accuracy=0.1019
C=10, gamma=0.1, coef0=0 → Accuracy=0.6130
C=10, gamma=0.1, coef0=1 → Accuracy=0.4296
C=10, gamma=0.1, coef0=5 → Accuracy=0.3296
C=10, gamma=1, coef0=0 → Accuracy=0.3926
C=10, gamma=1, coef0=1 → Accuracy=0.3093
C=10, gamma=1, coef0=5 → Accuracy=0.2593
C=100, gamma=scale, coef0=0 → Accuracy=0.8889
C=100, gamma=scale, coef0=1 → Accuracy=0.7926
C=100, gamma=scale, coef0=5 → Accuracy=0.6796
C=100, gamma=auto, coef0=0 → Accuracy=0.9037
C=100, gamma=auto, coef0=1 → Accuracy=0.7852
C=100, gamma=auto, coef0=5 → Accuracy=0.6667
C=100, gamma=0.01, coef0=0 → Accuracy=0.9407
C=100, gamma=0.01, coef0=1 → Accuracy=0.8593
C=100, gamma=0.01, coef0=5 → Accuracy=0.4481
C=100, gamma=0.1, coef0=0 → Accuracy=0.6111
C=100, gamma=0.1, coef0=1 → Accuracy=0.4241
C=100, gamma=0.1, coef0=5 → Accuracy=0.4426
C=100, gamma=1, coef0=0 → Accuracy=0.3648
C=100, gamma=1, coef0=1 → Accuracy=0.3148
C=100, gamma=1, coef0=5 → Accuracy=0.2574

 Best Parameters:
C               1.0
gamma          0.01
coef0             0
accuracy    0.97037
Name: 21, dtype: object
```




```
(        C  gamma  coef0  accuracy
 0     0.1  scale      0  0.925926
 1     0.1  scale      1  0.877778
 2     0.1  scale      5  0.101852
 3     0.1   auto      0  0.925926
 4     0.1   auto      1  0.875926
 5     0.1   auto      5  0.101852
 6     0.1   0.01      0  0.916667
 7     0.1   0.01      1  0.811111
 8     0.1   0.01      5  0.101852
 9     0.1    0.1      0  0.753704
 10    0.1    0.1      1  0.629630
 11    0.1    0.1      5  0.101852
 12    0.1      1      0  0.385185
 13    0.1      1      1  0.338889
 14    0.1      1      5  0.246296
 15    1.0  scale      0  0.955556
 16    1.0  scale      1  0.892593
```

```
17    1.0  scale      5  0.101852
18    1.0   auto      0  0.961111
19    1.0   auto      1  0.898148
20    1.0   auto      5  0.101852
21    1.0   0.01      0  0.970370
22    1.0   0.01      1  0.927778
23    1.0   0.01      5  0.101852
24    1.0    0.1      0  0.622222
25    1.0    0.1      1  0.483333
26    1.0    0.1      5  0.211111
27    1.0      1      0  0.392593
28    1.0      1      1  0.305556
29    1.0      1      5  0.264815
30   10.0  scale      0  0.907407
31   10.0  scale      1  0.805556
32   10.0  scale      5  0.101852
33   10.0   auto      0  0.912963
34   10.0   auto      1  0.829630
35   10.0   auto      5  0.101852
36   10.0   0.01      0  0.957407
37   10.0   0.01      1  0.894444
38   10.0   0.01      5  0.101852
39   10.0    0.1      0  0.612963
40   10.0    0.1      1  0.429630
41   10.0    0.1      5  0.329630
42   10.0      1      0  0.392593
43   10.0      1      1  0.309259
44   10.0      1      5  0.259259
45  100.0  scale      0  0.888889
46  100.0  scale      1  0.792593
47  100.0  scale      5  0.679630
48  100.0   auto      0  0.903704
49  100.0   auto      1  0.785185
50  100.0   auto      5  0.666667
51  100.0   0.01      0  0.940741
52  100.0   0.01      1  0.859259
53  100.0   0.01      5  0.448148
54  100.0    0.1      0  0.611111
55  100.0    0.1      1  0.424074
56  100.0    0.1      5  0.442593
57  100.0      1      0  0.364815
58  100.0      1      1  0.314815
59  100.0      1      5  0.257407,
C               1.0
gamma          0.01
coef0             0
accuracy    0.97037
Name: 21, dtype: object)
```

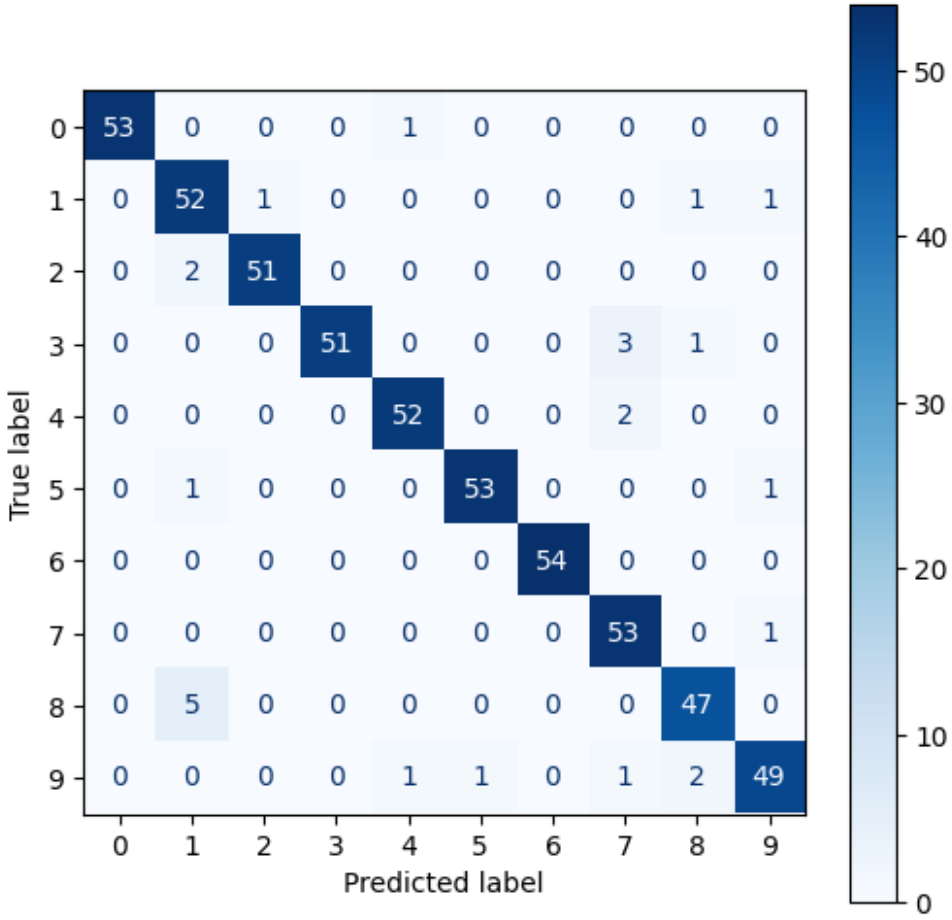
Output without Hyperparameter

```
svm_clf = SVC(kernel="sigmoid")

X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.30, shuffle=True, random_state=10,
stratify=y
    )

svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        54
           1       0.87      0.95      0.90        55
           2       0.98      0.96      0.97        53
           3       1.00      0.93      0.96        55
           4       0.96      0.96      0.96        54
           5       0.98      0.96      0.97        55
           6       1.00      1.00      1.00        54
           7       0.90      0.98      0.94        54
           8       0.92      0.90      0.91        52
           9       0.94      0.91      0.92        54

    accuracy                           0.95       540
   macro avg       0.96      0.95      0.95       540
weighted avg       0.96      0.95      0.95       540
```

```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(svm_clf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7a18156b7bf0>
```



Output with Hyperparameter Tuning

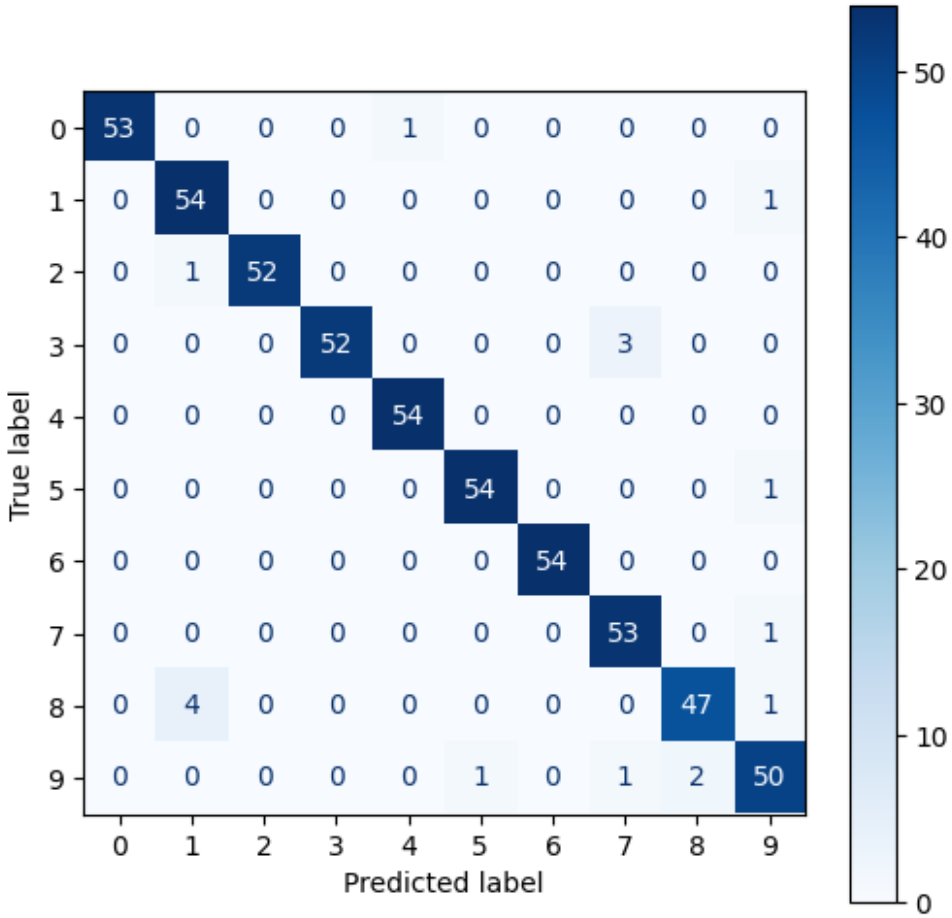
```
svm_clf = SVC(kernel="sigmoid", C=1.0, gamma=0.01, coef0=0)

X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.30, shuffle=True, random_state=10,
stratify=y
    )

svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        54
           1       0.92      0.98      0.95        55
           2       1.00      0.98      0.99        53
           3       1.00      0.95      0.97        55
           4       0.98      1.00      0.99        54
           5       0.98      0.98      0.98        55
```

```
           6       1.00      1.00      1.00        54
           7       0.93      0.98      0.95        54
           8       0.96      0.90      0.93        52
           9       0.93      0.93      0.93        54

    accuracy                           0.97       540
   macro avg       0.97      0.97      0.97       540
weighted avg       0.97      0.97      0.97       540
```

```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(svm_clf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7a18154054c0>
```



Random Forest

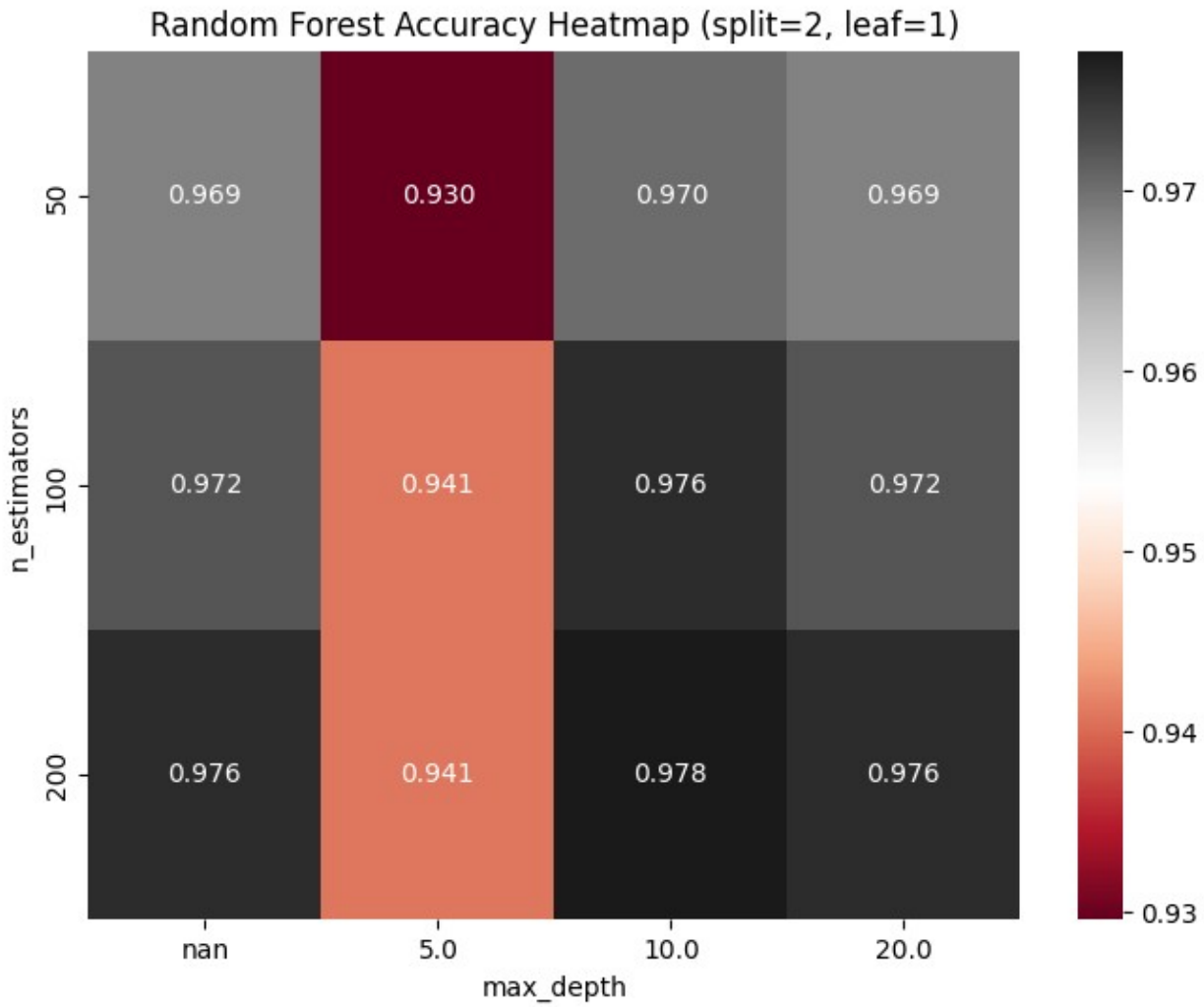
```
param_grid_rf = {
    "n_estimators": [50, 100, 200],
    "max_depth": [None, 5, 10, 20],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4]
}

def train_random_forest(X, y, split=0.30, shuffle=True,
random_state=10):
    # Train/test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=split, shuffle=shuffle,
        random_state=random_state, stratify=y
    )

    results = []

    for n in param_grid_rf["n_estimators"]:
        for depth in param_grid_rf["max_depth"]:
            for split_val in param_grid_rf["min_samples_split"]:
                for leaf_val in param_grid_rf["min_samples_leaf"]:
                    rf_clf = RandomForestClassifier(
                        n_estimators=n,
                        max_depth=depth,
                        min_samples_split=split_val,
                        min_samples_leaf=leaf_val,
                        random_state=random_state,
                        n_jobs=-1
                    )
                    rf_clf.fit(X_train, y_train)
                    y_pred = rf_clf.predict(X_test)
                    acc = accuracy_score(y_test, y_pred)

                    results.append({
                        "n_estimators": n,
                        "max_depth": depth,
                        "min_samples_split": split_val,
                        "min_samples_leaf": leaf_val,
                        "accuracy": acc
                    })

                    print(f"n={n}, depth={depth}, split={split_val},
leaf={leaf_val} → Accuracy={acc:.4f}")

    df = pd.DataFrame(results)

    # Best params
    best = df.loc[df["accuracy"].idxmax()]
```

```
    print("\n Best Parameters:")
    print(best)

    # Pivot table (for fixed min_samples_split=2, min_samples_leaf=1
just for visualization)
    subset = df[(df["min_samples_split"]==2) &
(df["min_samples_leaf"]==1)]
    pivot = subset.pivot(index="n_estimators", columns="max_depth",
values="accuracy")

    plt.figure(figsize=(8,6))
    sns.heatmap(pivot, annot=True, cmap="RdGy", fmt=".3f")
    plt.title("Random Forest Accuracy Heatmap (split=2, leaf=1)")
    plt.ylabel("n_estimators")
    plt.xlabel("max_depth")
    plt.show()

    return df, best

train_random_forest(X, y)
```

```
n=50, depth=None, split=2, leaf=1 → Accuracy=0.9685
n=50, depth=None, split=2, leaf=2 → Accuracy=0.9667
n=50, depth=None, split=2, leaf=4 → Accuracy=0.9667
n=50, depth=None, split=5, leaf=1 → Accuracy=0.9630
n=50, depth=None, split=5, leaf=2 → Accuracy=0.9667
n=50, depth=None, split=5, leaf=4 → Accuracy=0.9667
n=50, depth=None, split=10, leaf=1 → Accuracy=0.9685
n=50, depth=None, split=10, leaf=2 → Accuracy=0.9704
n=50, depth=None, split=10, leaf=4 → Accuracy=0.9574
n=50, depth=5, split=2, leaf=1 → Accuracy=0.9296
n=50, depth=5, split=2, leaf=2 → Accuracy=0.9296
n=50, depth=5, split=2, leaf=4 → Accuracy=0.9463
n=50, depth=5, split=5, leaf=1 → Accuracy=0.9389
n=50, depth=5, split=5, leaf=2 → Accuracy=0.9352
n=50, depth=5, split=5, leaf=4 → Accuracy=0.9463
n=50, depth=5, split=10, leaf=1 → Accuracy=0.9278
n=50, depth=5, split=10, leaf=2 → Accuracy=0.9315
n=50, depth=5, split=10, leaf=4 → Accuracy=0.9333
n=50, depth=10, split=2, leaf=1 → Accuracy=0.9704
n=50, depth=10, split=2, leaf=2 → Accuracy=0.9630
n=50, depth=10, split=2, leaf=4 → Accuracy=0.9630
n=50, depth=10, split=5, leaf=1 → Accuracy=0.9630
n=50, depth=10, split=5, leaf=2 → Accuracy=0.9704
n=50, depth=10, split=5, leaf=4 → Accuracy=0.9630
n=50, depth=10, split=10, leaf=1 → Accuracy=0.9704
n=50, depth=10, split=10, leaf=2 → Accuracy=0.9722
n=50, depth=10, split=10, leaf=4 → Accuracy=0.9593
n=50, depth=20, split=2, leaf=1 → Accuracy=0.9685
n=50, depth=20, split=2, leaf=2 → Accuracy=0.9667
```

```
n=50, depth=20, split=2, leaf=4 → Accuracy=0.9667
n=50, depth=20, split=5, leaf=1 → Accuracy=0.9630
n=50, depth=20, split=5, leaf=2 → Accuracy=0.9667
n=50, depth=20, split=5, leaf=4 → Accuracy=0.9667
n=50, depth=20, split=10, leaf=1 → Accuracy=0.9685
n=50, depth=20, split=10, leaf=2 → Accuracy=0.9704
n=50, depth=20, split=10, leaf=4 → Accuracy=0.9574
n=100, depth=None, split=2, leaf=1 → Accuracy=0.9722
n=100, depth=None, split=2, leaf=2 → Accuracy=0.9722
n=100, depth=None, split=2, leaf=4 → Accuracy=0.9630
n=100, depth=None, split=5, leaf=1 → Accuracy=0.9685
n=100, depth=None, split=5, leaf=2 → Accuracy=0.9648
n=100, depth=None, split=5, leaf=4 → Accuracy=0.9630
n=100, depth=None, split=10, leaf=1 → Accuracy=0.9648
n=100, depth=None, split=10, leaf=2 → Accuracy=0.9685
n=100, depth=None, split=10, leaf=4 → Accuracy=0.9630
n=100, depth=5, split=2, leaf=1 → Accuracy=0.9407
n=100, depth=5, split=2, leaf=2 → Accuracy=0.9352
n=100, depth=5, split=2, leaf=4 → Accuracy=0.9370
n=100, depth=5, split=5, leaf=1 → Accuracy=0.9389
n=100, depth=5, split=5, leaf=2 → Accuracy=0.9370
n=100, depth=5, split=5, leaf=4 → Accuracy=0.9370
n=100, depth=5, split=10, leaf=1 → Accuracy=0.9352
n=100, depth=5, split=10, leaf=2 → Accuracy=0.9296
n=100, depth=5, split=10, leaf=4 → Accuracy=0.9352
n=100, depth=10, split=2, leaf=1 → Accuracy=0.9759
n=100, depth=10, split=2, leaf=2 → Accuracy=0.9685
n=100, depth=10, split=2, leaf=4 → Accuracy=0.9593
n=100, depth=10, split=5, leaf=1 → Accuracy=0.9648
n=100, depth=10, split=5, leaf=2 → Accuracy=0.9648
n=100, depth=10, split=5, leaf=4 → Accuracy=0.9593
n=100, depth=10, split=10, leaf=1 → Accuracy=0.9648
n=100, depth=10, split=10, leaf=2 → Accuracy=0.9704
n=100, depth=10, split=10, leaf=4 → Accuracy=0.9630
n=100, depth=20, split=2, leaf=1 → Accuracy=0.9722
n=100, depth=20, split=2, leaf=2 → Accuracy=0.9722
n=100, depth=20, split=2, leaf=4 → Accuracy=0.9630
n=100, depth=20, split=5, leaf=1 → Accuracy=0.9685
n=100, depth=20, split=5, leaf=2 → Accuracy=0.9648
n=100, depth=20, split=5, leaf=4 → Accuracy=0.9630
n=100, depth=20, split=10, leaf=1 → Accuracy=0.9648
n=100, depth=20, split=10, leaf=2 → Accuracy=0.9685
n=100, depth=20, split=10, leaf=4 → Accuracy=0.9630
n=200, depth=None, split=2, leaf=1 → Accuracy=0.9759
n=200, depth=None, split=2, leaf=2 → Accuracy=0.9722
n=200, depth=None, split=2, leaf=4 → Accuracy=0.9630
n=200, depth=None, split=5, leaf=1 → Accuracy=0.9741
n=200, depth=None, split=5, leaf=2 → Accuracy=0.9704
n=200, depth=None, split=5, leaf=4 → Accuracy=0.9630
```

```
n=200, depth=None, split=10, leaf=1 → Accuracy=0.9685
n=200, depth=None, split=10, leaf=2 → Accuracy=0.9704
n=200, depth=None, split=10, leaf=4 → Accuracy=0.9667
n=200, depth=5, split=2, leaf=1 → Accuracy=0.9407
n=200, depth=5, split=2, leaf=2 → Accuracy=0.9444
n=200, depth=5, split=2, leaf=4 → Accuracy=0.9407
n=200, depth=5, split=5, leaf=1 → Accuracy=0.9426
n=200, depth=5, split=5, leaf=2 → Accuracy=0.9389
n=200, depth=5, split=5, leaf=4 → Accuracy=0.9407
n=200, depth=5, split=10, leaf=1 → Accuracy=0.9370
n=200, depth=5, split=10, leaf=2 → Accuracy=0.9352
n=200, depth=5, split=10, leaf=4 → Accuracy=0.9370
n=200, depth=10, split=2, leaf=1 → Accuracy=0.9778
n=200, depth=10, split=2, leaf=2 → Accuracy=0.9722
n=200, depth=10, split=2, leaf=4 → Accuracy=0.9611
n=200, depth=10, split=5, leaf=1 → Accuracy=0.9685
n=200, depth=10, split=5, leaf=2 → Accuracy=0.9648
n=200, depth=10, split=5, leaf=4 → Accuracy=0.9611
n=200, depth=10, split=10, leaf=1 → Accuracy=0.9648
n=200, depth=10, split=10, leaf=2 → Accuracy=0.9667
n=200, depth=10, split=10, leaf=4 → Accuracy=0.9648
n=200, depth=20, split=2, leaf=1 → Accuracy=0.9759
n=200, depth=20, split=2, leaf=2 → Accuracy=0.9722
n=200, depth=20, split=2, leaf=4 → Accuracy=0.9630
n=200, depth=20, split=5, leaf=1 → Accuracy=0.9741
n=200, depth=20, split=5, leaf=2 → Accuracy=0.9704
n=200, depth=20, split=5, leaf=4 → Accuracy=0.9630
n=200, depth=20, split=10, leaf=1 → Accuracy=0.9685
n=200, depth=20, split=10, leaf=2 → Accuracy=0.9704
n=200, depth=20, split=10, leaf=4 → Accuracy=0.9667

 Best Parameters:
n_estimators         200.000000
max_depth             10.000000
min_samples_split      2.000000
min_samples_leaf       1.000000
accuracy               0.977778
Name: 90, dtype: float64
```




```
(     n_estimators  max_depth  min_samples_split  min_samples_leaf
accuracy
 0              50        NaN                  2                 1
0.968519
 1              50        NaN                  2                 2
0.966667
 2              50        NaN                  2                 4
0.966667
 3              50        NaN                  5                 1
0.962963
 4              50        NaN                  5                 2
0.966667
 ..            ...        ...                ...               ...
...
 103           200       20.0                  5                 2
0.970370
 104           200       20.0                  5                 4
0.962963
```

```
 105           200       20.0                  10                   1
0.968519
 106           200       20.0                  10                   2
0.970370
 107           200       20.0                  10                   4
0.966667

 [108 rows x 5 columns],
 n_estimators         200.000000
 max_depth             10.000000
 min_samples_split      2.000000
 min_samples_leaf       1.000000
 accuracy               0.977778
 Name: 90, dtype: float64)
```

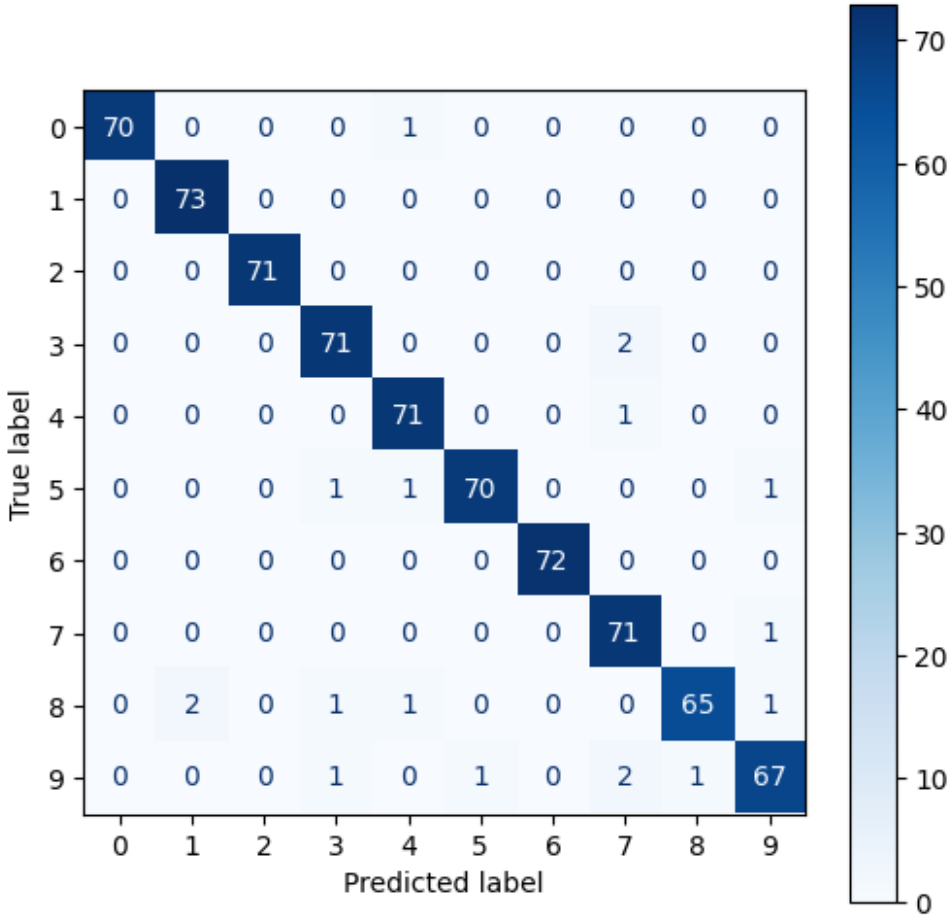
Output without Hyperparameter Tuning

