# AI ASSISTED CODING

# LAB TEST – 03

**Name: Likhitha Pothunuri**

**Roll no: 2403A510D1**

**Batch 05**

**CSE 2nd year**

**Set E4:**

**Q1:**

**Prompt:**

"In the education sector, a company is facing a challenge related to code refactoring.
Use AI-assisted tools to improve and refactor an existing piece of Python code used for managing student or course data.
The refactored code should be cleaner, more efficient, and easier to maintain.
Include:

- The original (unoptimized) code
- The refactored code generated or improved with AI assistance
- Explanation of how AI helped in refactoring
- Sample output showing the working of the refactored program."

**Code Generated:**

your_file.py > ...

```python
from dataclasses import dataclass, asdict
from flask import Flask, request, jsonify, abort
from itertools import count
from typing import Dict, Any, Optional

# Minimal Student model
@dataclass
class Student:
    id: int
    name: str
    age: int
    major: str

    def to_dict(self) -> Dict[str, Any]:
        return asdict(self)

# Very small store with in-memory dict
class Store:
    def __init__(self):
        self._data: Dict[int, Student] = {}
        self._ids = count[int](1)

    def create(self, payload: Dict[str, Any]) -> Student:
        name = str(payload.get("name", "")).strip()
        major = str(payload.get("major", "")).strip()
        try:
            age = int(payload.get("ag
        except (TypeError, ValueErro
            age = -1
```

Review next file >

```
        except (TypeError, ValueError):
            age = -1
        if not name or not major or age < 0:
            raise ValueError("invalid payload")
        sid = next(self._ids)
        student = Student(id=sid, name=name, age=age, major=major)
        self._data[sid] = student
        return student

    def list(self):
        return list[Student](self._data.values())

    def get(self, sid: int) -> Optional[Student]:
        return self._data.get(sid)

    def update(self, sid: int, payload: Dict[str, Any]) -> Student:
        s = self.get(sid)
        if not s:
            raise KeyError("not found")
        if "name" in payload:
            name = str(payload["name"]).strip()
            if not name: raise ValueError("invalid name")
            s.name = name
        if "age" in payload:
            try:
                age = int(payload["a
                if age < 0: raise Va
                s.age = age
```

Review next file >

```
            except (TypeError, ValueError):
                raise ValueError("invalid age")
        if "major" in payload:
            major = str(payload["major"]).strip()
            if not major: raise ValueError("invalid major")
            s.major = major
        return s

    def delete(self, sid: int) -> Student:
        s = self._data.pop(sid, None)
        if not s:
            raise KeyError("not found")
        return s

def create_app() -> Flask:
    app = Flask(__name__)
    app.config["JSON_SORT_KEYS"] = False
    store = Store()

    @app.get("/students")
    def list_students():
        return jsonify([s.to_dict() for s in store.list()]), 200

    @app.post("/students")
```

```python
@app.post("/students")
def create_student():
    payload = request.get_json(silent=True) or {}
    try:
        s = store.create(payload)
    except ValueError:
        abort(400, "invalid or missing fields")
    return jsonify(s.to_dict()), 201

@app.put("/students/<int:sid>")
def update_student(sid: int):
    if not store.get(sid):
        abort(404, "student not found")
    payload = request.get_json(silent=True) or {}
    try:
        s = store.update(sid, payload)
    except ValueError:
        abort(400, "invalid fields")
    return jsonify(s.to_dict()), 200

@app.delete("/students/<int:sid>")
def delete_student(sid: int):
    try:
        s = store.delete(sid)
    except KeyError:
        abort(404, "student not
    return jsonify({"deleted": s.to_dict()}), 200
```

```
        return jsonify({"deleted": s.to_dict()}), 200

    # quick demo without running a server
    @app.get("/demo")
    def demo():
        store._data.clear()
        s1 = store.create({"name": "Alice", "age": 21, "major": "CS"})
        s2 = store.create({"name": "Bob", "age": 22, "major": "Math"})
        store.update(s1.id, {"major": "Data Science"})
        store.delete(s2.id)
        return jsonify([s.to_dict() for s in store.list()])

    return app

if __name__ == "__main__":
    # Run a tiny demo using Flask test client (no server)
    app = create_app()
    c = app.test_client()

    print("Create:")
    r = c.post("/students", json={"name": "Alice", "age": 21, "major": "CS"})
    print(r.status_code, r.get_json())

    print("List:")
    r = c.get("/students")
    print(r.status_code, r.get_json(
```

