ARDUINO CODE:

```
#include <Wire.h>
#include <LiquidCrystal.h>
#include <Adafruit_ADXL345_U.h>

// LCD (RS, E, D4, D5, D6, D7)
LiquidCrystal lcd(33, 32, 14, 27, 26, 25);

// ADXL345
Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified(123);

// Pins
#define FLAME_SENSOR_PIN 13
#define GAS_SENSOR_PIN 34
#define TRIG_PIN 18
#define ECHO_PIN 19
#define FLOW_SENSOR_PIN 23
#define BUZZER_PIN 12
#define LED_WARNING 4
#define LED_CRITICAL 5

// Flow
volatile int flowPulseCount = 0;
unsigned long lastFlowMillis = 0;
float flowRate = 0.0;

void IRAM_ATTR countFlowPulse() {
  flowPulseCount++;
}

void setup() {
  Serial.begin(115200);
  lcd.begin(16, 2);
  lcd.print("Disaster Monitor");

  pinMode(FLAME_SENSOR_PIN, INPUT);
  pinMode(GAS_SENSOR_PIN, INPUT);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  pinMode(FLOW_SENSOR_PIN, INPUT_PULLUP);
```

```
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(LED_WARNING, OUTPUT);
  pinMode(LED_CRITICAL, OUTPUT);

  // Blink test for LEDs
  digitalWrite(LED_WARNING, HIGH);
  digitalWrite(LED_CRITICAL, HIGH);
  delay(1000);
  digitalWrite(LED_WARNING, LOW);
  digitalWrite(LED_CRITICAL, LOW);

  attachInterrupt(digitalPinToInterrupt(FLOW_SENSOR_PIN), countFlowPulse,
RISING);

  if (!accel.begin()) {
    Serial.println("No ADXL345 found");
    while (1);
  }

  accel.setRange(ADXL345_RANGE_2_G);
  delay(2000); // MQ2 warm-up time
}

void loop() {
  bool warning = false;
  bool critical = false;

  // 1. Flame
  bool flameDetected = digitalRead(FLAME_SENSOR_PIN) == LOW;

  // 2. Gas
  int gasLevel = analogRead(GAS_SENSOR_PIN);
  int gasStatus = (gasLevel > 2000) ? 2 : (gasLevel > 1500) ? 1 : 0;

  // 3. Ultrasonic
  digitalWrite(TRIG_PIN, LOW); delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH); delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  long duration = pulseIn(ECHO_PIN, HIGH, 30000);
```

```
  float distance = duration * 0.034 / 2.0;
  float distanceToWater = (distance < 2 || distance > 400) ? -1 :
distance;
  bool floodingDetected = (distanceToWater < 90 && distanceToWater > 0);

  // 4. Flow Rate
  unsigned long currentMillis = millis();
  if (currentMillis - lastFlowMillis >= 1000) {
    flowRate = (flowPulseCount / 7.5); // L/min approx
    flowPulseCount = 0;
    lastFlowMillis = currentMillis;
  }

  if (flowRate > 10.0) {
    critical = true;
  } else if (flowRate > 5.0) {
    warning = true;
  }

  // 5. Earthquake
  sensors_event_t event;
  accel.getEvent(&event);
  float accel_x = event.acceleration.x;
  float accel_y = event.acceleration.y;
  float accel_z = event.acceleration.z;
  float accMag = sqrt(accel_x * accel_x + accel_y * accel_y + accel_z *
accel_z);
  float quakeMagnitude = abs(accMag - 9.8);
  quakeMagnitude = constrain(quakeMagnitude, 0, 5);
  bool earthquakeDetected = quakeMagnitude > 1.5;

  // Set warning and critical flags
  if (flameDetected) critical = true;
  if (gasStatus == 2) critical = true;
  else if (gasStatus == 1) warning = true;

  if (distanceToWater > 0) {
    if (distanceToWater < 30) critical = true;
    else if (distanceToWater < 60) warning = true;
  }
```

```
  if (earthquakeDetected) critical = true;

  // Outputs
  digitalWrite(LED_WARNING, warning ? HIGH : LOW);
  digitalWrite(LED_CRITICAL, critical ? HIGH : LOW);

  if (critical) {
    digitalWrite(BUZZER_PIN, HIGH);
    delay(2000);
    digitalWrite(BUZZER_PIN, LOW);
  } else if (warning) {
    digitalWrite(BUZZER_PIN, HIGH);
    delay(1000);
    digitalWrite(BUZZER_PIN, LOW);
  } else {
    digitalWrite(BUZZER_PIN, LOW);
  }

  // LCD Display
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Gas:"); lcd.print(gasStatus);
  lcd.print(" Fire:"); lcd.print(flameDetected);

  lcd.setCursor(0, 1);
  if (distanceToWater > 0) {
    lcd.print("Dist:"); lcd.print((int)distanceToWater); lcd.print("cm");
  } else {
    lcd.print("Dist:ERR ");
  }
  lcd.setCursor(9, 1);
  lcd.print("F:"); lcd.print((int)flowRate);

  // Serial Monitor Output (human-readable logs)
  Serial.print("Raw Gas Level: "); Serial.println(gasLevel);
  Serial.println(flowRate > 10 ? "Flow Critical." : flowRate > 5 ? "Flow
Warning." : "Flow Normal.");

  Serial.print("Flame: "); Serial.print(flameDetected);
```

```
  Serial.print(", Gas: "); Serial.print(gasLevel);
  Serial.print(", Dist: "); Serial.print(distanceToWater, 2);
Serial.print(" cm");
  Serial.print(", Flow: "); Serial.print(flowRate, 2);
  Serial.print(", Quake: "); Serial.println(quakeMagnitude, 2);

  Serial.print("Warning: "); Serial.print(warning);
  Serial.print(", Critical: "); Serial.println(critical);

  // CSV Serial Output (for logging)

  Serial.print(accel_x); Serial.print(",");
  Serial.print(accel_y); Serial.print(",");
  Serial.print(accel_z); Serial.print(",");
  Serial.print(gasLevel); Serial.print(",");
  Serial.print(gasStatus); Serial.print(",");
  Serial.print(flameDetected); Serial.print(",");
  Serial.print(floodingDetected); Serial.print(",");
  Serial.print(distanceToWater); Serial.print(",");
  Serial.print(earthquakeDetected); Serial.print(",");
  Serial.print(quakeMagnitude); Serial.print(",");
  Serial.println(flowRate);

  delay(1000);
}
```

