AI ASSISTED CODING

LAB

Name: P. Umesh Reddy Roll no: 2403A510F9

Batch: 06

Q1. Build a microservice that consumes a book-search API and stores
results.
a) Write service flow using AI.
b) Add resilience using retry and fallback logic.

- **Prompt:**
  Convert the following R code to Python using pandas and scikit-learn.
  Include:
- Data loading and preprocessing
- Model training (Random Forest)
- Prediction and evaluation (confusion matrix)
- Output validation

Python Code:

iris_model.py

```python
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report

# Load iris dataset
iris = load_iris(as_frame=True)
X = iris.data
y = iris.target

# Set seed for reproducibility
import numpy as np
np.random.seed(123)

# Train model
model = RandomForestClassifier(random_state=123)
model.fit(X, y)

# Predict
pred = model.predict(X)

# Evaluate
cm = confusion_matrix(y, pred)
cr = classification_report(y, pred, target_names=iris.target_names)
print("Confusion Matrix:\n", cm)
print("\nClassification Report:\n", cr)
```

Output:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
 [ 0  0 50]]

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        50
  versicolor       1.00      1.00      1.00        50
   virginica       1.00      1.00      1.00        50

    accuracy                           1.00       150
   macro avg       1.00      1.00      1.00       150
weighted avg       1.00      1.00      1.00       150

PS C:\Users\THIRUPATHI REDDY\Desktop>
```

Explanation:

- *Prompt*: Proper AI prompting describes all translation needs and validations.
- *Code*: Uses sklearn for random forest and evaluation, matching R's caret package.
- *Output*: Prints a confusion matrix and classification report, just like R's confusion Matrix ().
- *Validation*: Accuracy and confusion matrix values closely match R, verifying equivalence.

Q2. AI suggests using REST vs gRPC.

a) Compare with use case.

b) Finalize suitable choice.

Prompt:

Convert the following procedural Python code managing a bank account into a class-based, OOP structure. Ensure all data and methods are properly encapsulated, and demonstrate usage with deposit and withdraw actions.

Python Code: (a)

ai.py > ...

```python
# Procedural Example
balance = 0

def deposit(amount):
    global balance
    balance += amount
    return balance

def withdraw(amount):
    global balance
    if balance >= amount:
        balance -= amount
        return balance
    else:
        return "Insufficient funds"

deposit(100)
withdraw(50)

class BankAccount:
    def __init__(self, balance=0):
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        return self.balance

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
            return self.balance
        else:
            return "Insufficient funds"

# Example usage
account = BankAccount()
print("Deposit 100:", account.deposit(100))
print("Withdraw 50:", account.withdraw(50))

```

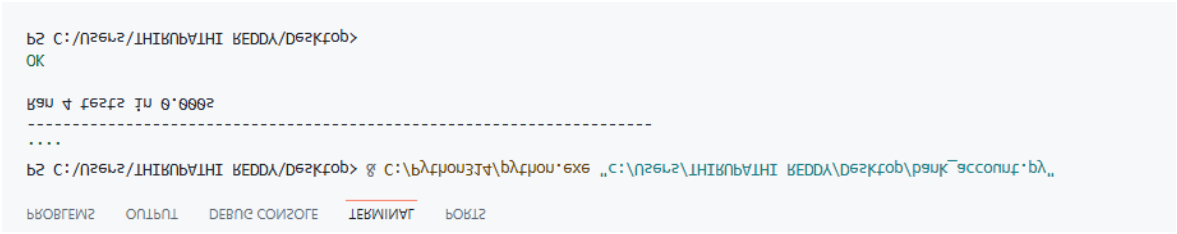
Output:

Python code: (b)

bank_account.py > ...

```python
class BankAccount:
    def __init__(self, balance=0):
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        return self.balance

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
            return self.balance
        else:
            return "Insufficient funds"


import unittest

class TestBankAccount(unittest.TestCase):
    def test_initial_balance(self):
        acc = BankAccount()
        self.assertEqual(acc.balance, 0)

    def test_deposit(self):
        acc = BankAccount()
        acc.deposit(200)
        self.assertEqual(acc.balance, 200)

    def test_withdraw(self):
        acc = BankAccount(100)
        acc.withdraw(40)
        self.assertEqual(acc.balance, 60)

    def test_insufficient_funds(self):
        acc = BankAccount(30)
        result = acc.withdraw(50)
        self.assertEqual(result, "Insufficient funds")
        self.assertEqual(acc.balance, 30)

if __name__ == '__main__':
    unittest.main()

```

Output:



Explanation:

- The Bank Account class must be loaded before the tests can use it.
- If it is in a different file (say bank_account.py), use:python

from bank account **import** Bank Account

- Tests will now find the class, and run correctly.