```
# from sklearn.datasets import load_wine
# data = load_wine()
# X, y = data.data, data.target
```

```
# from sklearn.datasets import load_digits
# data = load_digits()
# X, y = data.data, data.target
```

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split


data = load_digits()
X, y = data.data, data.target


#splits = [0.5, 0.4, 0.3, 0.2]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
```

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

kernels = ['linear', 'poly', 'rbf', 'sigmoid']
for k in kernels:
    model = SVC(kernel=k)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average='macro')   # precision
    rec = recall_score(y_test, y_pred, average='macro')       # recall
    f1 = f1_score(y_test, y_pred, average='macro')            # f1-score
    cm = confusion_matrix(y_test, y_pred)

    print(f"Kernel: {k}")
    print(f"Accuracy : {acc:.4f}")
    print(f"Precision: {prec:.4f}")
    print(f"Recall   : {rec:.4f}")
    print(f"F1-score : {f1:.4f}")
    print("Confusion Matrix:\n", cm)
    print("-"*40)
```

```
Kernel: linear
Accuracy : 0.9759
Precision: 0.9761
Recall   : 0.9755
F1-score : 0.9755
Confusion Matrix:
 [[53  0  0  0  1  0  0  0  0  0]
 [ 0 54  0  0  0  0  0  0  0  1]
 [ 0  0 53  0  0  0  0  0  0  0]
 [ 0  0  0 54  0  0  0  0  0  1]
 [ 0  0  0  0 54  0  0  0  0  0]
 [ 0  0  0  0  0 54  0  0  0  1]
 [ 0  0  0  0  0  0 54  0  0  0]
 [ 0  0  0  0  0  0  0 54  0  0]
 [ 0  4  0  0  0  1  1  1 45  0]
 [ 0  0  0  0  0  0  0  0  2 52]]
----------------------------------------
Kernel: poly
Accuracy : 0.9889
Precision: 0.9890
Recall   : 0.9888
F1-score : 0.9888
Confusion Matrix:
 [[53  0  0  0  1  0  0  0  0  0]
 [ 0 55  0  0  0  0  0  0  0  0]
 [ 0  0 53  0  0  0  0  0  0  0]
 [ 0  0  0 55  0  0  0  0  0  0]
 [ 0  0  0  0 54  0  0  0  0  0]
 [ 0  0  0  0  0 54  0  0  0  1]
 [ 0  0  0  0  0  0 53  0  1  0]
```

```
 [ 0  0  0  0  0  0  0 54  0  0]
 [ 0  2  0  0  0  0  0  0 50  0]
 [ 0  0  0  0  0  0  0  1  0 53]]
----------------------------------------
Kernel: rbf
Accuracy : 0.9889
Precision: 0.9890
Recall   : 0.9888
F1-score : 0.9888
Confusion Matrix:
 [[53  0  0  0  1  0  0  0  0  0]
 [ 0 55  0  0  0  0  0  0  0  0]
 [ 0  0 53  0  0  0  0  0  0  0]
 [ 0  0  0 55  0  0  0  0  0  0]
 [ 0  0  0  0 53  0  0  0  1  0]
 [ 0  0  0  0  0 54  0  0  0  1]
 [ 0  0  0  0  0  0 54  0  0  0]
 [ 0  0  0  0  0  0  0 54  0  0]
 [ 0  2  0  0  0  0  0  0 50  0]
 [ 0  0  0  0  0  0  0  1  0 53]]
----------------------------------------
Kernel: sigmoid
Accuracy : 0.8944
Precision: 0.9010
Recall   : 0.8936
F1-score : 0.8937
Confusion Matrix:
```

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split


data = load_digits()
X, y = data.data, data.target


#splits = [0.5, 0.4, 0.3, 0.2]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)


from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

kernels = ['linear', 'poly', 'rbf', 'sigmoid']
for k in kernels:
    model = SVC(kernel=k)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average='macro')   # precision
    rec = recall_score(y_test, y_pred, average='macro')       # recall
    f1 = f1_score(y_test, y_pred, average='macro')            # f1-score
    cm = confusion_matrix(y_test, y_pred)

    print(f"Kernel: {k}")
    print(f"Accuracy : {acc:.4f}")
    print(f"Precision: {prec:.4f}")
    print(f"Recall   : {rec:.4f}")
    print(f"F1-score : {f1:.4f}")
    print("Confusion Matrix:\n", cm)
    print("-"*40)
```

```
Kernel: linear
Accuracy : 0.9778
Precision: 0.9779
Recall   : 0.9775
F1-score : 0.9775
Confusion Matrix:
 [[36  0  0  0  0  0  0  0  0  0]
 [ 0 34  0  0  0  0  0  0  1  1]
 [ 0  0 35  0  0  0  0  0  0  0]
 [ 0  0  0 36  0  0  0  0  0  1]
 [ 0  0  0  0 36  0  0  0  0  0]
 [ 0  0  0  0  0 37  0  0  0  0]
 [ 0  0  0  0  0  0 35  0  1  0]
 [ 0  0  0  0  0  0  0 36  0  0]
 [ 0  3  0  0  0  0  0  1 31  0]
 [ 0  0  0  0  0  0  0  0  0 36]]
----------------------------------------
Kernel: poly
Accuracy : 0.9861
Precision: 0.9864
```

```
Recall   : 0.9860
F1-score : 0.9860
Confusion Matrix:
 [[35  0  0  0  1  0  0  0  0  0]
 [ 0 36  0  0  0  0  0  0  0  0]
 [ 0  0 35  0  0  0  0  0  0  0]
 [ 0  0  0 37  0  0  0  0  0  0]
 [ 0  0  0  0 36  0  0  0  0  0]
 [ 0  0  0  0  0 37  0  0  0  0]
 [ 0  0  0  0  0  0 35  0  1  0]
 [ 0  0  0  0  0  0  0 36  0  0]
 [ 0  2  0  0  0  0  0  0 33  0]
 [ 0  0  0  0  0  0  0  1  0 35]]
----------------------------------------
Kernel: rbf
Accuracy : 0.9917
Precision: 0.9920
Recall   : 0.9915
F1-score : 0.9916
Confusion Matrix:
 [[36  0  0  0  0  0  0  0  0  0]
 [ 0 36  0  0  0  0  0  0  0  0]
 [ 0  0 35  0  0  0  0  0  0  0]
 [ 0  0  0 37  0  0  0  0  0  0]
 [ 0  0  0  0 36  0  0  0  0  0]
 [ 0  0  0  0  0 37  0  0  0  0]
 [ 0  0  0  0  0  0 36  0  0  0]
 [ 0  0  0  0  0  0  0 36  0  0]
 [ 0  2  0  0  0  0  0  0 33  0]
 [ 0  0  0  0  0  0  0  1  0 35]]
----------------------------------------
Kernel: sigmoid
Accuracy : 0.8861
Precision: 0.8925
Recall   : 0.8851
F1-score : 0.8855
Confusion Matrix:
```

```python
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split


data = load_digits()
X, y = data.data, data.target


#splits = [0.5, 0.4, 0.3, 0.2]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.4, random_state=42, stratify=y
)


from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

kernels = ['linear', 'poly', 'rbf', 'sigmoid']
for k in kernels:
    model = SVC(kernel=k)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average='macro')   # precision
    rec = recall_score(y_test, y_pred, average='macro')       # recall
    f1 = f1_score(y_test, y_pred, average='macro')            # f1-score
    cm = confusion_matrix(y_test, y_pred)

    print(f"Kernel: {k}")
    print(f"Accuracy : {acc:.4f}")
    print(f"Precision: {prec:.4f}")
    print(f"Recall   : {rec:.4f}")
    print(f"F1-score : {f1:.4f}")
    print("Confusion Matrix:\n", cm)
    print("-"*40)
```

```
Kernel: linear
Accuracy : 0.9694
Precision: 0.9711
Recall   : 0.9691
F1-score : 0.9693
Confusion Matrix:
 [[70  0  0  0  1  0  0  0  0  0]
 [ 0 72  0  0  0  0  0  0  0  1]
 [ 0  2 69  0  0  0  0  0  0  0]
 [ 0  0  0 71  0  0  0  0  0  2]
```

```
 [ 0  0  0  0 72  0  0  0  0  0]
 [ 0  0  0  0  0 72  0  0  0  1]
 [ 0  1  0  0  0  0 71  0  0  0]
 [ 0  0  0  0  0  0  0 72  0  0]
 [ 0  7  0  0  0  2  1  0 60  0]
 [ 0  0  0  0  0  0  0  3  1 68]]
----------------------------------------
Kernel: poly
Accuracy : 0.9861
Precision: 0.9866
Recall   : 0.9860
F1-score : 0.9861
Confusion Matrix:
 [[70  0  0  0  1  0  0  0  0  0]
 [ 0 73  0  0  0  0  0  0  0  0]
 [ 0  1 69  0  0  0  0  1  0  0]
 [ 0  0  0 73  0  0  0  0  0  0]
 [ 0  0  0  0 72  0  0  0  0  0]
 [ 0  0  0  0  0 72  0  0  0  1]
 [ 0  0  0  0  0  0 71  0  1  0]
 [ 0  0  0  0  0  0  0 72  0  0]
 [ 0  3  0  0  0  0  0  0 67  0]
 [ 0  0  0  0  0  0  0  2  0 70]]
----------------------------------------
Kernel: rbf
Accuracy : 0.9861
Precision: 0.9867
Recall   : 0.9860
F1-score : 0.9861
Confusion Matrix:
 [[70  0  0  0  1  0  0  0  0  0]
 [ 0 73  0  0  0  0  0  0  0  0]
 [ 0  1 70  0  0  0  0  0  0  0]
 [ 0  0  0 73  0  0  0  0  0  0]
 [ 0  0  0  0 72  0  0  0  0  0]
 [ 0  0  0  0  0 72  0  0  0  1]
 [ 0  0  0  0  0  0 72  0  0  0]
 [ 0  0  0  0  0  0  0 72  0  0]
 [ 0  3  0  0  0  0  0  0 67  0]
 [ 0  0  0  0  0  0  0  4  0 68]]
----------------------------------------
Kernel: sigmoid
Accuracy : 0.9026
Precision: 0.9075
Recall   : 0.9021
F1-score : 0.9027
Confusion Matrix:
 [[67  0  0  0  4  0  0  0  0  0]
```

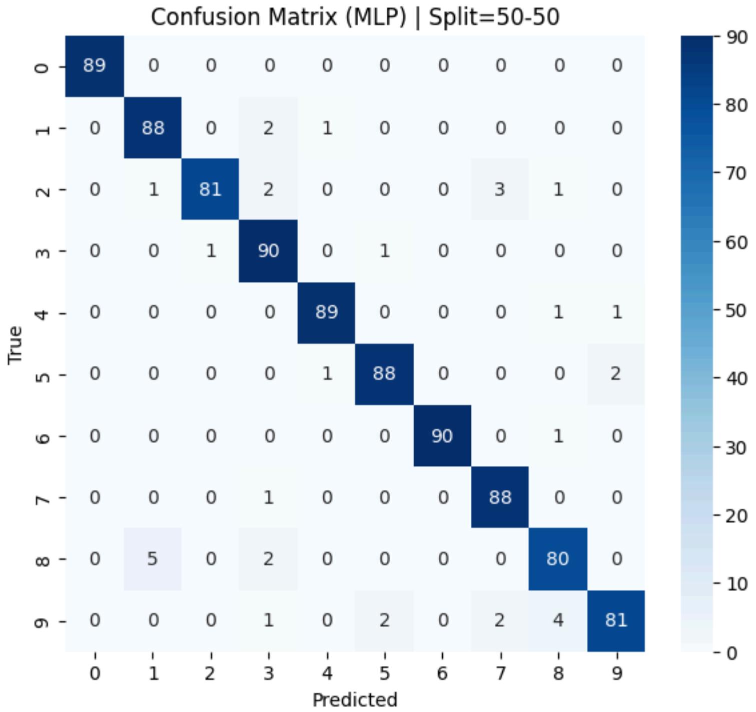
```python
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split