FLOOD WATER LEVEL PREDICTION:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
```

```
import xgboost as xgb

# Load and clean data
df = pd.read_csv("disaster_data_2.1.csv")
df_cleaned = df[(df["distanceToWater"] != -1) & (df["distanceToWater"] <=
400) & (df["distanceToWater"] >= 2)][["distanceToWater",
"flowRate"]].reset_index(drop=True)

# Feature extraction
window_size = 10
predict_ahead = 26

def build_features(data, target_col, aux_col, window, ahead):
    X, y = [], []
    for i in range(len(data) - window - ahead):
        water = data[target_col].iloc[i:i+window].tolist()
        flow = data[aux_col].iloc[i:i+window].tolist()
        water_diff = [water[j] - water[j-1] for j in range(1, window)]
        flow_diff = [flow[j] - flow[j-1] for j in range(1, window)]
        features = water + flow + water_diff + flow_diff
        label = data[target_col].iloc[i+window+ahead]
        X.append(features)
        y.append(label)
    return np.array(X), np.array(y)

X, y = build_features(df_cleaned, "distanceToWater", "flowRate",
window_size, predict_ahead)

# Random split (shuffle = True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, shuffle=True)

# Scale for Linear Regression
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Models
linreg = LinearRegression()
linreg.fit(X_train_scaled, y_train)
```

