

# 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:

The screenshot shows a Jupyter Notebook interface with several tabs at the top: 12.11.py, Untitled-2.py, Search by ID (102): Untitled-4, 13.3.1.py, 18.1.py, 13.3.3.py, and LABTEST4.py. The LABTEST4.py tab is active. The code cell contains Python code for loading the Iris dataset, creating a Random Forest classifier, performing 5-fold cross-validation, and printing the confusion matrix and classification report. The output cell shows the resulting confusion matrix and classification report.

```
C: > Users > user > AI > LABTEST4.py > ...
1  from sklearn.datasets import load_iris
2  from sklearn.ensemble import RandomForestClassifier
3  from sklearn.model_selection import cross_val_predict, KFold
4  from sklearn.metrics import confusion_matrix, classification_report
5
6  data = load_iris()
7  X, y = data.data, data.target
8
9  model = RandomForestClassifier(random_state=42)
10
11 kf = KFold(n_splits=5, shuffle=True, random_state=42)
12 pred = cross_val_predict(model, X, y, cv=kf)
13
14 print("Confusion Matrix:")
15 print(confusion_matrix(y, pred))
16
17 print("\nClassification Report:")
18 print(classification_report(y, pred))
```

Output:

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

Classification Report:
precision    recall    f1-score   support
          1       1       1       1
```

### 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:

```
C: > Users > user > AI > LABTEST4.py > ...

1  def add(x, y):
2      return x + y
3
4  def multiply(x, y):
5      return x * y
6  print("procedural example code:")
7  print(add(5, 3))
8  print(multiply(5, 3))
9  #Converted OOP code
10
11 class Calculator:
12     def add(self, x, y):
13         return x + y
14
15     def multiply(self, x, y):
16         return x * y
17
18
19 calc = Calculator()
20 print("OOP Code:")
21 print(calc.add(5, 3))
22 print(calc.multiply(5, 3))
23
```

## Expected Output:

```
OOP Code:
8
15
```

Ctrl+K to generate command

## 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.