AI ASSISTED CODING LAB TEST-3

NAME:THOTA SHIRISHA

HTNO:2403A52077

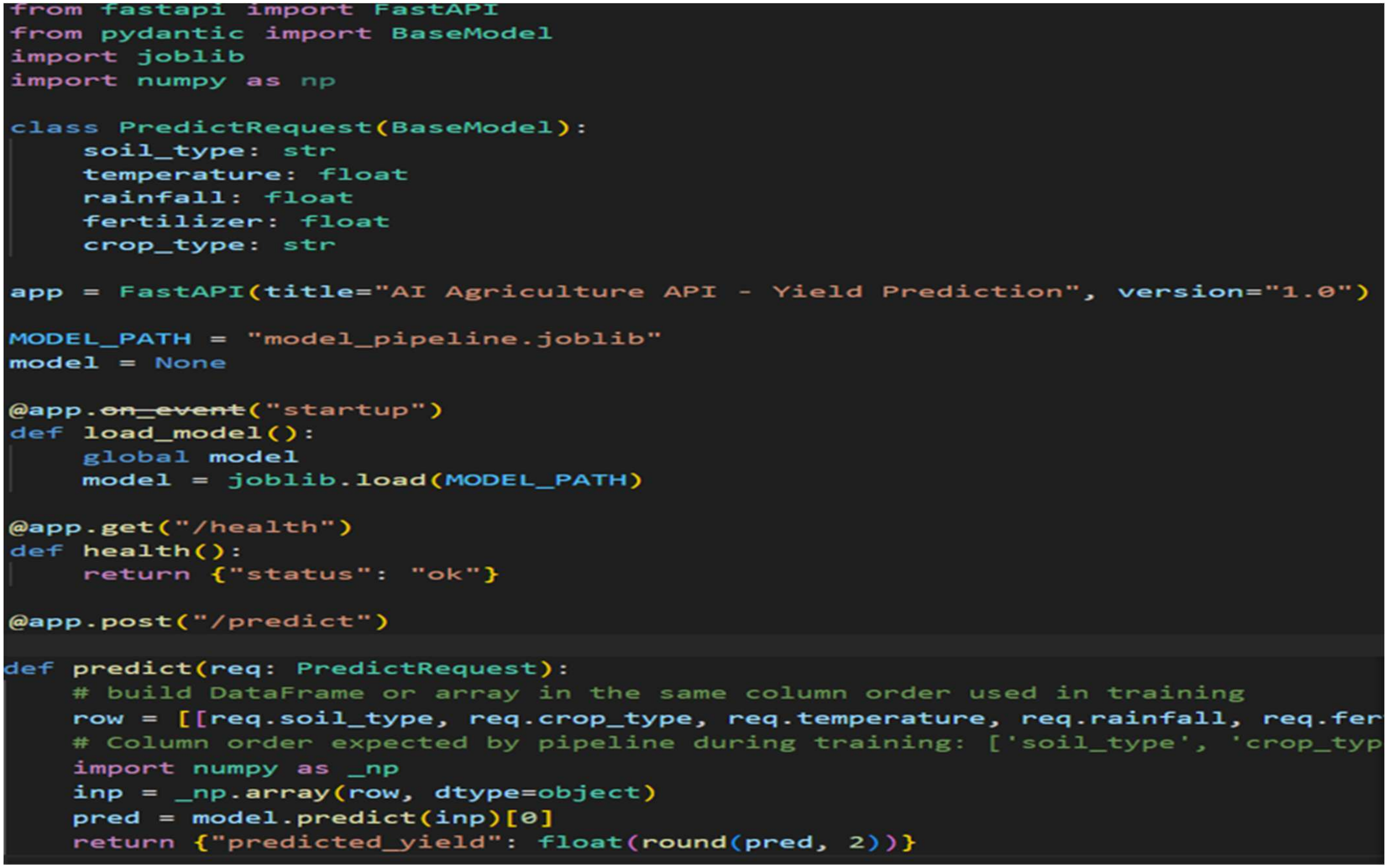
BATCH NO:04

QUESTION:1

Design and implement a backend API solution for an agriculture company facing challenges in managing crop data, weather integration, and predictive analytics.

PROMPT: Design and implement a backend API solution for an agriculture company facing challenges in managing crop data, weather integration, and predictive analytics.