```python
y_pred_lin = linreg.predict(X_test_scaled)

rf = RandomForestRegressor(n_estimators=300, max_depth=15,
random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

gbr = GradientBoostingRegressor(n_estimators=300, learning_rate=0.05,
max_depth=5, random_state=42)
gbr.fit(X_train, y_train)
y_pred_gbr = gbr.predict(X_test)

xgbr = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=400,
max_depth=6, learning_rate=0.05, random_state=42)
xgbr.fit(X_train, y_train)
y_pred_xgb = xgbr.predict(X_test)

# Evaluation
def evaluate(y_true, y_pred, name):
    print(f"\n✅ {name}")
    print(f"  MSE: {mean_squared_error(y_true, y_pred):.2f}")
    print(f"  MAE: {mean_absolute_error(y_true, y_pred):.2f}")
    print(f"  R²: {r2_score(y_true, y_pred):.3f}")

evaluate(y_test, y_pred_lin, "Linear Regression")
evaluate(y_test, y_pred_rf, "Random Forest")
evaluate(y_test, y_pred_gbr, "Gradient Boosting")
evaluate(y_test, y_pred_xgb, "XGBoost")

# Predict current 5-min forecast
latest = df_cleaned.tail(window_size).copy()
latest_water = latest["distanceToWater"].tolist()
latest_flow = latest["flowRate"].tolist()
latest_water_diff = [latest_water[j] - latest_water[j-1] for j in range(1,
window_size)]
latest_flow_diff = [latest_flow[j] - latest_flow[j-1] for j in range(1,
window_size)]

latest_features = np.array([latest_water + latest_flow + latest_water_diff
+ latest_flow_diff])
```

```python
latest_scaled = scaler.transform(latest_features)

print(f"\n📡 5-Min Forecast:")
print(f"  Linear Regression: {linreg.predict(latest_scaled)[0]:.2f} cm")
print(f"  Random Forest:     {rf.predict(latest_features)[0]:.2f} cm")
print(f"  Gradient Boosting: {gbr.predict(latest_features)[0]:.2f} cm")
print(f"  XGBoost:           {xgbr.predict(latest_features)[0]:.2f} cm")

# Plotting actual vs predicted for all models on one page
plt.figure(figsize=(12, 10))

plt.subplot(2, 2, 1)
plt.plot(y_test[:100], label="Actual", color='black')
plt.plot(y_pred_xgb[:100], label="XGBoost", color='blue')
plt.title("XGBoost Prediction vs Actual (First 100 Samples)")
plt.xlabel("Sample Index")
plt.ylabel("Water Level")
plt.legend()
plt.grid(True)

plt.subplot(2, 2, 2)
plt.plot(y_test[:100], label="Actual", color='black')
plt.plot(y_pred_lin[:100], label="Linear Regression", color='red')
plt.title("Linear Regression Prediction vs Actual (First 100 Samples)")
plt.xlabel("Sample Index")
plt.ylabel("Water Level")
plt.legend()
plt.grid(True)

plt.subplot(2, 2, 3)
plt.plot(y_test[:100], label="Actual", color='black')
plt.plot(y_pred_rf[:100], label="Random Forest", color='green')
plt.title("Random Forest Prediction vs Actual (First 100 Samples)")
plt.xlabel("Sample Index")
plt.ylabel("Water Level")
plt.legend()
plt.grid(True)

plt.subplot(2, 2, 4)
plt.plot(y_test[:100], label="Actual", color='black')
```

```python
plt.plot(y_pred_gbr[:100], label="Gradient Boosting", color='purple')
plt.title("Gradient Boosting Prediction vs Actual (First 100 Samples)")
plt.xlabel("Sample Index")
plt.ylabel("Water Level")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```

