```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

# ----------------- Load Wine Dataset -----------------
data = load_wine()
X, y = data.data, data.target

# ----------------- Train/Test Split -----------------
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# ----------------- Try Different Kernels -----------------
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 : 0.9444
Precision: 0.9522
Recall   : 0.9397
F1-score : 0.9439
Confusion Matrix:
 [[18  0  0]
 [ 1 20  0]
 [ 0  2 13]]
----------------------------------------
Kernel: poly
Accuracy : 0.6667
Precision: 0.5128
Recall   : 0.6111
F1-score : 0.5364
Confusion Matrix:
 [[15  3  0]
 [ 0 21  0]
 [ 0 15  0]]
----------------------------------------
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
Kernel: rbf
Accuracy : 0.6667
Precision: 0.4833
Recall   : 0.6111
F1-score : 0.5271
Confusion Matrix:
 [[15  3  0]
 [ 0 21  0]
 [ 2 13  0]]
----------------------------------------
Kernel: sigmoid
Accuracy : 0.2037
Precision: 0.1146
Recall   : 0.1746
F1-score : 0.1384
Confusion Matrix:
 [[ 0 18  0]
 [10 11  0]
 [12  3  0]]
----------------------------------------
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

# ----------------- Load Wine Dataset -----------------
data = load_wine()
X, y = data.data, data.target

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

# ----------------- Try Different Kernels -----------------
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 : 0.9444
Precision: 0.9583
Recall   : 0.9333
F1-score : 0.9407
Confusion Matrix:
 [[12  0  0]
 [ 0 14  0]
 [ 0  2  8]]
----------------------------------------
Kernel: poly
Accuracy : 0.6944
Precision: 0.5000
Recall   : 0.6389
F1-score : 0.5512
Confusion Matrix:
 [[11  1  0]
 [ 0 14  0]
 [ 1  9  0]]
----------------------------------------
Kernel: rbf
Accuracy : 0.6944
Precision: 0.5085
Recall   : 0.6389
F1-score : 0.5578
Confusion Matrix:
 [[11  0  1]
 [ 0 14  0]
 [ 1  9  0]]
----------------------------------------
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
Kernel: sigmoid
Accuracy : 0.1667
Precision: 0.1000
Recall   : 0.1429
F1-score : 0.1176
Confusion Matrix:
 [[ 0 12  0]
 [ 8  6  0]
 [ 8  2  0]]
----------------------------------------
```

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
```

```
# ----------------- Load Wine Dataset -----------------
data = load_wine()
X, y = data.data, data.target

# ----------------- Train/Test Split -----------------
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.4, random_state=42, stratify=y
)

# ----------------- Try Different Kernels -----------------
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 : 0.9444
Precision: 0.9514
Recall   : 0.9419
F1-score : 0.9452
Confusion Matrix:
 [[24  0  0]
 [ 2 27  0]
 [ 0  2 17]]
----------------------------------------
Kernel: poly
Accuracy : 0.6806
Precision: 0.6359
Recall   : 0.6269
F1-score : 0.6056
Confusion Matrix:
 [[19  1  4]
 [ 1 27  1]
 [ 1 15  3]]
----------------------------------------
Kernel: rbf
Accuracy : 0.6806
Precision: 0.6359
Recall   : 0.6269
F1-score : 0.6056
Confusion Matrix:
 [[19  1  4]
 [ 1 27  1]
 [ 1 15  3]]
----------------------------------------
Kernel: sigmoid
Accuracy : 0.2639
Precision: 0.1410
Recall   : 0.2208
F1-score : 0.1715
Confusion Matrix:
 [[ 1 23  0]
 [11 18  0]
 [13  6  0]]
----------------------------------------
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

