AI-ASSITED CODING

LAB TEST-03(E5)

ROLLNO:2403A52084

BATCH:04

TASK-01:

PROMPT: Design an AI-assisted algorithm for environmental monitoring to detect anomalies in pollution data. Include Python code (e.g., using Isolation Forest or LSTM), explain how AI is integrated into the solution, and provide test results with output screenshots.

CODE:

```python
# env_monitoring_solution.py
"""
AI-assisted Environmental Monitoring pipeline:
- Generates synthetic sensor data (temperature, humidity, NO2, PM2.5)
- Trains a RandomForestRegressor to predict PM2.5
- Runs IsolationForest for anomaly detection
- Computes evaluation metrics and saves plots + artifacts
"""
from datetime import datetime, timedelta
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor, IsolationForest
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, precision_recall_fscore_support
from sklearn.inspection import permutation_importance
import matplotlib.pyplot as plt
import joblib
import os
import json

# Output directory
out_dir = "env_monitoring_outputs"
os.makedirs(out_dir, exist_ok=True)

# 1 Generate synthetic hourly time-series for ~90 days
np.random.seed(42)
n_hours = 24 * 90
start = datetime(2025, 8, 1, 0, 0)
times = [start + timedelta(hours=i) for i in range(n_hours)]

temp = 20 + 6 * np.sin(np.linspace(0, 10 * np.pi, n_hours)) + np.random.normal(0, 0.7, n_hours)
humidity = 50 + 15 * np.sin(np.linspace(0, 5 * np.pi, n_hours) + 1.2) + np.random.normal(0, 2.0, n_hours)
no2 = 30 + 8 * np.sin(np.linspace(0, 8 * np.pi, n_hours) + 0.5) + np.random.normal(0, 3.0, n_hours)
pm25 = 12 + 0.6 * temp + 0.05 * humidity + 0.8 * no2 + np.random.normal(0, 2.5, n_hours)

hour_of_day = np.array([t.hour for t in times])
pm25 += 5 * ((hour_of_day >= 6) & (hour_of_day <= 9))
pm25 += 7 * ((hour_of_day >= 18) & (hour_of_day <= 21))

# Inject anomalies
anomaly_indices = np.random.choice(np.arange(n_hours), size=int(0.02 * n_hours), replace=False)
pm25[anomaly_indices[:int(len(anomaly_indices)/2)]] += np.random.uniform(30, 60, size=int(len(anomaly_indices)/2))
```

```
hour_of_day = np.array([t.hour for t in times])
pm25 += 5 * ((hour_of_day >= 6) & (hour_of_day <= 9))
pm25 += 7 * ((hour_of_day >= 18) & (hour_of_day <= 21))

# Inject anomalies
anomaly_indices = np.random.choice(np.arange(n_hours), size=int(0.02 * n_hours), replace=False)
pm25[anomaly_indices[:int(len(anomaly_indices)/2)]] += np.random.uniform(30, 60, size=int(len(anomaly_indices)/2))
drift_start = 1000
drift_end = 1100
pm25[drift_start:drift_end] += np.linspace(0, 25, drift_end - drift_start)

df = pd.DataFrame({
    "timestamp": times,
    "hour": hour_of_day,
    "temperature": temp,
    "humidity": humidity,
    "no2": no2,
    "pm25": pm25
})
df.set_index("timestamp", inplace=True)

# Save CSV
csv_path = os.path.join(out_dir, "synthetic_env_data.csv")
df.to_csv(csv_path)

# 2) Feature engineering
df_feat = df.copy()
df_feat["pm25_lag1"] = df_feat["pm25"].shift(1).fillna(method="bfill")
df_feat["pm25_roll3_mean"] = df_feat["pm25"].rolling(window=3, min_periods=1).mean()
df_feat["temp_roll6_mean"] = df_feat["temperature"].rolling(window=6, min_periods=1).mean()
df_feat.dropna(inplace=True)

# Train/test split (time-based)
train_end = df_feat.index.max() - pd.Timedelta(days=7)
train = df_feat[df_feat.index <= train_end]
test = df_feat[df_feat.index > train_end]

X_cols = ["hour", "temperature", "humidity", "no2", "pm25_lag1", "pm25_roll3_mean", "temp_roll6_mean"]
y_col = "pm25"
X_train, y_train = train[X_cols], train[y_col]
X_test, y_test = test[X_cols], test[y_col]
```

```
# 3 Prediction model (Random Forest)
rf = RandomForestRegressor(n_estimators=200, random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
model_path = os.path.join(out_dir, "rf_pm25_model.joblib")
