assignment_1

August 5, 2025

[ ]:
```python
import numpy as np

class LoanRepaymentSimulator:
    """
    A class to simulate and calculate home loan repayments under different
    interest rate scenarios.
    """

    def __init__(self, principal: float, tenure_years: int):
        """
        Initializes the simulator with core loan parameters.

        Args:
            principal (float): The total loan amount.
            tenure_years (int): The total loan tenure in years.
        """
        self.principal = principal
        self.total_tenure_months = tenure_years * 12

    def _calculate_emi(self, p: float, annual_rate: float, n_months: int) ->␣
↪float:
        """
        Calculates the Equated Monthly Installment (EMI).

        Args:
            p (float): The principal amount for calculation.
            annual_rate (float): The annual interest rate (e.g., 8.5 for 8.5%).
            n_months (int): The number of months for repayment.

        Returns:
            float: The calculated EMI amount.
        """
        if p <= 0 or n_months <= 0:
            return 0

        monthly_rate = annual_rate / 12 / 100
        if monthly_rate == 0:
```

```python
            return p / n_months

        emi = p * monthly_rate * (1 + monthly_rate)**n_months / ((1 +␣
↪monthly_rate)**n_months - 1)
        return emi

    def calculate_fixed_rate_scenario(self, fixed_annual_rate: float) -> dict:
        """
        Calculates repayment for a purely fixed-rate loan.

        Args:
            fixed_annual_rate (float): The fixed annual interest rate for the␣
↪entire tenure.

        Returns:
            dict: A dictionary containing the calculated EMI and total␣
↪repayment amount.
        """
        emi = self._calculate_emi(self.principal, fixed_annual_rate, self.
↪total_tenure_months)
        total_payment = emi * self.total_tenure_months

        return {
            "scenario": "Purely Fixed Rate",
            "fixed_rate_pa": fixed_annual_rate,
            "monthly_emi": round(emi, 2),
            "total_repayment": round(total_payment, 2)
        }

    def calculate_variable_rate_scenario(self, initial_repo_rate_x: float,␣
↪fixed_margin_y: float) -> dict:
        """
        Simulates repayment for a purely variable-rate loan.

        Args:
            initial_repo_rate_x (float): The starting repo rate component.
            fixed_margin_y (float): The fixed credit margin component.

        Returns:
            dict: A dictionary containing the total repayment amount from the␣
↪simulation.
        """
        outstanding_principal = self.principal
        total_payment = 0
        current_repo_rate_x = initial_repo_rate_x
```

```
        # Define the stochastic process for repo rate changes
        rate_changes = [0.5, 0.25, 0, -0.25, -0.5]
        probabilities = [1/8, 2/8, 2/8, 2/8, 1/8]

        current_emi = 0

        for month in range(1, self.total_tenure_months + 1):
            # Update rate and recalculate EMI every 2 months
            if (month - 1) % 2 == 0:
                # On the first month, use initial rate. Subsequently, change it.
                if month > 1:
                    change = np.random.choice(rate_changes, p=probabilities)
                    current_repo_rate_x += change

                current_annual_rate = fixed_margin_y + current_repo_rate_x
                remaining_months = self.total_tenure_months - month + 1
                current_emi = self._calculate_emi(outstanding_principal,␣
↪current_annual_rate, remaining_months)

            # Calculate interest for the current month
            monthly_rate = (fixed_margin_y + current_repo_rate_x) / 12 / 100
            interest_for_month = outstanding_principal * monthly_rate

            # Ensure final payment clears the loan exactly
            if month == self.total_tenure_months:
                payment_this_month = outstanding_principal + interest_for_month
            else:
                payment_this_month = current_emi

            principal_paid = payment_this_month - interest_for_month
            outstanding_principal -= principal_paid
            total_payment += payment_this_month

        return {
            "scenario": "Purely Variable Rate",
            "total_repayment": round(total_payment, 2)
        }

    def calculate_hybrid_scenario(self, fixed_rate: float, fixed_tenure_months:␣
↪int,
                                  initial_repo_rate_x: float, fixed_margin_y:␣
↪float) -> dict:
        """
        Simulates repayment for a hybrid (Fixed + Variable) loan.

        Args:
            fixed_rate (float): The interest rate for the initial fixed period.
```

```
            fixed_tenure_months (int): The duration of the fixed period in␣
↪months.
            initial_repo_rate_x (float): The starting repo rate for the␣
↪variable part.
            fixed_margin_y (float): The fixed credit margin for the variable␣
↪part.

        Returns:
            dict: A dictionary containing the total repayment amount from the␣
↪simulation.
        """
        outstanding_principal = self.principal
        total_payment = 0

        # Phase 1: Fixed Rate Period
        fixed_emi = self._calculate_emi(self.principal, fixed_rate, self.
↪total_tenure_months)
        fixed_monthly_rate = fixed_rate / 12 / 100

        for _ in range(fixed_tenure_months):
            interest_for_month = outstanding_principal * fixed_monthly_rate
            principal_paid = fixed_emi - interest_for_month
            outstanding_principal -= principal_paid
            total_payment += fixed_emi

        # Phase 2: Variable Rate Period
        current_repo_rate_x = initial_repo_rate_x
        rate_changes = [0.5, 0.25, 0, -0.25, -0.5]
        probabilities = [1/8, 2/8, 2/8, 2/8, 1/8]
        current_emi = 0
        variable_period_start_month = fixed_tenure_months + 1

        for month in range(variable_period_start_month, self.
↪total_tenure_months + 1):
            if (month - variable_period_start_month) % 2 == 0:
                if month > variable_period_start_month:
                    change = np.random.choice(rate_changes, p=probabilities)
                    current_repo_rate_x += change

                current_annual_rate = fixed_margin_y + current_repo_rate_x
                remaining_months = self.total_tenure_months - month + 1
                current_emi = self._calculate_emi(outstanding_principal,␣
↪current_annual_rate, remaining_months)

            monthly_rate = (fixed_margin_y + current_repo_rate_x) / 12 / 100
            interest_for_month = outstanding_principal * monthly_rate
```

```
            if month == self.total_tenure_months:
                payment_this_month = outstanding_principal + interest_for_month
            else:
                payment_this_month = current_emi

            principal_paid = payment_this_month - interest_for_month
            outstanding_principal -= principal_paid
            total_payment += payment_this_month

        return {
            "scenario": "Hybrid (Fixed + Variable) Rate",
            "total_repayment": round(total_payment, 2)
        }
```