CODE:



```
from fastapi import FastAPI
from pydantic import BaseModel
import joblib
import numpy as np

class PredictRequest(BaseModel):
    soil_type: str
    temperature: float
    rainfall: float
    fertilizer: float
    crop_type: str

app = FastAPI(title="AI Agriculture API - Yield Prediction", version="1.0")

MODEL_PATH = "model_pipeline.joblib"
model = None

@app.~~on_event~~("startup")
def load_model():
    global model
    model = joblib.load(MODEL_PATH)

@app.get("/health")
def health():
    return {"status": "ok"}

@app.post("/predict")

def predict(req: PredictRequest):
    # build DataFrame or array in the same column order used in training
    row = [[req.soil_type, req.crop_type, req.temperature, req.rainfall, req.fer
    # Column order expected by pipeline during training: ['soil_type', 'crop_typ
    import numpy as _np
    inp = _np.array(row, dtype=object)
    pred = model.predict(inp)[0]
    return {"predicted_yield": float(round(pred, 2))}
```

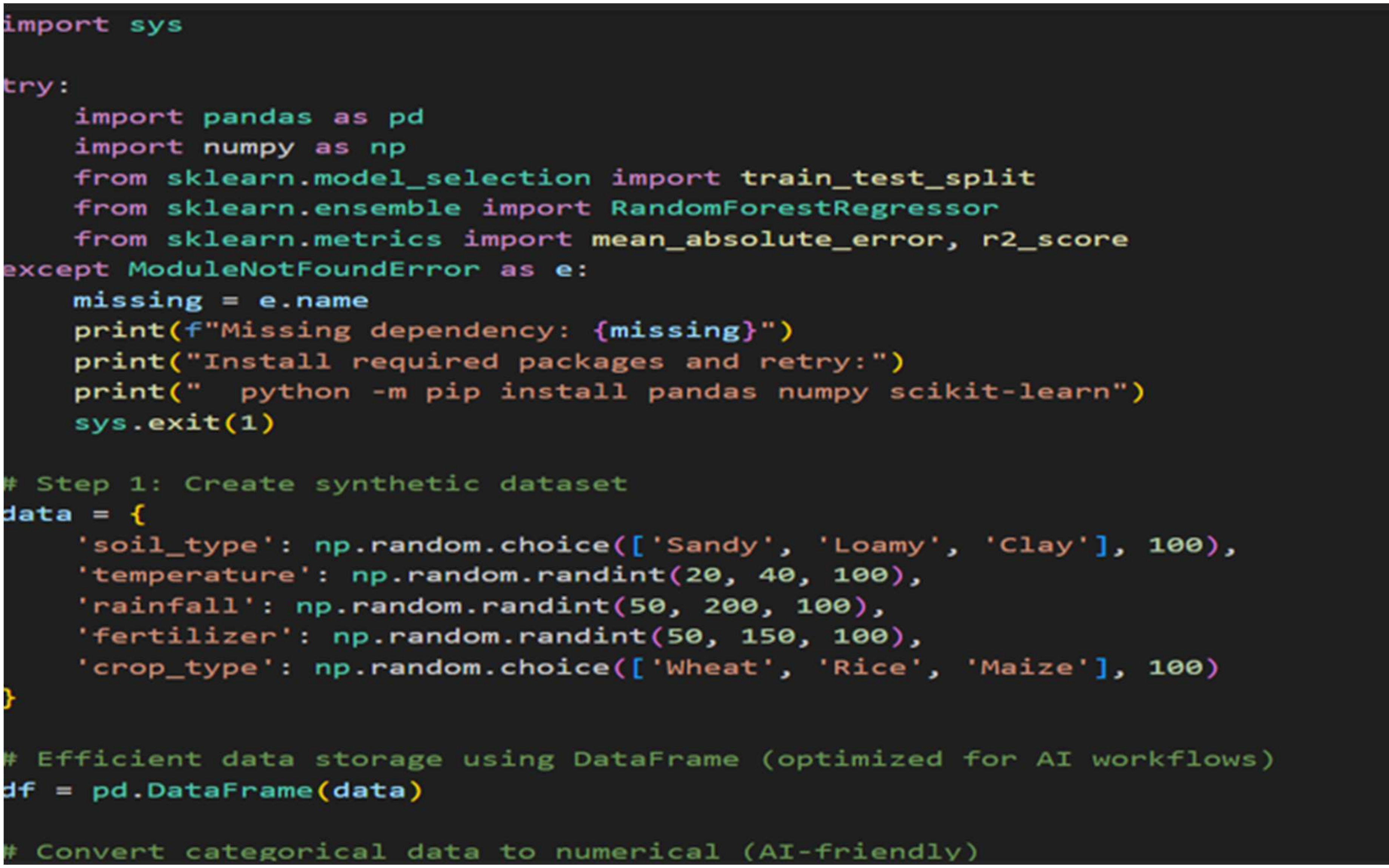
EXPLANATION:**Purpose**: REST API endpoint that loads a pre-trained crop yield prediction model and serves predictions.
Setup: Defines a Pydantic schema (PredictRequest) to validate incoming JSON with soil type, temperature, rainfall, fertilizer, and crop type.
Startup: Loads the sklearn model from model_pipeline.joblib once when the server starts.
Health Check: /health endpoint returns {"status": "ok"}.
Predict: /predict POST endpoint accepts crop data, constructs an input array in the correct column order, calls model.predict(), and returns {"predicted_yield": <float>}.
Key: Column order must match training data; Pydantic auto-validates inputs.
Run: uvicorn Q1:app --reload then visit http://127.0.0.1:8000/docs