```
X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.40, shuffle=True, random_state=10,
stratify=y
    )

rf_clf = RandomForestClassifier()

rf_clf.fit(X_train, y_train)

rf_clf.fit(X_train, y_train)
y_pred = rf_clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99        71
           1       0.97      1.00      0.99        73
           2       1.00      1.00      1.00        71
           3       0.96      0.97      0.97        73
           4       0.96      0.99      0.97        72
           5       0.99      0.96      0.97        73
           6       1.00      1.00      1.00        72
           7       0.93      0.99      0.96        72
           8       0.98      0.93      0.96        70
           9       0.96      0.93      0.94        72

    accuracy                           0.97       719
   macro avg       0.98      0.97      0.97       719
weighted avg       0.98      0.97      0.97       719
```

```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(rf_clf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7a181e0764b0>
```



Output with Hyperparameter Tuning

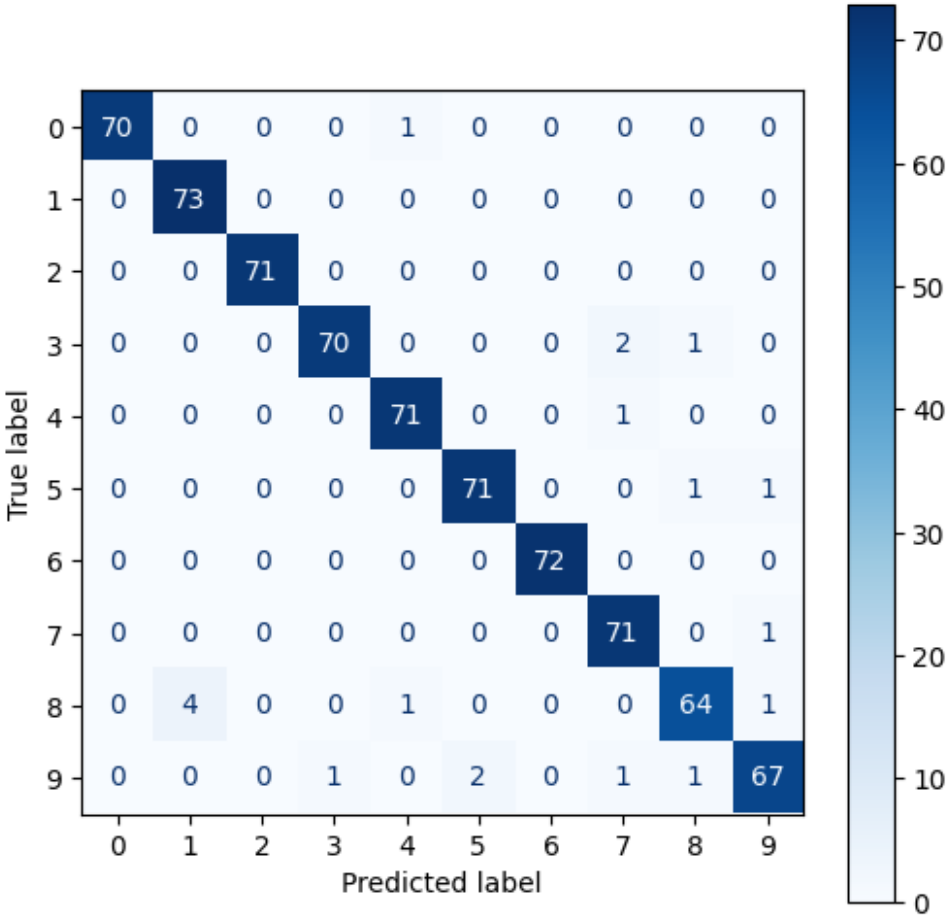
```
X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.40, shuffle=True, random_state=10,
stratify=y
    )

rf_clf = RandomForestClassifier(n_estimators=200, max_depth=10,
min_samples_split=2, min_samples_leaf=1)

rf_clf.fit(X_train, y_train)

rf_clf.fit(X_train, y_train)
y_pred = rf_clf.predict(X_test)
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99        71
           1       0.95      1.00      0.97        73
           2       1.00      1.00      1.00        71
           3       0.99      0.96      0.97        73
           4       0.97      0.99      0.98        72
           5       0.97      0.97      0.97        73
           6       1.00      1.00      1.00        72
           7       0.95      0.99      0.97        72
           8       0.96      0.91      0.93        70
           9       0.96      0.93      0.94        72

    accuracy                           0.97       719
   macro avg       0.97      0.97      0.97       719
weighted avg       0.97      0.97      0.97       719
```

```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(rf_clf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7a181f93bd70>
```



Principal Component Analysis on Handwritten Digits Dataset

```
from sklearn.decomposition import PCA
```

```
len(X)
```

```
1797
```

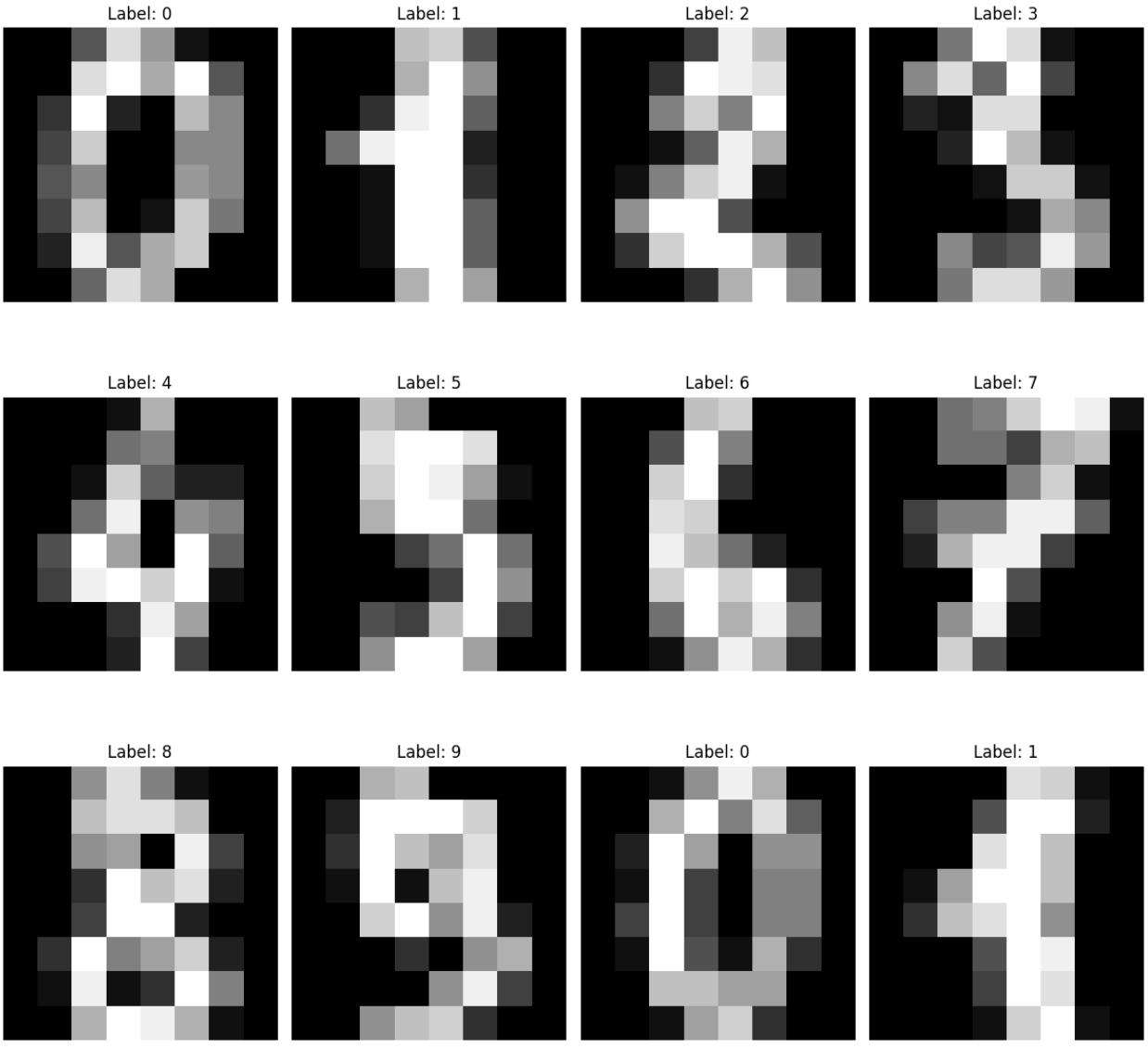
```
len(X[0])
```

```
64
```

So, there are 1797 samples and 64 features (8 X 8 flattened to 64).

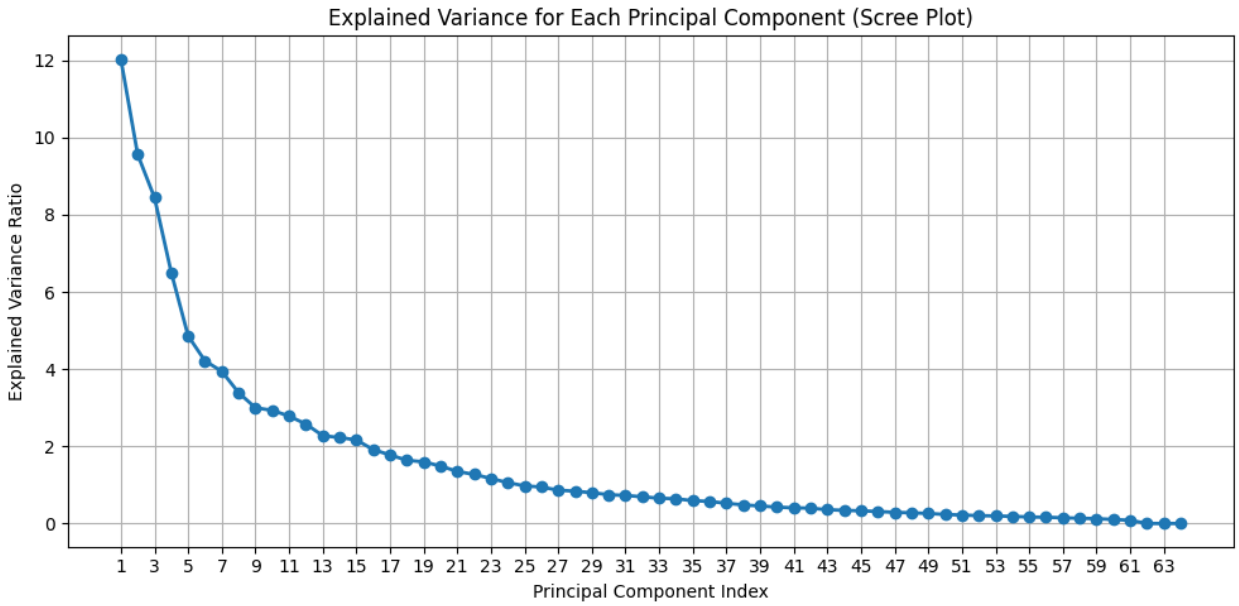
Plotting the first few images

```
plt.figure(figsize=(12, 12))

for i in range(12):
  plt.subplot(3, 4, i + 1)
  plt.imshow(digits.images[i], cmap='gray')
  plt.title(f"Label: {digits.target[i]}")
  plt.axis('off')

plt.tight_layout()
plt.show()
```



Above are the first 12 samples

```
X_scaled = scaler.fit_transform(X)

pca = PCA(n_components=None)
X_pca = pca.fit_transform(X_scaled)

explained_variance = pca.explained_variance_ratio_
explained_variance *= 100

plt.figure(figsize=(10, 5))
plt.plot(np.arange(1, len(explained_variance) + 1),
explained_variance, marker='o', linewidth=2)
plt.title("Explained Variance for Each Principal Component (Scree
Plot)")
plt.xlabel("Principal Component Index")
plt.ylabel("Explained Variance Ratio")
plt.xticks(np.arange(1, len(explained_variance) + 1, step=2))
plt.grid(True)
plt.tight_layout()
plt.show()
```

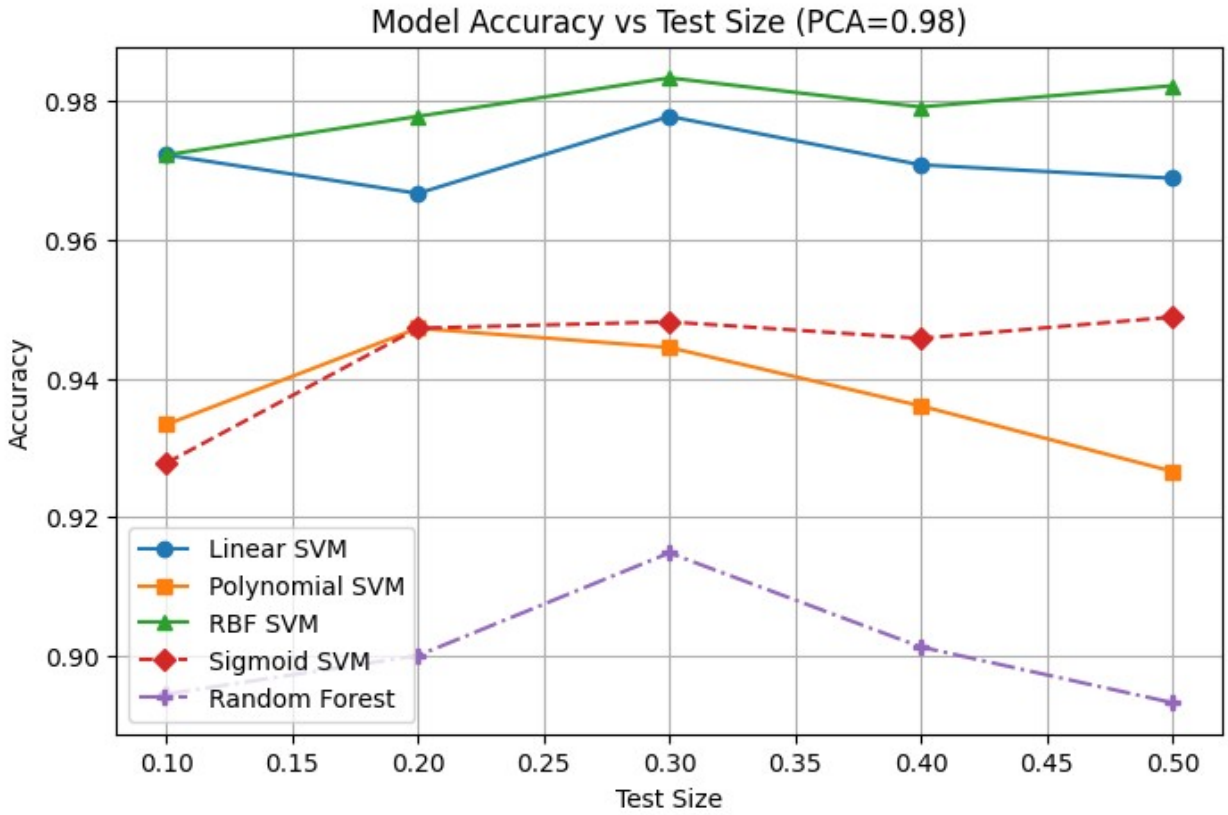



```
sum = 0

for i, var in enumerate(explained_variance[:10], start=1):
    sum += var
    print(f"PC{i}: {var:.4f} ({var*100:.2f}% of total variance)")

    if sum >= 98.0:
      break
```

```
PC1: 12.0339 (1203.39% of total variance)
PC2: 9.5611 (956.11% of total variance)
PC3: 8.4444 (844.44% of total variance)
PC4: 6.4984 (649.84% of total variance)
PC5: 4.8602 (486.02% of total variance)
PC6: 4.2141 (421.41% of total variance)
PC7: 3.9421 (394.21% of total variance)
PC8: 3.3894 (338.94% of total variance)
PC9: 2.9982 (299.82% of total variance)
PC10: 2.9320 (293.20% of total variance)
```

Since, it is an image data, we are taking the minimum number of principal components, which cover 98% of variance. It is 10 in this case.

```
pca = PCA(n_components=10)
X_pca = pca.fit_transform(X_scaled)

analyse_train_test_split(X, y, pca_n=0.98)
```



Conclusion:

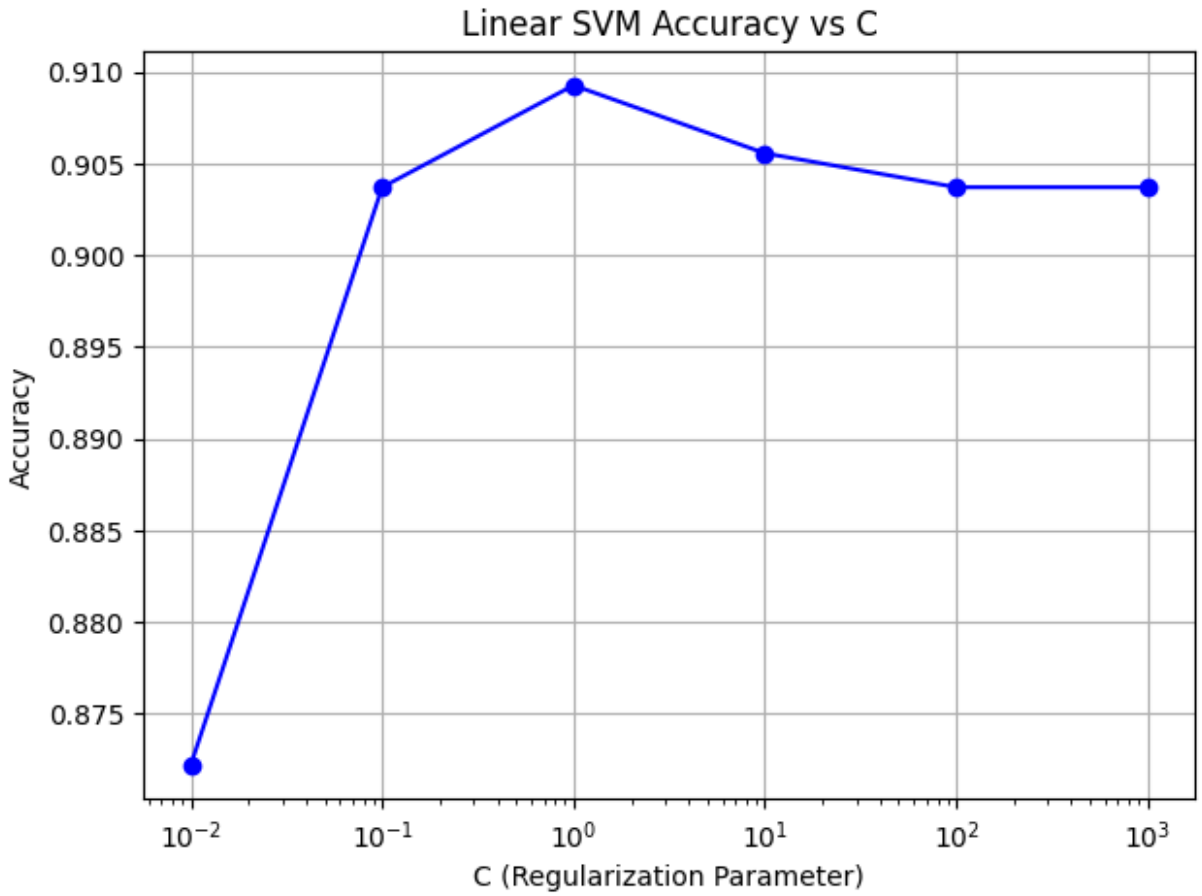
All the classifiers work best at 30% split

Linear SVM

```
def train_linear_svm(X, y, split=0.30, shuffle=True, random_state=10):
    # Train/test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=split, shuffle=shuffle,
random_state=random_state, stratify=y
    )

    # Scale features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # C values
    c_params = [0.01, 0.1, 1, 10, 100, 1000]
    linear_svm_acc = []

    # Train SVM for each C
    for c in c_params:
        svm_clf = SVC(kernel="linear", C=c)
        svm_clf.fit(X_train, y_train)
        y_pred = svm_clf.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        linear_svm_acc.append(acc)
        print(f"C={c}: Accuracy={acc:.4f}")

    # Plot results
    plt.figure(figsize=(7,5))
    plt.plot(c_params, linear_svm_acc, marker="o", linestyle="-",
color="b")
    plt.xscale("log")  # better visualization (log scale for C)
    plt.xlabel("C (Regularization Parameter)")
    plt.ylabel("Accuracy")
    plt.title("Linear SVM Accuracy vs C")
    plt.grid(True)
    plt.show()

    return c_params, linear_svm_acc

train_linear_svm(X_pca, y)
```

```
C=0.01: Accuracy=0.8722
C=0.1: Accuracy=0.9037
```

```
C=1: Accuracy=0.9093
C=10: Accuracy=0.9056
C=100: Accuracy=0.9037
C=1000: Accuracy=0.9037
```




```
([0.01, 0.1, 1, 10, 100, 1000],
 [0.8722222222222222,
  0.9037037037037037,
  0.9092592592592592,
  0.9055555555555556,
  0.9037037037037037,
  0.9037037037037037])
```

Output without hyperparameter tuning

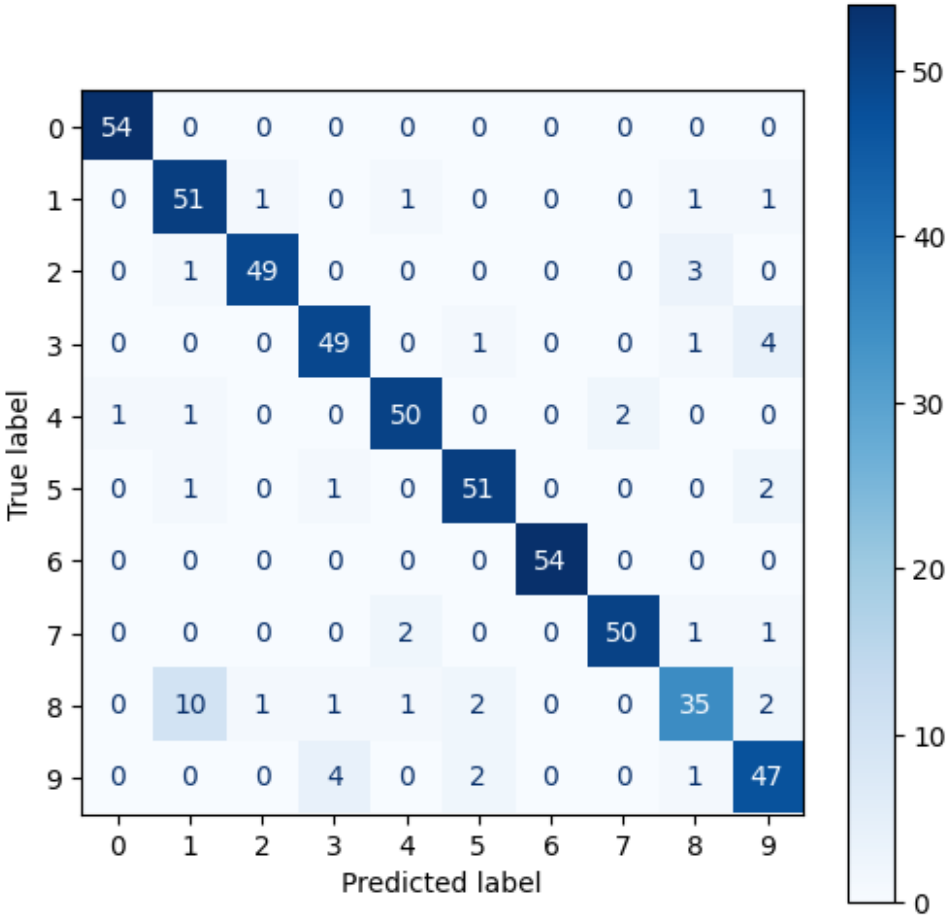
```
svm_clf = SVC(kernel="linear")

X_train, X_test, y_train, y_test = train_test_split(
        X_pca, y, test_size=0.30, shuffle=True, random_state=10,
stratify=y
    )
```

```
svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        54
           1       0.80      0.93      0.86        55
           2       0.96      0.92      0.94        53
           3       0.89      0.89      0.89        55
           4       0.93      0.93      0.93        54
           5       0.91      0.93      0.92        55
           6       1.00      1.00      1.00        54
           7       0.96      0.93      0.94        54
           8       0.83      0.67      0.74        52
           9       0.82      0.87      0.85        54

    accuracy                           0.91       540
   macro avg       0.91      0.91      0.91       540
weighted avg       0.91      0.91      0.91       540
```

```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(svm_clf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7a1815405220>
```



Output with Hyperparameter Tuning

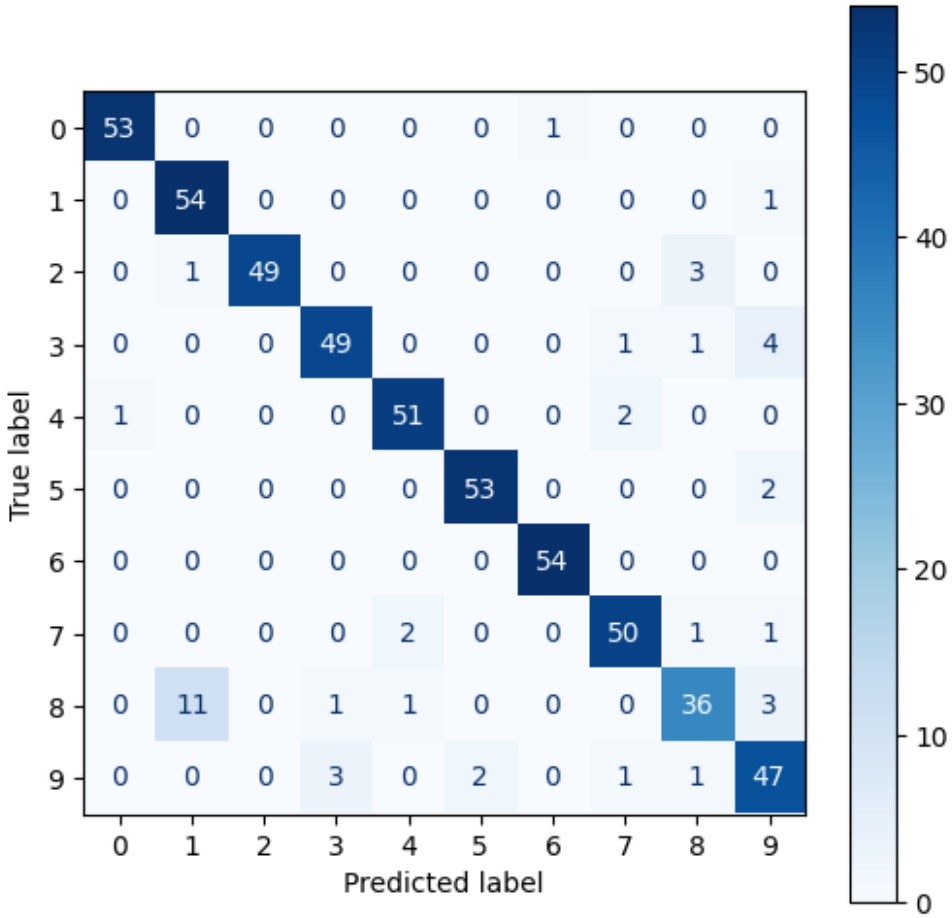
```
svm_clf = SVC(kernel="linear", C=0.1)