data = load_digits()
X, y = data.data, data.target


#splits = [0.5, 0.4, 0.3, 0.2]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.5, random_state=42, stratify=y
)


from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

kernels = ['linear', 'poly', 'rbf', 'sigmoid']
for k in kernels:
    model = SVC(kernel=k)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average='macro')   # precision
    rec = recall_score(y_test, y_pred, average='macro')       # recall
    f1 = f1_score(y_test, y_pred, average='macro')            # f1-score
    cm = confusion_matrix(y_test, y_pred)

    print(f"Kernel: {k}")
    print(f"Accuracy : {acc:.4f}")
    print(f"Precision: {prec:.4f}")
    print(f"Recall   : {rec:.4f}")
    print(f"F1-score : {f1:.4f}")
    print("Confusion Matrix:\n", cm)
    print("-"*40)
```

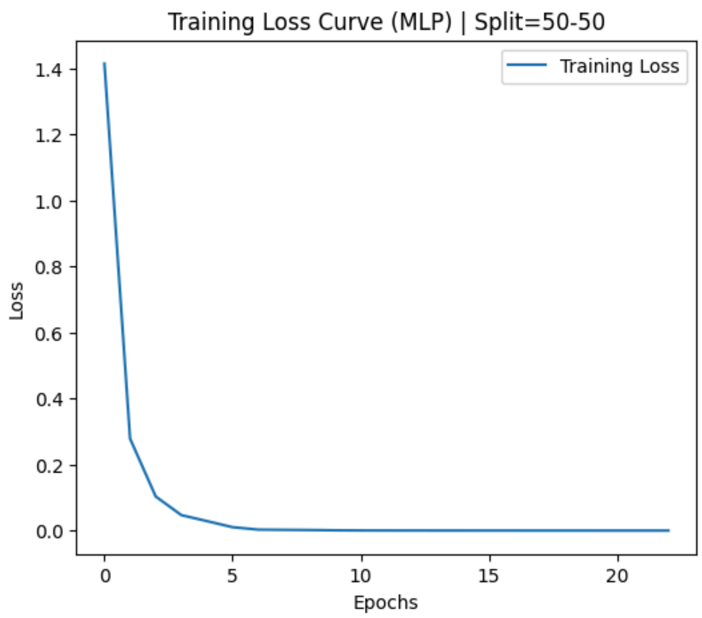
```
Kernel: linear
Accuracy : 0.9733
Precision: 0.9736
Recall   : 0.9731
F1-score : 0.9731
Confusion Matrix:
 [[88  0  0  0  1  0  0  0  0  0]
 [ 0 90  0  0  0  0  0  0  1  0]
 [ 0  1 87  0  0  0  0  0  0  0]
 [ 0  0  1 90  0  0  0  0  0  1]
 [ 0  0  0  0 91  0  0  0  0  0]
 [ 0  0  0  0  0 89  0  0  0  2]
 [ 0  0  0  0  0  0 90  0  1  0]
 [ 0  0  0  0  0  0  0 88  1  0]
 [ 0  6  0  1  0  0  1  0 79  0]
 [ 0  1  0  0  0  1  0  4  1 83]]
---------------------------------------
Kernel: poly
Accuracy : 0.9822
Precision: 0.9825
Recall   : 0.9822
F1-score : 0.9822
Confusion Matrix:
 [[88  0  0  0  1  0  0  0  0  0]
 [ 0 91  0  0  0  0  0  0  0  0]
 [ 0  1 86  0  0  0  0  1  0  0]
 [ 0  0  1 90  0  0  0  0  0  1]
 [ 0  0  0  0 91  0  0  0  0  0]
 [ 0  0  0  0  0 89  0  0  0  2]
 [ 0  0  0  0  0  0 90  0  1  0]
 [ 0  0  0  0  0  0  0 89  0  0]
 [ 0  2  0  0  0  0  0  0 85  0]
 [ 0  0  0  0  0  1  0  5  0 84]]
---------------------------------------
Kernel: rbf
Accuracy : 0.9811
Precision: 0.9817
Recall   : 0.9810
F1-score : 0.9810
Confusion Matrix:
 [[88  0  0  0  1  0  0  0  0  0]
 [ 0 91  0  0  0  0  0  0  0  0]
 [ 0  1 87  0  0  0  0  0  0  0]
 [ 0  0  0 91  0  0  0  0  1  0]
 [ 0  0  0  0 90  0  0  1  0  0]
 [ 0  0  0  0  0 90  0  0  0  1]
 [ 0  0  0  0  0  0 90  0  1  0]
 [ 0  0  0  0  0  0  0 89  0  0]
 [ 0  3  0  0  0  0  0  0 84  0]
 [ 0  0  0  0  0  2  0  6  0 82]]
---------------------------------------
Kernel: sigmoid
Accuracy : 0.9066
Precision: 0.9106
Recall   : 0.9062
F1-score : 0.9072
Confusion Matrix:
 [[86  0  0  0  3  0  0  0  0  0]
```

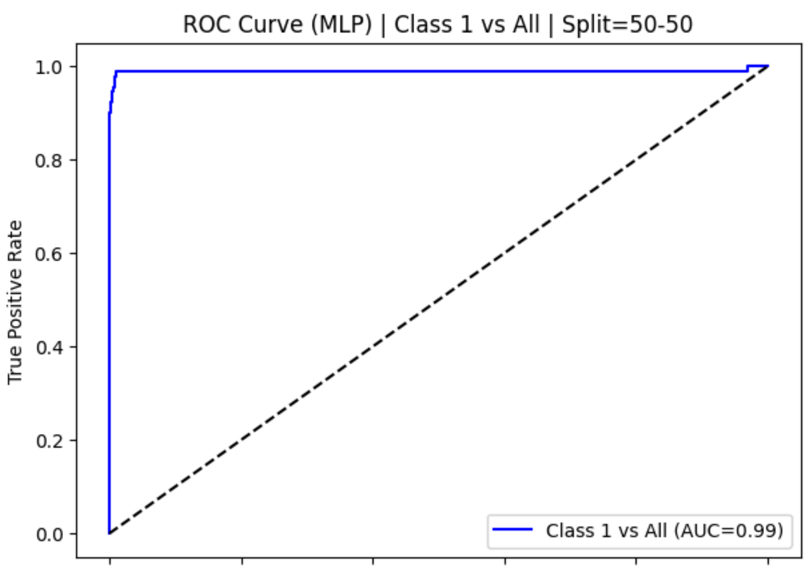
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, confusion_matrix, roc_curve, auc)

# ---------------------- LOAD DATA ----------------------
digits = load_digits()
X, y = digits.data, digits.target
classes = np.unique(y)

# ---------------------- PREPROCESS ----------------------
scaler = StandardScaler()
X = scaler.fit_transform(X)

# ---------------------- PARAMETERS ----------------------
splits = [0.5, 0.6, 0.7, 0.8]   # Train ratios
results = []

for split in splits:
    print(f"\n================ Train-Test Split: {int(split*100)}-{100-int(split*100)} ================\n")
```

```
        # Train-test split
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=1-split, stratify=y, random_state=42
        )

        # ---------------------- MODEL ----------------------
        mlp = MLPClassifier(hidden_layer_sizes=(100,100),
                            solver="adam",
                            learning_rate_init=0.01,
                            max_iter=500,
                            random_state=42,
                            verbose=False)   # training progress off for clarity

        mlp.fit(X_train, y_train)

        # ---------------------- PREDICTION ----------------------
        y_pred = mlp.predict(X_test)

        # ---------------------- METRICS ----------------------
        acc = accuracy_score(y_test, y_pred)
        prec = precision_score(y_test, y_pred, average="macro")
        rec = recall_score(y_test, y_pred, average="macro")
        f1 = f1_score(y_test, y_pred, average="macro")

        results.append([f"{int(split*100)}-{100-int(split*100)}", acc, prec, rec, f1])

        print(f"Accuracy : {acc:.4f}")
        print(f"Precision: {prec:.4f}")
        print(f"Recall   : {rec:.4f}")
        print(f"F1-score : {f1:.4f}")

        # ---------------------- CONFUSION MATRIX ----------------------
        cm = confusion_matrix(y_test, y_pred)
        plt.figure(figsize=(7,6))
        sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                    xticklabels=classes, yticklabels=classes)
        plt.title(f"Confusion Matrix (MLP) | Split={int(split*100)}-{100-int(split*100)}")
        plt.xlabel("Predicted")
        plt.ylabel("True")
        plt.show()

        # ---------------------- TRAINING LOSS CURVE ----------------------
        plt.figure(figsize=(6,5))
        plt.plot(mlp.loss_curve_, label="Training Loss")
        plt.title(f"Training Loss Curve (MLP) | Split={int(split*100)}-{100-int(split*100)}")
        plt.xlabel("Epochs")
        plt.ylabel("Loss")
        plt.legend()
        plt.show()


               # ---------------------- ROC & AUC (Class 1 vs All only) ----------------------
        y_test_bin = label_binarize(y_test, classes=classes)
        y_score = mlp.predict_proba(X_test)

        plt.figure(figsize=(7,5))
        fpr, tpr, _ = roc_curve(y_test_bin[:, 1], y_score[:, 1])   # class 1 vs all
        roc_auc = auc(fpr, tpr)

        plt.plot(fpr, tpr, label=f"Class 1 vs All (AUC={roc_auc:.2f})", color="blue")
        plt.plot([0,1], [0,1], 'k--')

        plt.title(f"ROC Curve (MLP) | Class 1 vs All | Split={int(split*100)}-{100-int(split*100)}")
        plt.xlabel("False Positive Rate")
        plt.ylabel("True Positive Rate")
        plt.legend(loc="lower right")
        plt.show()

    # ---------------------- COMPARISON TABLE ----------------------
    df = pd.DataFrame(results, columns=["Split", "Accuracy", "Precision", "Recall", "F1-score"])
    print("\n===== Performance Comparison =====")
    print(df)
```

```
=============== Train-Test Split: 50-50 ===============

Accuracy : 0.9611
Precision: 0.9617
Recall   : 0.9608
F1-score : 0.9609
```