Review next file >

```

    print("Update:")
    r = c.put("/students/1", json={"major": "Data Science"})
    print(r.status_code, r.get_json())

    print("Delete:")
    r = c.delete("/students/1")
    print(r.status_code, r.get_json())

    print("Final list:")
    r = c.get("/students")
    print(r.status_code, r.get_json())
```

**Output:**

```
200 []
PS C:\Users\likhi\OneDrive\Desktop\Ai_LT3> python your_file.py
Create:
201 {'age': 21, 'id': 1, 'major': 'CS', 'name': 'Alice'}
List:
200 [{'age': 21, 'id': 1, 'major': 'CS', 'name': 'Alice'}]
Update:
200 {'age': 21, 'id': 1, 'major': 'Data Science', 'name': 'Alice'}
Delete:
200 {'deleted': {'age': 21, 'id': 1, 'major': 'Data Science', 'name': 'Alice'}}
Final list:
200 []
PS C:\Users\likhi\OneDrive\Desktop\Ai_LT3>
```

**Observation:**

The program successfully demonstrates the process of **code refactoring** using AI assistance in the education sector.
The original code for managing student data was improved into a cleaner and more efficient version.
All operations such as **Create, Read, Update, and Delete** were executed correctly, showing proper handling of student records.
The final output confirms that the refactored code is easier to understand, maintain, and produces accurate results.

**Q2:**

**Prompt:**

“In the agriculture sector, a company is facing a challenge related to algorithms.
Use AI-assisted tools to create a Python program that applies suitable algorithms (such as sorting, searching, or prediction) along with AI or machine learning to solve an agricultural problem.
The program should demonstrate how AI assistance helps in selecting or optimizing the algorithm for tasks like crop yield prediction, soil data analysis, or weather-based decision-making.
Include explanation, AI assistance used, and sample output.”

**Code Generated:**

agricultural_ai_fixed.py > ...

```python
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error

def generate_crop_data(n_samples=200):
    """Generate agricultural data"""
    np.random.seed(42)
    data = {
        'soil_ph': np.random.normal(6.5, 1.0, n_samples),
        'soil_moisture': np.random.normal(0.4, 0.1, n_samples),
        'nitrogen': np.random.normal(150, 30, n_samples),
        'temperature': np.random.normal(25, 5, n_samples),
        'rainfall': np.random.normal(800, 200, n_samples),
        'crop_type': np.random.choice(['Wheat', 'Corn'], n_samples)
    }
    df = pd.DataFrame(data)
    df['yield'] = (
        df['soil_moisture'] * 1000 +
        df['nitrogen'] * 2 +
        df['rainfall'] * 0.5 +
        np.where(df['crop_type'] == 'Wheat', 200, 150) +
        np.random.normal(0, 50, n_samples)
    )

    df['yield'] = np.maximum(df['yie
    return df
```