X_train, X_test, y_train, y_test = train_test_split(
        X_pca, y, test_size=0.30, shuffle=True, random_state=10,
stratify=y
    )

svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        54
           1       0.82      0.98      0.89        55
           2       1.00      0.92      0.96        53
           3       0.92      0.89      0.91        55
           4       0.94      0.94      0.94        54
           5       0.96      0.96      0.96        55
```

```
           6       0.98      1.00      0.99        54
           7       0.93      0.93      0.93        54
           8       0.86      0.69      0.77        52
           9       0.81      0.87      0.84        54

    accuracy                           0.92       540
   macro avg       0.92      0.92      0.92       540
weighted avg       0.92      0.92      0.92       540
```

```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(svm_clf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7a18143709e0>
```



Polynomial SVM

```
param_grid = {
    "C": [0.1, 1, 10, 100],
```

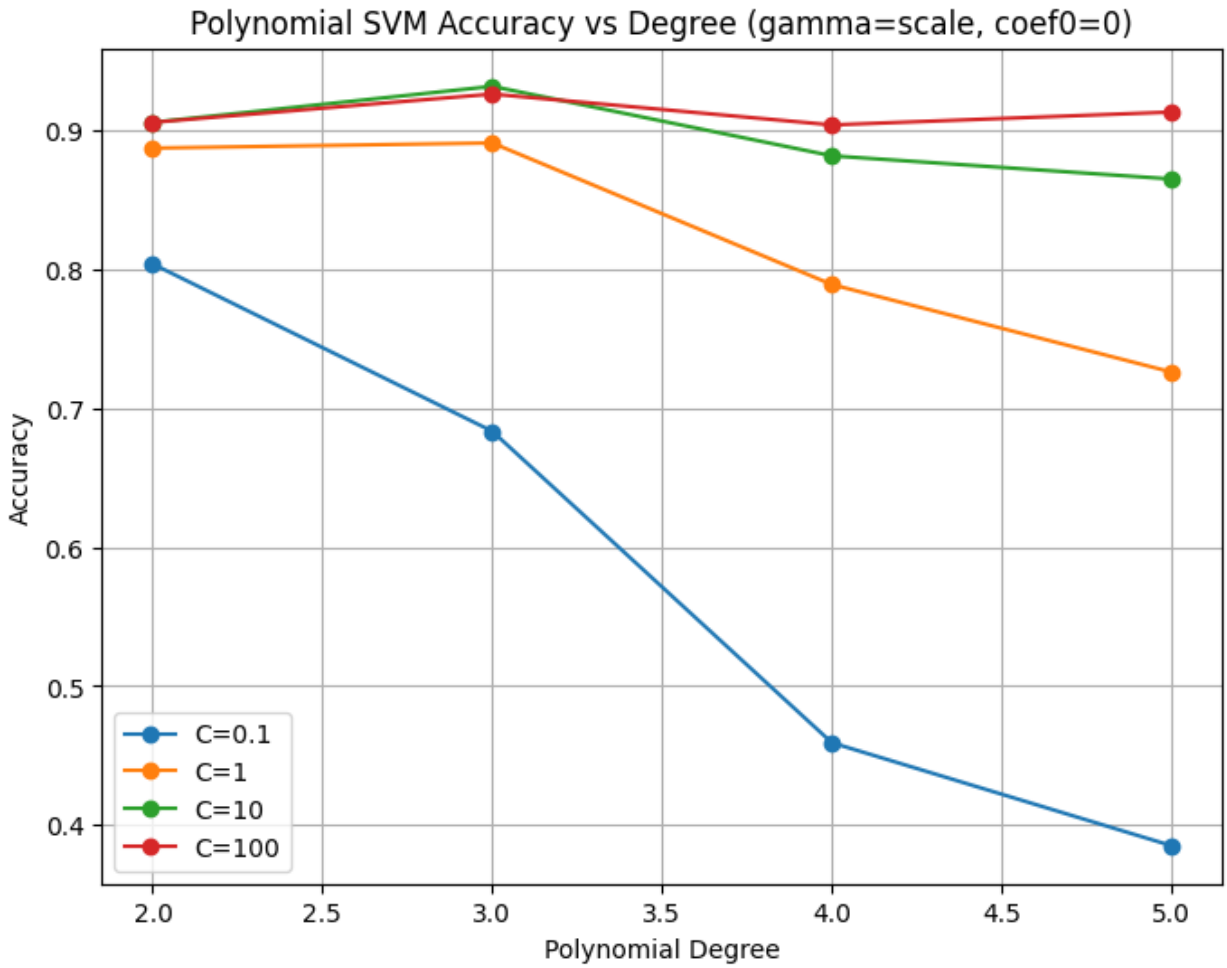
```
    "degree": [2, 3, 4, 5],
    "gamma": ["scale", "auto"],
    "coef0": [0, 1, 2]
}

def train_polynomial_svm(X, y, split=0.30, shuffle=True,
random_state=10):
    # Train/test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=split, shuffle=shuffle,
random_state=random_state, stratify=y
    )

    # Scale features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    results = []

    for c in param_grid["C"]:
        for degree in param_grid["degree"]:
            for gamma in param_grid["gamma"]:
                for coef in param_grid["coef0"]:

                    svm_clf = SVC(kernel="poly", C=c, degree=degree,
gamma=gamma, coef0=coef)
                    svm_clf.fit(X_train, y_train)
                    y_pred = svm_clf.predict(X_test)
                    acc = accuracy_score(y_test, y_pred)

                    results.append({
                        "C": c,
                        "degree": degree,
                        "gamma": gamma,
                        "coef0": coef,
                        "accuracy": acc
                    })

                    print(f"C={c}, degree={degree}, gamma={gamma},
coef0={coef} → Accuracy={acc:.4f}")

    df = pd.DataFrame(results)

    best = df.loc[df["accuracy"].idxmax()]
    print("\n Best Parameters:")
    print(best)

    plt.figure(figsize=(8,6))
    for c in param_grid["C"]:
```

```
        subset = df[(df["C"]==c) & (df["gamma"]=="scale") &
(df["coef0"]==0)]
        plt.plot(subset["degree"], subset["accuracy"], marker="o",
label=f"C={c}")

    plt.xlabel("Polynomial Degree")
    plt.ylabel("Accuracy")
    plt.title("Polynomial SVM Accuracy vs Degree (gamma=scale,
coef0=0)")
    plt.legend()
    plt.grid(True)
    plt.show()

    return df, best

best = train_polynomial_svm(X_pca, y)
```

```
C=0.1, degree=2, gamma=scale, coef0=0 → Accuracy=0.8037
C=0.1, degree=2, gamma=scale, coef0=1 → Accuracy=0.9093
C=0.1, degree=2, gamma=scale, coef0=2 → Accuracy=0.9074
C=0.1, degree=2, gamma=auto, coef0=0 → Accuracy=0.8037
C=0.1, degree=2, gamma=auto, coef0=1 → Accuracy=0.9093
C=0.1, degree=2, gamma=auto, coef0=2 → Accuracy=0.9074
C=0.1, degree=3, gamma=scale, coef0=0 → Accuracy=0.6833
C=0.1, degree=3, gamma=scale, coef0=1 → Accuracy=0.9241
C=0.1, degree=3, gamma=scale, coef0=2 → Accuracy=0.9352
C=0.1, degree=3, gamma=auto, coef0=0 → Accuracy=0.6833
C=0.1, degree=3, gamma=auto, coef0=1 → Accuracy=0.9241
C=0.1, degree=3, gamma=auto, coef0=2 → Accuracy=0.9352
C=0.1, degree=4, gamma=scale, coef0=0 → Accuracy=0.4593
C=0.1, degree=4, gamma=scale, coef0=1 → Accuracy=0.9370
C=0.1, degree=4, gamma=scale, coef0=2 → Accuracy=0.9444
C=0.1, degree=4, gamma=auto, coef0=0 → Accuracy=0.4593
C=0.1, degree=4, gamma=auto, coef0=1 → Accuracy=0.9370
C=0.1, degree=4, gamma=auto, coef0=2 → Accuracy=0.9444
C=0.1, degree=5, gamma=scale, coef0=0 → Accuracy=0.3852
C=0.1, degree=5, gamma=scale, coef0=1 → Accuracy=0.9500
C=0.1, degree=5, gamma=scale, coef0=2 → Accuracy=0.9463
C=0.1, degree=5, gamma=auto, coef0=0 → Accuracy=0.3852
C=0.1, degree=5, gamma=auto, coef0=1 → Accuracy=0.9500
C=0.1, degree=5, gamma=auto, coef0=2 → Accuracy=0.9463
C=1, degree=2, gamma=scale, coef0=0 → Accuracy=0.8870
C=1, degree=2, gamma=scale, coef0=1 → Accuracy=0.9444
C=1, degree=2, gamma=scale, coef0=2 → Accuracy=0.9444
C=1, degree=2, gamma=auto, coef0=0 → Accuracy=0.8870
C=1, degree=2, gamma=auto, coef0=1 → Accuracy=0.9444
C=1, degree=2, gamma=auto, coef0=2 → Accuracy=0.9444
C=1, degree=3, gamma=scale, coef0=0 → Accuracy=0.8907
C=1, degree=3, gamma=scale, coef0=1 → Accuracy=0.9389
C=1, degree=3, gamma=scale, coef0=2 → Accuracy=0.9407
```

```
C=1, degree=3, gamma=auto, coef0=0 → Accuracy=0.8907
C=1, degree=3, gamma=auto, coef0=1 → Accuracy=0.9389
C=1, degree=3, gamma=auto, coef0=2 → Accuracy=0.9407
C=1, degree=4, gamma=scale, coef0=0 → Accuracy=0.7889
C=1, degree=4, gamma=scale, coef0=1 → Accuracy=0.9444
C=1, degree=4, gamma=scale, coef0=2 → Accuracy=0.9444
C=1, degree=4, gamma=auto, coef0=0 → Accuracy=0.7889
C=1, degree=4, gamma=auto, coef0=1 → Accuracy=0.9444
C=1, degree=4, gamma=auto, coef0=2 → Accuracy=0.9444
C=1, degree=5, gamma=scale, coef0=0 → Accuracy=0.7259
C=1, degree=5, gamma=scale, coef0=1 → Accuracy=0.9407
C=1, degree=5, gamma=scale, coef0=2 → Accuracy=0.9444
C=1, degree=5, gamma=auto, coef0=0 → Accuracy=0.7259
C=1, degree=5, gamma=auto, coef0=1 → Accuracy=0.9407
C=1, degree=5, gamma=auto, coef0=2 → Accuracy=0.9444
C=10, degree=2, gamma=scale, coef0=0 → Accuracy=0.9056
C=10, degree=2, gamma=scale, coef0=1 → Accuracy=0.9444
C=10, degree=2, gamma=scale, coef0=2 → Accuracy=0.9426
C=10, degree=2, gamma=auto, coef0=0 → Accuracy=0.9056
C=10, degree=2, gamma=auto, coef0=1 → Accuracy=0.9444
C=10, degree=2, gamma=auto, coef0=2 → Accuracy=0.9426
C=10, degree=3, gamma=scale, coef0=0 → Accuracy=0.9315
C=10, degree=3, gamma=scale, coef0=1 → Accuracy=0.9426
C=10, degree=3, gamma=scale, coef0=2 → Accuracy=0.9444
C=10, degree=3, gamma=auto, coef0=0 → Accuracy=0.9315
C=10, degree=3, gamma=auto, coef0=1 → Accuracy=0.9426
C=10, degree=3, gamma=auto, coef0=2 → Accuracy=0.9444
C=10, degree=4, gamma=scale, coef0=0 → Accuracy=0.8815
C=10, degree=4, gamma=scale, coef0=1 → Accuracy=0.9444
C=10, degree=4, gamma=scale, coef0=2 → Accuracy=0.9426
C=10, degree=4, gamma=auto, coef0=0 → Accuracy=0.8815
C=10, degree=4, gamma=auto, coef0=1 → Accuracy=0.9444
C=10, degree=4, gamma=auto, coef0=2 → Accuracy=0.9426
C=10, degree=5, gamma=scale, coef0=0 → Accuracy=0.8648
C=10, degree=5, gamma=scale, coef0=1 → Accuracy=0.9426
C=10, degree=5, gamma=scale, coef0=2 → Accuracy=0.9407
C=10, degree=5, gamma=auto, coef0=0 → Accuracy=0.8648
C=10, degree=5, gamma=auto, coef0=1 → Accuracy=0.9426
C=10, degree=5, gamma=auto, coef0=2 → Accuracy=0.9407
C=100, degree=2, gamma=scale, coef0=0 → Accuracy=0.9056
C=100, degree=2, gamma=scale, coef0=1 → Accuracy=0.9463
C=100, degree=2, gamma=scale, coef0=2 → Accuracy=0.9426
C=100, degree=2, gamma=auto, coef0=0 → Accuracy=0.9056
C=100, degree=2, gamma=auto, coef0=1 → Accuracy=0.9463
C=100, degree=2, gamma=auto, coef0=2 → Accuracy=0.9426
C=100, degree=3, gamma=scale, coef0=0 → Accuracy=0.9259
C=100, degree=3, gamma=scale, coef0=1 → Accuracy=0.9389
C=100, degree=3, gamma=scale, coef0=2 → Accuracy=0.9389
C=100, degree=3, gamma=auto, coef0=0 → Accuracy=0.9259
```

```
C=100, degree=3, gamma=auto, coef0=1 → Accuracy=0.9389
C=100, degree=3, gamma=auto, coef0=2 → Accuracy=0.9389
C=100, degree=4, gamma=scale, coef0=0 → Accuracy=0.9037
C=100, degree=4, gamma=scale, coef0=1 → Accuracy=0.9389
C=100, degree=4, gamma=scale, coef0=2 → Accuracy=0.9407
C=100, degree=4, gamma=auto, coef0=0 → Accuracy=0.9037
C=100, degree=4, gamma=auto, coef0=1 → Accuracy=0.9389
C=100, degree=4, gamma=auto, coef0=2 → Accuracy=0.9407
C=100, degree=5, gamma=scale, coef0=0 → Accuracy=0.9130
C=100, degree=5, gamma=scale, coef0=1 → Accuracy=0.9407
C=100, degree=5, gamma=scale, coef0=2 → Accuracy=0.9407
C=100, degree=5, gamma=auto, coef0=0 → Accuracy=0.9130
C=100, degree=5, gamma=auto, coef0=1 → Accuracy=0.9407
C=100, degree=5, gamma=auto, coef0=2 → Accuracy=0.9407

 Best Parameters:
C             0.1
degree          5
gamma       scale
coef0           1
accuracy     0.95
Name: 19, dtype: object
```




```
best

(         C  degree  gamma  coef0  accuracy
 0      0.1       2  scale      0  0.803704
 1      0.1       2  scale      1  0.909259
 2      0.1       2  scale      2  0.907407
 3      0.1       2   auto      0  0.803704
 4      0.1       2   auto      1  0.909259
 ..     ...     ...    ...    ...       ...
 91   100.0       5  scale      1  0.940741
 92   100.0       5  scale      2  0.940741
 93   100.0       5   auto      0  0.912963
 94   100.0       5   auto      1  0.940741
 95   100.0       5   auto      2  0.940741

 [96 rows x 5 columns],
 C             0.1
 degree          5
 gamma       scale
 coef0           1
```

```
accuracy     0.95
Name: 19, dtype: object)
```

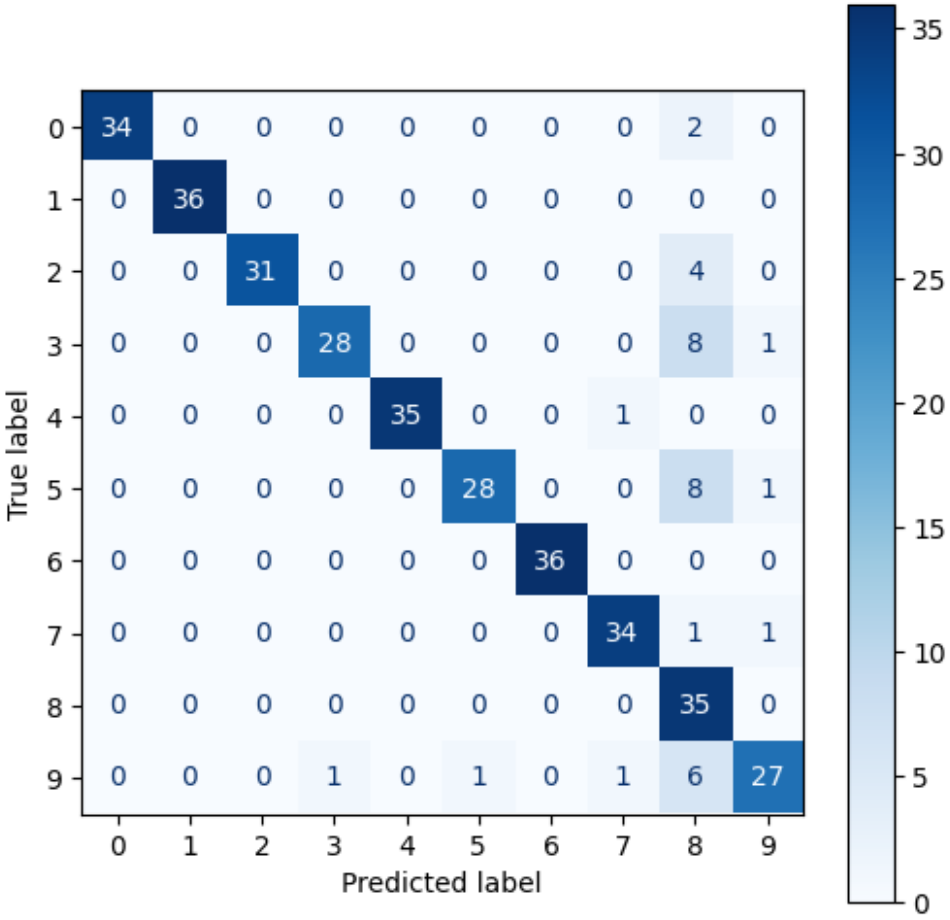
Output without Hyperparameter Tuning

```
svm_clf = SVC(kernel="poly")

X_train, X_test, y_train, y_test = train_test_split(
        X_pca, y, test_size=0.20, shuffle=True, random_state=10,
stratify=y
    )

svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        36
           1       1.00      1.00      1.00        36
           2       1.00      0.89      0.94        35
           3       0.97      0.76      0.85        37
           4       1.00      0.97      0.99        36
           5       0.97      0.76      0.85        37
           6       1.00      1.00      1.00        36
           7       0.94      0.94      0.94        36
           8       0.55      1.00      0.71        35
           9       0.90      0.75      0.82        36

    accuracy                           0.90       360
   macro avg       0.93      0.90      0.91       360
weighted avg       0.93      0.90      0.91       360
```

```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(svm_clf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7a1814abf530>
```



Output with Hyperparameter Tuning

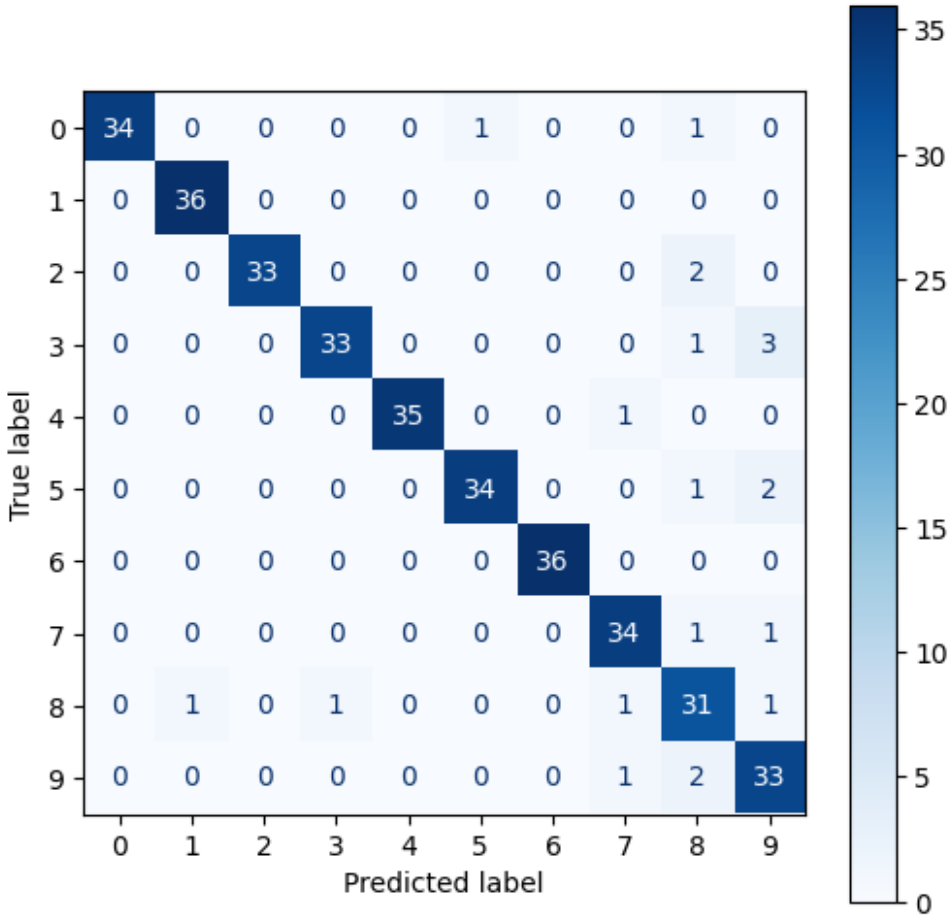
```
svm_clf = SVC(kernel="poly", C=0.1, degree=5, gamma="scale", coef0=1)

X_train, X_test, y_train, y_test = train_test_split(
        X_pca, y, test_size=0.20, shuffle=True, random_state=10,
stratify=y
    )

svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        36
           1       0.97      1.00      0.99        36
           2       1.00      0.94      0.97        35
           3       0.97      0.89      0.93        37
           4       1.00      0.97      0.99        36
           5       0.97      0.92      0.94        37
```

```
           6       1.00      1.00      1.00        36
           7       0.92      0.94      0.93        36
           8       0.79      0.89      0.84        35
           9       0.82      0.92      0.87        36

    accuracy                           0.94       360
   macro avg       0.95      0.94      0.94       360
weighted avg       0.95      0.94      0.94       360
```

```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(svm_clf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7a18150b4200>
```



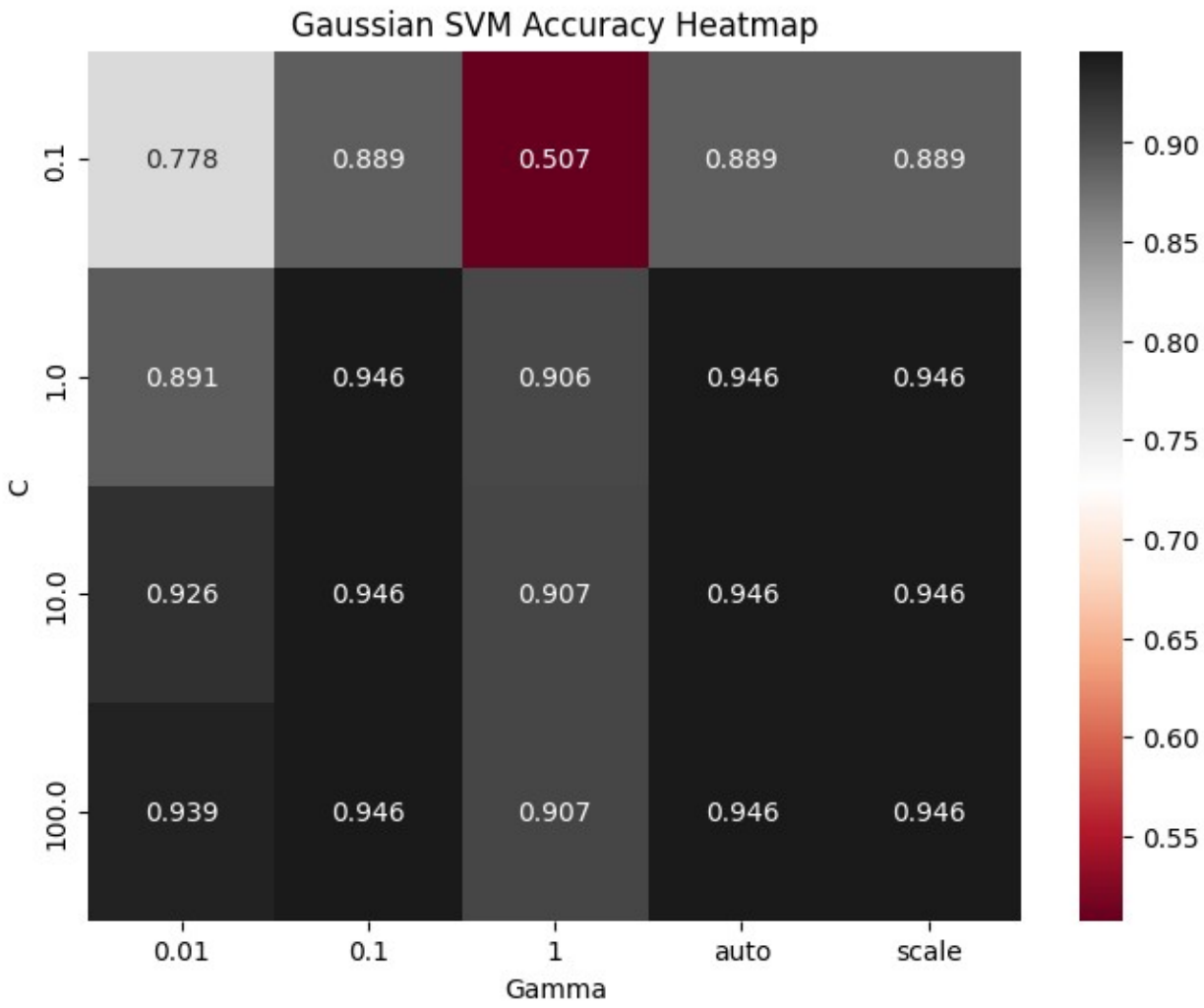
Gausian SVM

```
param_grid_rbf = {
    "C": [0.1, 1, 10, 100],
```

```
    "gamma": ["scale", "auto", 0.01, 0.1, 1]  # can also try numeric
values
}

def train_rbf_svm(X, y, split=0.30, shuffle=True, random_state=10):
    # Train/test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=split, shuffle=shuffle,
        random_state=random_state, stratify=y
    )

    # Scale features (important for SVMs)
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    results = []

    for c in param_grid_rbf["C"]:
        for gamma in param_grid_rbf["gamma"]:
            svm_clf = SVC(kernel="rbf", C=c, gamma=gamma)
            svm_clf.fit(X_train, y_train)
            y_pred = svm_clf.predict(X_test)
            acc = accuracy_score(y_test, y_pred)

            results.append({
                "C": c,
                "gamma": gamma,
                "accuracy": acc
            })

            print(f"C={c}, gamma={gamma} → Accuracy={acc:.4f}")

    df = pd.DataFrame(results)

    # pick best row
    best = df.loc[df["accuracy"].idxmax()]
    print("\n Best Parameters:")
    print(best)

    # Pivot for heatmap
    pivot = df.pivot(index="C", columns="gamma", values="accuracy")

    plt.figure(figsize=(8,6))
    sns.heatmap(pivot, annot=True, cmap="RdGy", fmt=".3f")
    plt.title("Gaussian SVM Accuracy Heatmap")
    plt.ylabel("C")
    plt.xlabel("Gamma")
    plt.show()
```

