AI ASSISTED CODING

LAB TEST-3

E4

Q1:

Scenario: In the Agriculture sector, a company faces a challenge related to data structures with ai.
Task: Use AI-assisted tools to solve a problem involving data structures with ai in this context.
Deliverables: Submit the source code, explanation of AI assistance used, and sample output

PROBLEM:An agriculture company collects large amounts of **sensor data** (temperature, humidity, soil moisture, rainfall, etc.) from multiple farms.
DATA STRUCTURE USED: LIST
AI ASSISTANCE USED:CHATGPT

SOURCE CODE:

```
# Crop Yield Prediction using AI and List Data Structures
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_absolute_error
# Step 1: Sensor Data (List of Dicts)
sensor_data = [
 {"farm_id":1,"temperature":30,"humidity":60,"soil_moisture":25,"rainfall":100,"yield":3.2},
 {"farm_id":2,"temperature":28,"humidity":70,"soil_moisture":30,"rainfall":120,"yield":3.6},
 {"farm_id":3,"temperature":32,"humidity":55,"soil_moisture":20,"rainfall":80,"yield":2.8},
 {"farm_id":4,"temperature":26,"humidity":75,"soil_moisture":35,"rainfall":150,"yield":4.0},
 {"farm_id":5,"temperature":29,"humidity":65,"soil_moisture":28,"rainfall":110,"yield":3.5},
 {"farm_id":6,"temperature":27,"humidity":68,"soil_moisture":32,"rainfall":140,"yield":3.8},
 {"farm_id":7,"temperature":31,"humidity":58,"soil_moisture":22,"rainfall":90,"yield":3.0},
 {"farm_id":8,"temperature":33,"humidity":50,"soil_moisture":18,"rainfall":70,"yield":2.5},
 {"farm_id":9,"temperature":25,"humidity":80,"soil_moisture":38,"rainfall":160,"yield":4.2},
 {"farm_id":10,"temperature":28,"humidity":72,"soil_moisture":33,"rainfall":130,"yield":3.7}
]
# Step 2: Convert to DataFrame
df = pd.DataFrame(sensor_data)
X = df[["temperature","humidity","soil_moisture","rainfall"]]
y = df["yield"]
# Step 3: Split & Train
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)
model = RandomForestRegressor(n_estimators=100,random_state=42)
model.fit(X_train,y_train)
# Step 4: Predict & Evaluate
pred = model.predict(X_test)
r2 = r2_score(y_test,pred)
mae = mean_absolute_error(y_test,pred)
print("R²:",round(r2,2)," MAE:",round(mae,2))
# Step 5: Show Results
for a,p in zip(y_test.values,pred):
    print("Actual:",round(a,2),"Predicted:",round(p,2))


# Step 6: New Farm Prediction
new = [[29,63,27,115]]
y_pred = model.predict(new)[0]
print("Predicted Yield:",round(y_pred,2),"tons/ha")
```

Output:
R^2: 0.42 MAE: 0.19 Actual: 4.2 Predicted:
3.88 Actual: 3.6 Predicted: 3.66 Predicted
Yield: 3.45 tons/ha

Explanation:
This code predicts crop yield using a RandomForestRegressor model based on sensor data.
Step 1: Sensor Data (List of Dicts): Defines a list of dictionaries called sensor_data, where each dictionary represents data from a farm including temperature, humidity, soil moisture, rainfall, and the actual yield.
Step 2: Convert to DataFrame: Converts the sensor_data list into a pandas DataFrame for easier manipulation. It then separates the features (temperature, humidity, soil moisture, rainfall) into X and the target variable (yield) into y.
Step 3: Split & Train: Splits the data into training and testing sets using train_test_split.
A RandomForestRegressor model is initialized and trained on the training data.
Step 4: Predict & Evaluate: Uses the trained model to make predictions on the test set. It then calculates and prints the R-squared (r2_score) and Mean Absolute Error (mean_absolute_error) to evaluate the model's performance.
Step 5: Show Results: Iterates through the actual and predicted yield values for the test set and prints them side-by-side.
Step 6: New Farm Prediction: Defines a new data point for a hypothetical farm and uses the trained model to predict its yield. The predicted yield is then printed.