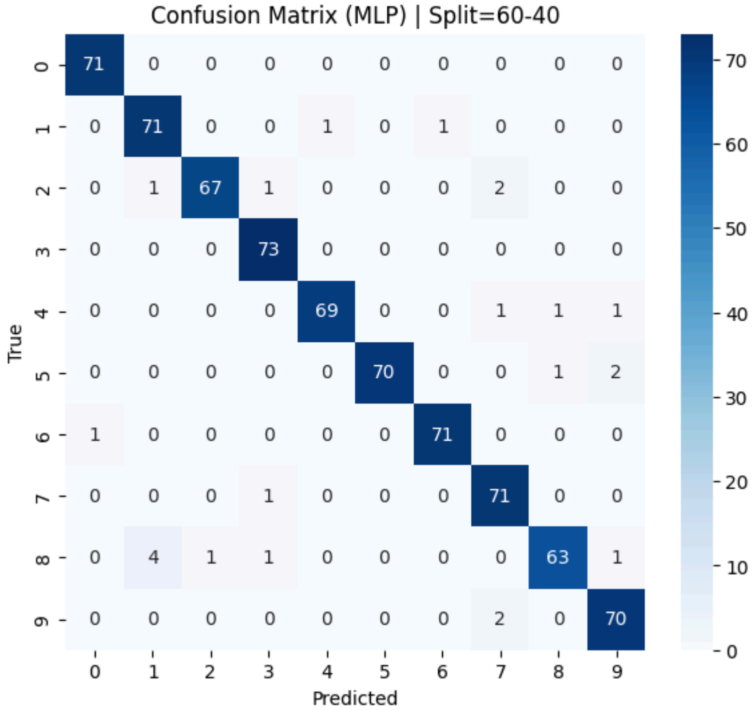


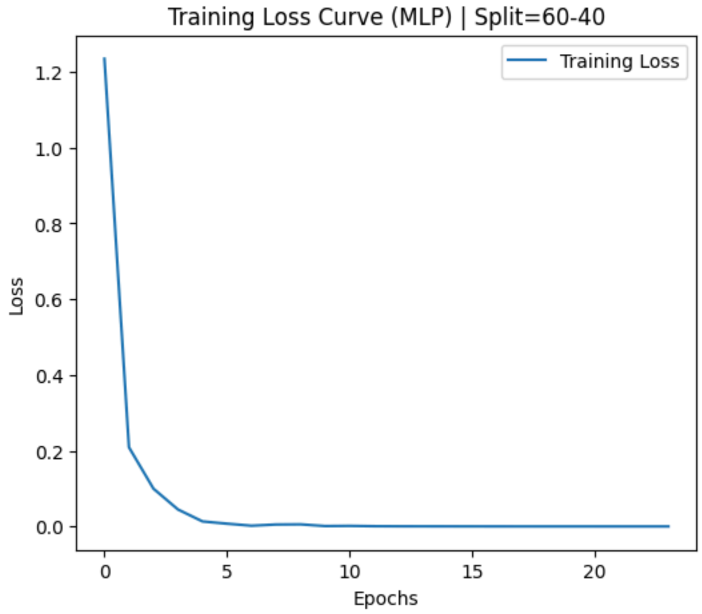


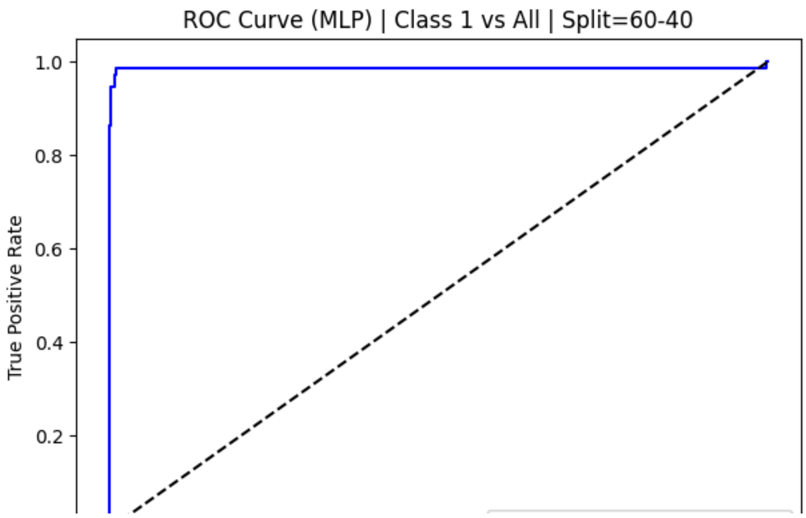


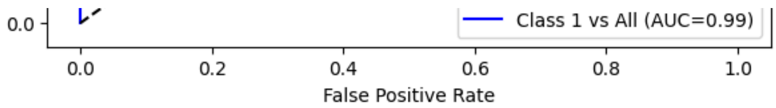
0.0 0.2 0.4 0.6 0.8 1.0
False Positive Rate

```
=============== Train-Test Split: 60-40 ===============

Accuracy : 0.9680
Precision: 0.9687
Recall   : 0.9678
F1-score : 0.9679
```

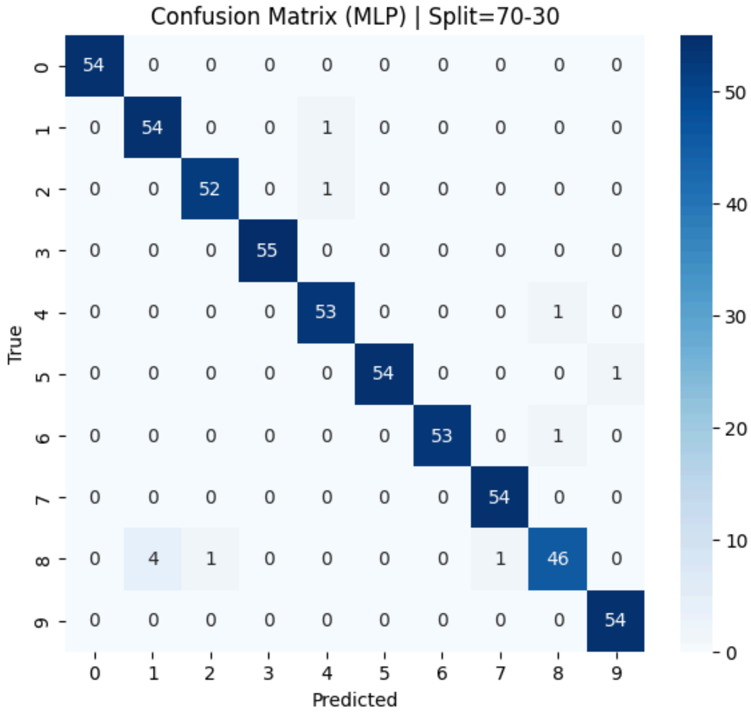


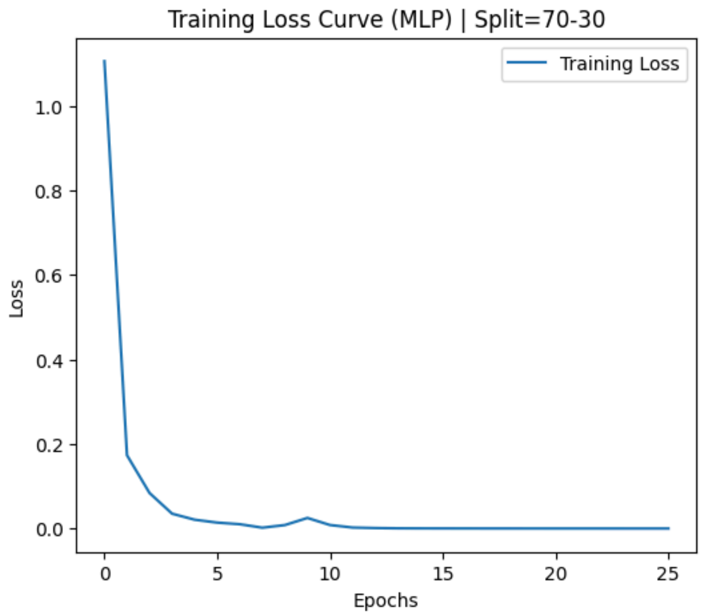


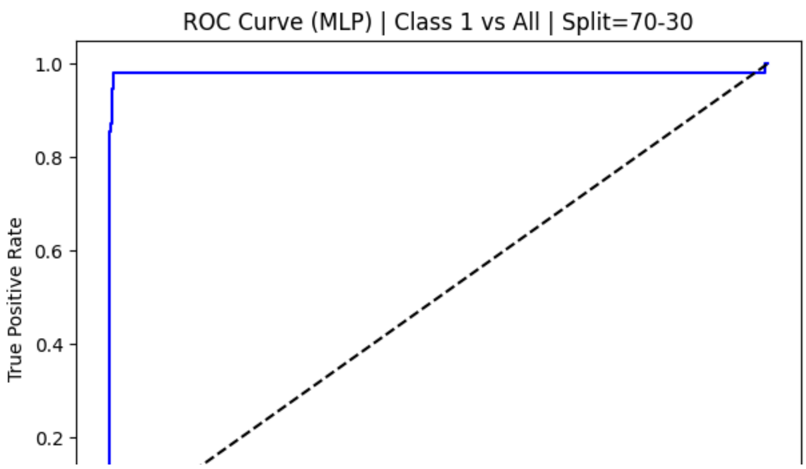


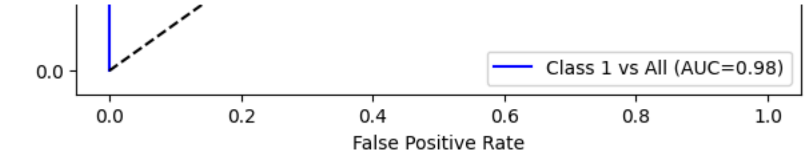



```
================ Train-Test Split: 70-30 ================

Accuracy : 0.9796
Precision: 0.9798
Recall   : 0.9792
F1-score : 0.9792
```