```
    return df, best

train_rbf_svm(X_pca, y)
```

```
C=0.1, gamma=scale → Accuracy=0.8889
C=0.1, gamma=auto → Accuracy=0.8889
C=0.1, gamma=0.01 → Accuracy=0.7778
C=0.1, gamma=0.1 → Accuracy=0.8889
C=0.1, gamma=1 → Accuracy=0.5074
C=1, gamma=scale → Accuracy=0.9463
C=1, gamma=auto → Accuracy=0.9463
C=1, gamma=0.01 → Accuracy=0.8907
C=1, gamma=0.1 → Accuracy=0.9463
C=1, gamma=1 → Accuracy=0.9056
C=10, gamma=scale → Accuracy=0.9463
C=10, gamma=auto → Accuracy=0.9463
C=10, gamma=0.01 → Accuracy=0.9259
C=10, gamma=0.1 → Accuracy=0.9463
C=10, gamma=1 → Accuracy=0.9074
C=100, gamma=scale → Accuracy=0.9463
C=100, gamma=auto → Accuracy=0.9463
C=100, gamma=0.01 → Accuracy=0.9389
C=100, gamma=0.1 → Accuracy=0.9463
C=100, gamma=1 → Accuracy=0.9074

 Best Parameters:
C                 1.0
gamma           scale
accuracy    0.946296
Name: 5, dtype: object
```




```
(        C  gamma  accuracy
 0     0.1  scale  0.888889
 1     0.1   auto  0.888889
 2     0.1   0.01  0.777778
 3     0.1    0.1  0.888889
 4     0.1      1  0.507407
 5     1.0  scale  0.946296
 6     1.0   auto  0.946296
 7     1.0   0.01  0.890741
 8     1.0    0.1  0.946296
 9     1.0      1  0.905556
 10   10.0  scale  0.946296
 11   10.0   auto  0.946296
 12   10.0   0.01  0.925926
 13   10.0    0.1  0.946296
 14   10.0      1  0.907407
 15  100.0  scale  0.946296
 16  100.0   auto  0.946296
```

```
 17  100.0   0.01  0.938889
 18  100.0    0.1  0.946296
 19  100.0      1  0.907407,
 C                 1.0
 gamma           scale
 accuracy    0.946296
 Name: 5, dtype: object)
```

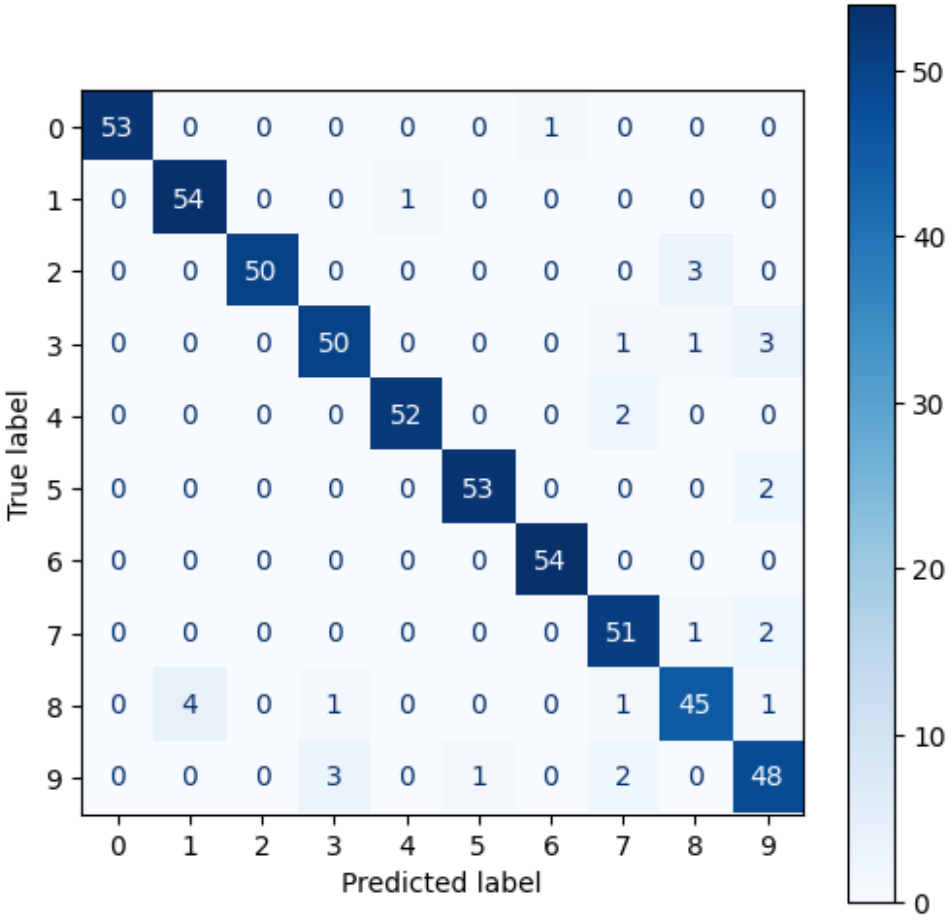
Output without Hyperparameter Tuning

```
svm_clf = SVC(kernel="rbf")

X_train, X_test, y_train, y_test = train_test_split(
        X_pca, y, test_size=0.30, shuffle=True, random_state=10,
stratify=y
    )

svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        54
           1       0.93      0.98      0.96        55
           2       1.00      0.94      0.97        53
           3       0.93      0.91      0.92        55
           4       0.98      0.96      0.97        54
           5       0.98      0.96      0.97        55
           6       0.98      1.00      0.99        54
           7       0.89      0.94      0.92        54
           8       0.90      0.87      0.88        52
           9       0.86      0.89      0.87        54

    accuracy                           0.94       540
   macro avg       0.95      0.94      0.94       540
weighted avg       0.95      0.94      0.94       540
```

```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(svm_clf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7a1814ec5220>
```



Output with Hyperparamter Tuning

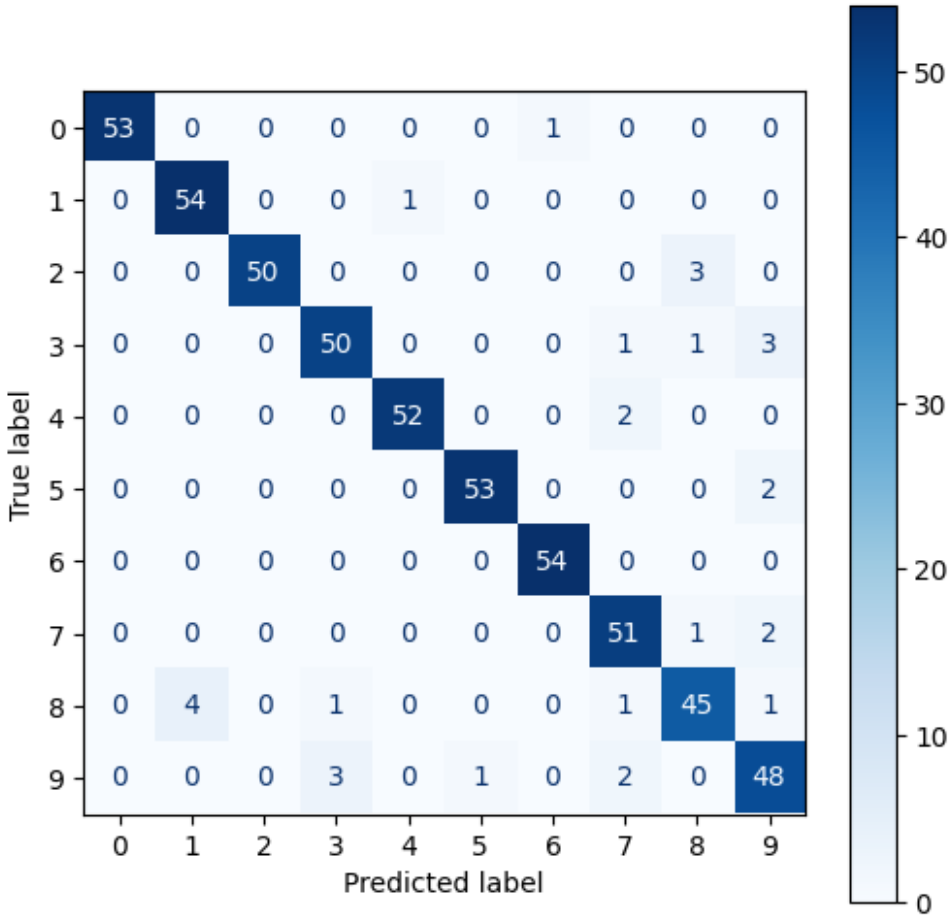
```
svm_clf = SVC(kernel="rbf", C=1, gamma='scale')

X_train, X_test, y_train, y_test = train_test_split(
        X_pca, y, test_size=0.30, shuffle=True, random_state=10,
stratify=y
    )

svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        54
           1       0.93      0.98      0.96        55
           2       1.00      0.94      0.97        53
           3       0.93      0.91      0.92        55
           4       0.98      0.96      0.97        54
           5       0.98      0.96      0.97        55
```

```
           6       0.98      1.00      0.99        54
           7       0.89      0.94      0.92        54
           8       0.90      0.87      0.88        52
           9       0.86      0.89      0.87        54

    accuracy                           0.94       540
   macro avg       0.95      0.94      0.94       540
weighted avg       0.95      0.94      0.94       540
```

```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(svm_clf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7a1814e67ef0>
```



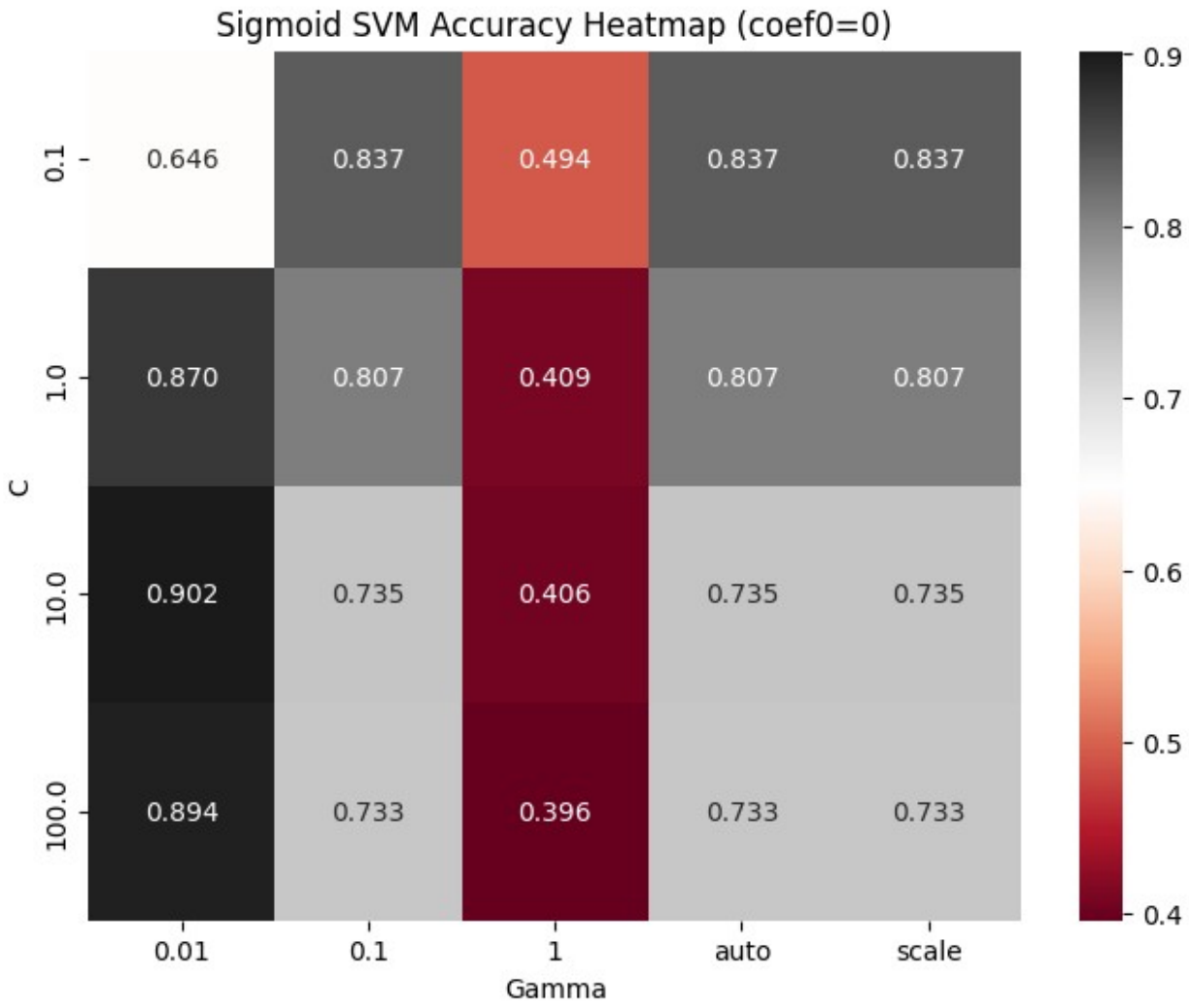
Sigmoid SVM

```
param_grid_sigmoid = {
    "C": [0.1, 1, 10, 100],
    "gamma": ["scale", "auto", 0.01, 0.1, 1],
    "coef0": [0, 1, 5]
}

def train_sigmoid_svm(X, y, split=0.30, shuffle=True,
random_state=10):
    # Train/test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=split, shuffle=shuffle,
        random_state=random_state, stratify=y
    )

    # Scale features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    results = []

    # Loop through params
    for c in param_grid_sigmoid["C"]:
        for gamma in param_grid_sigmoid["gamma"]:
            for coef in param_grid_sigmoid["coef0"]:
                svm_clf = SVC(kernel="sigmoid", C=c, gamma=gamma,
coef0=coef)
                svm_clf.fit(X_train, y_train)
                y_pred = svm_clf.predict(X_test)
                acc = accuracy_score(y_test, y_pred)

                results.append({
                    "C": c,
                    "gamma": gamma,
                    "coef0": coef,
                    "accuracy": acc
                })

                print(f"C={c}, gamma={gamma}, coef0={coef} →
Accuracy={acc:.4f}")

    df = pd.DataFrame(results)

    # Best params
    best = df.loc[df["accuracy"].idxmax()]
    print("\n Best Parameters:")
    print(best)
```

```
    # Pivot table (for fixed coef0=0 just for visualization)
    subset = df[df["coef0"]==0]
    pivot = subset.pivot(index="C", columns="gamma",
values="accuracy")

    plt.figure(figsize=(8,6))
    sns.heatmap(pivot, annot=True, cmap="RdGy", fmt=".3f")
    plt.title("Sigmoid SVM Accuracy Heatmap (coef0=0)")
    plt.ylabel("C")
    plt.xlabel("Gamma")
    plt.show()

    return df, best

train_sigmoid_svm(X_pca, y)
```

```
C=0.1, gamma=scale, coef0=0 → Accuracy=0.8370
C=0.1, gamma=scale, coef0=1 → Accuracy=0.7889
C=0.1, gamma=scale, coef0=5 → Accuracy=0.1019
C=0.1, gamma=auto, coef0=0 → Accuracy=0.8370
C=0.1, gamma=auto, coef0=1 → Accuracy=0.7889
C=0.1, gamma=auto, coef0=5 → Accuracy=0.1019
C=0.1, gamma=0.01, coef0=0 → Accuracy=0.6463
C=0.1, gamma=0.01, coef0=1 → Accuracy=0.1037
C=0.1, gamma=0.01, coef0=5 → Accuracy=0.1019
C=0.1, gamma=0.1, coef0=0 → Accuracy=0.8370
C=0.1, gamma=0.1, coef0=1 → Accuracy=0.7889
C=0.1, gamma=0.1, coef0=5 → Accuracy=0.1019
C=0.1, gamma=1, coef0=0 → Accuracy=0.4944
C=0.1, gamma=1, coef0=1 → Accuracy=0.3537
C=0.1, gamma=1, coef0=5 → Accuracy=0.1556
C=1, gamma=scale, coef0=0 → Accuracy=0.8074
C=1, gamma=scale, coef0=1 → Accuracy=0.7333
C=1, gamma=scale, coef0=5 → Accuracy=0.1019
C=1, gamma=auto, coef0=0 → Accuracy=0.8074
C=1, gamma=auto, coef0=1 → Accuracy=0.7333
C=1, gamma=auto, coef0=5 → Accuracy=0.1019
C=1, gamma=0.01, coef0=0 → Accuracy=0.8704
C=1, gamma=0.01, coef0=1 → Accuracy=0.8444
C=1, gamma=0.01, coef0=5 → Accuracy=0.1019
C=1, gamma=0.1, coef0=0 → Accuracy=0.8074
C=1, gamma=0.1, coef0=1 → Accuracy=0.7333
C=1, gamma=0.1, coef0=5 → Accuracy=0.1019
C=1, gamma=1, coef0=0 → Accuracy=0.4093
C=1, gamma=1, coef0=1 → Accuracy=0.3111
C=1, gamma=1, coef0=5 → Accuracy=0.1519
C=10, gamma=scale, coef0=0 → Accuracy=0.7352
C=10, gamma=scale, coef0=1 → Accuracy=0.6278
C=10, gamma=scale, coef0=5 → Accuracy=0.1019
C=10, gamma=auto, coef0=0 → Accuracy=0.7352
```

```
C=10, gamma=auto, coef0=1 → Accuracy=0.6278
C=10, gamma=auto, coef0=5 → Accuracy=0.1019
C=10, gamma=0.01, coef0=0 → Accuracy=0.9019
C=10, gamma=0.01, coef0=1 → Accuracy=0.8741
C=10, gamma=0.01, coef0=5 → Accuracy=0.1019
C=10, gamma=0.1, coef0=0 → Accuracy=0.7352
C=10, gamma=0.1, coef0=1 → Accuracy=0.6278
C=10, gamma=0.1, coef0=5 → Accuracy=0.1019
C=10, gamma=1, coef0=0 → Accuracy=0.4056
C=10, gamma=1, coef0=1 → Accuracy=0.2926
C=10, gamma=1, coef0=5 → Accuracy=0.1889
C=100, gamma=scale, coef0=0 → Accuracy=0.7333
C=100, gamma=scale, coef0=1 → Accuracy=0.6093
C=100, gamma=scale, coef0=5 → Accuracy=0.7093
C=100, gamma=auto, coef0=0 → Accuracy=0.7333
C=100, gamma=auto, coef0=1 → Accuracy=0.6093
C=100, gamma=auto, coef0=5 → Accuracy=0.7093
C=100, gamma=0.01, coef0=0 → Accuracy=0.8944
C=100, gamma=0.01, coef0=1 → Accuracy=0.8611
C=100, gamma=0.01, coef0=5 → Accuracy=0.1019
C=100, gamma=0.1, coef0=0 → Accuracy=0.7333
C=100, gamma=0.1, coef0=1 → Accuracy=0.6093
C=100, gamma=0.1, coef0=5 → Accuracy=0.7093
C=100, gamma=1, coef0=0 → Accuracy=0.3963
C=100, gamma=1, coef0=1 → Accuracy=0.3037
C=100, gamma=1, coef0=5 → Accuracy=0.1870

 Best Parameters:
C               10.0
gamma           0.01
coef0              0
accuracy    0.901852
Name: 36, dtype: object
```




```
(        C  gamma  coef0  accuracy
 0     0.1  scale      0  0.837037
 1     0.1  scale      1  0.788889
 2     0.1  scale      5  0.101852
 3     0.1   auto      0  0.837037
 4     0.1   auto      1  0.788889
 5     0.1   auto      5  0.101852
 6     0.1   0.01      0  0.646296
 7     0.1   0.01      1  0.103704
 8     0.1   0.01      5  0.101852
 9     0.1    0.1      0  0.837037
 10    0.1    0.1      1  0.788889
 11    0.1    0.1      5  0.101852
 12    0.1      1      0  0.494444
 13    0.1      1      1  0.353704
 14    0.1      1      5  0.155556
 15    1.0  scale      0  0.807407
 16    1.0  scale      1  0.733333
```

```
17    1.0  scale      5  0.101852
18    1.0   auto      0  0.807407
19    1.0   auto      1  0.733333
20    1.0   auto      5  0.101852
21    1.0   0.01      0  0.870370
22    1.0   0.01      1  0.844444
23    1.0   0.01      5  0.101852
24    1.0    0.1      0  0.807407
25    1.0    0.1      1  0.733333
26    1.0    0.1      5  0.101852
27    1.0      1      0  0.409259
28    1.0      1      1  0.311111
29    1.0      1      5  0.151852
30   10.0  scale      0  0.735185
31   10.0  scale      1  0.627778
32   10.0  scale      5  0.101852
33   10.0   auto      0  0.735185
34   10.0   auto      1  0.627778
35   10.0   auto      5  0.101852
36   10.0   0.01      0  0.901852
37   10.0   0.01      1  0.874074
38   10.0   0.01      5  0.101852
39   10.0    0.1      0  0.735185
40   10.0    0.1      1  0.627778
41   10.0    0.1      5  0.101852
42   10.0      1      0  0.405556
43   10.0      1      1  0.292593
44   10.0      1      5  0.188889
45  100.0  scale      0  0.733333
46  100.0  scale      1  0.609259
47  100.0  scale      5  0.709259
48  100.0   auto      0  0.733333
49  100.0   auto      1  0.609259
50  100.0   auto      5  0.709259
51  100.0   0.01      0  0.894444
52  100.0   0.01      1  0.861111
53  100.0   0.01      5  0.101852
54  100.0    0.1      0  0.733333
55  100.0    0.1      1  0.609259
56  100.0    0.1      5  0.709259
57  100.0      1      0  0.396296
58  100.0      1      1  0.303704
59  100.0      1      5  0.187037,
C               10.0
gamma            0.01
coef0               0
accuracy     0.901852
Name: 36, dtype: object)
```

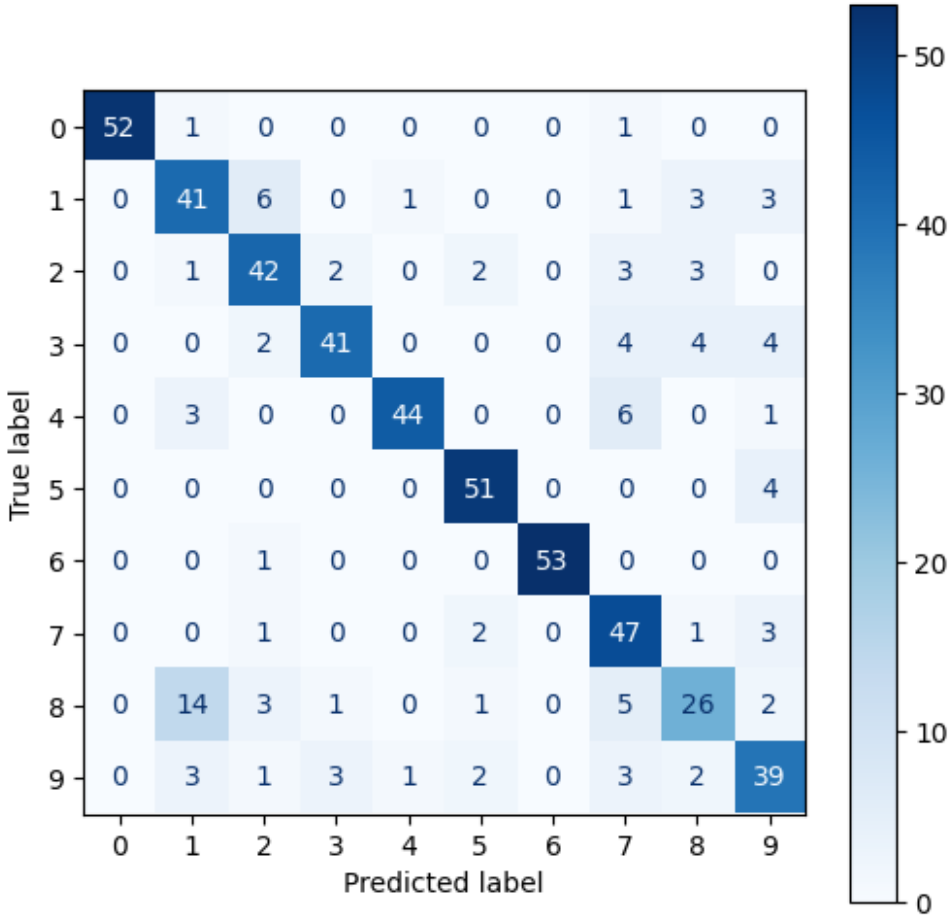
Output without Hyperparameter

```
svm_clf = SVC(kernel="sigmoid")

X_train, X_test, y_train, y_test = train_test_split(
        X_pca, y, test_size=0.30, shuffle=True, random_state=10,
stratify=y
    )

svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.96      0.98        54
           1       0.65      0.75      0.69        55
           2       0.75      0.79      0.77        53
           3       0.87      0.75      0.80        55
           4       0.96      0.81      0.88        54
           5       0.88      0.93      0.90        55
           6       1.00      0.98      0.99        54
           7       0.67      0.87      0.76        54
           8       0.67      0.50      0.57        52
           9       0.70      0.72      0.71        54

    accuracy                           0.81       540
   macro avg       0.81      0.81      0.81       540
weighted avg       0.81      0.81      0.81       540
```

```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(svm_clf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7a181567b1a0>
```



Output with Hyperparameter Tuning

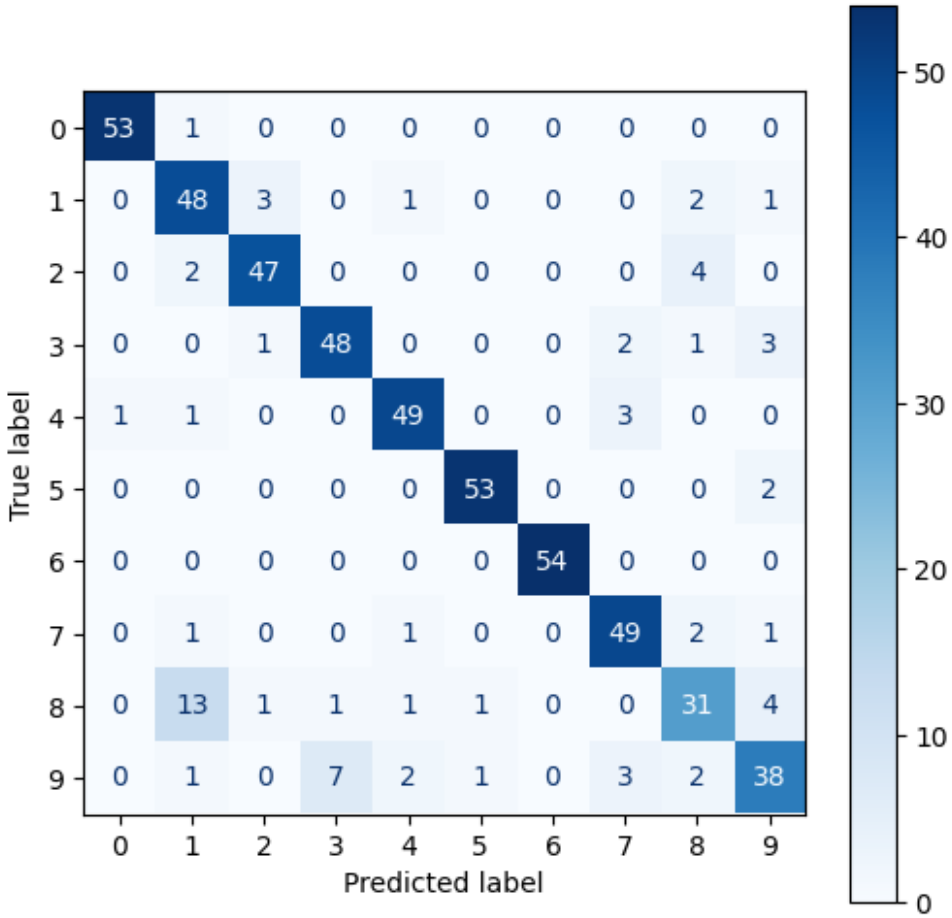
```
svm_clf = SVC(kernel="sigmoid", C=10, gamma=0.01, coef0=0)