# ----------------- Load Wine Dataset -----------------
data = load_wine()
X, y = data.data, data.target

# ----------------- Train/Test Split -----------------
```

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

# ----------------- Try Different Kernels -----------------
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 : 0.9101
Precision: 0.9064
Recall   : 0.9151
F1-score : 0.9088
Confusion Matrix:
 [[30  0  0]
 [ 2 29  4]
 [ 0  2 22]]
----------------------------------------
Kernel: poly
Accuracy : 0.6629
Precision: 0.6390
Recall   : 0.6218
F1-score : 0.6139
Confusion Matrix:
 [[24  1  5]
 [ 1 30  4]
 [ 0 19  5]]
----------------------------------------
Kernel: rbf
Accuracy : 0.6742
Precision: 0.6602
Recall   : 0.6444
F1-score : 0.6460
Confusion Matrix:
 [[24  0  6]
 [ 2 28  5]
 [ 0 16  8]]
----------------------------------------
Kernel: sigmoid
Accuracy : 0.3258
Precision: 0.1239
Recall   : 0.2762
F1-score : 0.1711
Confusion Matrix:
 [[ 0 30  0]
 [ 6 29  0]
 [ 5 19  0]]
----------------------------------------
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

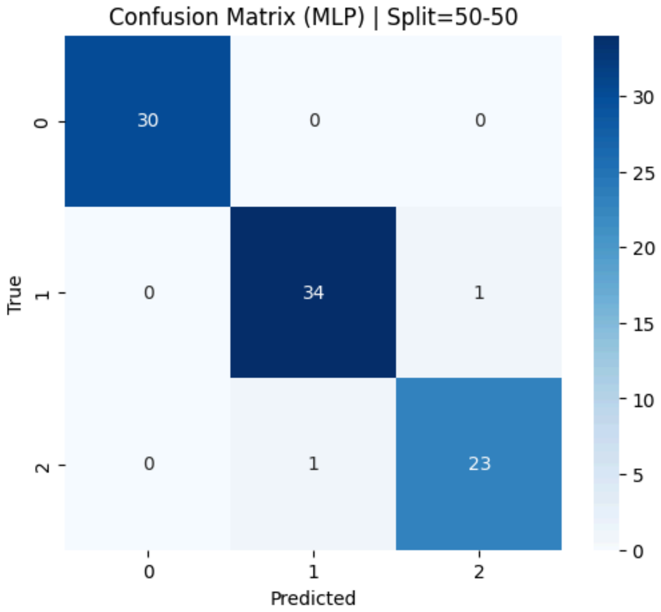
Start coding or generate with AI.

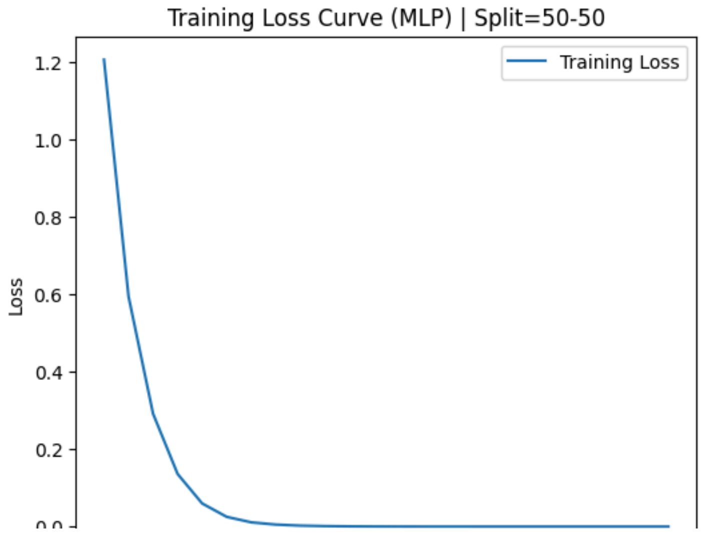
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

from sklearn.datasets import load_wine
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 ----------------------
```

```
wine = load_wine()
X, y = wine.data, wine.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)

    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=(6,5))
    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 (One-vs-All for all classes) ----------------------
    y_test_bin = label_binarize(y_test, classes=classes)
    y_score = mlp.predict_proba(X_test)

    plt.figure(figsize=(7,5))
    for i, c in enumerate(classes):
        fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_score[:, i])
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, label=f"Class {c} vs All (AUC={roc_auc:.2f})")

    plt.plot([0,1], [0,1], 'k--')
    plt.title(f"ROC Curves (MLP) | 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()
```

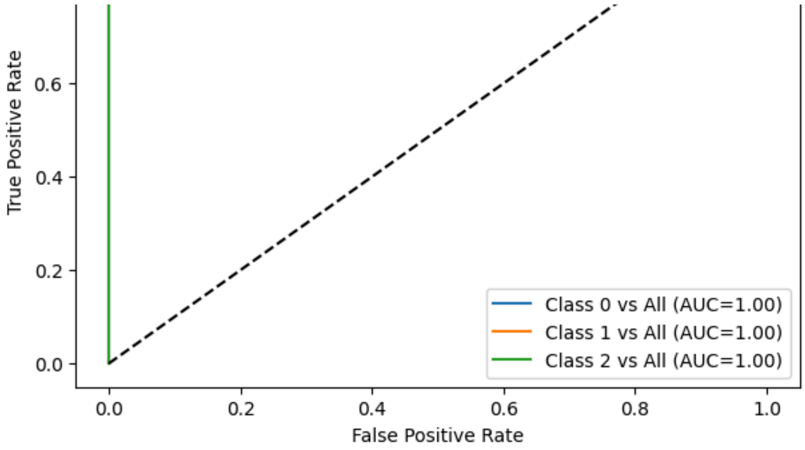
```
# ---------------------- COMPARISON TABLE ----------------------
df = pd.DataFrame(results, columns=["Split", "Accuracy", "Precision", "Recall", "F1-score"])
print("\n===== Performance Comparison =====")
print(df)
```