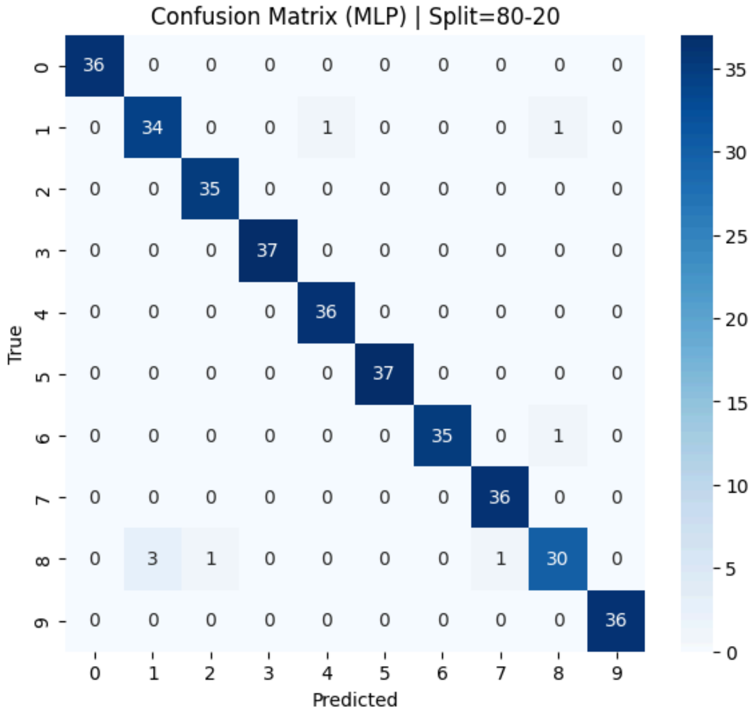


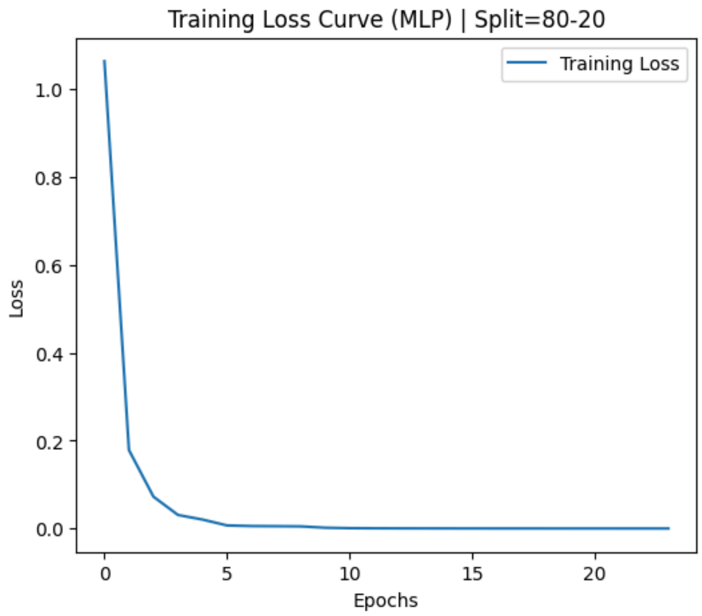


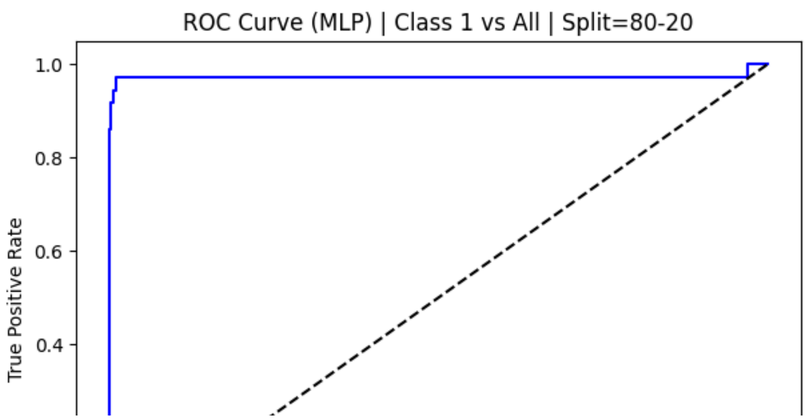


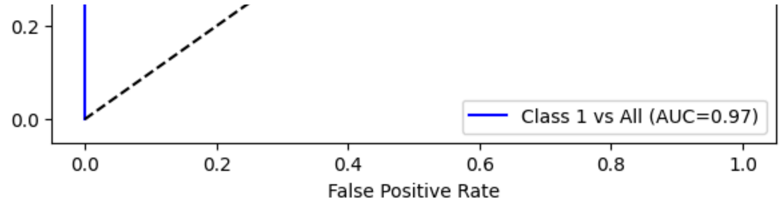



```
=============== Train-Test Split: 80-20 ===============

Accuracy : 0.9778
Precision: 0.9775
Recall   : 0.9774
F1-score : 0.9771
```







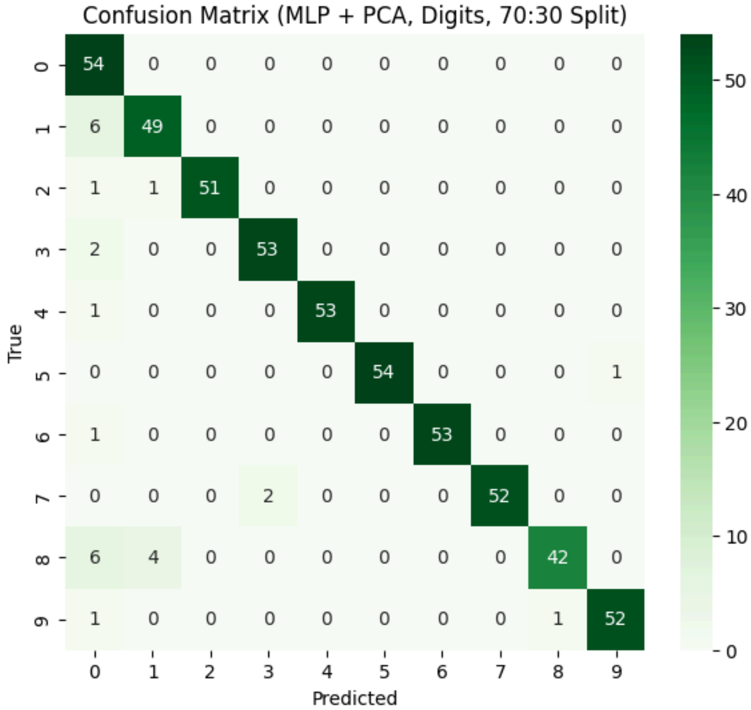


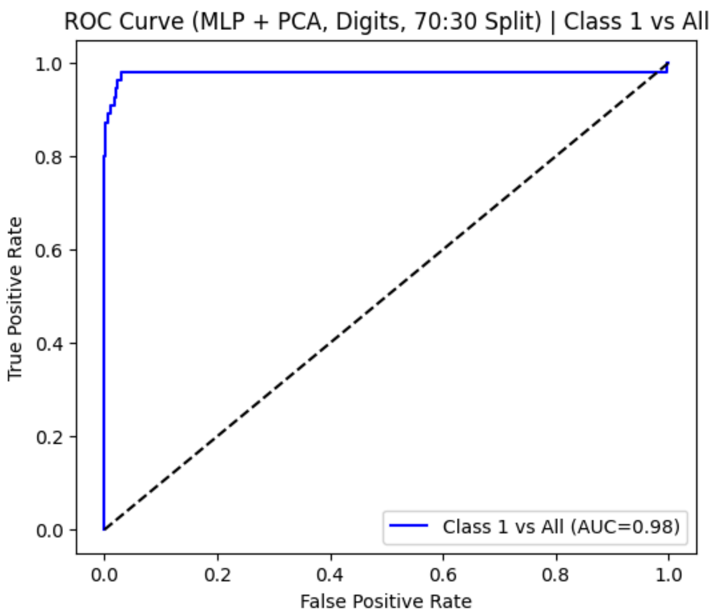

```
===== Performance Comparison =====
   Split  Accuracy  Precision    Recall  F1-score
0  50-50  0.961068   0.961746  0.960812  0.960880
1  60-40  0.968011   0.968735  0.967795  0.967858
2  70-30  0.979630   0.979777  0.979235  0.979249
3  80-20  0.977778   0.977459  0.977381  0.977146
```

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

from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.neural_network import MLPClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, roc_curve, auc

# ---------------------- LOAD DATA ----------------------
digits = load_digits()
X, y = digits.data, digits.target
classes = np.unique(y)
n_classes = len(classes)

# ---------------------- PREPROCESS ----------------------
scaler = StandardScaler()
X = scaler.fit_transform(X)

# ---------------------- PCA (95% variance) ----------------------
pca = PCA(n_components=0.95, random_state=42)
X_pca = pca.fit_transform(X)
print(f"Original shape: {X.shape}, After PCA: {X_pca.shape}")

# ---------------------- BEST SPLIT (70:30) ----------------------
X_train, X_test, y_train, y_test = train_test_split(
    X_pca, y, test_size=0.3, stratify=y, random_state=42
)

# Binarize for ROC
y_train_bin = label_binarize(y_train, classes=classes)
y_test_bin = label_binarize(y_test, classes=classes)

# ---------------------- MODEL ----------------------
mlp = OneVsRestClassifier(
    MLPClassifier(hidden_layer_sizes=(100,100),
                  solver="adam",
                  learning_rate_init=0.01,
                  max_iter=500,
                  random_state=42)
)

mlp.fit(X_train, y_train_bin)
y_pred_bin = mlp.predict(X_test)
y_score = mlp.predict_proba(X_test)

# Convert one-hot predictions back to class labels
y_pred = np.argmax(y_pred_bin, axis=1)

# ---------------------- METRICS ----------------------
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred, average="macro")
rec = recall_score(y_test, y_pred, average="macro")
f1 = f1_score(y_test, y_pred, average="macro")

print("\n===== PCA + MLP (Digits, Best Split: 70-30) =====")
print(f"Accuracy : {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall   : {rec:.4f}")
print(f"F1-score : {f1:.4f}")

# ---------------------- CONFUSION MATRIX ----------------------
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(7,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Greens",
            xticklabels=classes, yticklabels=classes)
plt.title("Confusion Matrix (MLP + PCA, Digits, 70:30 Split)")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()

# ---------------------- ROC & AUC (Only One-vs-All, e.g., Class 1 vs All) ----------------------
target_class = 1   # change this to any digit (0-9) you want to test

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

plt.figure(figsize=(6,5))
```

```
plt.plot(fpr, tpr, label=f"Class {target_class} vs All (AUC={roc_auc:.2f})", color="blue")
plt.plot([0,1], [0,1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title(f"ROC Curve (MLP + PCA, Digits, 70:30 Split) | Class {target_class} vs All")
plt.legend(loc="lower right")
plt.show()
```

```
Original shape: (1797, 64), After PCA: (1797, 40)

===== PCA + MLP (Digits, Best Split: 70-30) =====
Accuracy : 0.9500
Precision: 0.9579
Recall   : 0.9495
F1-score : 0.9510
```



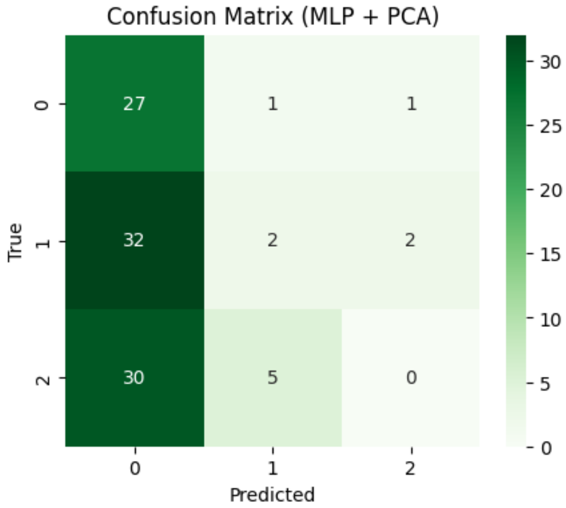


Start coding or generate with AI.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import label_binarize, StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.multiclass import OneVsRestClassifier
from sklearn.decomposition import PCA
```

```
import seaborn as sns

# ---------------------- PARAMETERS ----------------------
mom = 0.9
lr = 0.001
ep = 100
split = 0.5   # 50:50 split

# ---------------------- DATA (Dummy for Example) ----------------------
# Replace this with actual dataset (Wine/Digits)
X = np.random.rand(200, 10)   # 200 samples, 10 features
y = np.random.randint(0, 3, 200)  # 3-class labels (0,1,2)

# ---------------------- PREPROCESS ----------------------
scaler = StandardScaler()
X = scaler.fit_transform(X)

# PCA (reduce features, e.g. 95% variance retained)
pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X)
print(f"Original shape: {X.shape}, After PCA: {X_pca.shape}")

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=split, random_state=42)

# Binarize labels (for multi-class handling)
classes = np.unique(y)
n_classes = len(classes)
y_train_bin = label_binarize(y_train, classes=classes)
y_test_bin = label_binarize(y_test, classes=classes)

# ---------------------- MODEL ----------------------
mlp_pca = OneVsRestClassifier(
    MLPClassifier(hidden_layer_sizes=(100,),
                  max_iter=ep,
                  learning_rate_init=lr,
                  momentum=mom,
                  solver='sgd',
                  random_state=42)
)

mlp_pca.fit(X_train, y_train_bin)
y_pred = mlp_pca.predict(X_test)

# ---------------------- METRICS ----------------------
# Convert predictions back to class labels
y_pred_classes = np.argmax(y_pred, axis=1)

acc = accuracy_score(y_test, y_pred_classes)
print(f"Accuracy with PCA: {acc:.2f}")

print("\nClassification Report (PCA):")
print(classification_report(y_test, y_pred_classes, target_names=[f"Class {c}" for c in classes]))

# ---------------------- CONFUSION MATRIX ----------------------
cm = confusion_matrix(y_test, y_pred_classes)

plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Greens", xticklabels=classes, yticklabels=classes)
plt.title("Confusion Matrix (MLP + PCA)")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
Original shape: (200, 10), After PCA: (200, 10)
Accuracy with PCA: 0.29

Classification Report (PCA):
              precision    recall  f1-score   support

     Class 0       0.30      0.93      0.46        29
     Class 1       0.25      0.06      0.09        36
     Class 2       0.00      0.00      0.00        35

    accuracy                           0.29       100
   macro avg       0.18      0.33      0.18       100
weighted avg       0.18      0.29      0.17       100

/usr/local/lib/python3.12/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimiz
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimiz
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimiz
  warnings.warn(
```