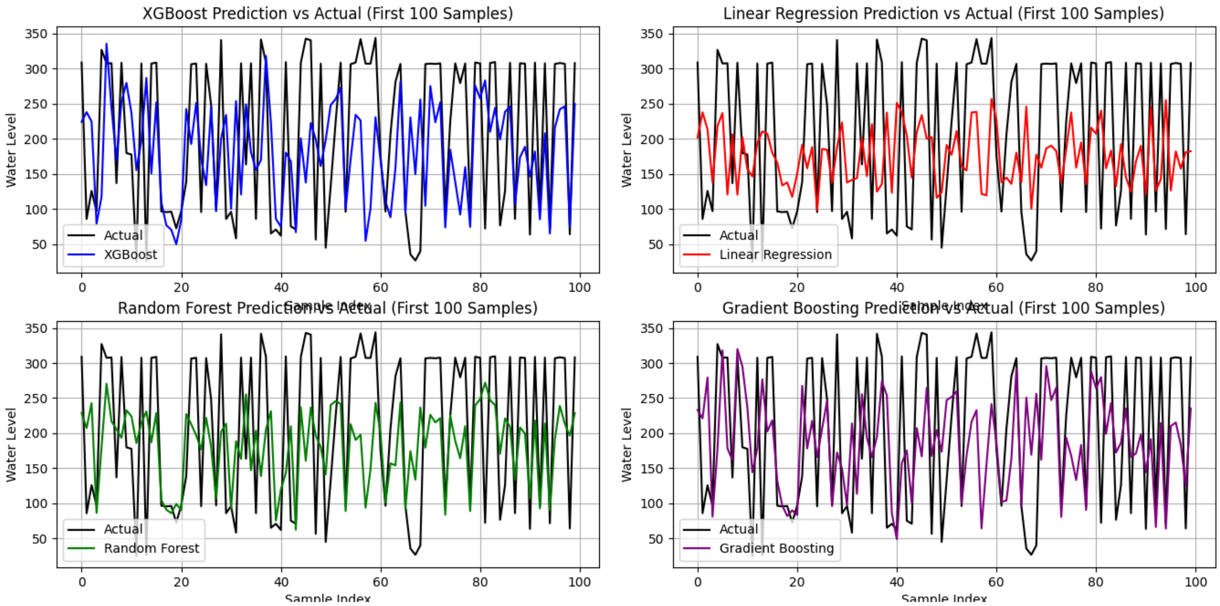
OUTPUT
✅ Linear Regression
  MSE: 12856.31
  MAE: 98.90
  R²: -0.011

✅ Random Forest
  MSE: 9367.90
  MAE: 80.70
  R²: 0.263

✅ Gradient Boosting
  MSE: 10150.70
  MAE: 81.57
  R²: 0.202

✅ XGBoost
  MSE: 11039.17
  MAE: 83.73
  R²: 0.132

📡 5-Min Forecast:
  Linear Regression: 95.70 cm
  Random Forest: 76.62 cm
  Gradient Boosting: 61.58 cm
  XGBoost: 95.58 cm



Disaster classification:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score,
confusion_matrix
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns

# --- LOAD YOUR DATA HERE ---
try:
    df = pd.read_csv('disaster_data_2.1.csv')
except FileNotFoundError:
    print("Error: 'disaster_data_2.1.csv' not found. Please make sure the
file is in the correct directory or provide the correct path.")
    exit()

# Convert timestamp to datetime and set as index
df['timestamp'] = pd.to_datetime(df['timestamp'])
df = df.set_index('timestamp')