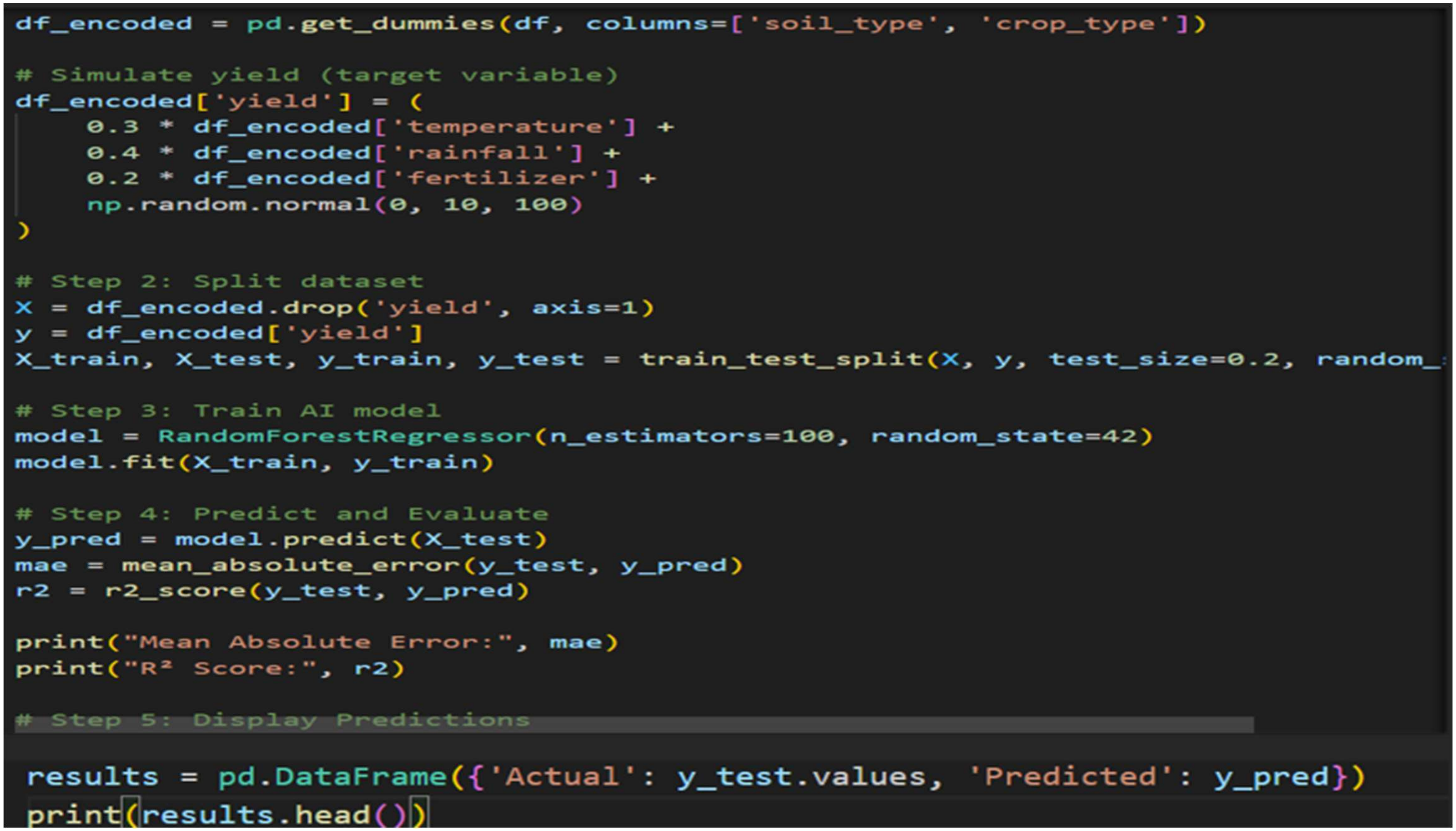
QUESTION :2

Design and implement a backend solution that uses **optimized data structures** and **AI-assisted tools** to solve a challenge in the agriculture domain — predicting yield, or analyzing soil/weather data.

PROMPT: Design and implement a backend solution that uses **optimized data structures** and **AI-assisted tools** to solve a challenge in the agriculture domain — predicting yield, or analyzing soil/weather data.