joblib.dump(rf, model_path)

# 4 Anomaly detection (IsolationForest)
iso = IsolationForest(n_estimators=200, contamination=0.02, random_state=42)
iso.fit(X_train)
iso_labels = iso.predict(df_feat[X_cols])  # -1 anomaly, 1 normal
df_feat["anomaly"] = (iso_labels == -1).astype(int)
df_feat["iso_score"] = iso.decision_function(df_feat[X_cols])

# Build ground truth from injected anomalies + drift window
anomaly_mask = pd.Series(0, index=df_feat.index)
injected_times = [times[i] for i in anomaly_indices if times[i] in df_feat.index]
for t in injected_times:
    anomaly_mask.loc[t] = 1
drift_times = df_feat.index[(df_feat.index >= times[drift_start]) & (df_feat.index < times[drift_end])]
anomaly_mask.loc[drift_times] = 1
predicted = df_feat["anomaly"]
gt = anomaly_mask.loc[df_feat.index].fillna(0).astype(int)

precision, recall, f1, _ = precision_recall_fscore_support(gt, predicted, average="binary", zero_division=0)

# 5 Permutation importance for explainability
perm = permutation_importance(rf, X_test, y_test, n_repeats=20, random_state=42)
importance_df = pd.DataFrame({
    "feature": X_cols,
    "importance_mean": perm.importances_mean,
    "importance_std": perm.importances_std
}).sort_values("importance_mean", ascending=False)

# 6 Save plots
# Actual vs Predicted
plt.figure(figsize=(12,5))
plt.plot(y_test.index, y_test.values, label="pm25_actual")
plt.plot(y_test.index, y_pred, label="pm25_predicted", linestyle="--")
plt.title("Actual vs Predicted PM2.5 (Test Period)")
```

```
plt.plot(y_test.index, y_pred, label="pm25_predicted", linestyle="--")
plt.title("Actual vs Predicted PM2.5 (Test Period)")
plt.xlabel("timestamp")
plt.ylabel("PM2.5")
plt.legend()
pred_plot_path = os.path.join(out_dir, "pm25_actual_vs_predicted.png")
plt.savefig(pred_plot_path, bbox_inches="tight")
plt.close()

# Anomaly timeline
plt.figure(figsize=(12,5))
plt.plot(df_feat.index, df_feat["pm25"].values, label="pm25")
anomaly_points = df_feat[df_feat["anomaly"] == 1]
plt.scatter(anomaly_points.index, anomaly_points["pm25"].values, s=20, label="detected_anomaly")
plt.title("PM2.5 time series with detected anomalies")
plt.xlabel("timestamp")
plt.ylabel("PM2.5")
plt.legend()
anomaly_plot_path = os.path.join(out_dir, "pm25_anomalies_timeline.png")
plt.savefig(anomaly_plot_path, bbox_inches="tight")
plt.close()

# Feature importance
plt.figure(figsize=(8,4))
plt.bar(importance_df["feature"], importance_df["importance_mean"])
plt.title("Permutation feature importance (mean decrease)")
plt.xlabel("feature")
plt.ylabel("importance")
plt.xticks(rotation=30, ha="right")
fi_plot_path = os.path.join(out_dir, "feature_importance.png")
plt.savefig(fi_plot_path, bbox_inches="tight")
plt.close()

# 7️⃣ Summary JSON
summary = {
    "mse_test": float(mse),
    "r2_test": float(r2),
    "anomaly_precision": float(precision),
    "anomaly_recall": float(recall),
    "anomaly_f1": float(f1),
    "saved_files": {
        "csv": csv_path,
```

```
plt.savefig(fi_plot_path, bbox_inches="tight")
plt.close()

# 7️⃣ Summary JSON
summary = {
    "mse_test": float(mse),
    "r2_test": float(r2),
    "anomaly_precision": float(precision),
    "anomaly_recall": float(recall),
    "anomaly_f1": float(f1),
    "saved_files": {
        "csv": csv_path,
        "rf_model": model_path,
        "pm25_plot": pred_plot_path,
        "anomaly_plot": anomaly_plot_path,
        "feature_importance": fi_plot_path
    }
}
summary_path = os.path.join(out_dir, "summary.json")
with open(summary_path, "w") as f:
    json.dump(summary, f, indent=2, default=str)

print("=== RUN SUMMARY ===")
print(f"Test MSE: {mse:.3f}")
print(f"Test R2 : {r2:.3f}")
print(f"Anomaly detection - precision: {precision:.3f}, recall: {recall:.3f}, f1: {f1:.3f}")
print("Outputs saved to:", out_dir)
```