# --- FEATURE ENGINEERING ---
# Example: Creating a simple lagged feature for gasLevel
df['gasLevel_lagged'] = df['gasLevel'].shift(1)
```

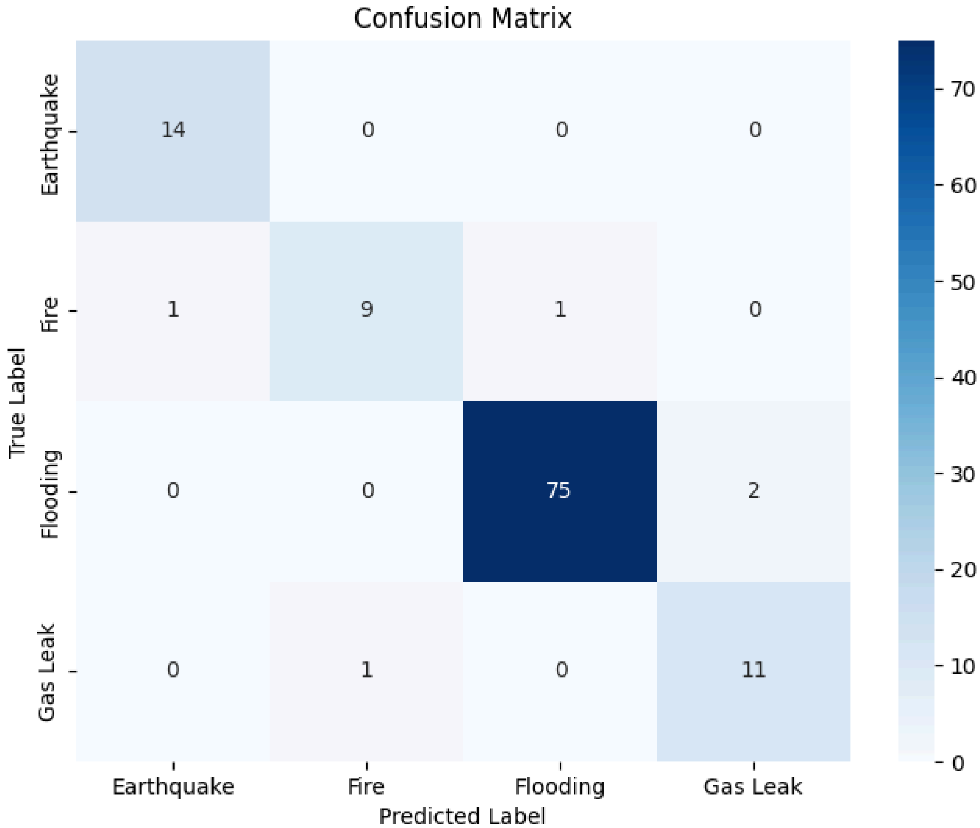
```
df = df.fillna(method='bfill') # Corrected fillna

# Define target variable based on detection flags
def get_disaster_type(row):
    if row['floodingDetected'] == 1:
        return 'Flooding'
    elif row['fireDetected'] == 1:
        return 'Fire'
    elif row['gasLeakDetected'] == 1:
        return 'Gas Leak'
    elif row['earthquakeDetected'] == 1:
        return 'Earthquake'
    else:
        return 'No Disaster'

df['disaster_type'] = df.apply(get_disaster_type, axis=1)

# For simplicity, let's focus on detecting disasters and exclude 'No
Disaster' for now
df_disaster = df[df['disaster_type'] != 'No Disaster'].copy()

if df_disaster.empty:
    print("No disaster events found in the data based on the detection
flags.")
    exit()

# Select features and target for the disaster classification
features = ['accel_x', 'accel_y', 'accel_z', 'gasLevel',
'distanceToWater', 'flowRate', 'earthquakeMagnitude', 'gasLevel_lagged']
target = 'disaster_type'
X = df_disaster[features].fillna(df_disaster[features].mean())
y = df_disaster[target]

# Data preprocessing
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.3, random_state=42, stratify=y)
```

```python
# --- MODEL TRAINING ---
# Train a Random Forest model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# --- MODEL EVALUATION ---
# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# --- CONFUSION MATRIX ---
cm = confusion_matrix(y_test, y_pred)
class_labels = sorted(y.unique()) # Get the unique class labels in order

# Visualize the confusion matrix using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_labels, yticklabels=class_labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

OUTPUT:
Accuracy: 0.956140350877193

Classification Report:

	precision	recall	f1-score	support
Earthquake	0.93	1.00	0.97	14
Fire	0.90	0.82	0.86	11
Flooding	0.99	0.97	0.98	77
Gas Leak	0.85	0.92	0.88	12
accuracy			0.96	114
macro avg	0.92	0.93	0.92	114
weighted avg	0.96	0.96	0.96	114



Confusion Matrix
True Label
Predicted Label
Earthquake
Fire
Flooding
Gas Leak
14
0
0
0
1
9
1
0
0
0
75
2
0
1
0
11
0
10
20
30
40
50
60
70