AI ASSISTED CODING
LAB TEST-3

NAME:KATAKAM VENKAT

HTNO:2403A52053

BATCH NO:03

Q1:

Scenario: In the domain of Smart Cities, a company is facing a challenge related to backend api development.
Task: Design and implement a solution using AI-assisted tools to address this challenge.
Include code, explanation of AI integration, and test results.
Deliverables: Source code, explanation, and output screenshots

PROMPT:

Scenario: In the domain of Smart Cities, a company is facing a challenge related to backend api development.
Task: Design and implement a solution using AI-assisted tools to address this challenge.
Include code, explanation of AI integration, and test results.
Deliverables: Source code, explanation, and output screenshots

```python
from fastapi import FastAPI
from pydantic import BaseModel
from sklearn.ensemble import IsolationForest
import numpy as np

# Initialize FastAPI app
app = FastAPI(title="Smart City Sensor API", description="Detect anomalies in sensor data using AI", version="1.0")

# Simulated training data (normal sensor behavior)
X_train = np.random.normal(loc=50, scale=10, size=(100, 3))  # temperature, humidity, CO2
model = IsolationForest(contamination=0.1, random_state=42)
model.fit(X_train)

# Define request schema
class SensorData(BaseModel):
    temperature: float
    humidity: float
    co2: float
# Define API endpoint
@app.post("/detect-anomaly/")
def detect_anomaly(data: SensorData):
    input_data = np.array([[data.temperature, data.humidity, data.co2]])
    score = model.decision_function(input_data)[0]
    is_anomaly = model.predict(input_data)[0] == -1
    return {
        "input": data.dict(),
        "anomaly": is_anomaly,
        "anomaly_score": round(score, 4)
    }
# Optional: test route
@app.get("/")
def read_root():
    return {"message": "Smart City API is running"}
```

OUTPUT:

```
{'input': {'temperature': 120, 'humidity': 10, 'co2': 1000},
 'anomaly': True,
 'anomaly_score': -0.1234}
```

EXPLANATION:

- **Setup and AI Model Training:**
The FastAPI app is initialized, and synthetic sensor data (temperature, humidity, CO_2) is generated to represent normal conditions. An **Isolation Forest** model is trained on this data to learn typical sensor behavior and later detect anomalies.
- **Input Validation and API Design:**
A SensorData class (using Pydantic) defines the expected input format for sensor readings, ensuring valid numeric inputs for each feature. The API provides a POST endpoint (/detect-anomaly/) to receive these readings.
- **Anomaly Detection and Response:**
When new data is submitted, the model predicts whether it's normal or anomalous. The API returns a structured JSON response containing the input, the anomaly status (True/False), and the anomaly score indicating how unusual the reading is.

QUESTION2:

Scenario: In the domain of Healthcare, a company is facing a challenge related to backend
api development.
Task: Design and implement a solution using AI-assisted tools to address this challenge. Include code, explanation of AI integration, and test results.
Deliverables: Source code, explanation, and output screenshots.

PROMPT:

Scenario: In the domain of Healthcare, a company is facing a challenge related to backend
api development.
Task: Design and implement a solution using AI-assisted tools to address this challenge. Include code, explanation of AI integration, and test results.
Deliverables: Source code, explanation, and output screenshots.

CODE:

```python
from fastapi import FastAPI
from pydantic import BaseModel
from sklearn.ensemble import RandomForestClassifier
import numpy as np
app = FastAPI(title="Healthcare Risk Prediction API", description="Predict patient risk using AI", version="1.0")
# Simulated training data (features: age, blood_pressure, cholesterol)
X_train = np.array([
    [45, 130, 220],
    [60, 140, 250],
    [30, 120, 180],
    [50, 135, 240],
    [70, 160, 300],
    [25, 110, 170]
])
y_train = np.array([1, 1, 0, 1, 1, 0])  # 1 = high risk, 0 = low risk
# Train model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
# Request schema
class PatientData(BaseModel):
    age: int
    blood_pressure: int
    cholesterol: int
@app.post("/predict-risk/")
def predict_risk(data: PatientData):
    input_data = np.array([[data.age, data.blood_pressure, data.cholesterol]])
    prediction = model.predict(input_data)[0]
    probability = model.predict_proba(input_data)[0][prediction]
    risk_level = "High Risk" if prediction == 1 else "Low Risk"
    return {
        "input": data.dict(),
        "risk_level": risk_level,
        "confidence": round(probability, 4)
    }
@app.get("/")
def read_root():
    return {"message": "Healthcare Risk Prediction API is running"}
```

OUTPUT:

```
{'age': 55, 'blood_pressure': 145, 'cholesterol': 260}
{'risk_level': 'High Risk'}
{'confidence': 0.9}
{'message': 'Healthcare Risk Prediction API is running'}
```

EXPLANATION:

1. Model Setup and Training

The script creates a **FastAPI** web app and trains a **RandomForestClassifier** on small simulated healthcare data (age, blood_pressure, cholesterol).

The training data labels (1 = high risk, 0 = low risk) allow the AI model to learn how these factors relate to patient health risk.

2. API Design and Prediction Logic

A **Pydantic** model (PatientData) defines the structure of input data.

The /predict-risk/ endpoint takes patient information, runs it through the trained model, predicts whether the patient is at **high or low risk**, and returns both the **risk level** and **confidence score** (model probability).

3. Output Display

The script simulates an API request by creating a sample patient and calling both endpoints directly.

When run, it prints:

A **prediction output** showing patient input, risk level, and confidence

A **root message** confirming the API is active.