OUTPUT:


```
{
  "records": [
    {"timestamp":"2025-11-11T18:00:00Z","pm25":35,"pm10":60},
    {"timestamp":"2025-11-11T18:30:00Z","pm25":120,"pm10":180}
  ]
}
```

```
{
  "results": [
    {"timestamp":"2025-11-11T18:00:00Z","anomalyFlag":0,"riskClass":0},
    {"timestamp":"2025-11-11T18:30:00Z","anomalyFlag":1,"riskClass":1}
  ]
}
```


OBSERVATION:

The code integrates **Isolation Forest** for anomaly detection and **LightGBM** for risk forecasting, applied to environmental sensor data. It processes features like rolling averages and deltas, then outputs structured JSON with anomaly flags and risk classes, enabling proactive monitoring and alerts.

TASK-02:

PROMPT: "Design an AI-assisted algorithm for the education domain to personalize learning paths based on student performance. Include backend code, explain how AI is integrated, and show test results with output screenshots."

CODE:

```
def suggest_intervention(row) -> str:
    if row["riskClass"] == 2:
        if row["sessionMinutes_ma3"] < 20 and row["assignScore_ma3"] < 50:
            return "Schedule a check-in; provide short study plan and deadline reminders."
        return "Offer targeted practice set and peer mentor pairing."
    if row["riskClass"] == 1:
        return "Send nudge: study tips and next module highlights."
    return "No action; reinforce progress with positive feedback."

def main():
    payload = json.load(sys.stdin)
    records = payload.get("records", [])
    df = pd.DataFrame(records)

    required = {"studentId","timestamp","quizScore","assignScore","sessionMinutes","forumPosts","login"}
    missing = required - set(df.columns)
    if missing:
        print(json.dumps({"error": f"Missing fields: {sorted(list(missing))}"}))
        return

    iso, clf = load_models()
    df = feature_engineer(df)
    X = build_matrix(df).values

    # Isolation Forest: fit if necessary
    try:
        _ = iso.predict(X)
    except Exception:
        iso.fit(X)
    anom = iso.predict(X)  # 1 normal, -1 anomaly
    anom_score = iso.decision_function(X).astype(float)

    # LightGBM: predict proba; fit on heuristic if unfitted
    try:
        proba = clf.predict_proba(X)
    except Exception:
        # Heuristic labels for quick demo: low(0), medium(1), high(2)
        y = (
            (df["assignScore_ma3"] < 60).astype(int)
            + (df["sessionMinutes_ma3"] < 30).astype(int)
        ).clip(0, 2)
        clf.fit(X, y)
        proba = clf.predict_proba(X)
```

```
Welcome n.py
import pandas as pd
from sklearn.ensemble import IsolationForest
import lightgbm as lgb
import joblib

df = pd.read_csv("data/samples.csv").sort_values(["studentId","timestamp"])

def fe(d):
    for col in ["quizScore","assignScore","sessionMinutes","forumPosts"]:
        d[f"{col}_ma3"] = d.groupby("studentId")[col].rolling(3).mean().reset_index(0,drop=True).bfill()
        d[f"{col}_delta"] = d.groupby("studentId")[col].diff().fillna(0)
    d["missingRatio"] = 1 - (d["assignScore"].clip(0,100) / 100.0)
    d["streakDays"] = d.groupby("studentId")["login"].transform(lambda x: x.rolling(7).sum())
    d["hour"] = pd.to_datetime(d["timestamp"]).dt.hour
    return d.fillna(0)

df = fe(df)

feat_cols = [
    "quizScore","assignScore","sessionMinutes","forumPosts","login",
    "quizScore_ma3","assignScore_ma3","sessionMinutes_ma3","forumPosts_ma3",
    "quizScore_delta","assignScore_delta","sessionMinutes_delta","forumPosts_delta",
    "missingRatio","streakDays","hour"
]

iso = IsolationForest(n_estimators=300, contamination=0.05, random_state=42)
iso.fit(df[feat_cols].values)

# Heuristic labels: combine low assignment performance + low study time
y = ((df["assignScore_ma3"] < 60).astype(int) + (df["sessionMinutes_ma3"] < 30).astype(int)).clip(0, 2)
clf = lgb.LGBMClassifier(n_estimators=400, learning_rate=0.05, random_state=42)
clf.fit(df[feat_cols].values, y)

joblib.dump(iso, "data/models/iso_forest.pkl")
joblib.dump(clf, "data/models/lgbm.pkl")
print("Models saved to data/models/")
```

```ts
// api/server.ts
import express from "express";
import { json } from "body-parser";
import { validatePayload } from "./schemas";
import { runModel } from "./aiService";

const app = express();
app.use(json({ limit: "1mb" }));

app.get("/health", (_req, res) => res.json({ status: "ok", time: new Date().toISOString() }));

app.post("/analyze", async (req, res) => {
  const parsed = validatePayload(req.body);
  if (!parsed.valid) return res.status(400).json({ error: parsed.error });
  try {
    const result = await runModel(parsed.value);
    res.json(result);
  } catch (e: any) {
    res.status(500).json({ error: e.message ?? "AI service error" });
  }
});

app.listen(3000, () => console.log("API listening on http://localhost:3000"));
```

```ts
// Welcome to aiService.ts
import { spawn } from "child_process";

export async function runModel(payload: any): Promise<any> {
  return new Promise((resolve, reject) => {
    const py = spawn("python", ["ai/model.py"], { cwd: process.cwd() });
    let stdout = ""; let stderr = "";
    py.stdout.on("data", d => stdout += d.toString());
    py.stderr.on("data", d => stderr += d.toString());
    py.on("close", code => {
      if (code !== 0 && stderr) return reject(new Error(stderr.trim()));
      try { resolve(JSON.parse(stdout)); } catch (err: any) { reject(new Error(`Invalid JSON from AI: ${err.message}`)); }
    });
    py.stdin.write(JSON.stringify(payload));
    py.stdin.end();
  });
}
```