X_train, X_test, y_train, y_test = train_test_split(
        X_pca, y, test_size=0.30, shuffle=True, random_state=10,
stratify=y
    )

svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        54
           1       0.72      0.87      0.79        55
           2       0.90      0.89      0.90        53
           3       0.86      0.87      0.86        55
           4       0.91      0.91      0.91        54
           5       0.96      0.96      0.96        55
```

```
           6       1.00      1.00      1.00        54
           7       0.86      0.91      0.88        54
           8       0.74      0.60      0.66        52
           9       0.78      0.70      0.74        54

    accuracy                           0.87       540
   macro avg       0.87      0.87      0.87       540
weighted avg       0.87      0.87      0.87       540
```

```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(svm_clf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7a181f8c69f0>
```



Random Forest

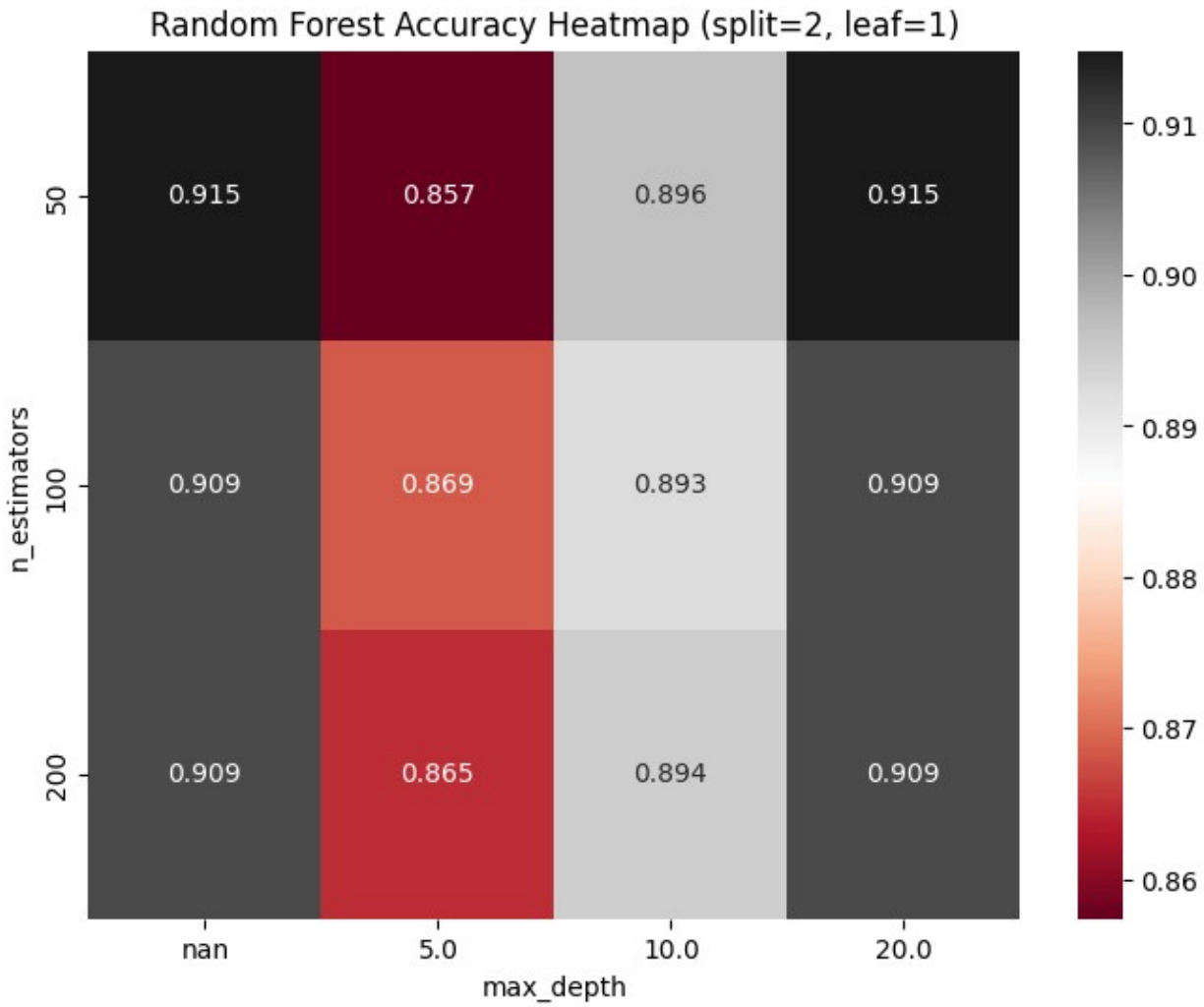
```
param_grid_rf = {
    "n_estimators": [50, 100, 200],
    "max_depth": [None, 5, 10, 20],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4]
}

def train_random_forest(X, y, split=0.30, shuffle=True,
random_state=10):
    # Train/test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=split, shuffle=shuffle,
        random_state=random_state, stratify=y
    )

    results = []

    for n in param_grid_rf["n_estimators"]:
        for depth in param_grid_rf["max_depth"]:
            for split_val in param_grid_rf["min_samples_split"]:
                for leaf_val in param_grid_rf["min_samples_leaf"]:
                    rf_clf = RandomForestClassifier(
                        n_estimators=n,
                        max_depth=depth,
                        min_samples_split=split_val,
                        min_samples_leaf=leaf_val,
                        random_state=random_state,
                        n_jobs=-1
                    )
                    rf_clf.fit(X_train, y_train)
                    y_pred = rf_clf.predict(X_test)
                    acc = accuracy_score(y_test, y_pred)

                    results.append({
                        "n_estimators": n,
                        "max_depth": depth,
                        "min_samples_split": split_val,
                        "min_samples_leaf": leaf_val,
                        "accuracy": acc
                    })

                    print(f"n={n}, depth={depth}, split={split_val},
leaf={leaf_val} → Accuracy={acc:.4f}")

    df = pd.DataFrame(results)

    # Best params
    best = df.loc[df["accuracy"].idxmax()]
```

```
    print("\n Best Parameters:")
    print(best)

    # Pivot table (for fixed min_samples_split=2, min_samples_leaf=1
just for visualization)
    subset = df[(df["min_samples_split"]==2) &
(df["min_samples_leaf"]==1)]
    pivot = subset.pivot(index="n_estimators", columns="max_depth",
values="accuracy")

    plt.figure(figsize=(8,6))
    sns.heatmap(pivot, annot=True, cmap="RdGy", fmt=".3f")
    plt.title("Random Forest Accuracy Heatmap (split=2, leaf=1)")
    plt.ylabel("n_estimators")
    plt.xlabel("max_depth")
    plt.show()

    return df, best

train_random_forest(X_pca, y)
```

```
n=50, depth=None, split=2, leaf=1 → Accuracy=0.9148
n=50, depth=None, split=2, leaf=2 → Accuracy=0.9111
n=50, depth=None, split=2, leaf=4 → Accuracy=0.9019
n=50, depth=None, split=5, leaf=1 → Accuracy=0.9019
n=50, depth=None, split=5, leaf=2 → Accuracy=0.9056
n=50, depth=None, split=5, leaf=4 → Accuracy=0.9019
n=50, depth=None, split=10, leaf=1 → Accuracy=0.9037
n=50, depth=None, split=10, leaf=2 → Accuracy=0.8944
n=50, depth=None, split=10, leaf=4 → Accuracy=0.8907
n=50, depth=5, split=2, leaf=1 → Accuracy=0.8574
n=50, depth=5, split=2, leaf=2 → Accuracy=0.8685
n=50, depth=5, split=2, leaf=4 → Accuracy=0.8537
n=50, depth=5, split=5, leaf=1 → Accuracy=0.8648
n=50, depth=5, split=5, leaf=2 → Accuracy=0.8667
n=50, depth=5, split=5, leaf=4 → Accuracy=0.8537
n=50, depth=5, split=10, leaf=1 → Accuracy=0.8630
n=50, depth=5, split=10, leaf=2 → Accuracy=0.8685
n=50, depth=5, split=10, leaf=4 → Accuracy=0.8630
n=50, depth=10, split=2, leaf=1 → Accuracy=0.8963
n=50, depth=10, split=2, leaf=2 → Accuracy=0.9056
n=50, depth=10, split=2, leaf=4 → Accuracy=0.9019
n=50, depth=10, split=5, leaf=1 → Accuracy=0.8852
n=50, depth=10, split=5, leaf=2 → Accuracy=0.8796
n=50, depth=10, split=5, leaf=4 → Accuracy=0.9019
n=50, depth=10, split=10, leaf=1 → Accuracy=0.8963
n=50, depth=10, split=10, leaf=2 → Accuracy=0.8963
n=50, depth=10, split=10, leaf=4 → Accuracy=0.8944
n=50, depth=20, split=2, leaf=1 → Accuracy=0.9148
n=50, depth=20, split=2, leaf=2 → Accuracy=0.9111
```

```
n=50, depth=20, split=2, leaf=4 → Accuracy=0.9019
n=50, depth=20, split=5, leaf=1 → Accuracy=0.9019
n=50, depth=20, split=5, leaf=2 → Accuracy=0.9056
n=50, depth=20, split=5, leaf=4 → Accuracy=0.9019
n=50, depth=20, split=10, leaf=1 → Accuracy=0.9037
n=50, depth=20, split=10, leaf=2 → Accuracy=0.8926
n=50, depth=20, split=10, leaf=4 → Accuracy=0.8907
n=100, depth=None, split=2, leaf=1 → Accuracy=0.9093
n=100, depth=None, split=2, leaf=2 → Accuracy=0.9056
n=100, depth=None, split=2, leaf=4 → Accuracy=0.8981
n=100, depth=None, split=5, leaf=1 → Accuracy=0.9037
n=100, depth=None, split=5, leaf=2 → Accuracy=0.9037
n=100, depth=None, split=5, leaf=4 → Accuracy=0.8981
n=100, depth=None, split=10, leaf=1 → Accuracy=0.9037
n=100, depth=None, split=10, leaf=2 → Accuracy=0.8963
n=100, depth=None, split=10, leaf=4 → Accuracy=0.8963
n=100, depth=5, split=2, leaf=1 → Accuracy=0.8685
n=100, depth=5, split=2, leaf=2 → Accuracy=0.8593
n=100, depth=5, split=2, leaf=4 → Accuracy=0.8500
n=100, depth=5, split=5, leaf=1 → Accuracy=0.8611
n=100, depth=5, split=5, leaf=2 → Accuracy=0.8630
n=100, depth=5, split=5, leaf=4 → Accuracy=0.8500
n=100, depth=5, split=10, leaf=1 → Accuracy=0.8574
n=100, depth=5, split=10, leaf=2 → Accuracy=0.8574
n=100, depth=5, split=10, leaf=4 → Accuracy=0.8556
n=100, depth=10, split=2, leaf=1 → Accuracy=0.8926
n=100, depth=10, split=2, leaf=2 → Accuracy=0.8963
n=100, depth=10, split=2, leaf=4 → Accuracy=0.8963
n=100, depth=10, split=5, leaf=1 → Accuracy=0.8963
n=100, depth=10, split=5, leaf=2 → Accuracy=0.8889
n=100, depth=10, split=5, leaf=4 → Accuracy=0.8963
n=100, depth=10, split=10, leaf=1 → Accuracy=0.9056
n=100, depth=10, split=10, leaf=2 → Accuracy=0.9056
n=100, depth=10, split=10, leaf=4 → Accuracy=0.8944
n=100, depth=20, split=2, leaf=1 → Accuracy=0.9093
n=100, depth=20, split=2, leaf=2 → Accuracy=0.9056
n=100, depth=20, split=2, leaf=4 → Accuracy=0.8981
n=100, depth=20, split=5, leaf=1 → Accuracy=0.9037
n=100, depth=20, split=5, leaf=2 → Accuracy=0.9037
n=100, depth=20, split=5, leaf=4 → Accuracy=0.8981
n=100, depth=20, split=10, leaf=1 → Accuracy=0.9037
n=100, depth=20, split=10, leaf=2 → Accuracy=0.8963
n=100, depth=20, split=10, leaf=4 → Accuracy=0.8963
n=200, depth=None, split=2, leaf=1 → Accuracy=0.9093
n=200, depth=None, split=2, leaf=2 → Accuracy=0.9056
n=200, depth=None, split=2, leaf=4 → Accuracy=0.9019
n=200, depth=None, split=5, leaf=1 → Accuracy=0.9056
n=200, depth=None, split=5, leaf=2 → Accuracy=0.9019
n=200, depth=None, split=5, leaf=4 → Accuracy=0.9019
```

```
n=200, depth=None, split=10, leaf=1 → Accuracy=0.9111
n=200, depth=None, split=10, leaf=2 → Accuracy=0.9074
n=200, depth=None, split=10, leaf=4 → Accuracy=0.9019
n=200, depth=5, split=2, leaf=1 → Accuracy=0.8648
n=200, depth=5, split=2, leaf=2 → Accuracy=0.8611
n=200, depth=5, split=2, leaf=4 → Accuracy=0.8648
n=200, depth=5, split=5, leaf=1 → Accuracy=0.8593
n=200, depth=5, split=5, leaf=2 → Accuracy=0.8611
n=200, depth=5, split=5, leaf=4 → Accuracy=0.8648
n=200, depth=5, split=10, leaf=1 → Accuracy=0.8667
n=200, depth=5, split=10, leaf=2 → Accuracy=0.8685
n=200, depth=5, split=10, leaf=4 → Accuracy=0.8704
n=200, depth=10, split=2, leaf=1 → Accuracy=0.8944
n=200, depth=10, split=2, leaf=2 → Accuracy=0.8963
n=200, depth=10, split=2, leaf=4 → Accuracy=0.8981
n=200, depth=10, split=5, leaf=1 → Accuracy=0.9000
n=200, depth=10, split=5, leaf=2 → Accuracy=0.9000
n=200, depth=10, split=5, leaf=4 → Accuracy=0.8981
n=200, depth=10, split=10, leaf=1 → Accuracy=0.9037
n=200, depth=10, split=10, leaf=2 → Accuracy=0.9111
n=200, depth=10, split=10, leaf=4 → Accuracy=0.8981
n=200, depth=20, split=2, leaf=1 → Accuracy=0.9093
n=200, depth=20, split=2, leaf=2 → Accuracy=0.9056
n=200, depth=20, split=2, leaf=4 → Accuracy=0.9019
n=200, depth=20, split=5, leaf=1 → Accuracy=0.9056
n=200, depth=20, split=5, leaf=2 → Accuracy=0.9019
n=200, depth=20, split=5, leaf=4 → Accuracy=0.9019
n=200, depth=20, split=10, leaf=1 → Accuracy=0.9111
n=200, depth=20, split=10, leaf=2 → Accuracy=0.9074
n=200, depth=20, split=10, leaf=4 → Accuracy=0.9019

 Best Parameters:
n_estimators         50.000000
max_depth                  NaN
min_samples_split     2.000000
min_samples_leaf      1.000000
accuracy              0.914815
Name: 0, dtype: float64
```




```
(     n_estimators  max_depth  min_samples_split  min_samples_leaf
accuracy
 0              50        NaN                  2                 1
0.914815
 1              50        NaN                  2                 2
0.911111
 2              50        NaN                  2                 4
0.901852
 3              50        NaN                  5                 1
0.901852
 4              50        NaN                  5                 2
0.905556
 ..            ...        ...                ...               ...
...
 103           200       20.0                  5                 2
0.901852
 104           200       20.0                  5                 4
0.901852
```

```
 105           200       20.0                  10                   1
0.911111
 106           200       20.0                  10                   2
0.907407
 107           200       20.0                  10                   4
0.901852

 [108 rows x 5 columns],
 n_estimators         50.000000
 max_depth                  NaN
 min_samples_split     2.000000
 min_samples_leaf      1.000000
 accuracy              0.914815
 Name: 0, dtype: float64)
```

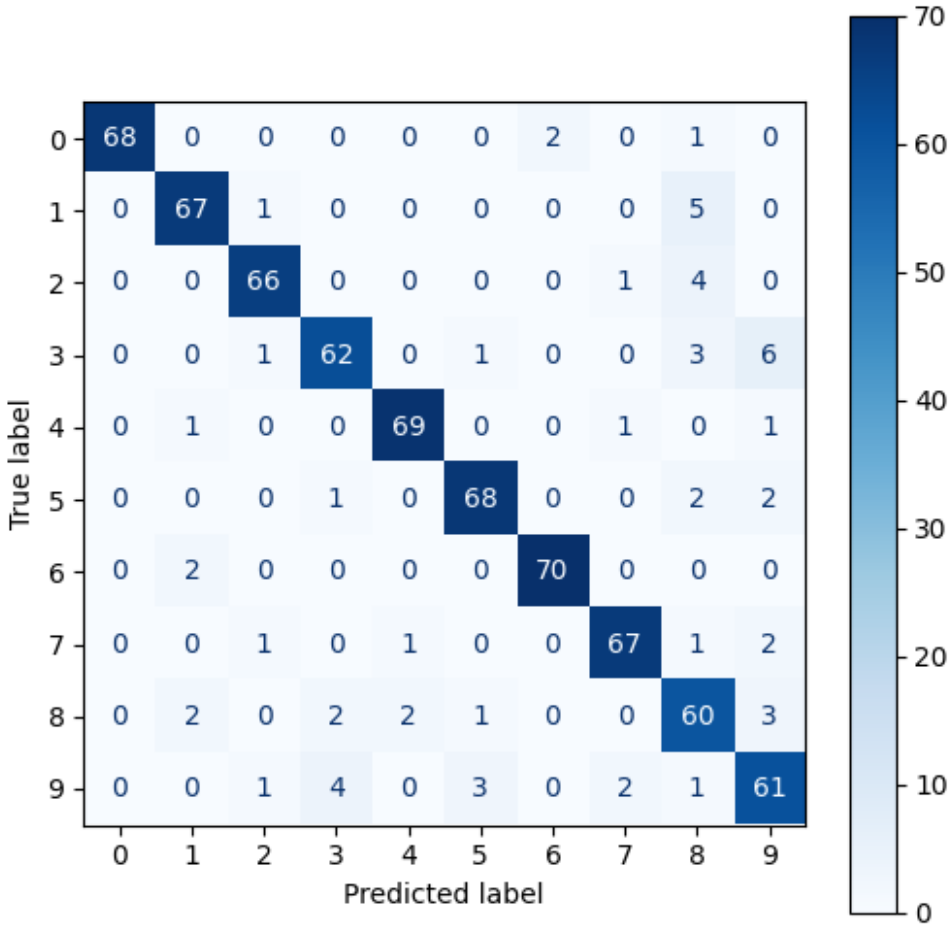
Output without Hyperparameter Tuning

```
X_train, X_test, y_train, y_test = train_test_split(
        X_pca, y, test_size=0.40, shuffle=True, random_state=10,
stratify=y
    )

rf_clf = RandomForestClassifier()

rf_clf.fit(X_train, y_train)

rf_clf.fit(X_train, y_train)
y_pred = rf_clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.96      0.98        71
           1       0.93      0.92      0.92        73
           2       0.94      0.93      0.94        71
           3       0.90      0.85      0.87        73
           4       0.96      0.96      0.96        72
           5       0.93      0.93      0.93        73
           6       0.97      0.97      0.97        72
           7       0.94      0.93      0.94        72
           8       0.78      0.86      0.82        70
           9       0.81      0.85      0.83        72

    accuracy                           0.92       719
   macro avg       0.92      0.92      0.92       719
weighted avg       0.92      0.92      0.92       719
```

```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(rf_clf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7a1813ffce30>
```



Output with Hyperparameter Tuning

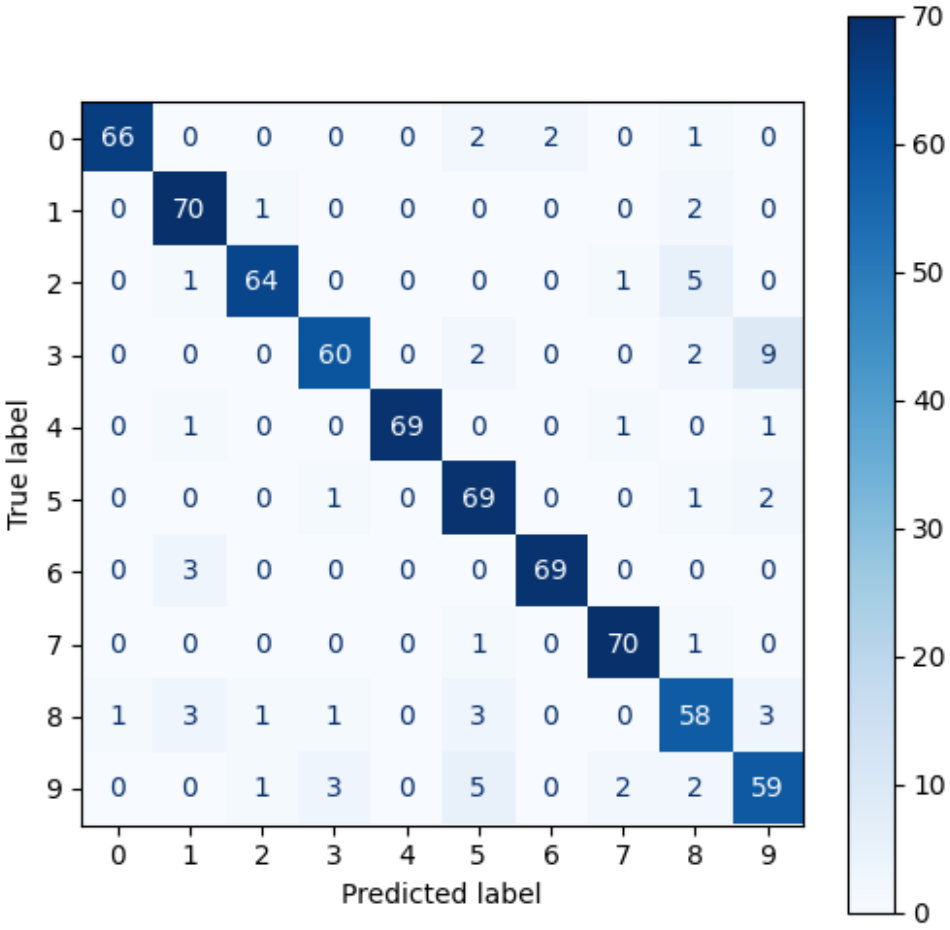
```
X_train, X_test, y_train, y_test = train_test_split(
        X_pca, y, test_size=0.40, shuffle=True, random_state=10,
stratify=y
    )

rf_clf = RandomForestClassifier(n_estimators=50, min_samples_split=2,
min_samples_leaf=1)

rf_clf.fit(X_train, y_train)

rf_clf.fit(X_train, y_train)
y_pred = rf_clf.predict(X_test)
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.93      0.96        71
           1       0.90      0.96      0.93        73
           2       0.96      0.90      0.93        71
           3       0.92      0.82      0.87        73
           4       1.00      0.96      0.98        72
           5       0.84      0.95      0.89        73
           6       0.97      0.96      0.97        72
           7       0.95      0.97      0.96        72
           8       0.81      0.83      0.82        70
           9       0.80      0.82      0.81        72

    accuracy                           0.91       719
   macro avg       0.91      0.91      0.91       719
weighted avg       0.91      0.91      0.91       719
```

```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(rf_clf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7a1815551460>
```



Wine Dataset

Loading the data

```
from sklearn.datasets import load_wine

wine = load_wine()

X = pd.DataFrame(wine.data, columns=wine.feature_names)
y = pd.DataFrame(wine.target, columns=["target"])
```

Number of data of each category

```
y.value_counts()
```

```
target
1         71
0         59
```

```
2         48
Name: count, dtype: int64
```

Checking for duplicate rows

```
X.duplicated().sum()
```

```
np.int64(0)
```

```
df = pd.concat([X,y],axis=1)
display(df)
df = df.rename(columns={"target": "class"})
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 178,\n  \"fields\": [\n    {\n      \"column\": \"alcohol\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.8118265380058577,\n        \"min\": 11.03,\n        \"max\": 14.83,\n        \"num_unique_values\": 126,\n        \"samples\": [\n          11.62,\n          13.64,\n          13.69\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"malic_acid\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 1.1171460976144627,\n        \"min\": 0.74,\n        \"max\": 5.8,\n        \"num_unique_values\": 133,\n        \"samples\": [\n          1.21,\n          2.83,\n          1.8\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"ash\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.2743440090608148,\n        \"min\": 1.36,\n        \"max\": 3.23,\n        \"num_unique_values\": 79,\n        \"samples\": [\n          2.31,\n          2.43,\n          2.52\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"alcalinity_of_ash\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 3.3395637671735052,\n        \"min\": 10.6,\n        \"max\": 30.0,\n        \"num_unique_values\": 63,\n        \"samples\": [\n          25.5,\n          28.5,\n          15.6\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"magnesium\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 14.282483515295668,\n        \"min\": 70.0,\n        \"max\": 162.0,\n        \"num_unique_values\": 53,\n        \"samples\": [\n          126.0,\n          85.0,\n          162.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"total_phenols\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.6258510488339891,\n        \"min\": 0.98,\n        \"max\": 3.88,\n        \"num_unique_values\": 97,\n        \"samples\": [\n          1.68,\n          2.11,\n          1.35\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"flavanoids\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.9988586850169465,\n        \"min\":
```

```
0.34,\n        \"max\": 5.08,\n        \"num_unique_values\": 132,\n
\"samples\": [\n          3.18,\n          2.5,\n          3.17\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"nonflavanoid_phenols\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.12445334029667939,\n        \"min\": 0.13,\n        \"max\": 0.66,\n
\"num_unique_values\": 39,\n        \"samples\": [\n          0.58,\n
0.41,\n          0.39\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"proanthocyanins\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.5723588626747611,\n        \"min\":
0.41,\n        \"max\": 3.58,\n        \"num_unique_values\": 101,\n
\"samples\": [\n          0.75,\n          1.77,\n          1.42\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"color_intensity\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2.318285871822413,\n        \"min\": 1.28,\n        \"max\": 13.0,\n
\"num_unique_values\": 132,\n        \"samples\": [\n          2.95,\n
3.3,\n          5.1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"hue\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.22857156582982338,\n        \"min\": 0.48,\n
\"max\": 1.71,\n        \"num_unique_values\": 78,\n
\"samples\": [\n          1.22,\n          1.04,\n          1.45\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"od280/od315_of_diluted_wines\",\
n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.7099904287650505,\n        \"min\": 1.27,\n        \"max\":
4.0,\n        \"num_unique_values\": 122,\n        \"samples\": [\n
4.0,\n          1.82,\n          1.59\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"proline\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
314.9074742768489,\n        \"min\": 278.0,\n        \"max\": 1680.0,\
n        \"num_unique_values\": 121,\n        \"samples\": [\n
1375.0,\n          1270.0,\n          735.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"target\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 0,\n
\"min\": 0,\n        \"max\": 2,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0,\n          1,\n          2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```


```
print(df.shape)
```

```
(178, 14)
```

