### SAMPLE SOURCE CODE

# 1. Imports

import pandas as pd import numpy as np

import matplotlib.pyplot as plt import seaborn as sns

import folium import datetime

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier, VotingClassifier from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, confusion\_matrix, roc\_curve

from sklearn.feature\_selection import RFE

# 2. Load and Preprocess Data print("Q Loading and preprocessing sensor data...")

X\_train\_raw = pd.read\_csv('UCI HAR Dataset/train/X\_train.txt', sep='\s+', header=None)

y\_train\_raw = pd.read\_csv('UCI HAR Dataset/train/y\_train.txt', header=None) X\_test\_raw = pd.read\_csv('UCI HAR Dataset/test/X\_test.txt', sep='\s+', header=None)

y\_test\_raw = pd.read\_csv('UCI HAR Dataset/test/y\_test.txt', header=None) activity\_labels = pd.read\_csv('UCI HAR Dataset/activity\_labels.txt', sep='\s+', header=None, names=['ID', 'Activity'])

X = pd.concat([X\_train\_raw, X\_test\_raw])

y = pd.concat([y\_train\_raw, y\_test\_raw]).values.ravel() y\_named = pd.Series(y).map(dict(zip(activity\_labels.ID, activity\_labels.Activity)))

scaler = StandardScaler() X\_scaled = scaler.fit\_transform(X)

rfe = RFE(RandomForestClassifier(), n\_features\_to\_select=50)

X\_selected = rfe.fit\_transform(X\_scaled, y)

# 3. Train-Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_selected, y, test\_size=0.2, random\_state=42)

# 4. Model Setup

rf = RandomForestClassifier(n\_estimators=100) svm = SVC(probability=True)

knn = KNeighborsClassifier(n\_neighbors=5)

ensemble = VotingClassifier(estimators=[('rf', rf), ('svm', svm), ('knn', knn)], voting='soft')

models = [rf, svm, knn, ensemble]

model\_names = ['Random Forest', 'SVM', 'KNN', 'Voting Ensemble']

# 5. Train Models

for model in models: model.fit(X\_train, y\_train)

# 6. Evaluate Models def evaluate(model, name):

y\_pred = model.predict(X\_test) y\_prob = model.predict\_proba(X\_test) print(f"\n📊 {name} Performance:")

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred):.2f}")

print(f"Precision: {precision\_score(y\_test, y\_pred, average='weighted'):.2f}") print(f"Recall: {recall\_score(y\_test, y\_pred, average='weighted'):.2f}") print(f"F1-Score: {f1\_score(y\_test, y\_pred, average='weighted'):.2f}") print(f"AUC: {roc\_auc\_score(pd.get\_dummies(y\_test), y\_prob,

multi\_class='ovr'):.2f}")

for model, name in zip(models, model\_names): evaluate(model, name)

# 7. Confusion Matrix y\_pred = ensemble.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred) plt.figure(figsize=(6, 4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.title("Confusion Matrix - Voting Ensemble") plt.xlabel("Predicted")

plt.ylabel("Actual") plt.tight\_layout() plt.show()

# 8. ROC Curve

y\_prob = ensemble.predict\_proba(X\_test)

fpr, tpr, \_ = roc\_curve(pd.get\_dummies(y\_test).iloc[:, 0], y\_prob[:, 0]) plt.figure(figsize=(6, 4))

plt.plot(fpr, tpr, label='Voting Ensemble') plt.plot([0, 1], [0, 1], 'k--')

plt.xlabel("False Positive Rate") plt.ylabel("True Positive Rate") plt.title("ROC Curve") plt.legend()

plt.tight\_layout() plt.show()

# 9. Alert Logic

alert\_log = []

def trigger\_alert(activity\_id, location):

activity = dict(zip(activity\_labels.ID, activity\_labels.Activity))[activity\_id] if activity in ['LAYING', 'SITTING']:

print(f"\n ALERT: {activity} detected")

print(f"Time: {datetime.datetime.now().strftime('%Y-%m-%d

%H:%M:%S')}")

print(f"Location: {location}") log\_alert(activity, location) show\_map(location)

def log\_alert(activity, location):

timestamp = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S') alert\_log.append({'Time': timestamp, 'Activity': activity, 'Location': location}) print("📋 Alert logged.")

def show\_map(location): lat, lon = location

map\_alert = folium.Map(location=[lat, lon], zoom\_start=16) folium.Marker([lat, lon], popup='Alert Triggered',

icon=folium.Icon(color='red')).add\_to(map\_alert) map\_alert.save('alert\_map.html')

print("🗺◻ Map saved as alert\_map.html")

# 10. Simulate Alert sample\_input = X\_test[0].reshape(1, -1)

predicted\_label = ensemble.predict(sample\_input)[0] sample\_location = (13.0827, 80.2707) # Chennai coordinates trigger\_alert(predicted\_label, sample\_location)

# 11. Export Logs

df\_log = pd.DataFrame(alert\_log) df\_log.to\_csv('alert\_log.csv', index=False) print("\n📁 Alert log exported to alert\_log.csv")

# 12. Visual Summary

metrics = ['Accuracy', 'Precision', 'Recall', 'F1-Score', 'AUC'] scores = []

for model in models:

y\_pred = model.predict(X\_test) y\_prob = model.predict\_proba(X\_test) scores.append([

accuracy\_score(y\_test, y\_pred), precision\_score(y\_test, y\_pred, average='weighted'), recall\_score(y\_test, y\_pred, average='weighted'), f1\_score(y\_test, y\_pred, average='weighted'),

roc\_auc\_score(pd.get\_dummies(y\_test), y\_prob, multi\_class='ovr')

])

df\_scores = pd.DataFrame(scores, columns=metrics, index=model\_names) df\_scores.plot(kind='bar', figsize=(10, 6), colormap='Dark2') plt.title("Figure A.2.4: Model Performance Comparison") plt.ylabel("Score")

plt.ylim(0.85, 1.0)

plt.grid(axis='y', linestyle='--', alpha=0.5) plt.tight\_layout()

plt.show()