Review next file >

```python
def preprocess_data(df):
    """Preprocess data for ML"""
    df['crop_wheat'] = (df['crop_type'] == 'Wheat').astype(int)
    df['crop_corn'] = (df['crop_type'] == 'Corn').astype(int)

    features = ['soil_ph', 'soil_moisture', 'nitrogen', 'temperature', 'rainfall', 'crop_wheat',
    X = df[features]
    y = df['yield']

    return X, y
def train_models(X, y):
    """Train and compare models"""
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    rf_model = RandomForestRegressor(n_estimators=50, random_state=42)
    lr_model = LinearRegression()

    rf_model.fit(X_train, y_train)
    lr_model.fit(X_train, y_train)

    rf_pred = rf_model.predict(X_test)
    lr_pred = lr_model.predict(X_test)

    rf_r2 = r2_score(y_test, rf_pred)
    lr_r2 = r2_score(y_test, lr_pred)
    rf_mse = mean_squared_error(y_test, rf_pred)
    lr_mse = mean_squared_error(y_te
                                Review next file >

    return {
```

```python
    return {
        'Random Forest': {'model': rf_model, 'r2': rf_r2, 'mse': rf_mse},
        'Linear Regression': {'model': lr_model, 'r2': lr_r2, 'mse': lr_mse},
        'y_test': y_test
    }
def analyze_results(df, results):
    """Analyze results and find best model"""
    print("\nCROP PERFORMANCE:")
    crop_stats = df.groupby('crop_type')['yield'].agg(['mean', 'std'])
    print(crop_stats)

    high_yield = df[df['yield'] > df['yield'].quantile(0.8)]
    print(f"\nOptimal Conditions:")
    print(f"pH: {high_yield['soil_ph'].mean():.2f}")
    print(f"Moisture: {high_yield['soil_moisture'].mean():.2f}")
    print(f"Nitrogen: {high_yield['nitrogen'].mean():.1f}")
    print(f"Temperature: {high_yield['temperature'].mean():.1f}C")

    print(f"\nMODEL PERFORMANCE:")
    for name, result in results.items():
        if name != 'y_test':
            print(f"{name}: R2={result['r2']:.4f}, MSE={result['mse']:.2f}")

    best_model = max([k for k in results.keys() if k != 'y_test'],
                     key=lambda x: results[x]['r2'])
    print(f"\nBest Model: {best_mode
                         Review next file >
```

```python
    return best_model


def generate_recommendations(best_model_name):
    """Generate AI recommendations"""
    return [
        "Maintain soil pH between 6.0-7.0",
        "Keep soil moisture around 0.4",
        "Ensure nitrogen levels 150+ ppm",
        "Plant when temperature is 20-30C",
        f"Use {best_model_name} for predictions"
    ]

Ctrl+L to chat, Ctrl+K to generate
def main():
    """Main demonstration"""
    print("=" * 60)
    print("AI-ASSISTED AGRICULTURAL CROP YIELD PREDICTION")
    print("=" * 60)

    print("\n1. GENERATING DATA")
    df = generate_crop_data(200)
    print(f"Generated {len(df)} records")
    print(f"Features: {list[Any](df.columns)}")
    print(f"\nSample data:")
    print(df.head())
    print("\n2. PREPROCESSING")
```

Review next file >

```
    X, y = preprocess_data(df)
    print(f"Features shape: {X.shape}")
    print(f"Yield range: {y.min():.1f} - {y.max():.1f}")


    print("\n3. AI MODEL TRAINING")
    results = train_models(X, y)

    print("\n4. ANALYSIS")
    best_model = analyze_results(df, results)

    print("\n5. AI RECOMMENDATIONS")
    recommendations = generate_recommendations(best_model)
    for i, rec in enumerate[str](recommendations, 1):
        print(f"{i}. {rec}")

    print("\n6. SAMPLE PREDICTIONS")
    best_model_obj = results[best_model]['model']
    sample_data = X.head(3)
    predictions = best_model_obj.predict(sample_data)

    print("Predictions:")
    for i, (idx, pred) in enumerate[tuple[int, Any]](zip[tuple[int, Any]](sample_data.index,
        actual = y.iloc[idx]
        error = abs(pred - actual) / actual * 100
        print(f"Sample {i+1}: Pred={
                                    al={actual:.1f}, Error={error:.1f}%")

    print("\n7. AI ASSISTANCE SUMMARY")
```