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

Accuracy : 0.9775
Precision: 0.9766
Recall   : 0.9766
F1-score : 0.9766
```

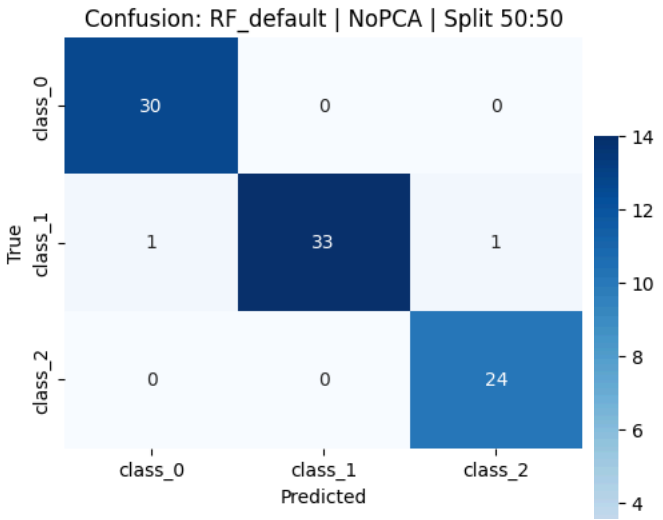




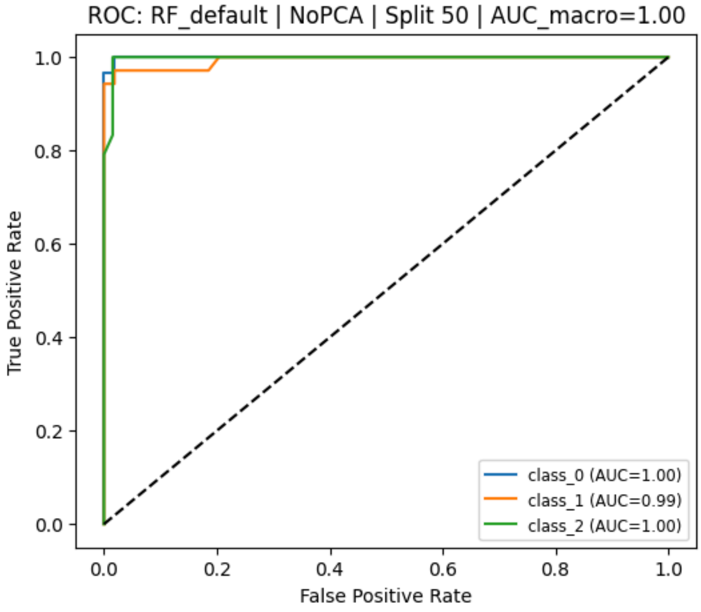
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import load_wine
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

# --- Load Wine Data ---
wine = load_wine()
X, y = wine.data, wine.target
n_classes = len(np.unique(y))
random_state = 42

# --- Configuration ---
splits = [0.5, 0.6, 0.7, 0.8]      # train ratios
do_grid_search = True              # toggle tuned RF
rf_param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}
save_plots = False                # set True to save figures instead of showing
do_pca_versions = True
pca_variance = 0.95               # keep 95% variance

results = []

# 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)
    else:
        plt.show()

# Main loop (no PCA + PCA version)
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 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

    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, stratify=y, random_state=random_state
        )

        # ---- 1) Default RF ----
        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)

        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)

        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,
                    xticklabels=wine.target_names, yticklabels=wine.target_names)
        plt.title(f"Confusion: RF_default | {version_tag} | Split {int(split*100)}:{int(test_size*100)}")
        plt.xlabel("Predicted"); plt.ylabel("True")
        show_or_save(fig, f"Confusion_RF_default_{version_tag}_Split{int(split*100)}")

        # ROC & AUC
        y_test_bin = label_binarize(y_test, classes=np.arange(n_classes))
        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])
        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()))

        fig = plt.figure(figsize=(6,5))
        for i in range(n_classes):
            plt.plot(fpr[i], tpr[i], label=f"{wine.target_names[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}_Split{int(split*100)}")

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

        # ---- 2) Tuned RF ----
        if do_grid_search:
            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
            })

            fig = plt.figure(figsize=(5,4))
            sns.heatmap(cm_t, annot=True, fmt='d', cmap='Blues', cbar=False,
                        xticklabels=wine.target_names, yticklabels=wine.target_names)
            plt.title(f"Confusion: RF_tuned | {version_tag} | Split {int(split*100)}:{int(test_size*100)}")
            plt.xlabel("Predicted"); plt.ylabel("True")
            show_or_save(fig, f"Confusion_RF_tuned_{version_tag}_Split{int(split*100)}")

            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"{wine.target_names[i]} (AUC={roc_auc_t[i]:.2f})")
            plt.plot([0,1],[0,1],'k--')
            plt.title(f"ROC: RF_tuned | {version_tag} | Split {int(split*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}_Split{int(split*100)}")

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

# Collect results
results_df = pd.DataFrame(results)
print("\n===== Performance Summary =====")
print(results_df.sort_values(['version','split','model']))
```



```
================ Train-Test Split: 80-20 ================
--- NoPCA | Train-Test: 50:50 ---
```

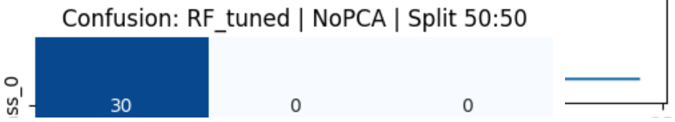


Confusion: RF_default | NoPCA | Split 50:50



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

```
RF GridSearch best params: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 50}
```



Confusion: RF_tuned | NoPCA | Split 50:50