```
import pandas as pd
import numpy as np
import os
import joblib
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score,
    f1_score, roc_auc_score, classification_report,
    precision_recall_curve
)

from sklearn.ensemble import RandomForestClassifier, StackingClassifier
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.calibration import CalibratedClassifierCV

# 1. Load dataset

df = pd.read_csv("/content/balanced_crime_data.csv")

# 2. Define target + drop irrelevant cols
target = "Arrest"

drop_cols = ["ID", "Case Number", "Date", "Updated On", "Location"]
df = df.drop(columns=[col for col in drop_cols if col in df.columns])

# 3. Split features & target

X = df.drop(columns=[target])
y = df[target].astype(int)
```

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

# 4. Preprocessor

numeric_features = X.select_dtypes(include=[np.number]).columns.tolist()
categorical_features = X.select_dtypes(exclude=[np.number]).columns.tolist()

numeric_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("encoder", OneHotEncoder(handle_unknown="ignore"))
])

preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features)
    ]
)

# 5. Base models with balancing

rf = RandomForestClassifier(
    n_estimators=200, random_state=42, n_jobs=-1,
    class_weight="balanced"
)

xgb = XGBClassifier(
    n_estimators=300, learning_rate=0.05, random_state=42,
    n_jobs=-1, use_label_encoder=False, eval_metric="logloss",
    scale_pos_weight=(len(y_train) - sum(y_train)) / sum(y_train)
```

```
)

base_models = [("rf", rf), ("xgb", xgb)]

# 6. Meta model (MLP) with calibration

meta_model = MLPClassifier(
    hidden_layer_sizes=(64, 32),
    activation="relu",
    solver="adam",
    max_iter=300,
    random_state=42,
    early_stopping=True
)
calibrated_meta = CalibratedClassifierCV(meta_model, method="isotonic", cv=3)

# 7. Stacking classifier

stacking_clf = StackingClassifier(
    estimators=base_models,
    final_estimator=calibrated_meta,
    passthrough=True,
    n_jobs=-1
)

# 8. Full pipeline

pipeline = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("stacking", stacking_clf)
])

# 9. Train

print("Training stacking classifier...")
pipeline.fit(X_train, y_train)
```

```
# ----------------------------
# 10. Create directory & save model
# ----------------------------

model_dir = "/mnt/data/saved_models"
model_filename = "crime_stacking_model_v2.pkl"
os.makedirs(model_dir, exist_ok=True)
model_path = os.path.join(model_dir, model_filename)

joblib.dump(pipeline, model_path)
print(f"Model directory ready: {model_dir}")
print(f"Model successfully saved at: {model_path}")

# 11. Evaluate
y_pred = pipeline.predict(X_test)
y_proba = pipeline.predict_proba(X_test)[:, 1]

print("\nClassification Report (Default Threshold 0.5):\n", classification_report(y_test,
y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
print(" Precision:", precision_score(y_test, y_pred))
print(" Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print(" ROC-AUC:", roc_auc_score(y_test, y_proba))

# 12. Threshold tuning function

def tune_threshold(y_true, y_proba, metric="f1"):
    precisions, recalls, thresholds = precision_recall_curve(y_true, y_proba)
    best_threshold, best_score = 0.5, 0
    for p, r, t in zip(precisions, recalls, thresholds):
        if metric == "f1":
            score = 2 * (p * r) / (p + r + 1e-6)
        elif metric == "recall":
            score = r
        elif metric == "precision":
            score = p
```

```
        else:
            raise ValueError("Metric must be 'f1', 'recall', or 'precision'")
        if score > best_score:
            best_threshold, best_score = t, score
    return best_threshold, best_score

# 13. Tune threshold for F1

best_t, best_f1 = tune_threshold(y_test, y_proba, metric="f1")
print(f"\n Best Threshold for F1: {best_t:.3f} | F1 Score: {best_f1:.4f}")

# Apply tuned threshold

y_pred_tuned = (y_proba >= best_t).astype(int)
print("\n Classification Report (Tuned Threshold):\n", classification_report(y_test,
y_pred_tuned))

# 14. Optional: plot precision–recall vs threshold

precisions, recalls, thresholds = precision_recall_curve(y_test, y_proba)
plt.figure(figsize=(8,6))
plt.plot(thresholds, precisions[:-1], label="Precision")
plt.plot(thresholds, recalls[:-1], label="Recall")
plt.axvline(x=best_t, color="red", linestyle="--", label=f"Best Threshold {best_t:.2f}")
plt.xlabel("Threshold")
plt.ylabel("Score")
plt.title("Precision-Recall vs Threshold")
plt.legend()
plt.grid(True)
plt.show()
```