Review next file >

```
    print("\n7. AI ASSISTANCE SUMMARY")
    print("Algorithm Selection: Compared Random Forest vs Linear Regression")
    print("Feature Engineering: Categorical to numerical conversion")
    print("Performance Optimization: Best model selection by R2 score")
    print("Predictive Analytics: Generated yield predictions")
    print("Decision Support: Provided farming recommendations")

    print("\n" + "=" * 60)
    print("CONCLUSION: AI-assisted system successfully predicts crop yields")
    print("and provides actionable recommendations for farmers.")
    print("=" * 60)
if __name__ == "__main__":
    main()
```

**Output:**

```
Problems  Output  Debug Console  Terminal  Ports        Python Debug Console

============================================================
AI-ASSISTED AGRICULTURAL CROP YIELD PREDICTION
============================================================

1. GENERATING DATA
Generated 200 records
Features: ['soil_ph', 'soil_moisture', 'nitrogen', 'temperature', 'rainfall', 'crop_type', 'yield']

Sample data:
    soil_ph  soil_moisture    nitrogen  ...    rainfall  crop_type        yield
0  6.996714       0.435779  102.167170  ...  987.656761      Wheat  1256.112362
1  6.361736       0.456078  132.018749  ...  696.791054       Corn  1197.105720
2  7.147689       0.508305  150.157311  ...  819.224155      Wheat  1493.269814
3  8.023030       0.505380  151.409418  ...  707.544942       Corn  1354.482599
4  6.265847       0.262233  136.498036  ...  713.100755      Wheat  1074.346905

[5 rows x 7 columns]

2. PREPROCESSING
Features shape: (200, 7)
Yield range: 866.5 - 1648.2

3. AI MODEL TRAINING

4. ANALYSIS

CROP PERFORMANCE:
                mean         std
crop_type
Corn       1269.931101  137.937751
Wheat      1320.931876  158.683569

Optimal Conditions:
pH: 6.30
Moisture: 0.49
Nitrogen: 162.3
Temperature: 25.6C

MODEL PERFORMANCE:
Random Forest: R2=0.7909, MSE=5343.92
Linear Regression: R2=0.9216, MSE=2005.32

Best Model: Linear Regression

5. AI RECOMMENDATIONS
1. Maintain soil pH between 6.0-7.0
2. Keep soil moisture around 0.4
3. Ensure nitrogen levels 150+ ppm
4. Plant when temperature is 20-30C
5. Use Linear Regression for predictions

6. SAMPLE PREDICTIONS
Predictions:
Sample 1: Pred=1328.6, Actual=1256.1, Error=5.8%
Sample 2: Pred=1225.0, Actual=1197.1, Error=2.3%
Sample 3: Pred=1417.1, Actual=1493.3, Error=5.1%

7. AI ASSISTANCE SUMMARY
Algorithm Selection: Compared Random Forest vs Linear Regression
Feature Engineering: Categorical to numerical conversion
Performance Optimization: Best model selection by R2 score
Predictive Analytics: Generated yield predictions
    `sion Support: Provided farming recommendations
```

```
Decision Support: Provided farming recommendations

============================================================
CONCLUSION: AI-assisted system successfully predicts crop yields
Feature Engineering: Categorical to numerical conversion
Performance Optimization: Best model selection by R2 score
Predictive Analytics: Generated yield predictions
Decision Support: Provided farming recommendations
```

**Observation:**

The program successfully demonstrates the use of **algorithms with AI assistance** to solve an agricultural problem.
AI-assisted tools helped in designing and optimizing the algorithm for better accuracy and efficiency.
The implemented algorithm correctly processed agricultural data and provided meaningful predictions or analysis results.
The output confirms that the algorithm works effectively and the AI-assisted improvements make the solution more reliable and optimized for real-world use.