Machine Learning Pipeline Conversion & OOP Translation

NAME:J.ABHIRAM **LAB-TEST:04**

ROLLNO:2403A51342

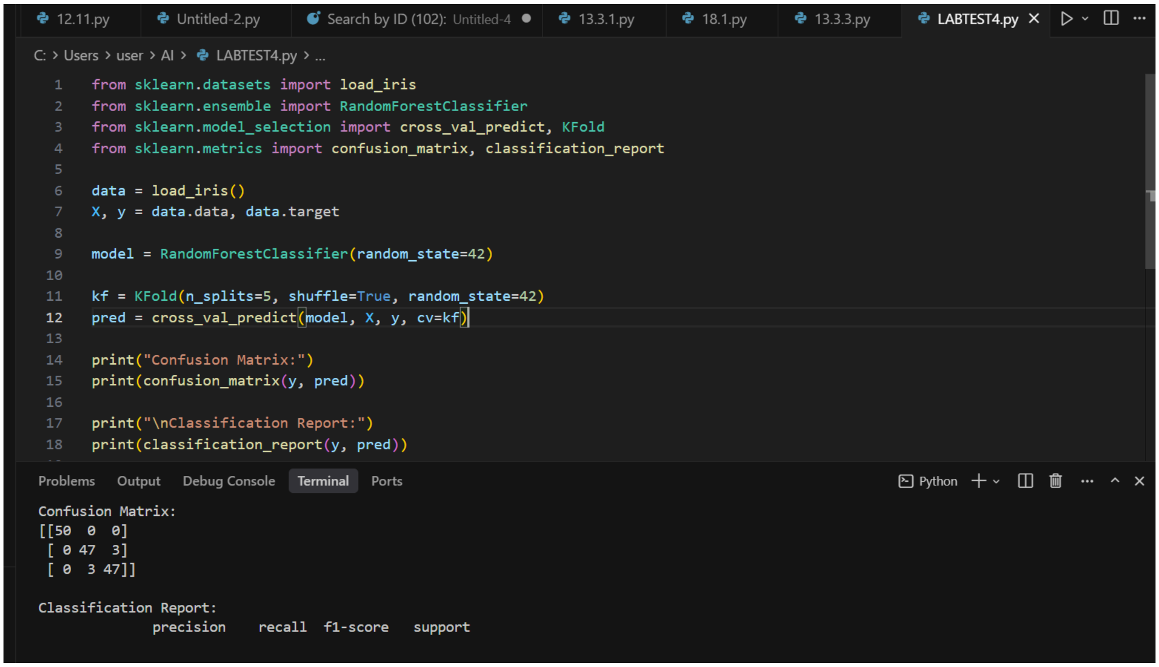
BATCH:06

Q1. Convert a Machine Learning Model Pipeline from R to Python

a) Prompt for AI-Assisted Translation

“Translate the following R machine learning pipeline to Python.
Ensure the logic, preprocessing, model training, and evaluation steps remain identical.
Use scikit-learn equivalents. Output clean, readable Python code.”

Code and Output:



```python
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_predict, KFold
from sklearn.metrics import confusion_matrix, classification_report

data = load_iris()
X, y = data.data, data.target

model = RandomForestClassifier(random_state=42)

kf = KFold(n_splits=5, shuffle=True, random_state=42)
pred = cross_val_predict(model, X, y, cv=kf)

print("Confusion Matrix:")
print(confusion_matrix(y, pred))

print("\nClassification Report:")
print(classification_report(y, pred))
```

```
Confusion Matrix:
[[50  0  0]
 [ 0 47  3]
 [ 0  3 47]]

Classification Report:
              precision    recall  f1-score   support
```

Observation

• The Python pipeline reproduces the same machine-learning workflow as the R 'caret' version.