There are 178 samples and 13 features

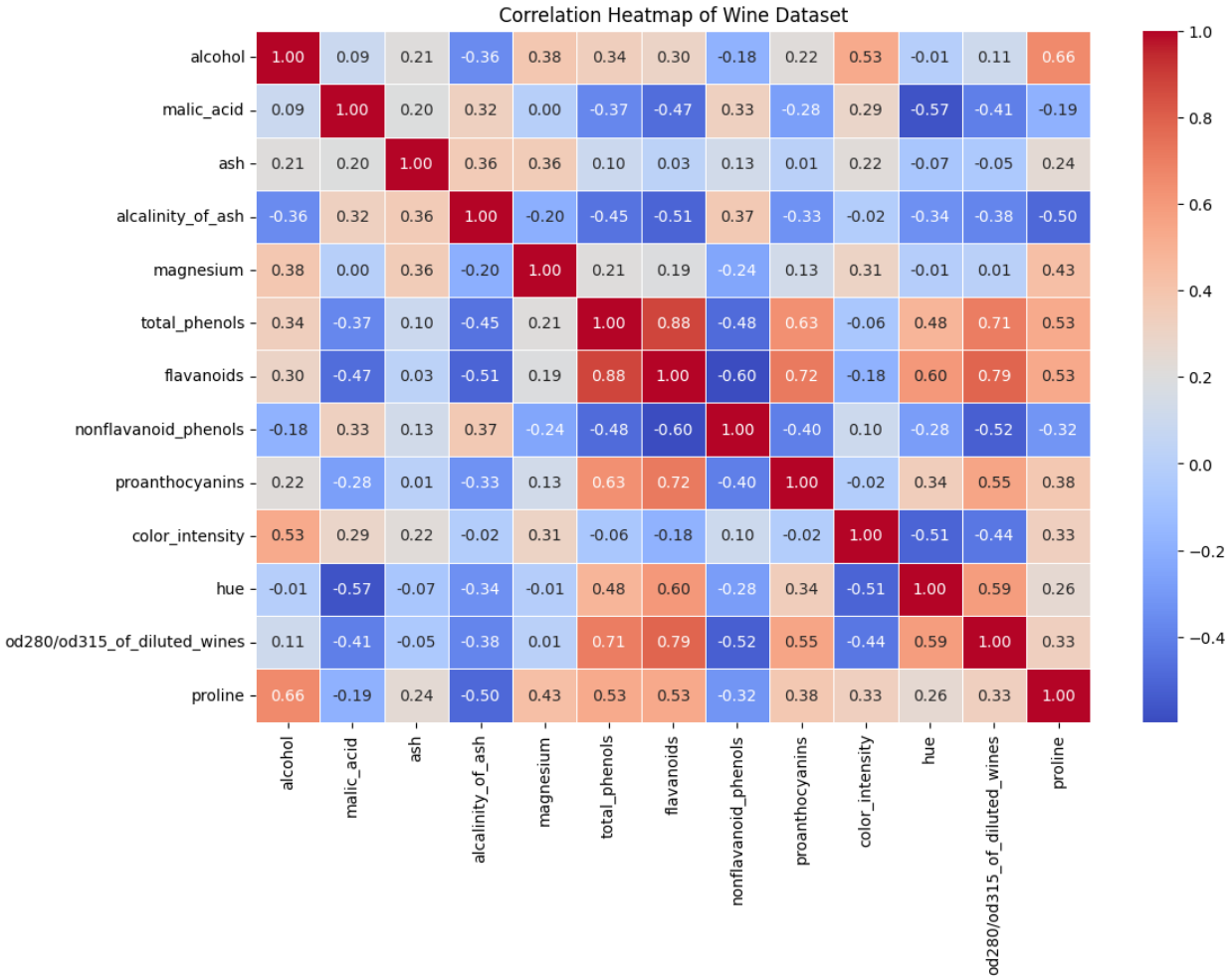
```
z_scores = np.abs(stats.zscore(df.drop("class", axis=1)))

df_no_outliers = df[(z_scores < 3).all(axis=1)]

print("Original shape:", df.shape)
print("After removing outliers (Z-score):", df_no_outliers.shape)
```

```
Original shape: (178, 14)
After removing outliers (Z-score): (168, 14)
```

```
X = df_no_outliers.drop("class",axis=1)
y = df_no_outliers["class"]
```

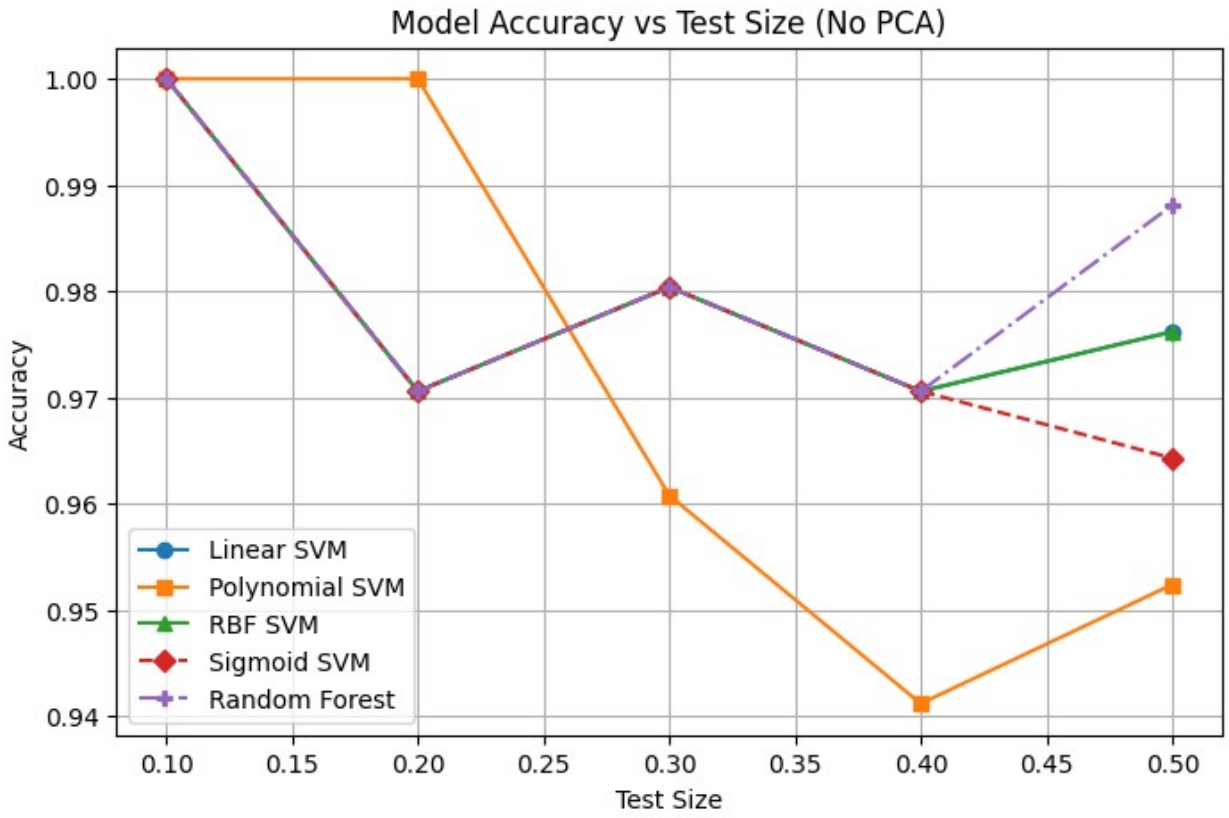
Heatmap

```
corr = X.corr()

# Plot heatmap
plt.figure(figsize=(12,8))
sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f",
linewidths=0.5)
plt.title("Correlation Heatmap of Wine Dataset")
plt.show()
```



Testing model accuracy at different splits

```
analyse_train_test_split(X, y)
```



Conclusion:

All the classifiers are working best at 10% split

Note: The above analysis was done without any hyperparameter tuning

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.1, shuffle=True, random_state=10, stratify=y
)
```

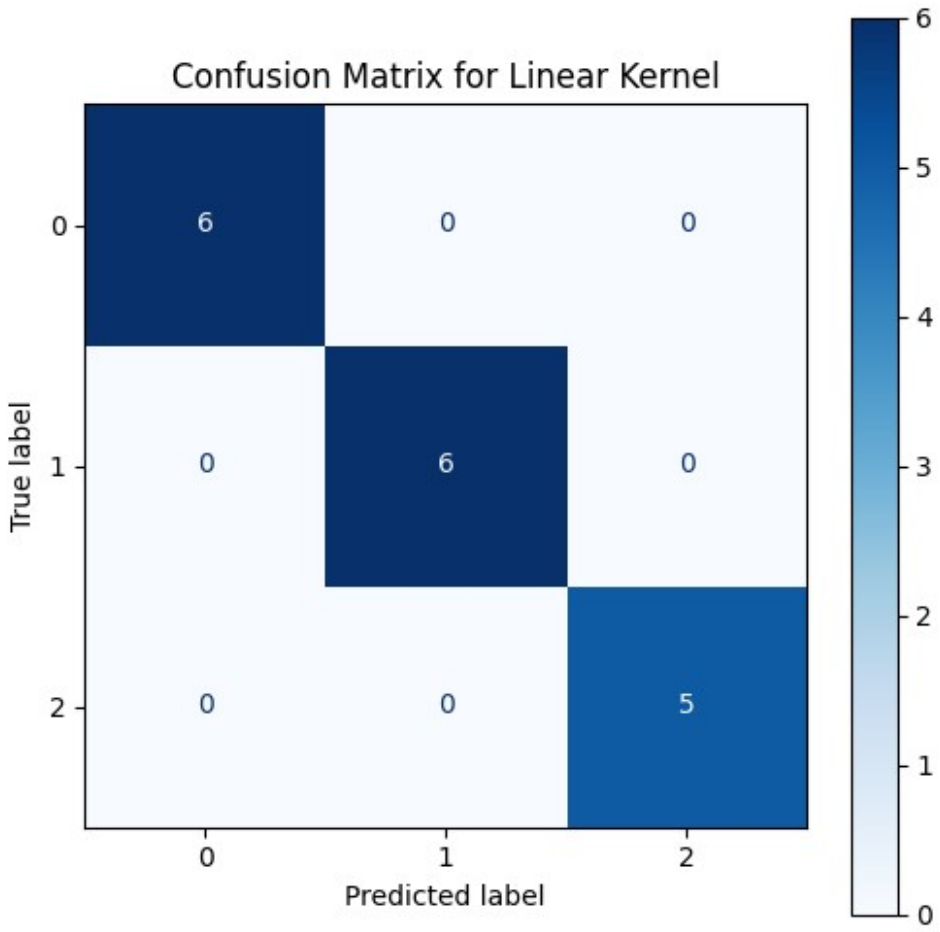
Normalizing the data

```
scale = StandardScaler()
scale.fit(X_train)
X_train = scale.transform(X_train)
X_test = scale.transform(X_test)
```

All SVMs

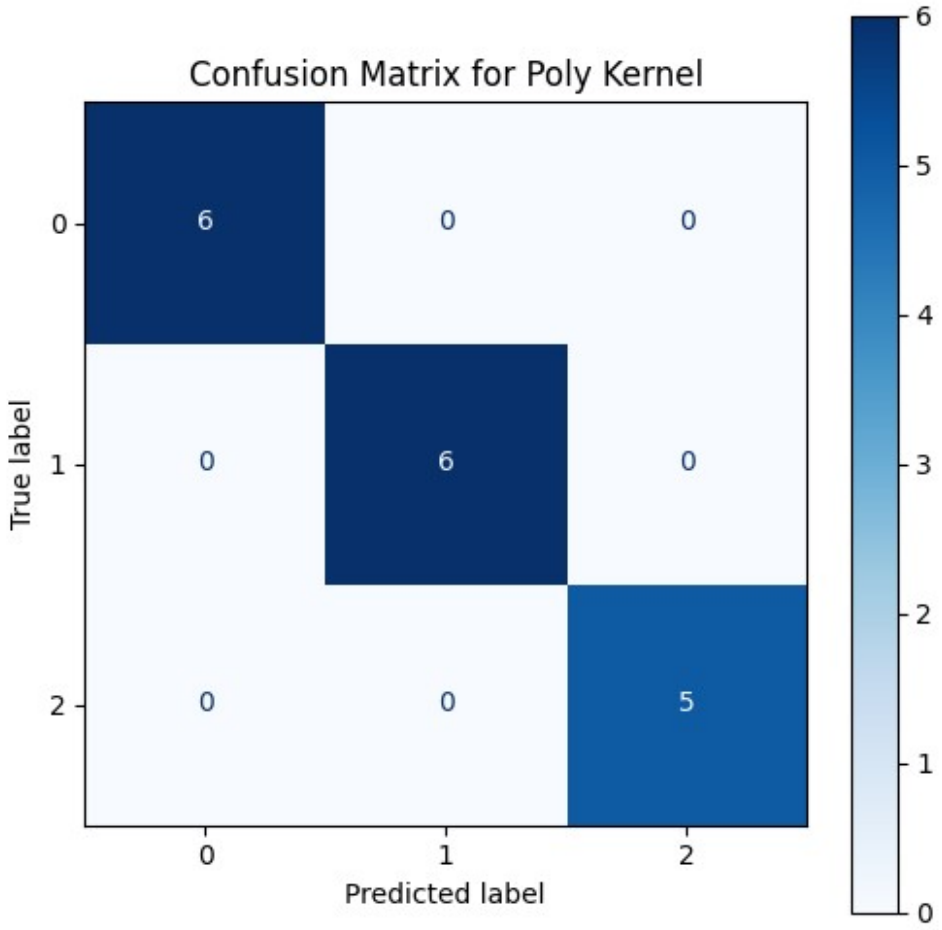
```
def svm_train():
    models = ["linear", "poly", "rbf", "sigmoid"]

    for model_name in models:
```

```
        print(f"\n{'=' * 20} Model: {model_name.upper()} {'=' * 20}\
n")

        svc = SVC(random_state=10, kernel=model_name)
        svc.fit(X_train, y_train)

        y_pred = svc.predict(X_test)


        print(" CONFUSION MATRIX ".center(80, "="))
        fig, ax = plt.subplots(figsize=(6, 6))
        disp = ConfusionMatrixDisplay.from_estimator(svc, X_test,
y_test, ax=ax, cmap=plt.cm.Blues)
        # disp.plot()
        plt.title(f'Confusion Matrix for {model_name.capitalize()}
Kernel')
        plt.show()
        print("=" * 80)


        print(" CLASSIFICATION REPORT ".center(80, "="))
        print(classification_report(y_test, y_pred))
        print("=" * 80)


        print(" ACCURACY SCORE ".center(80, "="))
        print(f"{accuracy_score(y_test, y_pred)}")
        print("=" * 80)

svm_train()


==================== Model: LINEAR ====================

================================ CONFUSION MATRIX
================================
```



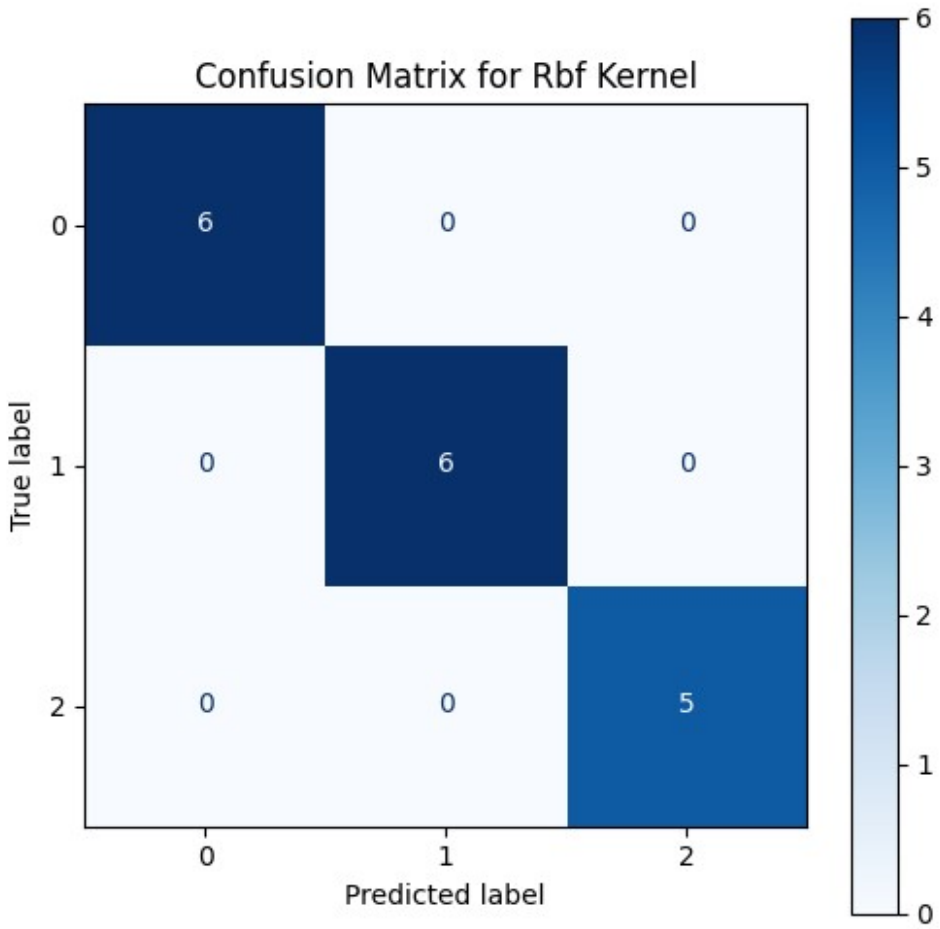

```
==========================================================================
==========
============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================== ACCURACY SCORE
==================================
1.0
==========================================================================
==========
```

```
==================== Model: POLY ====================

============================== CONFUSION MATRIX
==============================
```



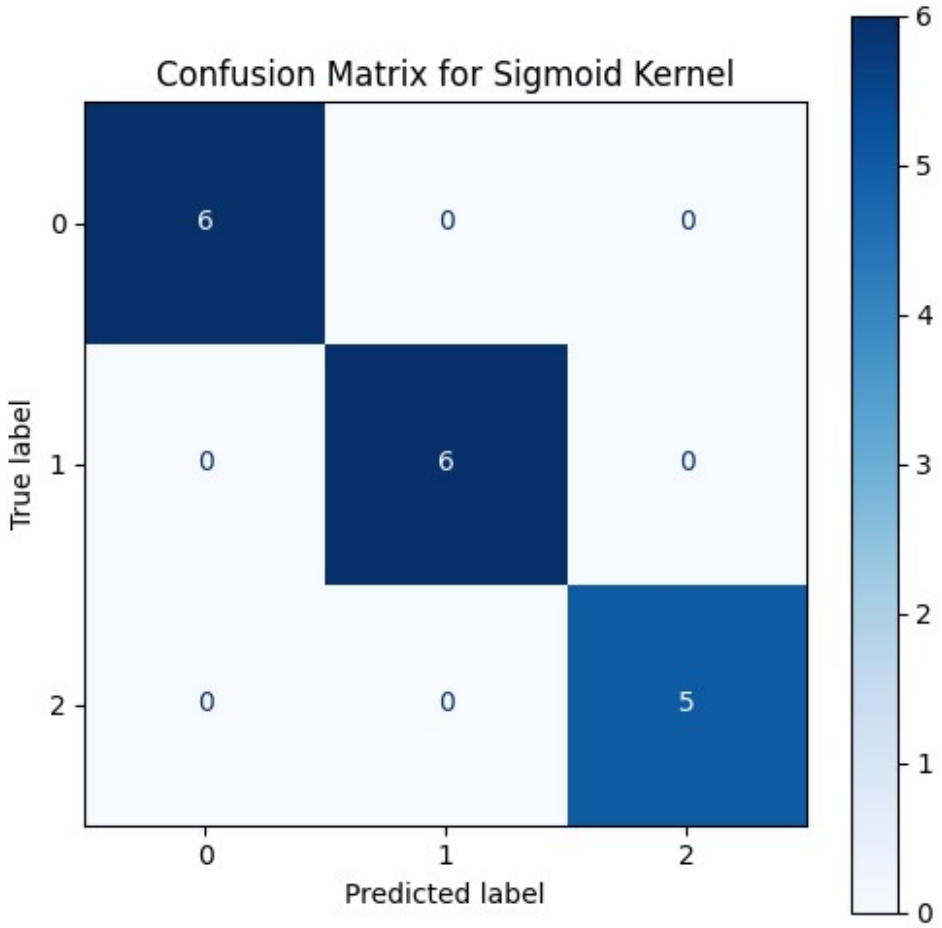

```
==========================================================================
==========
============================ CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17
```

```
================================================================================
==========
================================ ACCURACY SCORE
================================
1.0
================================================================================
==========

==================== Model: RBF ====================

================================ CONFUSION MATRIX
================================
```




```
================================================================================
==========
============================ CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
```

```
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================== ACCURACY SCORE
==================================
1.0
==========================================================================
==========

==================== Model: SIGMOID ====================

================================== CONFUSION MATRIX
==================================
```



```
================================================================================
==========
============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

================================================================================
==========
================================== ACCURACY SCORE
==================================
1.0
================================================================================
==========
```

```python
def plot_roc_curve(clf, X_train, X_test, y_train, y_test,
clf_name="Classifier"):
    """Handles both binary and multiclass ROC plotting."""
    clf.fit(X_train, y_train)

    # Get decision function or probabilities
    if hasattr(clf, "predict_proba"):
        y_scores = clf.predict_proba(X_test)
    else:
        y_scores = clf.decision_function(X_test)

    n_classes = len(np.unique(y_test))

    # Binary Classification
    if n_classes == 2:
        if y_scores.ndim > 1:
            y_scores = y_scores[:, 1]
        fpr, tpr, _ = roc_curve(y_test, y_scores)
        roc_auc = auc(fpr, tpr)

        plt.figure(figsize=(6, 5))
        plt.plot(fpr, tpr, label=f"{clf_name} (AUC = {roc_auc:.2f})")
        plt.plot([0, 1], [0, 1], "k--")
        plt.xlabel("False Positive Rate")
        plt.ylabel("True Positive Rate")
        plt.title(f"ROC Curve - {clf_name}")
        plt.legend(loc="lower right")
        plt.grid()
```

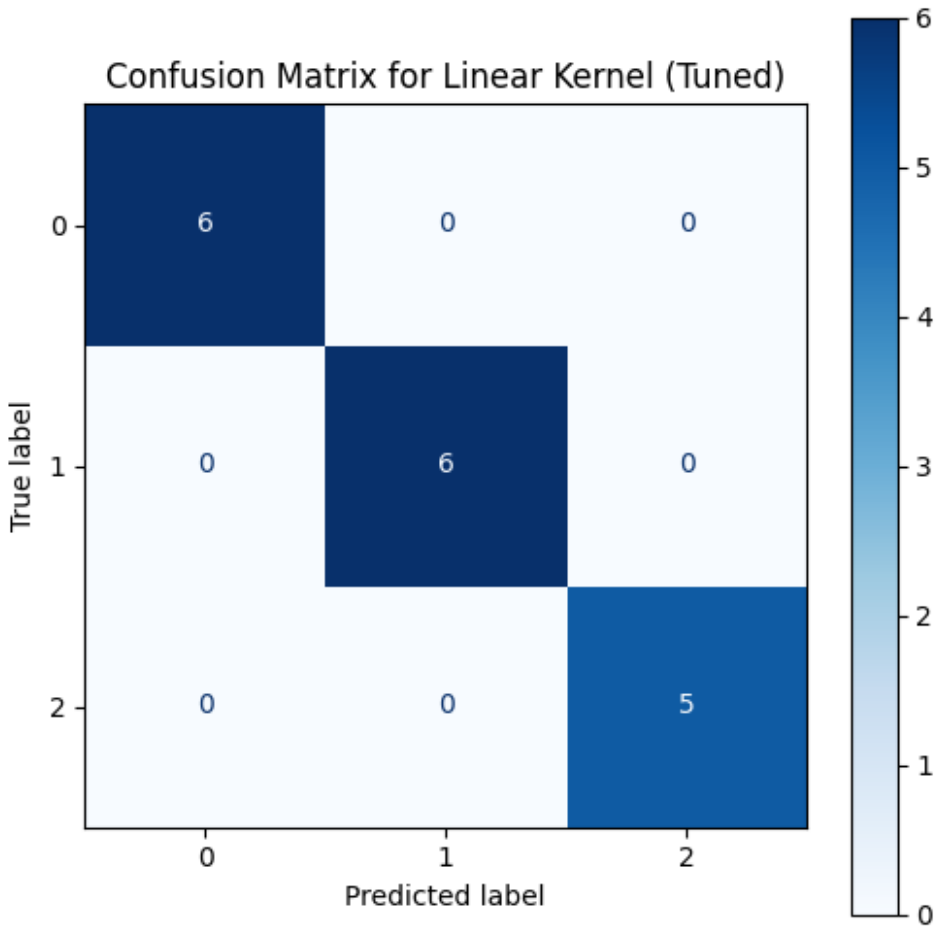
```
        plt.show()

    else:
        y_test_bin = label_binarize(y_test, classes=np.unique(y_test))

        plt.figure(figsize=(7, 6))
        for i in range(n_classes):
            fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_scores[:, i])
            roc_auc = auc(fpr, tpr)
            plt.plot(fpr, tpr, label=f"Class {i} (AUC =
{roc_auc:.2f})")

        plt.plot([0, 1], [0, 1], "k--")
        plt.xlabel("False Positive Rate")
        plt.ylabel("True Positive Rate")
        plt.title(f"ROC Curve - {clf_name} (Multi-class)")
        plt.legend(loc="lower right")
        plt.grid()
        plt.show()
```

Hyperparameter Tuning

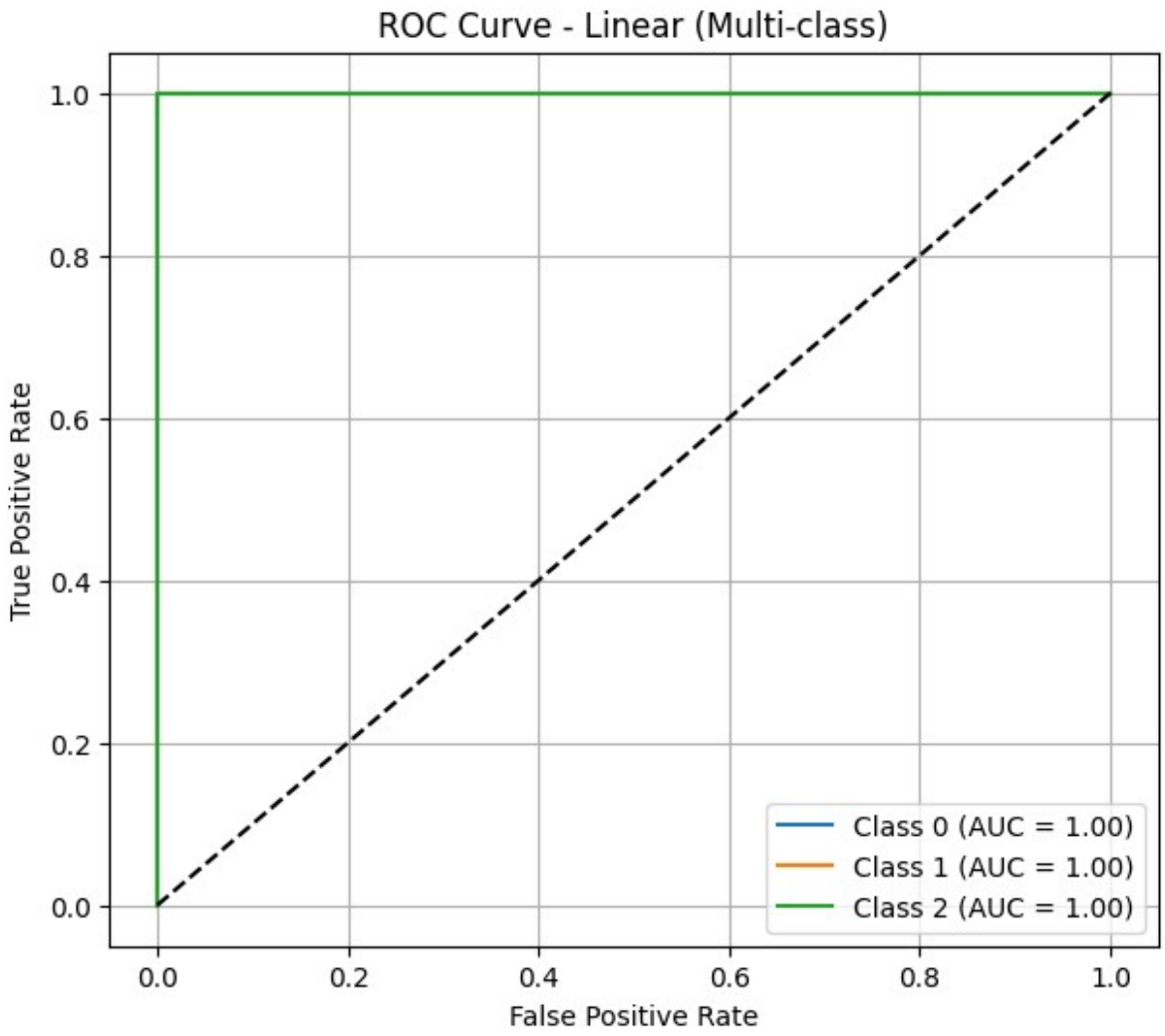
```
def svm_train_tuned():
    models = {
        "linear": {"C": [0.1, 1, 10, 100]},
        "poly": {"C": [0.1, 1, 10], "degree": [2, 3, 4], "gamma":
["scale", "auto"]},
        "rbf": {"C": [0.1, 1, 10, 100], "gamma": ["scale", "auto",
0.01, 0.1, 1]},
        "sigmoid": {"C": [0.1, 1, 10], "gamma": ["scale", "auto"]}
    }

    for model_name, param_grid in models.items():
        print(f"\n{'=' * 20} Model: {model_name.upper()} {'=' * 20}\
n")

        svc = SVC(kernel=model_name, random_state=10,
probability=True)  # □ probability=True ensures predict_proba
        grid = GridSearchCV(svc, param_grid, cv=5, n_jobs=-1,
scoring="accuracy")
        grid.fit(X_train, y_train)

        best_model = grid.best_estimator_
        y_pred = best_model.predict(X_test)

        print(f"Best Parameters: {grid.best_params_}\n")