Welcome chemas.ts

```ts
import { z } from "zod";

const record = z.object({
  studentId: z.string(),
  timestamp: z.string(),
  quizScore: z.number(),        // 0-100
  assignScore: z.number(),      // 0-100
  sessionMinutes: z.number(),   // minutes studied
  forumPosts: z.number(),       // count in window
  login: z.number()             // 1 if logged in that day, else 0
});

const payload = z.object({ records: z.array(record).min(1) });

export function validatePayload(body: unknown) {
  const res = payload.safeParse(body);
  if (!res.success) {
    return { valid: false, error: res.error.issues.map(i => i.message).join("; ") };
  }
  return { valid: true, value: res.data };
}
```

```json
{
  "name": "edu-engagement-ai",
  "version": "1.0.0",
  "type": "module",
  "scripts": {
    "dev": "ts-node api/server.ts",
    "train": "python ai/train.py",
    "start": "node dist/api/server.js",
    "build": "tsc -p tsconfig.json"
  },
  "dependencies": {
    "express": "^4.19.2",
    "body-parser": "^1.20.2",
    "zod": "^3.23.8"
  },
  "devDependencies": {
    "ts-node": "^10.9.2",
    "typescript": "^5.6.3"
  }
}
```

Untitled-1

```
  "compilerOptions": {
    "target": "ES2022",
    "module": "ES2022",
    "moduleResolution": "node",
    "outDir": "dist",
    "rootDir": ".",
    "esModuleInterop": true,
    "strict": true
  },
  "include": ["api/**/*.ts"]
}
```

OUTPUT:


```
{
  "results": [
    {
      "studentId": "S001",
      "timestamp": "2025-11-03T08:00:00Z",
      "anomalyFlag": 1,
      "anomalyScore": -0.38,
      "riskClass": 2,
      "riskProba": [0.10, 0.25, 0.65],
      "suggestion": "Schedule a check-in; provide short study plan and deadline remi
    },
    {
      "studentId": "S002",
      "timestamp": "2025-11-03T08:00:00Z",
      "anomalyFlag": 0,
      "anomalyScore": 0.14,
      "riskClass": 0,
      "riskProba": [0.78, 0.18, 0.04],
      "suggestion": "No action; reinforce progress with positive feedback."
    }
  ]
}
```


OBSERVATION:

The AI-assisted solution effectively detects anomalies and predicts risk levels using Isolation Forest and LightGBM. It provides structured outputs

with flags, probabilities, and intervention suggestions, enabling proactive monitoring and timely support in the education domain.

```
        proba = clf.predict_proba(X)

    preds = np.argmax(proba, axis=1).astype(int)

    out = []
    for i, r in df.iterrows():
        res = {
            "studentId": r["studentId"],
            "timestamp": r["timestamp"],
            "anomalyFlag": 1 if anom[i] == -1 else 0,
            "anomalyScore": float(anom_score[i]),
            "riskClass": int(preds[i]),
            "riskProba": [float(x) for x in proba[i].tolist()]
        }
        res["suggestion"] = suggest_intervention(res | {
            "sessionMinutes_ma3": r["sessionMinutes_ma3"],
            "assignScore_ma3": r["assignScore_ma3"]
        })
        out.append(res)

    print(json.dumps({"results": out}))

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