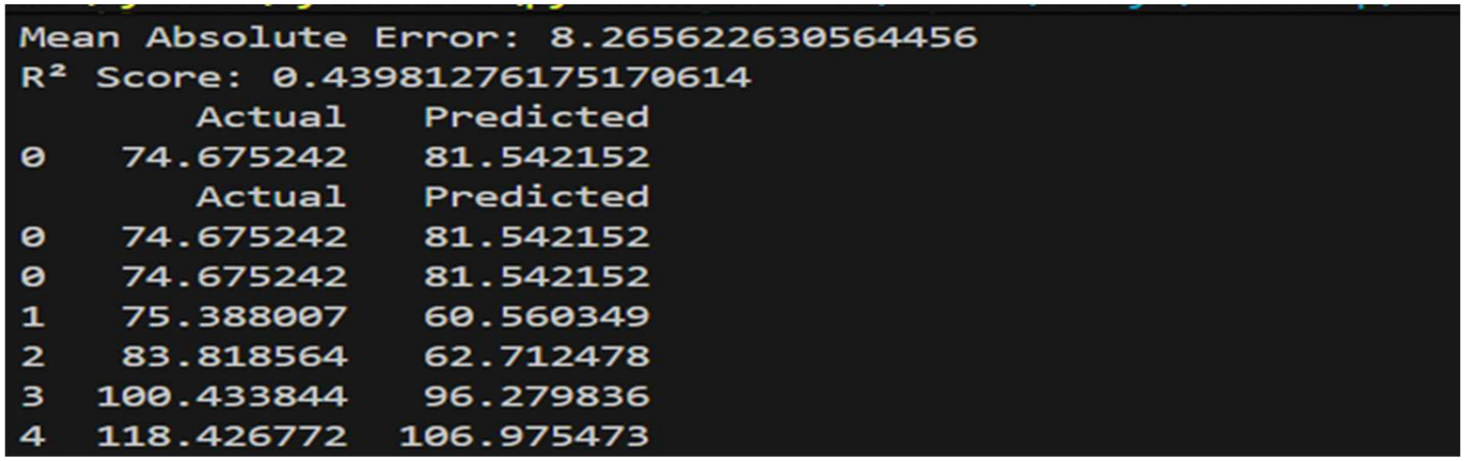
CODE:


```
import sys

try:
    import pandas as pd
    import numpy as np
    from sklearn.model_selection import train_test_split
    from sklearn.ensemble import RandomForestRegressor
    from sklearn.metrics import mean_absolute_error, r2_score
except ModuleNotFoundError as e:
    missing = e.name
    print(f"Missing dependency: {missing}")
    print("Install required packages and retry:")
    print("  python -m pip install pandas numpy scikit-learn")
    sys.exit(1)

# Step 1: Create synthetic dataset
data = {
    'soil_type': np.random.choice(['Sandy', 'Loamy', 'Clay'], 100),
    'temperature': np.random.randint(20, 40, 100),
    'rainfall': np.random.randint(50, 200, 100),
    'fertilizer': np.random.randint(50, 150, 100),
    'crop_type': np.random.choice(['Wheat', 'Rice', 'Maize'], 100)
}

# Efficient data storage using DataFrame (optimized for AI workflows)
df = pd.DataFrame(data)

# Convert categorical data to numerical (AI-friendly)
```



```python
df_encoded = pd.get_dummies(df, columns=['soil_type', 'crop_type'])

# Simulate yield (target variable)
df_encoded['yield'] = (
    0.3 * df_encoded['temperature'] +
    0.4 * df_encoded['rainfall'] +
    0.2 * df_encoded['fertilizer'] +
    np.random.normal(0, 10, 100)
)

# Step 2: Split dataset
X = df_encoded.drop('yield', axis=1)
y = df_encoded['yield']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Step 3: Train AI model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Step 4: Predict and Evaluate
y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Absolute Error:", mae)
print("R² Score:", r2)

# Step 5: Display Predictions

results = pd.DataFrame({'Actual': y_test.values, 'Predicted': y_pred})
print(results.head())
```

OUTPUT:



```
Mean Absolute Error: 8.265622630564456
R² Score: 0.43981276175170614
      Actual   Predicted
0   74.675242   81.542152
      Actual   Predicted
0   74.675242   81.542152
0   74.675242   81.542152
1   75.388007   60.560349
2   83.818564   62.712478
3  100.433844   96.279836
4  118.426772  106.975473
```

EXPLANATION:

The script generates a synthetic crop dataset (soil type, temperature, rainfall, fertilizer, crop type), encodes categorical features, simulates a yield target, splits data into train/test sets, trains a RandomForestRegressor, and prints evaluation metrics (MAE and R^2) and a sample of actual vs. predicted values; it also checks for missing dependencies (pandas/numpy/scikit-learn) and instructs how to install them.