        # Confusion Matrix
        print(" CONFUSION MATRIX ".center(80, "="))
        fig, ax = plt.subplots(figsize=(6, 6))
```

```
        ConfusionMatrixDisplay.from_estimator(best_model, X_test,
y_test, ax=ax, cmap=plt.cm.Blues)
        plt.title(f'Confusion Matrix for {model_name.capitalize()}
Kernel (Tuned)')
        plt.show()
        print("=" * 80)

        # Classification Report
        print(" CLASSIFICATION REPORT ".center(80, "="))
        print(classification_report(y_test, y_pred))
        print("=" * 80)

        # Accuracy
        print(" ACCURACY SCORE ".center(80, "="))
        print(f"{accuracy_score(y_test, y_pred)}")
        print("=" * 80)

        # ROC Curve (works for both binary & multiclass)
        plot_roc_curve(best_model, X_train, X_test, y_train, y_test,
clf_name=model_name.capitalize())

svm_train_tuned()
```

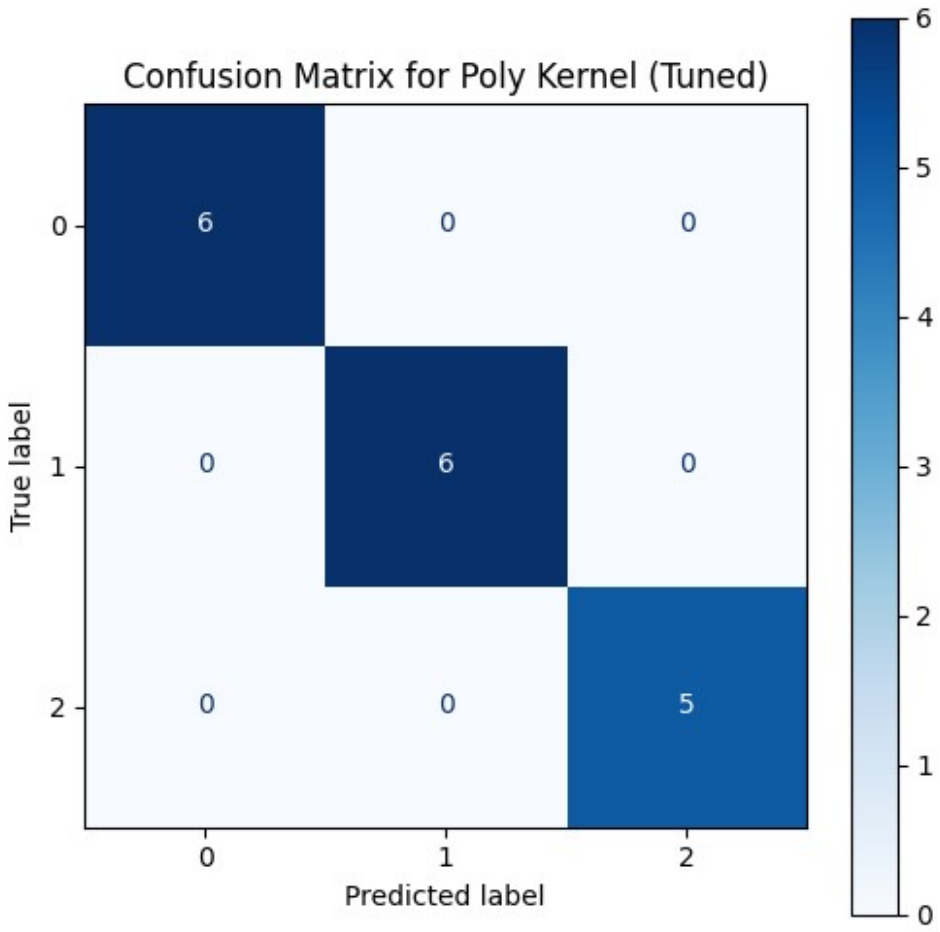
```
==================== Model: LINEAR ====================

Best Parameters: {'C': 0.1}

============================== CONFUSION MATRIX
==============================
```



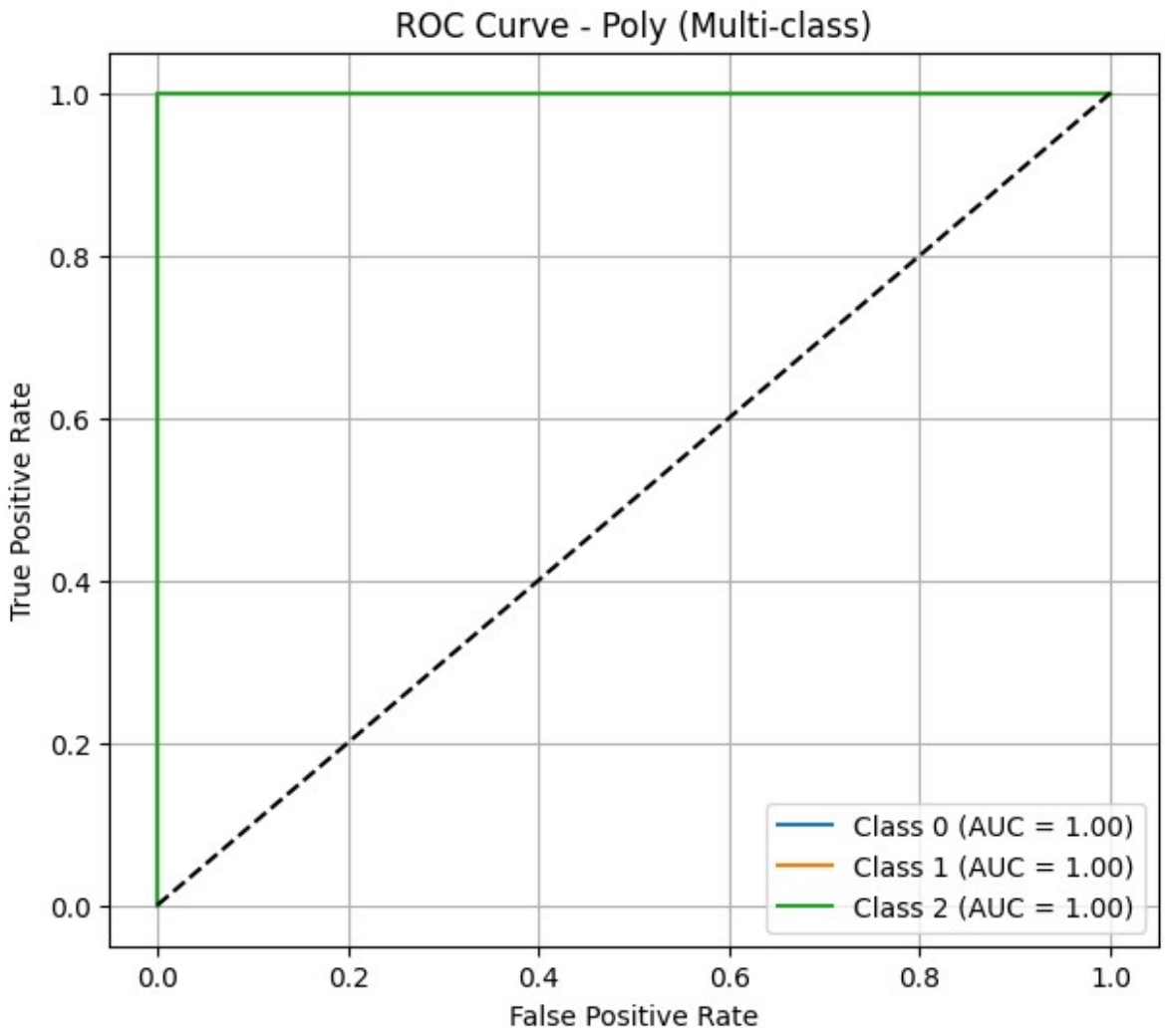

```
==========================================================================
==========
============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================== ACCURACY SCORE
==================================
1.0
==========================================================================
==========
```



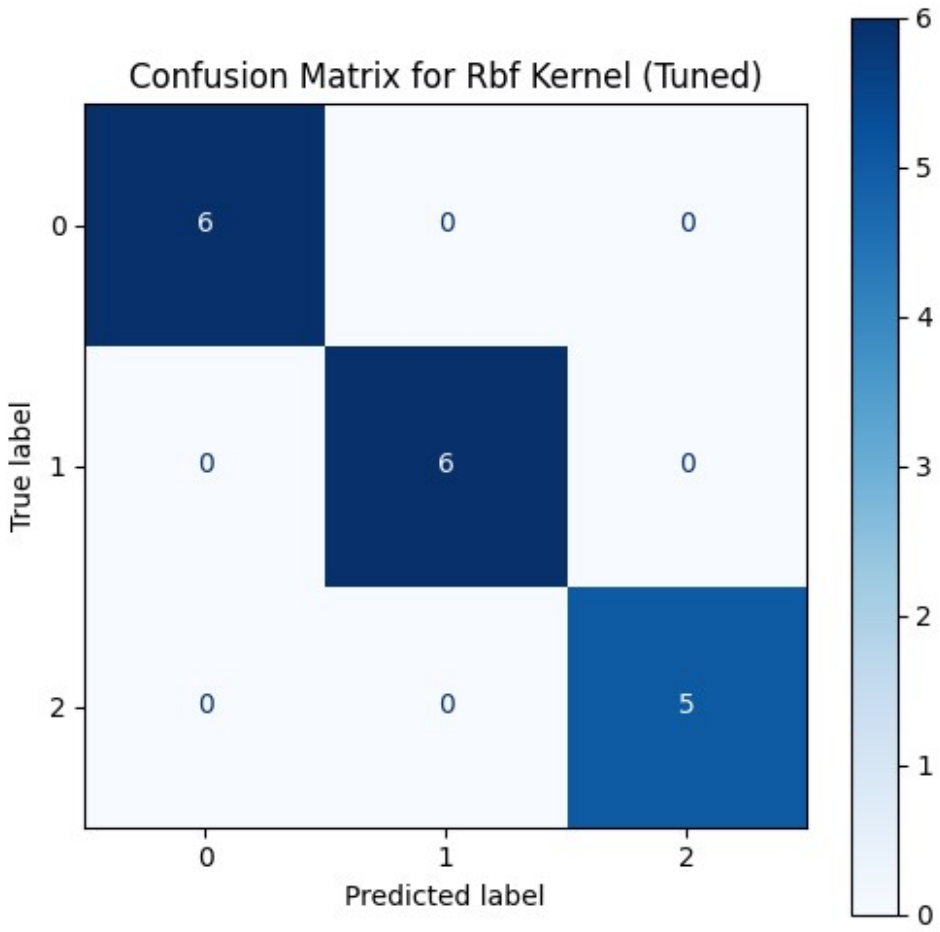

```
==================== Model: POLY ====================

Best Parameters: {'C': 1, 'degree': 3, 'gamma': 'scale'}

================================ CONFUSION MATRIX
================================
```



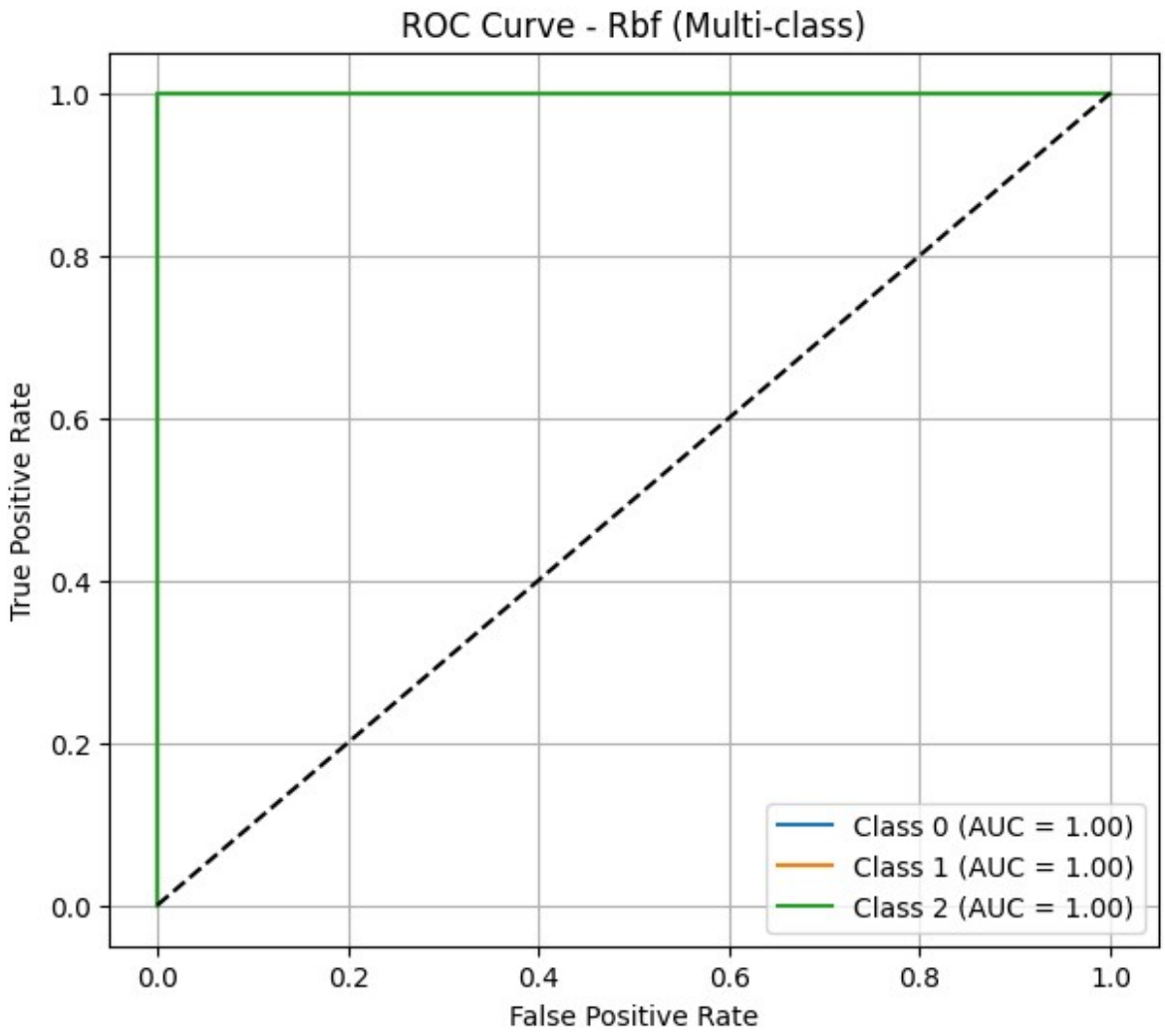

```
==========================================================================
==========
============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================== ACCURACY SCORE
==================================
1.0
==========================================================================
==========
```



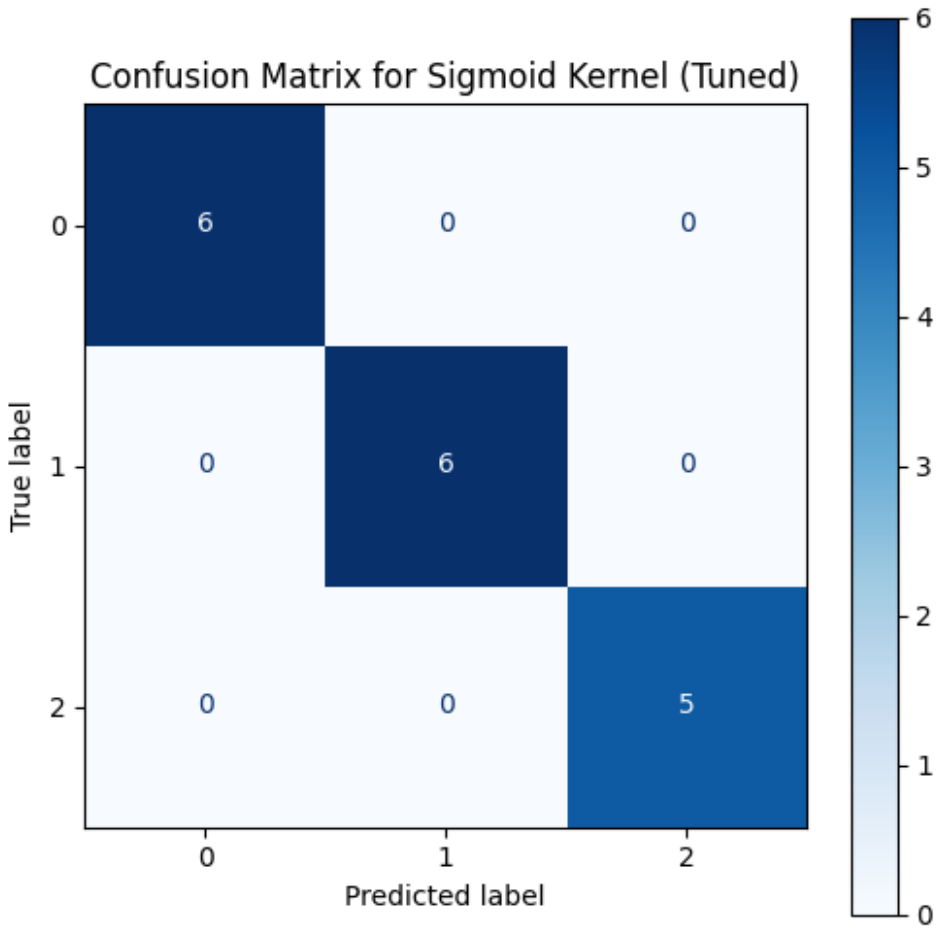

```
==================== Model: RBF ====================

Best Parameters: {'C': 1, 'gamma': 0.01}

================================ CONFUSION MATRIX
================================
```



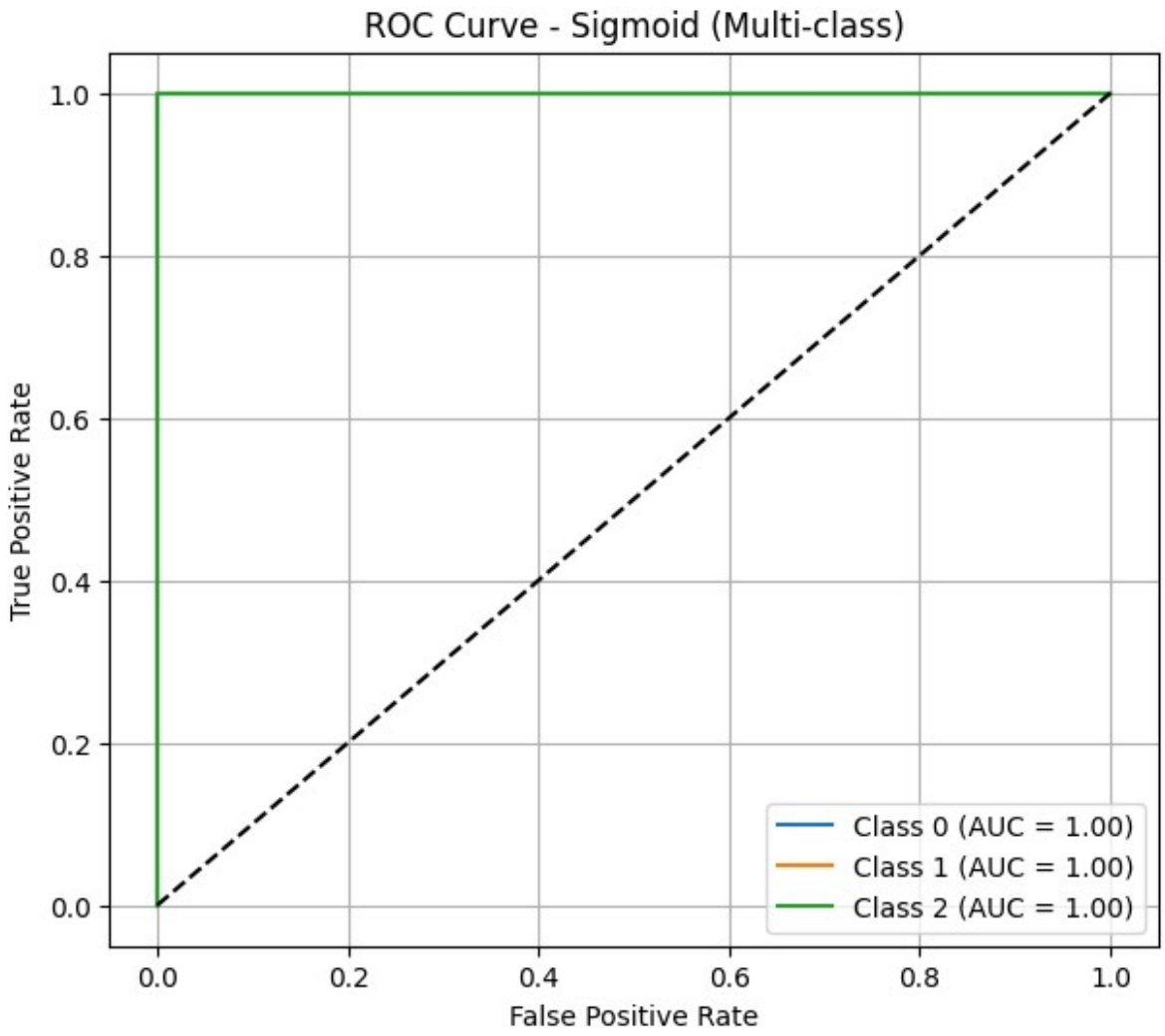

```
==========================================================================
==========
============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================== ACCURACY SCORE
==================================
1.0
==========================================================================
==========
```




```
==================== Model: SIGMOID ====================

Best Parameters: {'C': 0.1, 'gamma': 'scale'}

================================ CONFUSION MATRIX
================================
```




```
==========================================================================
==========
============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================== ACCURACY SCORE
==================================
1.0
==========================================================================
==========
```

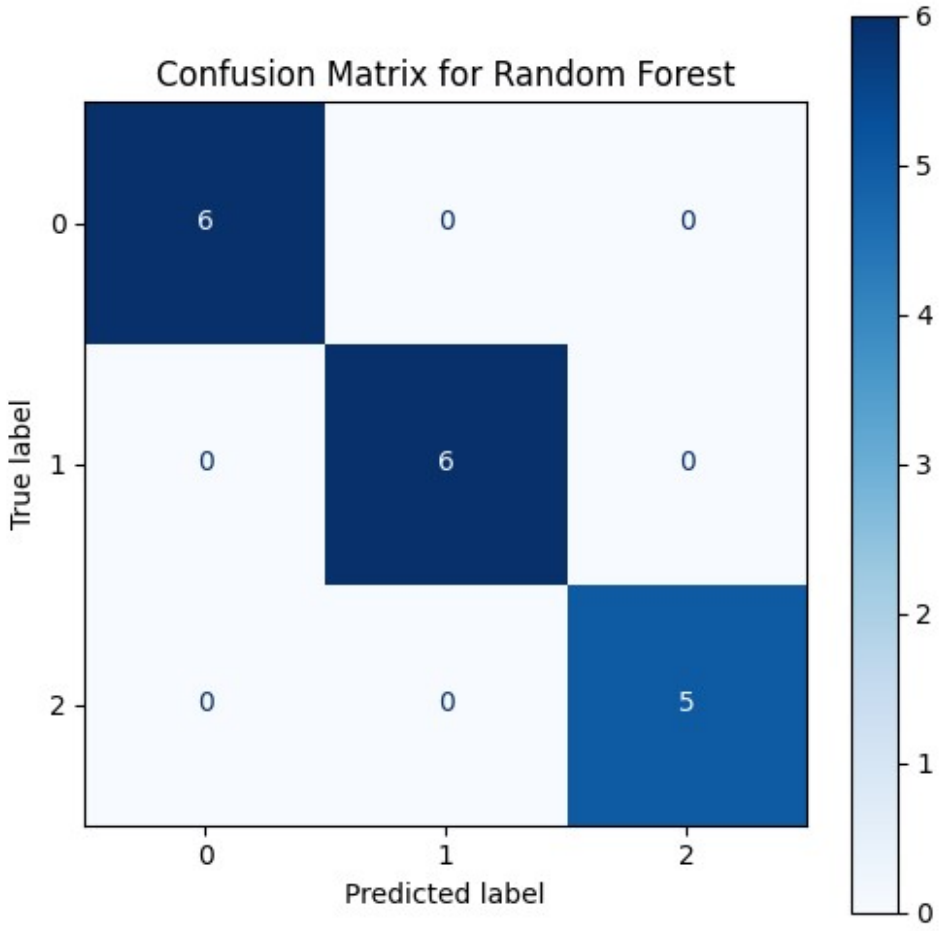


Random Forest

```
rf = RandomForestClassifier(n_estimators=100, random_state=10)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

print(" CONFUSION MATRIX ".center(80, "="))
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(rf, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
plt.title('Confusion Matrix for Random Forest')
plt.show()
print("=" * 80)

print(" CLASSIFICATION REPORT ".center(80, "="))
print(classification_report(y_test, y_pred))
```

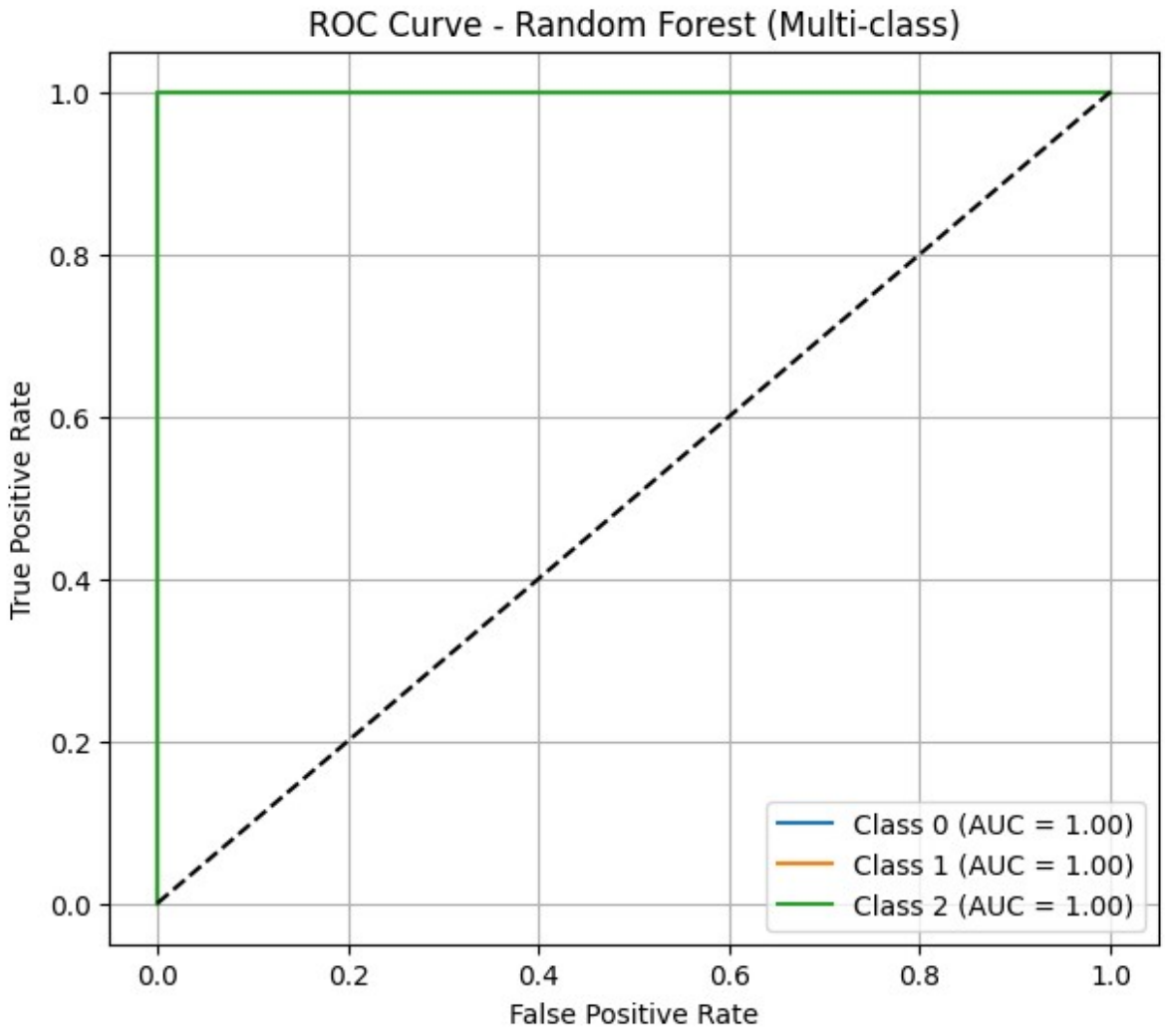
```
print("=" * 80)

print(" ACCURACY SCORE ".center(80, "="))
print(f"{accuracy_score(y_test, y_pred)}")
print("=" * 80)

plot_roc_curve(rf, X_train, X_test, y_train, y_test, clf_name="Random
Forest")
```