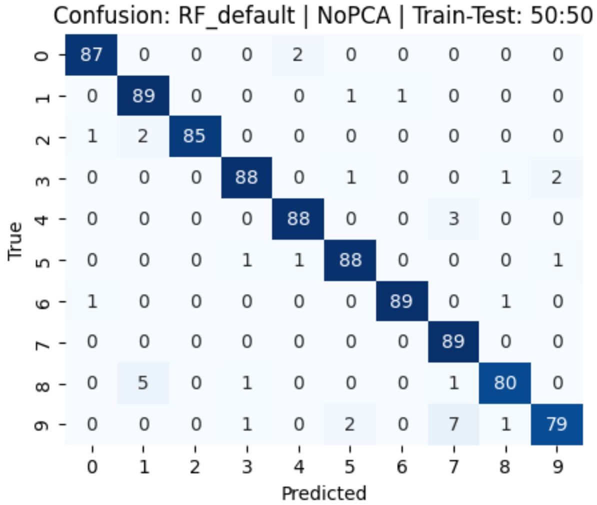


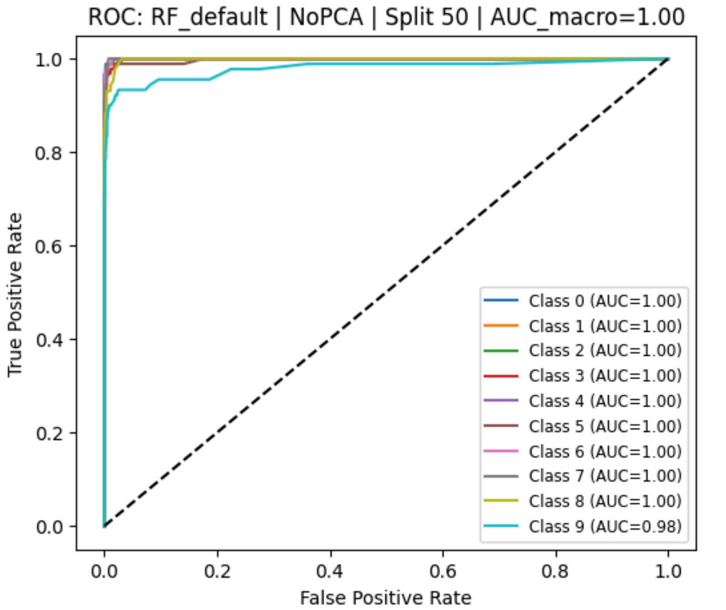

```
# Random Forest experiments: multiple splits, default + tuned, confusion heatmaps, ROC/AUC, optional PCA
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import (accuracy_score, precision_score, recall_score, f1_score,
                             confusion_matrix, roc_curve, auc, roc_auc_score)
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.decomposition import PCA
from sklearn.multiclass import OneVsRestClassifier

# --- Configuration ---
splits = [0.5, 0.6, 0.7, 0.8]                # train ratios
do_grid_search = True                        # toggle tuned RF
rf_param_grid = {                            # simple grid for tuning (modify as needed)
    'n_estimators': [50, 100],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}
random_state = 42
save_plots = False    # set True to save figures instead of plt.show()

# number of classes
n_classes = len(np.unique(y))

# results collection
results = []

# optional: do PCA version as well
do_pca_versions = True
pca_variance = 0.95   # keep components that explain 95% variance

# helper: plot + optionally save
def show_or_save(fig, title):
    if save_plots:
        fname = f"{title.replace(' ','_')}.png"
        fig.savefig(fname, bbox_inches='tight')
        plt.close(fig)
```

```
            plt.close(fig)
        else:
            plt.show()

    # Main loop (first without PCA, then with PCA if requested)
    for use_pca in [False, True] if do_pca_versions else [False]:
        version_tag = "PCA" if use_pca else "NoPCA"
        X_work = X.copy()
        # If using PCA, scale first (RF doesn't need scaling but PCA benefits)
        if use_pca:
            scaler = StandardScaler()
            X_scaled = scaler.fit_transform(X_work)
            pca = PCA(n_components=pca_variance, svd_solver='full', random_state=random_state)
            X_reduced = pca.fit_transform(X_scaled)
            print(f"[{version_tag}] PCA reduced dims: {X_reduced.shape[1]}")
            X_work = X_reduced
        else:
            # keep original X (no scaling needed for RF), but for ROC/AUC we won't scale
            pass

        for split in splits:
            train_size = split
            test_size = 1 - split
            print(f"\n--- {version_tag} | Train-Test: {int(split*100)}:{int(test_size*100)} ---")

            X_train, X_test, y_train, y_test = train_test_split(
                X_work, y, train_size=train_size, random_state=random_state, stratify=y
            )

            # ---- 1) Default Random Forest ----
            rf_default = RandomForestClassifier(random_state=random_state)
            rf_default.fit(X_train, y_train)
            y_pred = rf_default.predict(X_test)
            y_proba = rf_default.predict_proba(X_test)  # for ROC/AUC

            # metrics
            acc = accuracy_score(y_test, y_pred)
            prec = precision_score(y_test, y_pred, average='macro', zero_division=0)
            rec = recall_score(y_test, y_pred, average='macro', zero_division=0)
            f1 = f1_score(y_test, y_pred, average='macro', zero_division=0)
            cm = confusion_matrix(y_test, y_pred)

            # store
            results.append({
                'version': version_tag,
                'split': f"{int(split*100)}:{int(test_size*100)}",
                'model': 'RF_default',
                'acc': acc, 'precision': prec, 'recall': rec, 'f1': f1
            })

            # confusion heatmap
            fig = plt.figure(figsize=(5,4))
            sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
            plt.title(f"Confusion: RF_default | {version_tag} | Train-Test: {int(split*100)}:{int(test_size*100)}")
            plt.xlabel("Predicted"); plt.ylabel("True")
            show_or_save(fig, f"Confusion_RF_default_{version_tag}Train-Test: {int(split*100)}:{int(test_size*100)}")

            # ROC & AUC (multiclass)
            # Binarize true labels
            y_test_bin = label_binarize(y_test, classes=np.arange(n_classes))
            # compute ROC 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_proba[:, i])
                roc_auc[i] = auc(fpr[i], tpr[i])

            # macro AUC across classes
            try:
                overall_auc = roc_auc_score(y_test_bin, y_proba, average='macro', multi_class='ovr')
            except Exception:
                overall_auc = np.mean(list(roc_auc.values()))

            # plot multiclass ROC
            fig = plt.figure(figsize=(6,5))
            for i in range(n_classes):
                plt.plot(fpr[i], tpr[i], label=f"Class {i} (AUC={roc_auc[i]:.2f})")
            plt.plot([0,1],[0,1],'k--')
            plt.title(f"ROC: RF_default | {version_tag} | Split {int(split*100)} | AUC_macro={overall_auc:.2f}")
            plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
            plt.legend(loc='lower right', fontsize='small')
            show_or_save(fig, f"ROC_RF_default_{version_tag}_Train-Test: {int(split*100)}:{int(test_size*100)}")

            # ALSO add auc to results
```

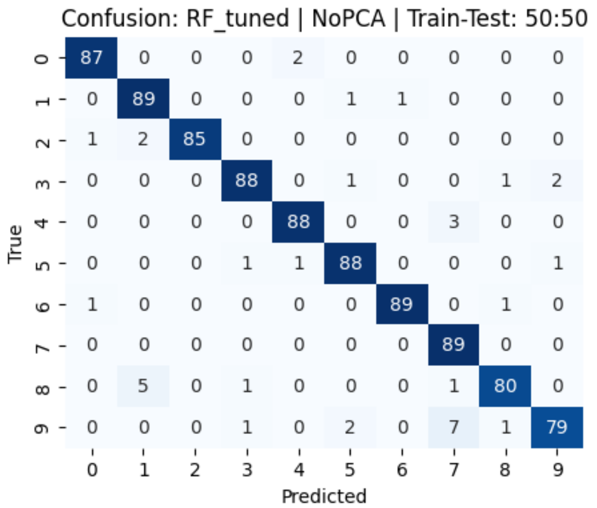
```
        results[-1].update({'auc_macro': overall_auc})

        # ---- 2) Tuned Random Forest (GridSearchCV) ----
        if do_grid_search:
            # Use a shallow GridSearch to save time; increase cv or grid if you want finer tuning
            gs = GridSearchCV(RandomForestClassifier(random_state=random_state),
                              rf_param_grid, cv=3, scoring='accuracy', n_jobs=-1, verbose=0)
            gs.fit(X_train, y_train)
            best = gs.best_estimator_
            print("RF GridSearch best params:", gs.best_params_)

            y_pred_t = best.predict(X_test)
            y_proba_t = best.predict_proba(X_test)

            acc_t = accuracy_score(y_test, y_pred_t)
            prec_t = precision_score(y_test, y_pred_t, average='macro', zero_division=0)
            rec_t = recall_score(y_test, y_pred_t, average='macro', zero_division=0)
            f1_t = f1_score(y_test, y_pred_t, average='macro', zero_division=0)
            cm_t = confusion_matrix(y_test, y_pred_t)

            results.append({
                'version': version_tag,
                'split': f"{int(split*100)}:{int(test_size*100)}",
                'model': 'RF_tuned',
                'acc': acc_t, 'precision': prec_t, 'recall': rec_t, 'f1': f1_t
            })

            # confusion heatmap tuned
            fig = plt.figure(figsize=(5,4))
            sns.heatmap(cm_t, annot=True, fmt='d', cmap='Blues', cbar=False)
            plt.title(f"Confusion: RF_tuned | {version_tag} | Train-Test: {int(split*100)}:{int(test_size*100)}")
            plt.xlabel("Predicted"); plt.ylabel("True")
            show_or_save(fig, f"Confusion_RF_tuned_{version_tag}_Train-Test: {int(split*100)}:{int(test_size*100)}")

            # ROC tuned
            fpr_t, tpr_t, roc_auc_t = dict(), dict(), dict()
            for i in range(n_classes):
                fpr_t[i], tpr_t[i], _ = roc_curve(y_test_bin[:, i], y_proba_t[:, i])
                roc_auc_t[i] = auc(fpr_t[i], tpr_t[i])
            try:
                overall_auc_t = roc_auc_score(y_test_bin, y_proba_t, average='macro', multi_class='ovr')
            except Exception:
                overall_auc_t = np.mean(list(roc_auc_t.values()))

            fig = plt.figure(figsize=(6,5))
            for i in range(n_classes):
                plt.plot(fpr_t[i], tpr_t[i], label=f"Class {i} (AUC={roc_auc_t[i]:.2f})")
            plt.plot([0,1],[0,1],'k--')
            plt.title(f"ROC: RF_tuned | {version_tag} | Train-Test: {int(split*100)}:{int(test_size*100)} | AUC_macro={overall_auc_t:.2f}
            plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
            plt.legend(loc='lower right', fontsize='small')
            show_or_save(fig, f"ROC_RF_tuned_{version_tag}_Train-Test: {int(split*100)}:{int(test_size*100)}")

            # add tuned auc
            results[-1].update({'auc_macro': overall_auc_t})

# Convert results to DataFrame and show
results_df = pd.DataFrame(results)
display(results_df.sort_values(['version','split','model'], ascending=[True,True,True]))
```

