```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, balanced_accuracy_score,
    matthews_corrcoef
from sklearn.model_selection import StratifiedKFold

np.random.seed(42)

# Load 14-object data (as in previous response)
data = {
    'Name': ['J0534+2200', 'J1853+1303', 'XTE J1810-197', '1E 2259+586', 'CXOU 
        J171405', 'SGR 0418+5729',
             'SGR 1806-20', 'SGR 0501+4516', '3C 273', 'SDSS J1220+2916', 'PKS 
                 2155-304', 'BL Lacertae', 'Sgr A*', 'M87*'],
    'Type': ['Pulsar', 'Pulsar', 'AXP', 'AXP', 'Slow Magnetar', 'Slow 
        Magnetar', 'Powerful Magnetar', 'Powerful Magnetar',
             'Quasar', 'Quasar', 'Blazar', 'Blazar', 'Black Hole', 'Black 
                 Hole'],
    'B': [12.6, 4.3, 15.0, 10.0, 8.0, 6.0, 20.0, 18.0, 0.0, 0.0, 0.0, 0.0,
        0.0, 0.0],
    'P': [0.033, 0.267, 5.54, 6.98, 3.82, 9.08, 7.56, 5.76, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0],
    'N_glt': [24, 2, 3, 4, 1, 0, 5, 6, 0, 0, 0, 0, 0, 0],
    'L_bol': [4.5, 0.1, 0.05, 0.03, 0.01, 0.005, 0.1, 0.08, 1000.0, 800.0,
        500.0, 300.0, 0.0001, 0.001],
    'rho_DM': [0.1, 0.08, 0.12, 0.11, 0.09, 0.07, 0.15, 0.14, 0.5, 0.4, 0.3,
        0.2, 1.0, 0.8],
    'Z': [1.0, 0.8, 1.2, 1.1, 0.9, 0.7, 1.3, 1.2, 2.0, 1.8, 1.5, 1.4, 0.5,
        0.6],
    'T': [1000, 800, 1200, 1100, 900, 700, 1300, 1200, 5000, 4500, 4000, 3500,
        100, 150],
    'P_env': [1.0, 0.9, 1.1, 1.0, 0.8, 0.7, 1.2, 1.1, 2.0, 1.8, 1.6, 1.5, 0.1,
        0.2]
}
df = pd.DataFrame(data)

# Calculate T_urb and sigma_env
def calculate_features(df, kappa=0.5, eta=0.3, gamma=0.2, epsilon=0.1):
    df['T_urb'] = np.where(df['Type'].isin(['Pulsar', 'AXP', 'Slow Magnetar',
        'Powerful Magnetar']),
                           (df['B'] / 10) * (df['P'] / 1) * (df['N_glt'] + 1) /
                               (df['L_bol'] + 0.01),
                           np.where(df['Type'].isin(['Quasar', 'Blazar']),
                                    0.1 * (df['L_bol'] / 100), 1e-3))
```

```
    df['sigma_env'] = kappa * df['rho_DM'] + eta * df['Z'] + gamma / df['T'] +
        epsilon / df['P_env']
    return df

df = calculate_features(df)

# Synthetic training data (700 samples, 100 per class)
synth_data = []
for cls in df['Type'].unique():
    if cls in ['Pulsar', 'AXP', 'Slow␣Magnetar', 'Powerful␣Magnetar']:
        T_urb = np.random.uniform(0.1, 3.0, 100)
        sigma_env = np.random.uniform(0.5, 1.5, 100)
    elif cls in ['Quasar', 'Blazar']:
        T_urb = np.random.uniform(0.01, 0.5, 100)
        sigma_env = np.random.uniform(1.0, 2.0, 100)
    else:
        T_urb = np.random.uniform(1e-4, 1e-3, 100)
        sigma_env = np.random.uniform(0.1, 0.5, 100)
    synth_data.append(pd.DataFrame({'T_urb': T_urb, 'sigma_env': sigma_env,
        'Type': [cls] * 100}))
synth_df = pd.concat(synth_data, ignore_index=True)

# Prepare features
X = df[['T_urb', 'sigma_env']].values
y = df['Type'].values
X_synth = synth_df[['T_urb', 'sigma_env']].values
y_synth = synth_df['Type'].values

scaler = StandardScaler()
X_synth_scaled = scaler.fit_transform(X_synth)
X_scaled = scaler.transform(X)

# Parameter sensitivity (50% for , , )
params = {'kappa': [0.25, 0.5, 0.75], 'eta': [0.15, 0.3, 0.45], 'gamma':
    [0.1, 0.2, 0.3]}
auc_results = []
for k in params['kappa']:
    for e in params['eta']:
        for g in params['gamma']:
            df_temp = calculate_features(df.copy(), kappa=k, eta=e, gamma=g)
            X_temp = scaler.transform(df_temp[['T_urb', 'sigma_env']].values)
            clf = LogisticRegression(multi_class='multinomial', max_iter=1000,
                random_state=42)
            clf.fit(X_synth_scaled, y_synth)
            probs = clf.predict_proba(X_temp)
            auc = roc_auc_score(y, probs, multi_class='ovr')
            auc_results.append((k, e, g, auc))

# Bootstrapped balanced accuracy and MCC
n_boot = 100
bal_accs, mccs = [], []
```

```
for _ in range(n_boot):
    idx = np.random.choice(len(y), len(y), replace=True)
    X_boot = X_scaled[idx]
    y_boot = y[idx]
    clf = LogisticRegression(multi_class='multinomial', max_iter=1000,
        random_state=42)
    clf.fit(X_synth_scaled, y_synth)
    y_pred = clf.predict(X_boot)
    bal_accs.append(balanced_accuracy_score(y_boot, y_pred))
    mccs.append(matthews_corrcoef(y_boot, y_pred))

# Jackknife test for _DM (0.3 dex)
jackknife_accs = []
for i in range(len(df)):
    df_jack = df.drop(i).copy()
    df_jack['rho_DM'] = df_jack['rho_DM'] * 10**np.random.uniform(-0.3, 0.3,
        len(df_jack))
    df_jack = calculate_features(df_jack)
    X_jack = scaler.transform(df_jack[['T_urb', 'sigma_env']].values)
    y_jack = df_jack['Type'].values
    clf.fit(X_synth_scaled, y_synth)
    y_pred = clf.predict(X_jack)
    jackknife_accs.append(accuracy_score(y_jack, y_pred))

# Results
print("AUC Sensitivity:", auc_results)
print("Bootstrapped Balanced Accuracy:", np.mean(bal_accs), "",
    np.std(bal_accs))
print("Bootstrapped MCC:", np.mean(mccs), "", np.std(mccs))
print("Jackknife Accuracy:", np.mean(jackknife_accs), "",
    np.std(jackknife_accs))
```