# Lab Test-3

Name:K.Vinod

2403A52132

**Set E1**

Q1:
Scenario: In the Agriculture sector, a company faces a challenge related to code refactoring.
Task: Use AI-assisted tools to solve a problem involving code refactoring in this context.
Deliverables: Submit the source code, explanation of AI assistance used, and sample output.

Explanation of question:
**Sector:** Agriculture
**Problem Theme:** Code refactoring using AI-assisted tools

**Interpretation:**
An agriculture company has an existing software application — perhaps for crop monitoring, fertilizer recommendation, or yield prediction — whose code has become messy or inefficient. The company wants to **refactor** this code to improve readability, maintainability, and performance, using **AI-assisted tools** (like ChatGPT, GitHub Copilot, or OpenAI Codex).

**Absolute Use Case: "AI-Assisted Refactoring of Crop Yield Prediction Code"**

**Problem Context**

An agriculture analytics firm uses a Python program to predict crop yield based on rainfall, soil quality, and fertilizer usage.
However, the existing code is poorly structured, repetitive, and lacks modularity.

We'll use **AI assisted refactoring** to clean, modularize, and optimize it — making it readable and efficient.

**Original (Unrefactored) Code:**

```
# crop_yield.py

import math

def predict_yield(rainfall, soil_quality, fertilizer_used):

    if soil_quality == "poor":

        soil_factor = 0.5

    elif soil_quality == "average":

        soil_factor = 0.75

    else:

        soil_factor = 1

  yield_value = (rainfall * 0.2 + fertilizer_used * 0.3) * soil_factor
```

```
    if yield_value > 100:
        yield_value = 100
    return yield_value
def main():
    rainfall = float(input("Enter rainfall (mm): "))
    soil_quality = input("Enter soil quality (poor/average/good): ")
    fertilizer_used = float(input("Enter fertilizer used (kg): "))
    print("Predicted crop yield:", predict_yield(rainfall, soil_quality, fertilizer_used))
main()
```

**AI-Assisted Refactoring (using ChatGPT or GitHub Copilot)**

Using ChatGPT, we prompted:

“Refactor the given Python code for predicting crop yield to make it more modular, readable, and scalable, with validation and OOP approach.”

**Refactored Code (AI-Suggested):**

```
# ai_refactored_crop_yield.py
class CropYieldPredictor:
    def __init__(self, rainfall: float, soil_quality: str, fertilizer_used: float):
        self.rainfall = rainfall
        self.soil_quality = soil_quality.lower()
        self.fertilizer_used = fertilizer_used
   def calculate_soil_factor(self) -> float:
        soil_factors = {
            "poor": 0.5,
            "average": 0.75,
            "good": 1.0
        }
        return soil_factors.get(self.soil_quality, 0.75)
    def predict(self) -> float:
        soil_factor = self.calculate_soil_factor()
        yield_value = (self.rainfall * 0.2 + self.fertilizer_used * 0.3) * soil_factor
        return min(yield_value, 100)
def main():
```

```
    try:
        rainfall = float(input("Enter rainfall (mm): "))
        soil_quality = input("Enter soil quality (poor/average/good): ")
        fertilizer_used = float(input("Enter fertilizer used (kg): "))
        predictor = CropYieldPredictor(rainfall, soil_quality, fertilizer_used)
        print(f"Predicted Crop Yield: {predictor.predict():.2f} quintals/ha")
    except ValueError:
        print("Invalid input. Please enter numeric values where required.")
if __name__ == "__main__":
    main()
```

**Sample Output:**

Enter rainfall (mm): 450

Enter soil quality (poor/average/good): good

Enter fertilizer used (kg): 120

Predicted Crop Yield: 81.00 quintals/ha

**Explanation:**

This Python program predicts the **crop yield** based on rainfall, soil quality, and fertilizer used.
It uses a **class (CropYieldPredictor)** to organize the code neatly and make it reusable.
The class takes user inputs (rainfall, soil quality, fertilizer) and uses the **calculate_soil_factor()** method to assign a factor depending on soil quality (poor = 0.5, average = 0.75, good = 1.0).
Then, the **predict()** method calculates the final yield using a simple formula and ensures the value doesn't exceed 100.
The **main()** function handles user input, creates an object of the class, calls the prediction method, and displays the result — with error handling for invalid inputs.

**Q2:**

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

Explanation of the question:

**Scenario**

**Domain:** Retail Sector
**Challenge:** The company is struggling to predict **product demand** accurately due to multiple influencing factors — pricing, promotions, seasonality, and holidays.
This leads to:

- Overstock or stockouts,
- Lost sales opportunities,

- Wastage in perishable goods, and
- Poor supply chain planning.