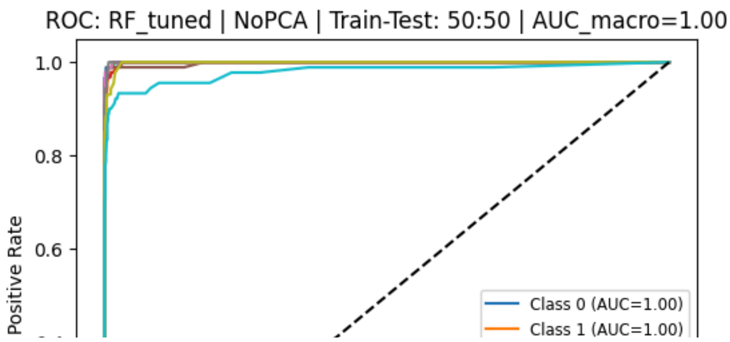
```
--- NoPCA | Train-Test: 50:50 ---
```

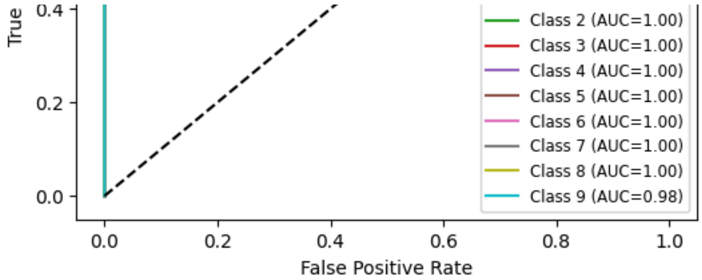




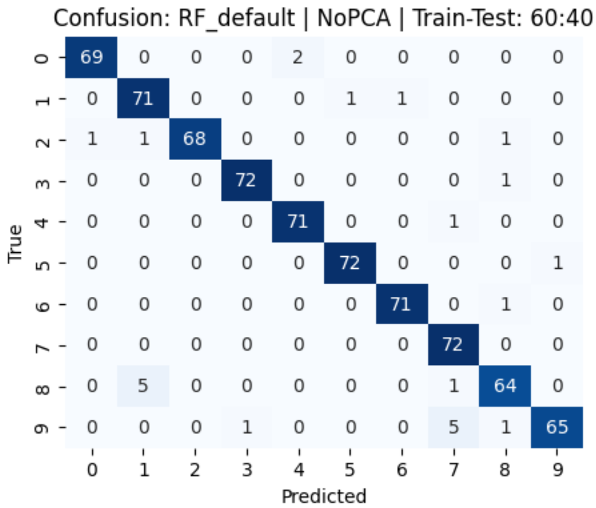

```
RF GridSearch best params: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}
```

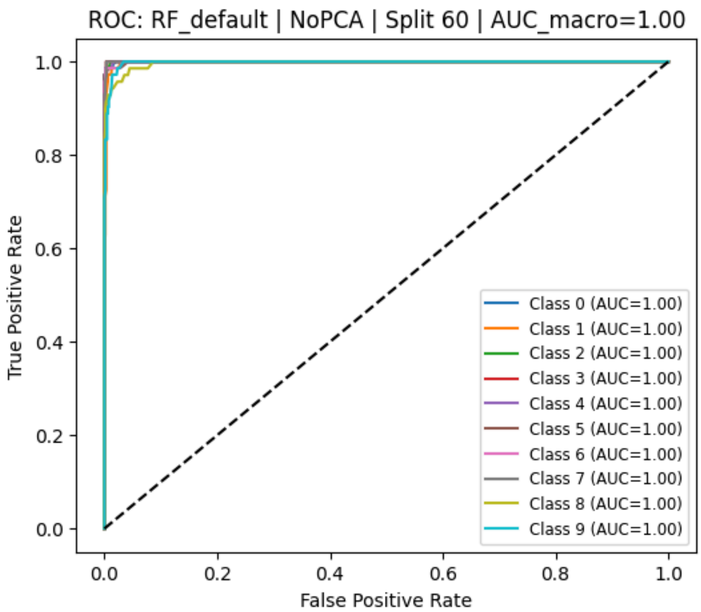




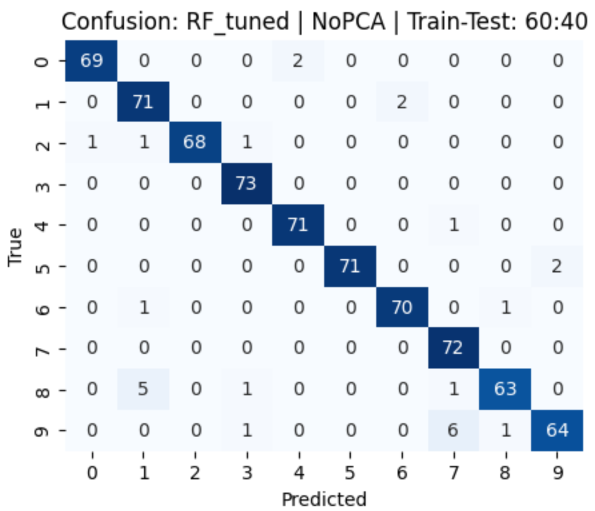


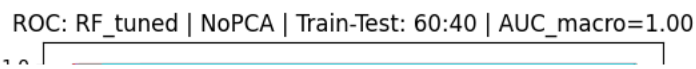

```
--- NoPCA | Train-Test: 60:40 ---
```

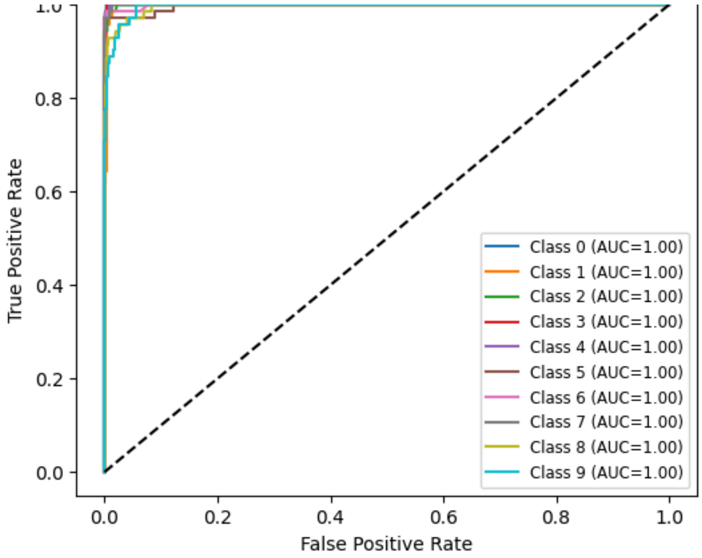




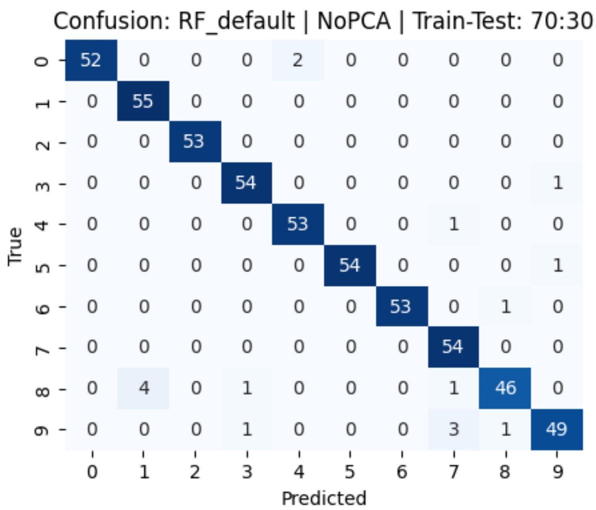

```
RF GridSearch best params: {'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 50}
```

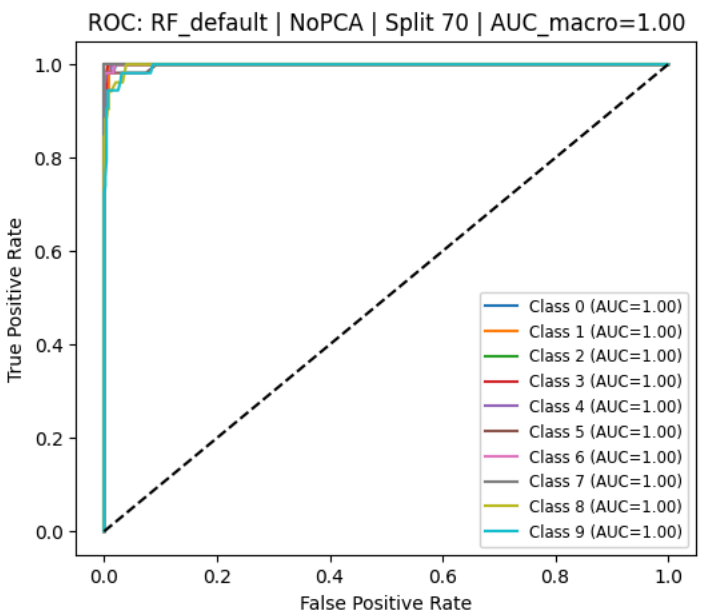




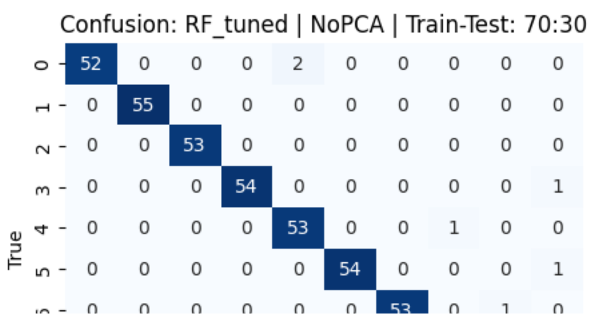


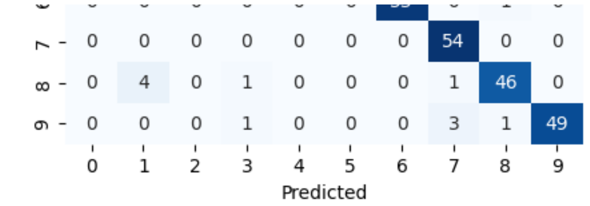

```
--- NoPCA | Train-Test: 70:30 ---
```