Q2:
Scenario: In the Education sector, a company faces a challenge related to backend api development.
Task: Use AI-assisted tools to solve a problem involving backend api development in this context.
Deliverables: Submit the source code, explanation of AI assistance used, and sample output

PROBLEM:An EdTech company offers an online learning platform.
They collect student quiz and course data

AI ASSISTANCE USED:CHATGPT

SOURCE CODE:

```
# File: student_performance_api.py
# Backend API for Student Performance Analysis in Education Sector
from flask import Flask, jsonify, request
app = Flask(__name__)
# Step 1: Mock Database (Structured Data)
students = [
    {"id": 1, "name": "Alice", "scores": {"math": 85, "science": 90, "english": 78}},
    {"id": 2, "name": "Bob", "scores": {"math": 65, "science": 70, "english": 60}},
    {"id": 3, "name": "Charlie", "scores": {"math": 95, "science": 92, "english": 88}},
]
def calculate_average(scores):
    return sum(scores.values()) / len(scores)
def ai_recommendation(avg):
    """Simple AI-assisted logic for recommendations"""
    if avg >= 85:
        return "Excellent! Keep challenging yourself with advanced topics."
    elif avg >= 70:
        return "Good work! Focus on improving weaker subjects."
    else:
        return "Needs improvement. Consider extra practice or tutoring."
@app.route("/students", methods=["GET"])
def get_students():
    """Fetch all students"""
    return jsonify({"students": students})
@app.route("/student/<int:student_id>", methods=["GET"])
def get_student(student_id):
    """Fetch individual student details"""
    student = next((s for s in students if s["id"] == student_id), None)
    if not student:
        return jsonify({"error": "Student not found"}), 404
avg = calculate_average(student["scores"])
    recommendation = ai_recommendation(avg)
  return jsonify({
        "student": student["name"],
        "average_score": round(avg, 2),
        "recommendation": recommendation
    })
@app.route("/add_student", methods=["POST"])
def add_student():
    """Add new student"""
    data = request.get_json()
    if not data or "name" not in data or "scores" not in data:
        return jsonify({"error": "Invalid input"}), 400
   new_id = max(s["id"] for s in students) + 1
    data["id"] = new_id
    students.append(data)
    return jsonify({"message": "Student added successfully", "student": data}), 201


if __name__ == "__main__":
    app.run(debug=True)
```

Output:


```
{
  "name": "Alice",
  "average": 84.33,
  "recommendation": "Good work! Improve weak subjects."
}
```


Explanation:

Initialization and Mock Data Setup

A. Import Flask: The necessary modules (Flask, jsonify, request) are imported to create the web server and handle JSON data/incoming requests.

B. App Creation: An instance of the Flask application (app = Flask(__name__)) is created.

C. Mock Database: The students list is defined. This in-memory list of dictionaries acts as a temporary database for the application.

2. Helper Functions (Business Logic)

A. calc_avg(scores): This function takes a student's score dictionary, calculates the sum of the scores, and divides by the number of subjects to determine the average performance.

B. ai_recommend(avg): This function uses conditional logic (if/elif/else) based on the calculated average (avg) to return a standardized text recommendation (e.g., "Excellent," "Good work," or "Needs improvement").

3. API Route: Read All (GET /students)

A. Define Route: The @app.route decorator maps the URL path /students to the get_all function, specifying it only responds to GET requests.

B. Response: The function uses jsonify({"students": students}) to convert the entire Python students list into a standard JSON array and sends it back to the client.

4. API Route: Read One & Analyze (GET /student/<int:sid>)

A. Define Route with Parameter: The route /student/<int:sid> is defined. The <int:sid> captures a student ID number from the URL.

B. Find Data: It uses a list comprehension (next()) to search the students list for a matching ID.

C. Handle Errors: If no student is found, it immediately returns a JSON error and the **404 Not Found** status code.

D. Analyze & Respond: If found, it calls the helper functions (calc_avg and ai_recommend) to process the scores and returns the results—name, rounded average, and recommendation—as a JSON object.

5. API Route: Create New Student (POST /add_student)

A. Define POST Route: The route /add_student is set up to only accept POST requests, expecting data in the body.

B. Get Data: It retrieves the JSON data sent by the user using request.get_json().

C. Validate Input: It checks if the required name and scores fields are present. If not, it returns a **400 Bad Request** error.

D. Assign ID and Append: It calculates a new unique ID (max existing ID + 1) and appends the new record to the global students list.

E. Confirmation: It returns a confirmation message and the full new student object with the **201 Created** status code.