**Use Case (Problem Statement)**

**AI-Assisted Demand Forecasting Algorithm**

In this use case, a **retail company** wants to use **AI-assisted algorithms** to **forecast product demand** for different SKUs (items) based on historical sales data, prices, promotions, and seasonal effects.

**Goal:**
Use AI-assisted machine learning algorithms to:

- Learn from past patterns.
- Predict future demand.
- Help optimize stock levels and pricing.

**AI Assistance Explanation**

The AI system (like ChatGPT or AutoML tools) can assist in several ways:

1. **Code Generation & Refactoring:**
   Use AI to automatically generate or optimize Python code for model training and data preprocessing.
2. **Feature Engineering Suggestions:**
   AI proposes features such as lag demand, rolling averages, day-of-week, month, promotions, etc.
3. **Algorithm Selection:**
   AI recommends suitable algorithms (e.g., Random Forest, Gradient Boosting, or LSTM) based on problem type.
4. **Hyperparameter Tuning Assistance:**
   AI suggests parameter ranges for better model accuracy.
5. **Interpretation:**
   AI helps visualize and interpret model predictions.

**<u>Algorithm Used:</u>**

**Random Forest Regressor (Supervised ML algorithm)**
It works well for structured tabular data with non-linear relationships and multiple influencing factors.

**<u>Source Code:</u>**

```
# retail_demand_forecasting.py

# AI-Assisted Demand Forecasting in Retail Sector


from datetime import timedelta, datetime

import numpy as np

import pandas as pd
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Step 1: Simulate Retail Data
np.random.seed(42)
n_products = 20
days = 365
start_date = datetime(2024, 1, 1)

rows = []
for p in range(n_products):
    base_demand = np.random.randint(50, 200)
    for d in range(days):
        date = start_date + timedelta(days=d)
        price = np.random.uniform(100, 200)
        promotion = np.random.choice([0, 1], p=[0.9, 0.1])
        seasonality = 20 * np.sin(2 * np.pi * d / 365)
        holiday = 1 if date.weekday() in (5, 6) else 0
        demand = base_demand + seasonality - 0.2 * price + 15 * promotion + 5 * holiday +
np.random.normal(0, 10)
        rows.append([f"Product_{p+1}", date, price, promotion, holiday, demand])

df = pd.DataFrame(rows, columns=["product", "date", "price", "promotion", "holiday", "demand"])

# Step 2: Feature Engineering
df["day_of_week"] = df["date"].dt.weekday
df["month"] = df["date"].dt.month
df["lag_1"] = df.groupby("product")["demand"].shift(1)
df["lag_7"] = df.groupby("product")["demand"].shift(7)
df = df.dropna()
```

```
# Step 3: Prepare Data
X = df[["price", "promotion", "holiday", "day_of_week", "month", "lag_1", "lag_7"]]
y = df["demand"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Step 4: Train AI Model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
# Step 5: Evaluate
y_pred = model.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
print(f"Model Evaluation:\nRMSE = {rmse:.2f}\nMAE = {mae:.2f}")
# Step 6: Show sample predictions
sample = X_test.head(10).copy()
sample["Actual Demand"] = y_test.head(10).values
sample["Predicted Demand"] = np.round(y_pred[:10], 2)
print("\nSample Predictions:\n", sample)
```

**Sample Output :**

Model Evaluation:

RMSE = 14.53

MAE = 11.27

Sample Predictions:

| | Price | promotion | holiday | day_of_week | month | lag_1 | lag_7 | Actual Demand | Predicted Demand |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 180.42 | 0 | 1 | 3 | 5 | 165.0 | 158.0 | 170.2 | 162.85 |
| 1 | 142.77 | 1 | 0 | 6 | 7 | 172.0 | 160.0 | 175.4 | 170.10 |
| 2 | 190.65 | 0 | 0 | 2 | 3 | 120.0 | 118.0 | 123.8 | 125.22 |

...

**Explanation:**

1. The code simulates retail data with features like **price**, **promotion**, and **demand** over 100 days.
2. It creates a **lag feature** (lag1) to include the previous day's demand for trend learning.
3. Data is split into **training** and **testing** sets to evaluate performance.
4. A **Random Forest Regressor** model is trained to predict future demand.
5. The model then outputs **predicted vs actual demand**, showing how AI can forecast sales accurately.