```
================================ CONFUSION MATRIX
================================
```




```
================================================================================
==========
============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
```

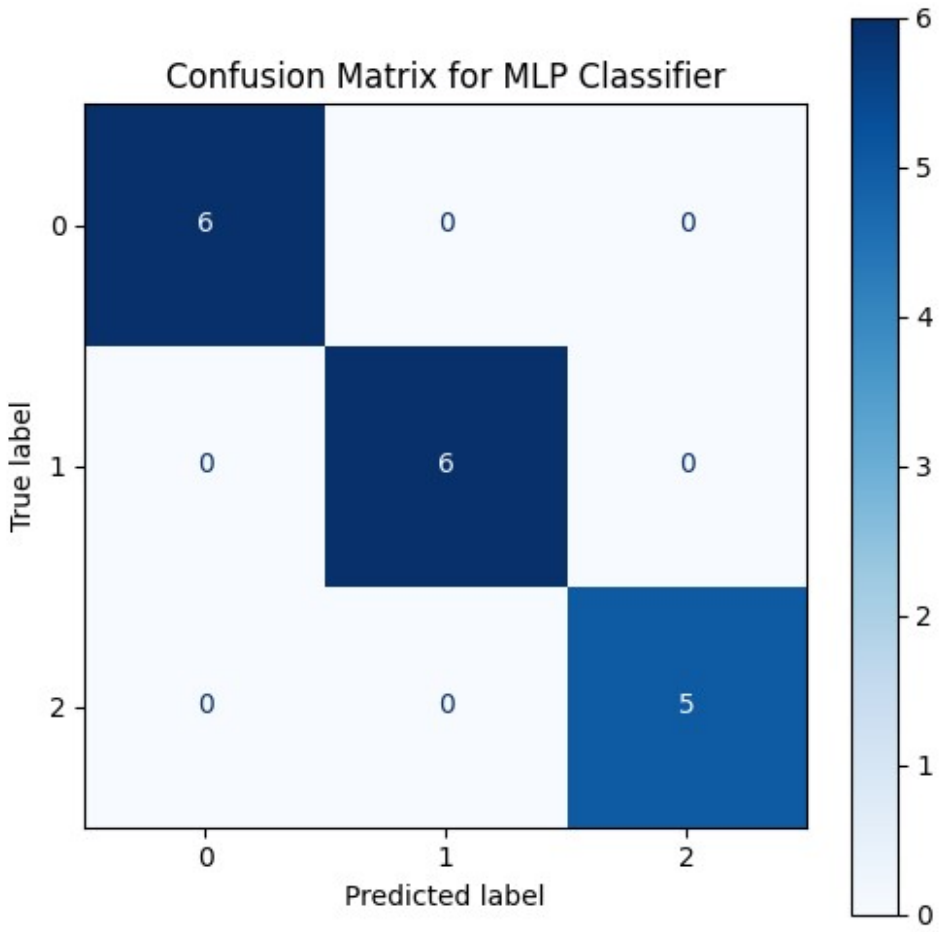
```
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================= ACCURACY SCORE
==================================
1.0
==========================================================================
==========
```



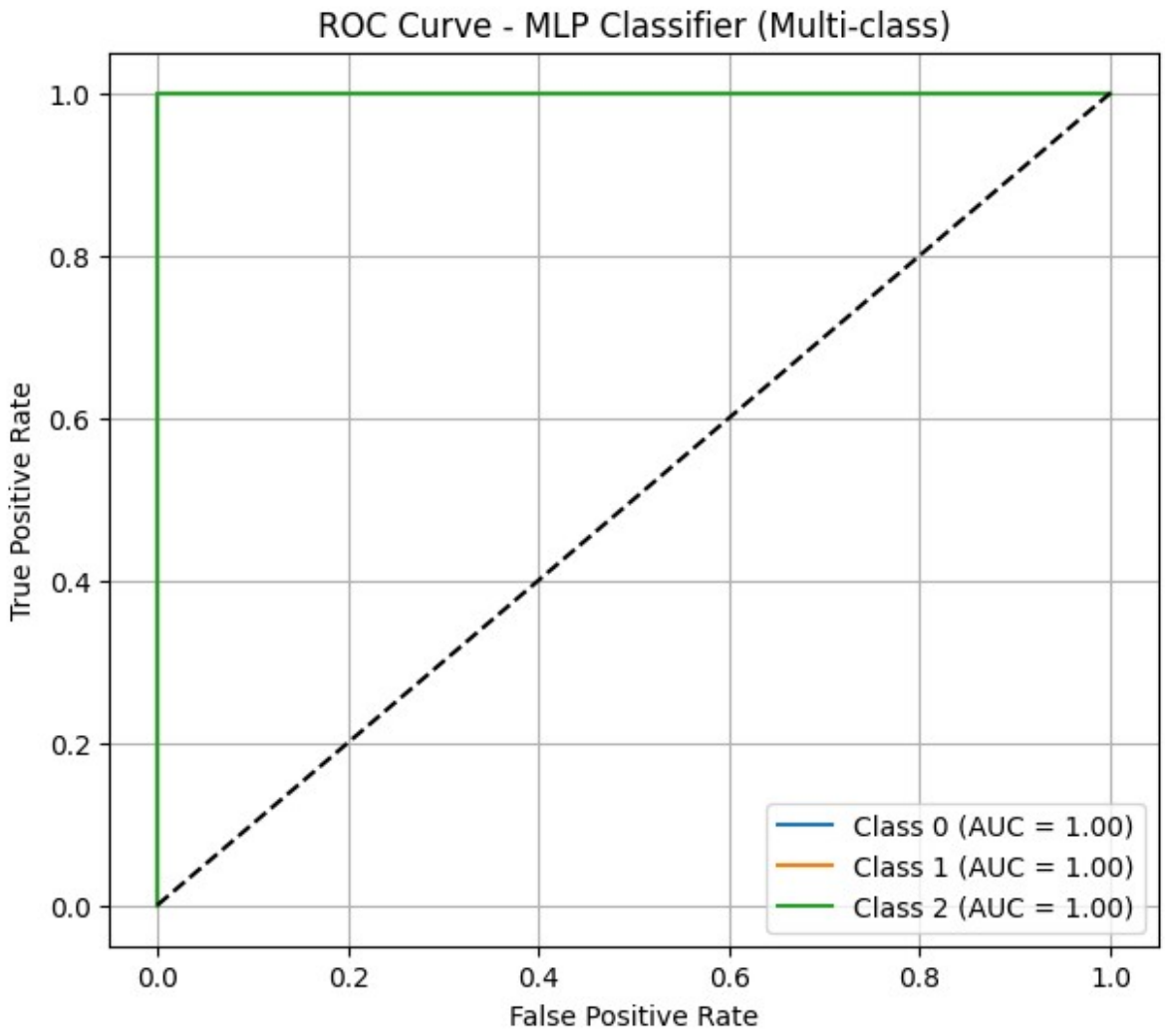
MLP

```
# Simple MLP Classifier Training and Evaluation

from sklearn.metrics import classification_report, accuracy_score,
ConfusionMatrixDisplay
import matplotlib.pyplot as plt

mlp = MLPClassifier(hidden_layer_sizes=(100,), max_iter=500,
random_state=10)
mlp.fit(X_train, y_train)
y_pred = mlp.predict(X_test)

print(" CONFUSION MATRIX ".center(80, "="))
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(mlp, X_test, y_test, ax=ax,
cmap=plt.cm.Blues)
plt.title('Confusion Matrix for MLP Classifier')
plt.show()
print("=" * 80)

print(" CLASSIFICATION REPORT ".center(80, "="))
print(classification_report(y_test, y_pred))
print("=" * 80)

print(" ACCURACY SCORE ".center(80, "="))
print(f"{accuracy_score(y_test, y_pred)}")
print("=" * 80)

plot_roc_curve(mlp, X_train, X_test, y_train, y_test, clf_name="MLP
Classifier")
```

```
============================== CONFUSION MATRIX
==============================
```




```
==========================================================================
==========
============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================== ACCURACY SCORE
==================================
1.0
==========================================================================
==========
```



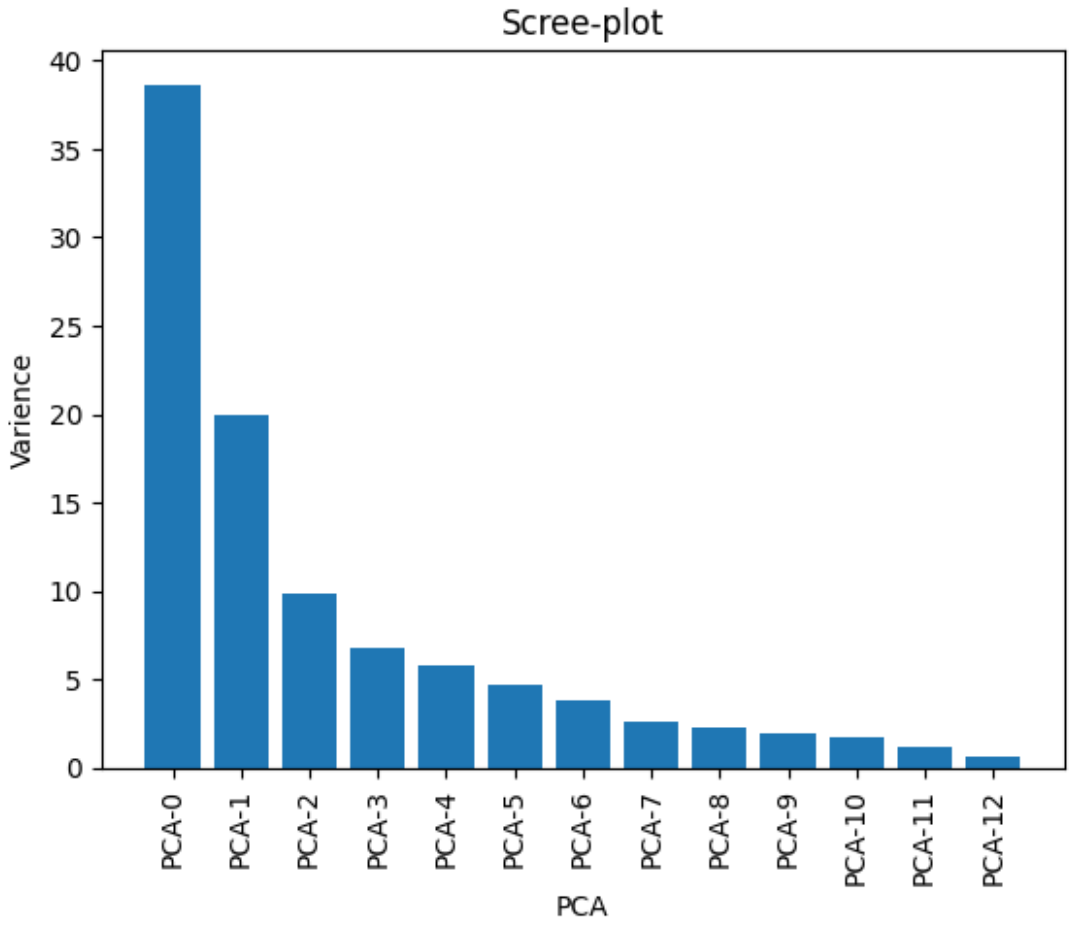
Has already attained 100% accuracy. Hence, fine Tuning is not needed.

Principal Component Analysis

```
pca = PCA()
pca.fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

captured_info = [var*100 for var in pca.explained_variance_ratio_]

pca_names = [f"PCA-{i}" for i in range(0,len(captured_info))]
print(pca_names)
```

```
['PCA-0', 'PCA-1', 'PCA-2', 'PCA-3', 'PCA-4', 'PCA-5', 'PCA-6', 'PCA-
7', 'PCA-8', 'PCA-9', 'PCA-10', 'PCA-11', 'PCA-12']
```

```
plt.bar(pca_names, captured_info)
plt.title('Scree-plot')
plt.xlabel('PCA')
plt.ylabel('Varience')

plt.xticks(rotation=90)

plt.show()
```




```
sum = 0
for i in range(8):
    sum+=captured_info[i]

print(sum)
```

```
92.2273015662882
```

Support Vector Machines

```
def svm_train_pca(X_train_pca, X_test_pca, y_train, y_test):
    models = ["linear", "poly", "rbf", "sigmoid"]

    for model_name in models: # Renamed 'model' to 'model_name' to
avoid confusion with the module
        print(f"\n{'=' * 20} Model: {model_name.upper()} {'=' * 20}\
n")

        svc = SVC(random_state=10, kernel=model_name)
        svc.fit(X_train_pca, y_train)

        y_pred = svc.predict(X_test_pca)


        print(" CLASSIFICATION REPORT ".center(80, "="))
        print(classification_report(y_test, y_pred))
        print("=" * 80)


        print(" ACCURACY SCORE ".center(80, "="))
        print(f"{accuracy_score(y_test, y_pred)}")
        print("=" * 80)

for i in range(1,9):
    pca = PCA(n_components=i)
    pca.fit(X_train)
    X_train_pca = pca.transform(X_train)
    X_test_pca = pca.transform(X_test)
    print(f"for {i} principal components:")
    svm_train_pca(X_train_pca, X_test_pca, y_train, y_test)
```

```
for 1 principal components:

==================== Model: LINEAR ====================

============================ CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      0.83      0.91         6
           1       0.75      1.00      0.86         6
           2       1.00      0.80      0.89         5

    accuracy                           0.88        17
   macro avg       0.92      0.88      0.89        17
weighted avg       0.91      0.88      0.88        17

================================================================================
==========
```

```
================================ ACCURACY SCORE
================================
0.8823529411764706
================================================================================
==========

==================== Model: POLY ====================

============================ CLASSIFICATION REPORT
============================
              precision    recall  f1-score   support

           0       1.00      0.83      0.91         6
           1       0.75      1.00      0.86         6
           2       1.00      0.80      0.89         5

    accuracy                           0.88        17
   macro avg       0.92      0.88      0.89        17
weighted avg       0.91      0.88      0.88        17

================================================================================
==========
================================ ACCURACY SCORE
================================
0.8823529411764706
================================================================================
==========

==================== Model: RBF ====================

============================ CLASSIFICATION REPORT
============================
              precision    recall  f1-score   support

           0       1.00      0.83      0.91         6
           1       0.75      1.00      0.86         6
           2       1.00      0.80      0.89         5

    accuracy                           0.88        17
   macro avg       0.92      0.88      0.89        17
weighted avg       0.91      0.88      0.88        17

================================================================================
==========
================================ ACCURACY SCORE
================================
0.8823529411764706
================================================================================
==========
```

```
==================== Model: SIGMOID ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       0.86      1.00      0.92         6
           1       1.00      0.50      0.67         6
           2       0.71      1.00      0.83         5

    accuracy                           0.82        17
   macro avg       0.86      0.83      0.81        17
weighted avg       0.87      0.82      0.81        17

================================================================================
==========
================================== ACCURACY SCORE
==================================
0.8235294117647058
================================================================================
==========
for 2 principal components:

==================== Model: LINEAR ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

================================================================================
==========
================================== ACCURACY SCORE
==================================
1.0
================================================================================
==========

==================== Model: POLY ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support
```

```
           0       1.00      0.83      0.91         6
           1       0.86      1.00      0.92         6
           2       1.00      1.00      1.00         5

    accuracy                           0.94        17
   macro avg       0.95      0.94      0.94        17
weighted avg       0.95      0.94      0.94        17

==========================================================================
==========
================================== ACCURACY SCORE
==================================
0.9411764705882353
==========================================================================
==========

==================== Model: RBF ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================== ACCURACY SCORE
==================================
1.0
==========================================================================
==========

==================== Model: SIGMOID ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
```

```
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================ ACCURACY SCORE
================================
1.0
==========================================================================
==========
for 3 principal components:

==================== Model: LINEAR ====================

============================ CLASSIFICATION REPORT
============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================ ACCURACY SCORE
================================
1.0
==========================================================================
==========

==================== Model: POLY ====================

============================ CLASSIFICATION REPORT
============================
              precision    recall  f1-score   support

           0       1.00      0.83      0.91         6
           1       0.86      1.00      0.92         6
           2       1.00      1.00      1.00         5

    accuracy                           0.94        17
   macro avg       0.95      0.94      0.94        17
weighted avg       0.95      0.94      0.94        17

==========================================================================
==========
```

```
================================ ACCURACY SCORE
================================
0.9411764705882353
================================================================================
==========

==================== Model: RBF ====================

============================ CLASSIFICATION REPORT
============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

================================================================================
==========
================================ ACCURACY SCORE
================================
1.0
================================================================================
==========

==================== Model: SIGMOID ====================

============================ CLASSIFICATION REPORT
============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

================================================================================
==========
================================ ACCURACY SCORE
================================
1.0
================================================================================
==========
for 4 principal components:
```

```
==================== Model: LINEAR ====================

============================ CLASSIFICATION REPORT
============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

======================================================================
==========
================================ ACCURACY SCORE
================================
1.0
======================================================================
==========

==================== Model: POLY ====================

============================ CLASSIFICATION REPORT
============================
              precision    recall  f1-score   support

           0       1.00      0.83      0.91         6
           1       0.86      1.00      0.92         6
           2       1.00      1.00      1.00         5

    accuracy                           0.94        17
   macro avg       0.95      0.94      0.94        17
weighted avg       0.95      0.94      0.94        17

======================================================================
==========
================================ ACCURACY SCORE
================================
0.9411764705882353
======================================================================
==========

==================== Model: RBF ====================

============================ CLASSIFICATION REPORT
============================
              precision    recall  f1-score   support
```

```
           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

======================================================================
==========
================================== ACCURACY SCORE
==================================
1.0
======================================================================
==========

==================== Model: SIGMOID ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

======================================================================
==========
================================== ACCURACY SCORE
==================================
1.0
======================================================================
==========
for 5 principal components:

==================== Model: LINEAR ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5
```

```
    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

======================================================================
==========
================================== ACCURACY SCORE
==================================
1.0
======================================================================
==========

==================== Model: POLY ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

======================================================================
==========
================================== ACCURACY SCORE
==================================
1.0
======================================================================
==========

==================== Model: RBF ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

======================================================================
==========
```

```
================================= ACCURACY SCORE
=================================
1.0
================================================================================
==========

==================== Model: SIGMOID ====================

============================= CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

================================================================================
==========
================================= ACCURACY SCORE
=================================
1.0
================================================================================
==========
for 6 principal components:

==================== Model: LINEAR ====================

============================= CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

================================================================================
==========
================================= ACCURACY SCORE
=================================
1.0
================================================================================
==========
```

```
==================== Model: POLY ====================

============================ CLASSIFICATION REPORT
============================
              precision    recall  f1-score   support

           0       1.00      0.83      0.91         6
           1       0.86      1.00      0.92         6
           2       1.00      1.00      1.00         5

    accuracy                           0.94        17
   macro avg       0.95      0.94      0.94        17
weighted avg       0.95      0.94      0.94        17

==========================================================================
==========
================================ ACCURACY SCORE
================================
0.9411764705882353
==========================================================================
==========

==================== Model: RBF ====================

============================ CLASSIFICATION REPORT
============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================ ACCURACY SCORE
================================
1.0
==========================================================================
==========

==================== Model: SIGMOID ====================

============================ CLASSIFICATION REPORT
============================
              precision    recall  f1-score   support
```

```
           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================ ACCURACY SCORE
================================
1.0
==========================================================================
==========
for 7 principal components:

==================== Model: LINEAR ====================

============================ CLASSIFICATION REPORT
============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================ ACCURACY SCORE
================================
1.0
==========================================================================
==========

==================== Model: POLY ====================

============================ CLASSIFICATION REPORT
============================
              precision    recall  f1-score   support

           0       1.00      0.83      0.91         6
           1       0.86      1.00      0.92         6
           2       1.00      1.00      1.00         5

    accuracy                           0.94        17
```

```
   macro avg       0.95      0.94      0.94        17
weighted avg       0.95      0.94      0.94        17

==========================================================================
==========
================================ ACCURACY SCORE
================================
0.9411764705882353
==========================================================================
==========

==================== Model: RBF ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================ ACCURACY SCORE
================================
1.0
==========================================================================
==========

==================== Model: SIGMOID ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================ ACCURACY SCORE
```

```
================================
1.0
================================================================================
==========
for 8 principal components:

==================== Model: LINEAR ====================

============================ CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

================================================================================
==========
================================ ACCURACY SCORE
================================
1.0
================================================================================
==========

==================== Model: POLY ====================

============================ CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

================================================================================
==========
================================ ACCURACY SCORE
================================
1.0
================================================================================
==========
```

```
==================== Model: RBF ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================ ACCURACY SCORE
================================
1.0
==========================================================================
==========

==================== Model: SIGMOID ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================ ACCURACY SCORE
================================
1.0
==========================================================================
==========
```

Conclusion:

With PCA, we can use only 5 features and end up getting the same results.

Random Forest

```
def random_forest_train_pca(X_train_pca, X_test_pca, y_train, y_test):
    print(f"\n{'=' * 20} Random Forest {'=' * 20}\n")
    rf = RandomForestClassifier(n_estimators=100, random_state=10)
    rf.fit(X_train_pca, y_train)
    y_pred = rf.predict(X_test_pca)

    print(" CLASSIFICATION REPORT ".center(80, "="))
    print(classification_report(y_test, y_pred))
    print("=" * 80)

    print(" ACCURACY SCORE ".center(80, "="))
    print(f"{accuracy_score(y_test, y_pred)}")
    print("=" * 80)

for i in range(1,9):
    pca = PCA(n_components=i)
    pca.fit(X_train)
    X_train_pca = pca.transform(X_train)
    X_test_pca = pca.transform(X_test)
    print(f"for {i} principal components:")
    random_forest_train_pca(X_train_pca, X_test_pca, y_train, y_test)
```

```
for 1 principal components:

==================== Random Forest ====================

============================ CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       0.80      0.67      0.73         6
           1       0.57      0.67      0.62         6
           2       0.80      0.80      0.80         5

    accuracy                           0.71        17
   macro avg       0.72      0.71      0.71        17
weighted avg       0.72      0.71      0.71        17

================================================================================
==========
================================ ACCURACY SCORE
================================
0.7058823529411765
```

```
==========================================================================
==========
for 2 principal components:

==================== Random Forest ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================ ACCURACY SCORE
================================
1.0
==========================================================================
==========
for 3 principal components:

==================== Random Forest ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================ ACCURACY SCORE
================================
1.0
==========================================================================
==========
for 4 principal components:
```

```
==================== Random Forest ====================

============================ CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

================================================================================
==========
================================ ACCURACY SCORE
================================
1.0
================================================================================
==========
for 5 principal components:

==================== Random Forest ====================

============================ CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

================================================================================
==========
================================ ACCURACY SCORE
================================
1.0
================================================================================
==========
for 6 principal components:

==================== Random Forest ====================

============================ CLASSIFICATION REPORT
==============================
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

================================================================================
==========
================================== ACCURACY SCORE
==================================
1.0
================================================================================
==========
for 7 principal components:

==================== Random Forest ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

================================================================================
==========
================================== ACCURACY SCORE
==================================
1.0
================================================================================
==========
for 8 principal components:

==================== Random Forest ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
```

```
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================ ACCURACY SCORE
================================
1.0
==========================================================================
==========
```

Conclusion:

With PCA, we can use only 3 features and end up getting the same results.

```
def mlp_train_pca(X_train_pca, X_test_pca, y_train, y_test):
    print(f"\n{'=' * 20} MLP Classifier {'=' * 20}\n")
    mlp = MLPClassifier(hidden_layer_sizes=(100,), max_iter=500,
random_state=10)
    mlp.fit(X_train_pca, y_train)
    y_pred = mlp.predict(X_test_pca)

    print(" CLASSIFICATION REPORT ".center(80, "="))
    print(classification_report(y_test, y_pred))
    print("=" * 80)

    print(" ACCURACY SCORE ".center(80, "="))
    print(f"{accuracy_score(y_test, y_pred)}")
    print("=" * 80)

for i in range(1,9):
    pca = PCA(n_components=i)
    pca.fit(X_train)
    X_train_pca = pca.transform(X_train)
    X_test_pca = pca.transform(X_test)
    print(f"for {i} principal components:")
    mlp_train_pca(X_train_pca, X_test_pca, y_train, y_test)
```

```
for 1 principal components:

==================== MLP Classifier ====================

============================ CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      0.83      0.91         6
           1       0.75      1.00      0.86         6
```

```
           2       1.00      0.80      0.89         5

    accuracy                           0.88        17
   macro avg       0.92      0.88      0.89        17
weighted avg       0.91      0.88      0.88        17

==========================================================================
==========
================================ ACCURACY SCORE
================================
0.8823529411764706
==========================================================================
==========
for 2 principal components:

==================== MLP Classifier ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================ ACCURACY SCORE
================================
1.0
==========================================================================
==========
for 3 principal components:

==================== MLP Classifier ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
```

```
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================== ACCURACY SCORE
==================================
1.0
==========================================================================
==========
for 4 principal components:

==================== MLP Classifier ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
================================== ACCURACY SCORE
==================================
1.0
==========================================================================
==========
for 5 principal components:

==================== MLP Classifier ====================

============================== CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

==========================================================================
==========
```

```
================================ ACCURACY SCORE
================================
1.0
================================================================================
==========
for 6 principal components:

==================== MLP Classifier ====================

============================ CLASSIFICATION REPORT
============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

================================================================================
==========
================================ ACCURACY SCORE
================================
1.0
================================================================================
==========
for 7 principal components:

==================== MLP Classifier ====================

============================ CLASSIFICATION REPORT
============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

================================================================================
==========
================================ ACCURACY SCORE
================================
1.0
================================================================================
```

```
==========
for 8 principal components:

==================== MLP Classifier ====================

============================ CLASSIFICATION REPORT
==============================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        17
   macro avg       1.00      1.00      1.00        17
weighted avg       1.00      1.00      1.00        17

================================================================================
==========
================================ ACCURACY SCORE
================================
1.0
================================================================================
==========
```

Conclusion:

With PCA, we can use only 3 features and end up getting the same results.