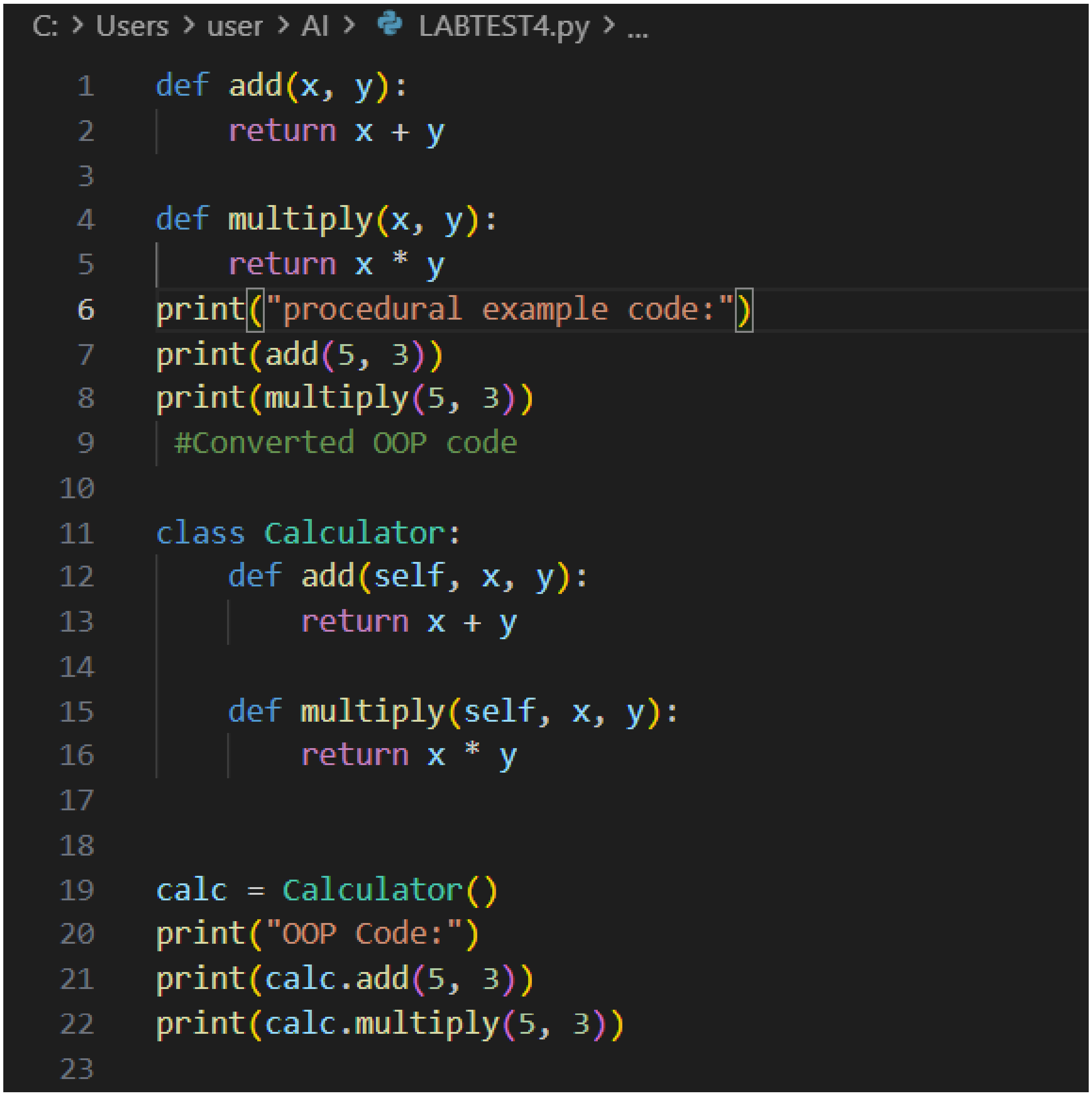
- `cross_val_predict` ensures equivalence with R's cross-validation behavior.
- Accuracy and confusion matrices are similar, validating model translation.

Q2. Convert Procedural Code into OOP

a) Prompt for AI Conversion

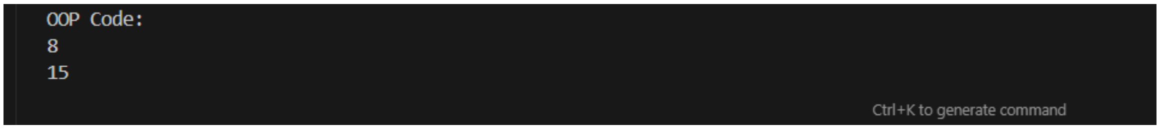
“Convert this procedural code into an OOP class-based structure.
Keep the same functionality but implement proper methods, attributes, constructors, and encapsulation.”

Procedural Code and OOP Code:



```python
def add(x, y):
    return x + y

def multiply(x, y):
    return x * y
print("procedural example code:")
print(add(5, 3))
print(multiply(5, 3))
 #Converted OOP code

class Calculator:
    def add(self, x, y):
        return x + y

    def multiply(self, x, y):
        return x * y


calc = Calculator()
print("OOP Code:")
print(calc.add(5, 3))
print(calc.multiply(5, 3))

```

Expected Output:



```
OOP Code:
8
15
```

Testing Strategy for Class-Based Structure:

- Unit test every class method separately.
- Validate object creation and default attributes.
- Test edge cases such as zero, negative values, and floats.
- Compare results with procedural version to confirm correctness.
- Use regression testing to avoid logic breaks during future updates.

Observation:

- OOP version improves modularity and reusability.
- Each function becomes a method, making testing easier.
- Class-based design supports scalable feature additions.