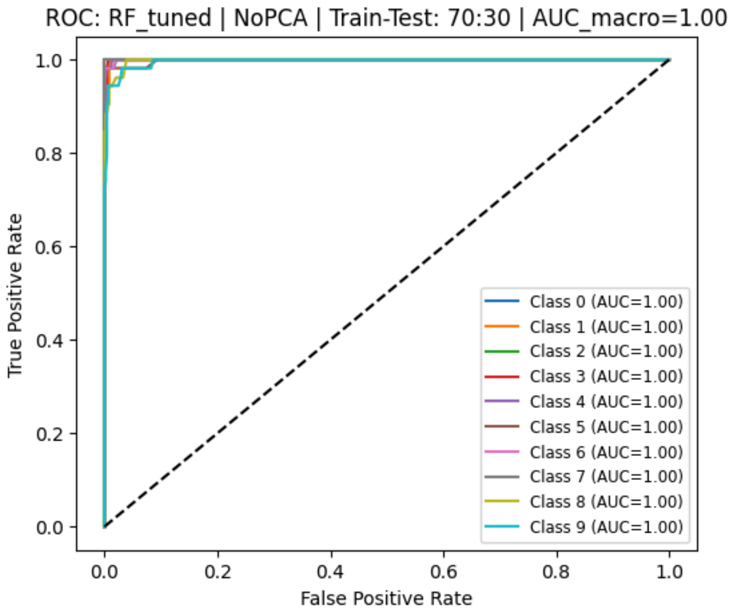




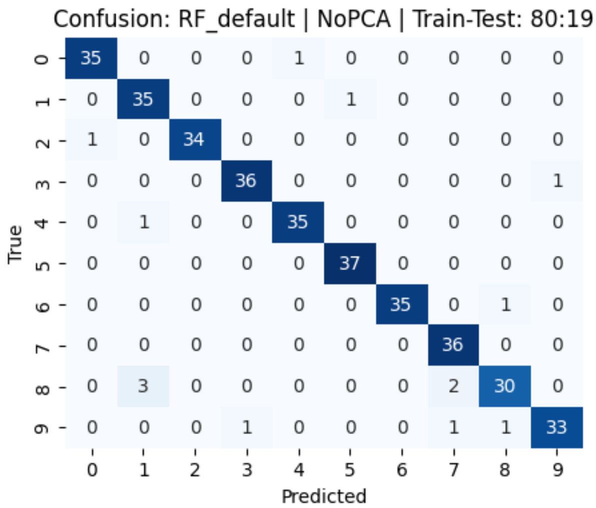

```
RF GridSearch best params: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}
```

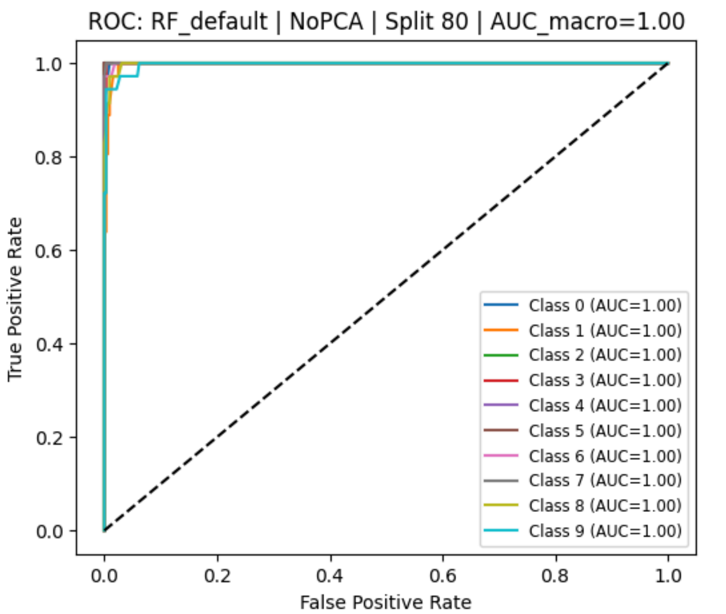






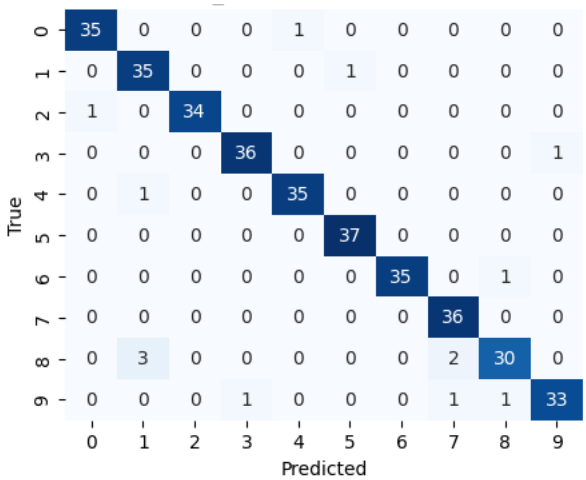

```
--- NoPCA | Train-Test: 80:19 ---
```



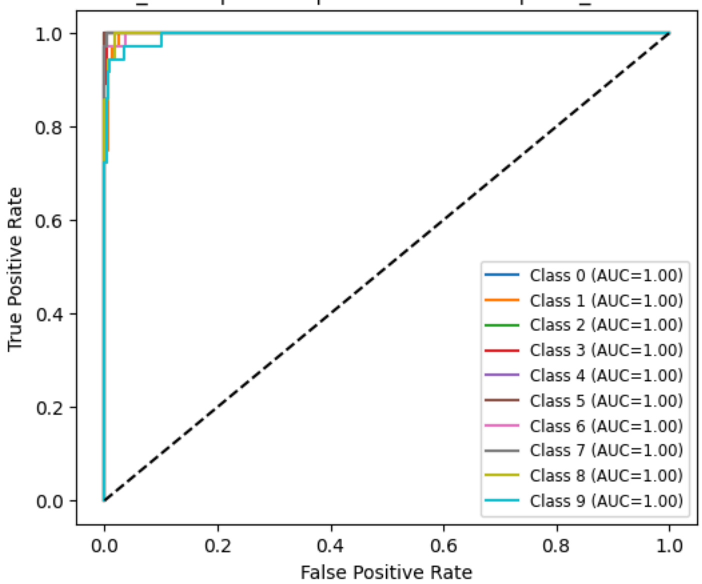



```
RF GridSearch best params: {'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 100}
```

Confusion: RF_tuned | NoPCA | Train-Test: 80:19

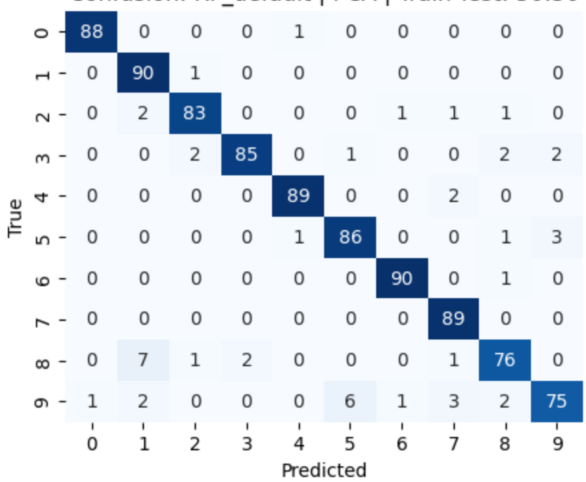


ROC: RF_tuned | NoPCA | Train-Test: 80:19 | AUC_macro=1.00

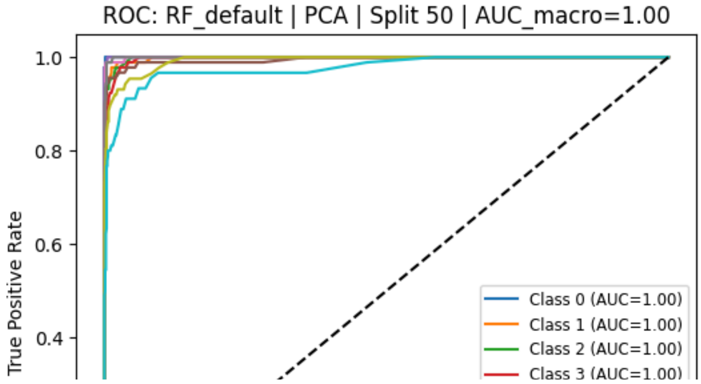


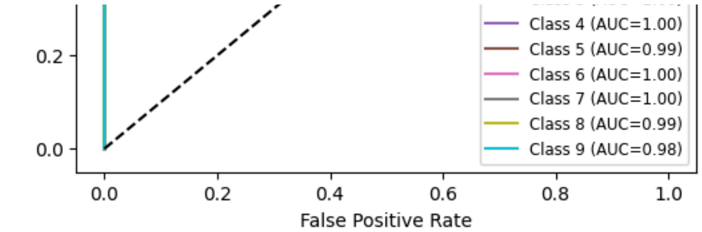

```
[PCA] PCA reduced dims: 40

--- PCA | Train-Test: 50:50 ---
```

Confusion: RF_default | PCA | Train-Test: 50:50

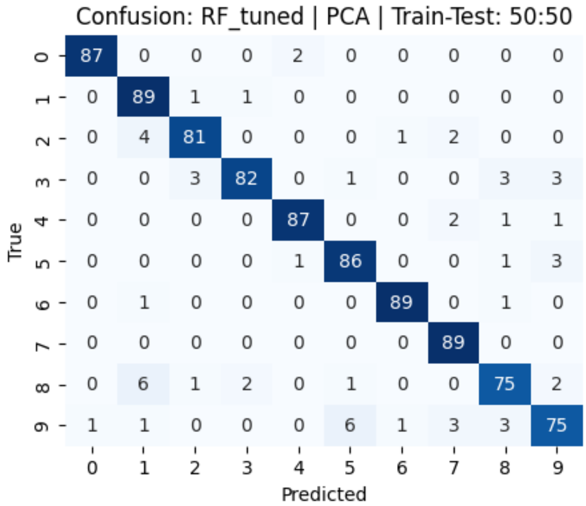


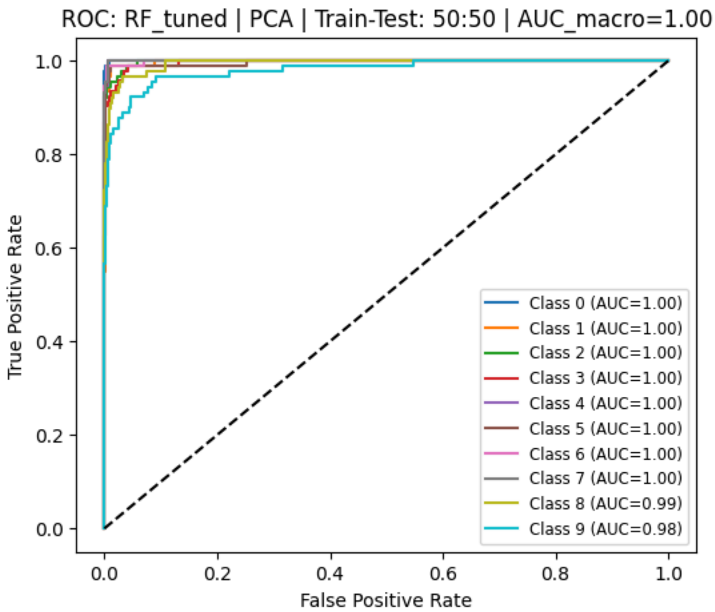
ROC: RF_default | PCA | Split 50 | AUC_macro=1.00



