```python
# final.py
# Simple phishing URL detection project
# (my course ML project using the PhiUSIIL dataset)

import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    accuracy_score, f1_score, precision_score, recall_score,
    roc_auc_score, confusion_matrix, classification_report
)

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier

import matplotlib.pyplot as plt


def load_and_clean_data(csv_path: str) -> tuple[pd.DataFrame, pd.Series]:
    """Load the PhiUSIIL phishing URL dataset and return X, y."""

    df = pd.read_csv(csv_path)
    print("Raw shape:", df.shape)

    # Drop text / id style columns that mess up scaling
    drop_cols = ["FILENAME", "URL", "Domain", "TLD"]
    for col in drop_cols:
        if col in df.columns:
            df = df.drop(columns=[col])

    # very simple missing value handling
    df = df.fillna(0)

    # features vs label
    target_col = "label"
    X = df.drop(columns=[target_col])
    y = df[target_col]

    print("Shape after dropping text columns:", X.shape)
    return X, y
```

```
def evaluate_model(name, model, X_train, X_test, y_train, y_test, results):
    """Train a model and print a few metrics."""

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

    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average="binary")
    prec = precision_score(y_test, y_pred, average="binary")
    rec = recall_score(y_test, y_pred, average="binary")

    # try to get ROC-AUC if possible
    try:
        if hasattr(model, "predict_proba"):
            y_scores = model.predict_proba(X_test)[:, 1]
        else:
            y_scores = model.decision_function(X_test)
        auc = roc_auc_score(y_test, y_scores)
    except Exception:
        auc = np.nan

    print(f"\n===== {name} =====")
    print(f"Accuracy : {acc:.4f}")
    print(f"F1-score : {f1:.4f}")
    print(f"Precision: {prec:.4f}")
    print(f"Recall   : {rec:.4f}")
    if not np.isnan(auc):
        print(f"ROC-AUC  : {auc:.4f}")
    else:
        print("ROC-AUC  : not available for this model setup")

    print("\nClassification report:")
    print(classification_report(y_test, y_pred))

    # confusion matrix just for a visual check
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(4, 3))
    plt.imshow(cm, cmap="Blues")
    plt.colorbar()
    plt.xticks([0, 1], ["Legit", "Phish"])
    plt.yticks([0, 1], ["Legit", "Phish"])
    plt.title(f"Confusion Matrix – {name}")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
```

```
            plt.text(j, i, cm[i, j], ha="center", va="center", color="black")
    plt.tight_layout()
    plt.show()

    results.append(
        {
            "Model": name,
            "Accuracy": acc,
            "F1-score": f1,
            "Precision": prec,
            "Recall": rec,
            "ROC-AUC": auc,
        }
    )


def main():
    # 1) load data
    X, y = load_and_clean_data("PhiUSIIL_Phishing_URL_Dataset.csv")

    # 2) 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
    )

    # 3) scale features for the models that care about it
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    results = []

    # 4) individual models
    log_reg = LogisticRegression(max_iter=1000)
    evaluate_model(
        "Logistic Regression",
        log_reg,
        X_train_scaled,
        X_test_scaled,
        y_train,
        y_test,
        results,
    )

    rf_clf = RandomForestClassifier(n_estimators=200, random_state=42, n_jobs=-1)
    evaluate_model(
```

```
    "Random Forest", rf_clf, X_train, X_test, y_train, y_test, results
)

svm_clf = SVC(kernel="rbf", C=1.0, probability=True, random_state=42)
evaluate_model(
    "SVM (RBF)", svm_clf, X_train_scaled, X_test_scaled, y_train, y_test, results
)

mlp_clf = MLPClassifier(
    hidden_layer_sizes=(64,), activation="relu", max_iter=200, random_state=42
)
evaluate_model(
    "MLP Neural Network",
    mlp_clf,
    X_train_scaled,
    X_test_scaled,
    y_train,
    y_test,
    results,
)

# 5) ensemble model (soft voting)
voting_clf = VotingClassifier(
    estimators=[
        ("lr", LogisticRegression(max_iter=1000)),
        ("rf", RandomForestClassifier(n_estimators=200, random_state=42)),
        ("svm", SVC(kernel="rbf", C=1.0, probability=True, random_state=42)),
    ],
    voting="soft",
)

evaluate_model(
    "Soft Voting Ensemble",
    voting_clf,
    X_train_scaled,
    X_test_scaled,
    y_train,
    y_test,
    results,
)

# 6) summary table
results_df = pd.DataFrame(results)
print("\n=== Summary table ===")
print(results_df)
```

```
if __name__ == "__main__":
    main()
```