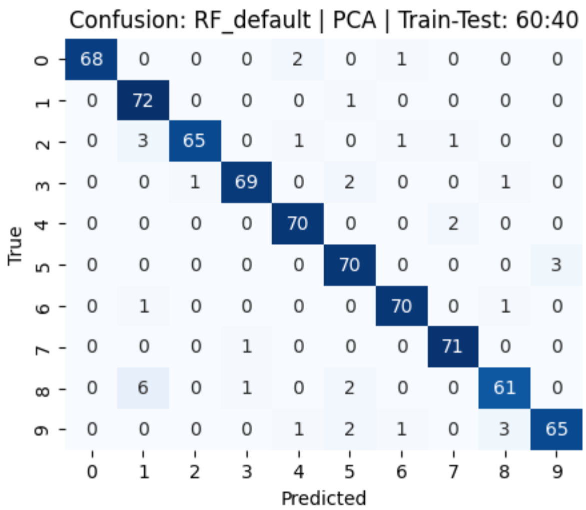


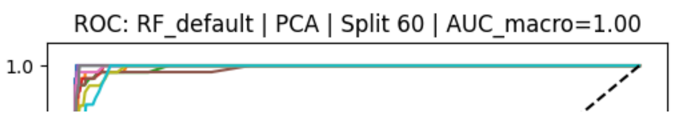

```
RF GridSearch best params: {'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 100}
```

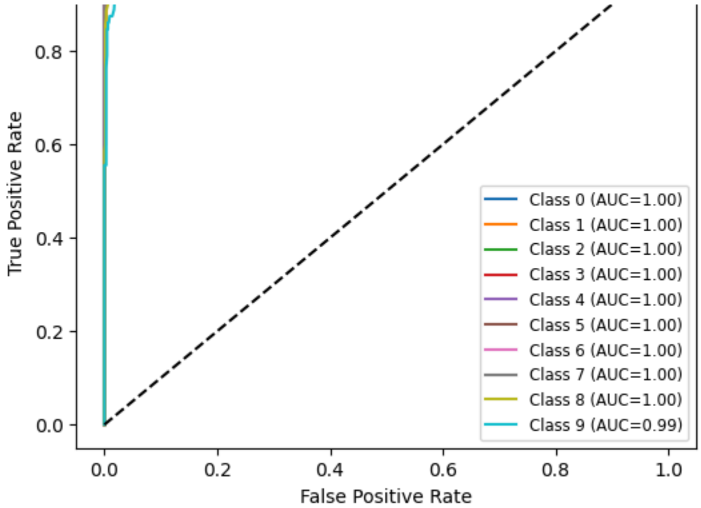




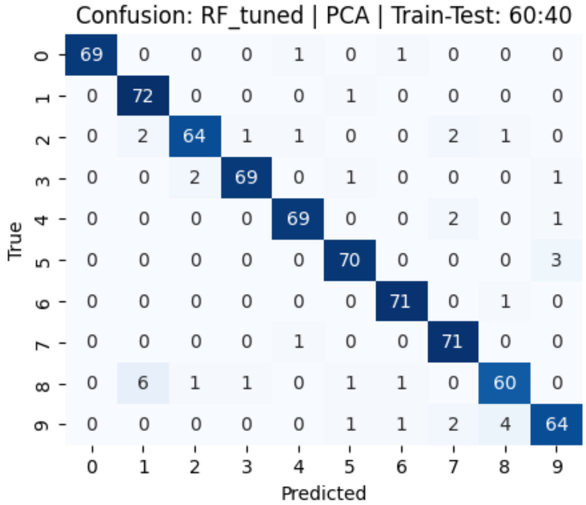

```
--- PCA | Train-Test: 60:40 ---
```

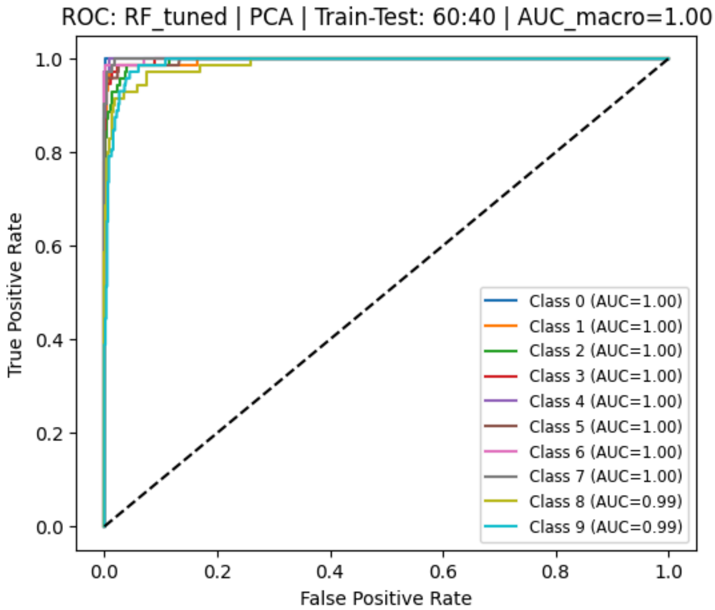




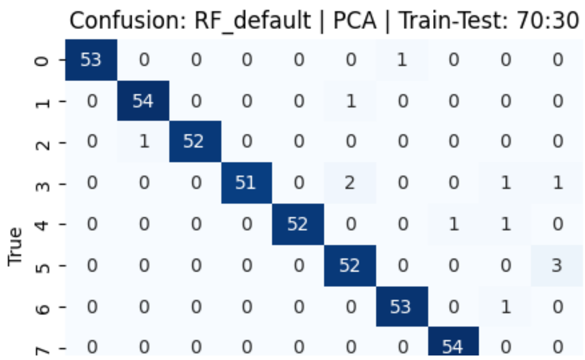


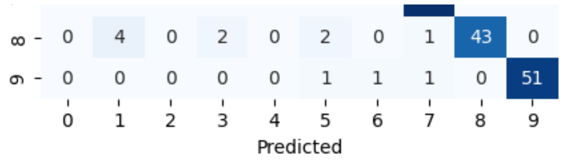

```
RF GridSearch best params: {'max_depth': None, 'min_samples_split': 5, 'n_estimators': 50}
```

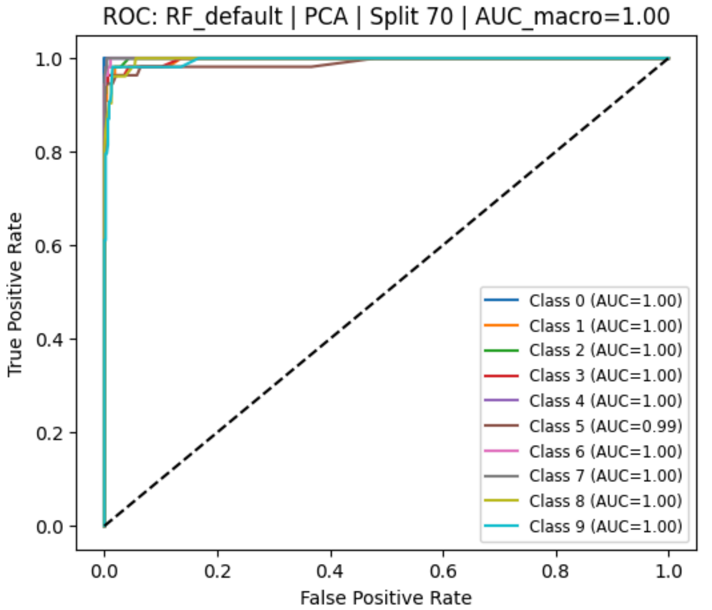




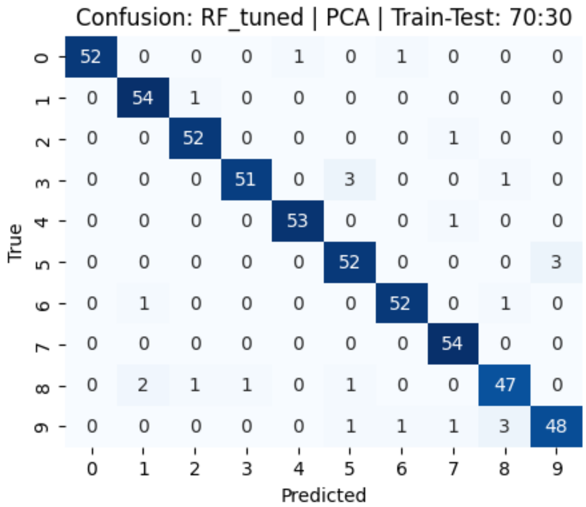

```
--- PCA | Train-Test: 70:30 ---
```

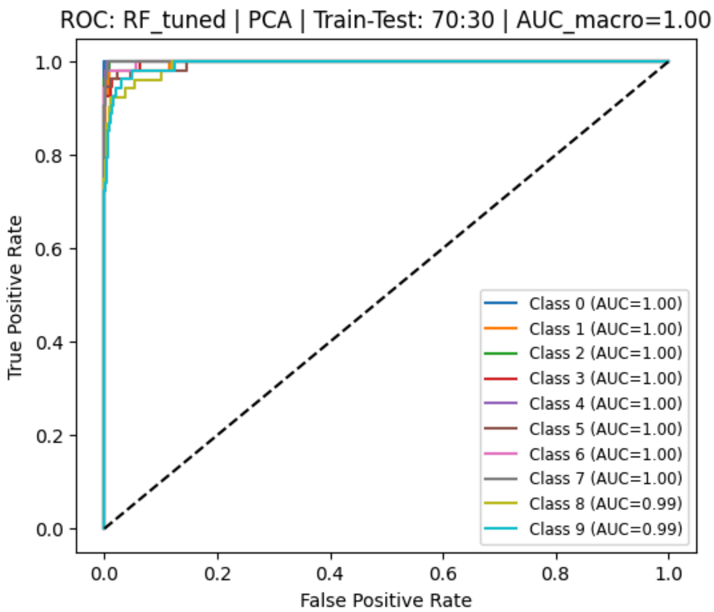




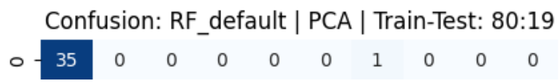


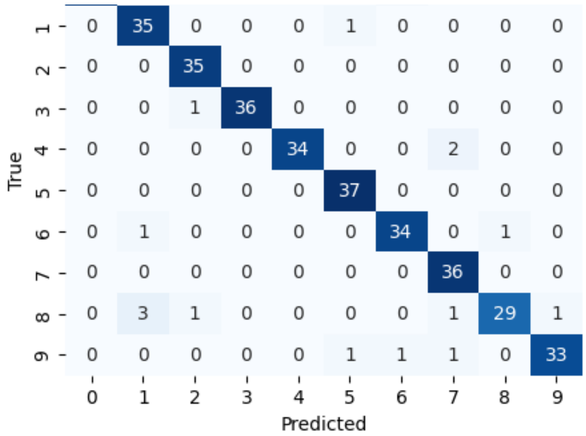

```
RF GridSearch best params: {'max_depth': 10, 'min_samples_split': 5, 'n_estimators': 100}
```

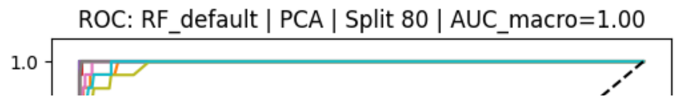





```
--- PCA | Train-Test: 80:19 ---
```




True
Predicted
1 - 0 35 0 0 0 1 0 0 0 0
2 - 0 0 35 0 0 0 0 0 0 0
3 - 0 0 1 36 0 0 0 0 0 0
4 - 0 0 0 0 34 0 0 2 0 0
5 - 0 0 0 0 0 37 0 0 0 0
6 - 0 1 0 0 0 0 34 0 1 0
7 - 0 0 0 0 0 0 0 36 0 0
8 - 0 3 1 0 0 0 0 1 29 1
9 - 0 0 0 0 0 1 1 1 0 33
0 1 2 3 4 5 6 7 8 9


ROC: RF_default | PCA | Split 80 | AUC_macro=1.00
1.0