[2]:

```
if __name__ == '__main__':
   # --- Define Core Simulation Parameters ---
    LOAN_PRINCIPAL = 2000000
    LOAN_TENURE_YEARS = 20
    NUM_SIMULATIONS = 1000  # Number of runs for stochastic scenarios for a␣
 ↪stable average

    # --- Define Variable Rate Parameters (Common for all banks as per prompt)␣
 ↪---
    INITIAL_REPO_RATE_X = 6.50
    FIXED_MARGIN_Y = 2.0

    # --- Define Bank-Specific Rate Data ---
    # This structure holds the unique rate offerings for each bank.
    bank_data = [
        {
            "name": "SBI",
            "fixed_full_tenure_rate": 9.25,
            "hybrid_fixed_rate": 8.95,
            "hybrid_fixed_months": 36 # 3 years
        },
        {
            "name": "PNB",
            "fixed_full_tenure_rate": 9.50,
            "hybrid_fixed_rate": 8.90,
            "hybrid_fixed_months": 60 # 5 years
        },
        {
            "name": "HDFC Bank",
            "fixed_full_tenure_rate": 8.70,
            "hybrid_fixed_rate": 8.70,
            "hybrid_fixed_months": 24 # 2 years
```

```
        },
        {
            "name": "Axis Bank",
            "fixed_full_tenure_rate": 14.00,
            "hybrid_fixed_rate": 8.85,
            "hybrid_fixed_months": 24 # 2 years
        },
        {
            "name": "Bank of Baroda",
            "fixed_full_tenure_rate": 9.50,
            "hybrid_fixed_rate": 9.25,
            "hybrid_fixed_months": 36 # 3 years
        },

    ]

    # --- Initialize Simulator and Results Storage ---
    simulator = LoanRepaymentSimulator(principal=LOAN_PRINCIPAL,␣
↪tenure_years=LOAN_TENURE_YEARS)
    final_results = {
        "Scenario 1 (Full Fixed)": {},
        "Scenario 2 (Pure Variable)": {},
        "Scenario 3 (Hybrid)": {}
    }

    print("="*60)
    print("Running Loan Repayment Simulations for Multiple Banks...")
    print(f"Number of simulations per bank for variable scenarios:␣
↪{NUM_SIMULATIONS}")
    print("="*60)

    # Since the variable rate formula is identical for all banks, we run it␣
↪once.
    print("Calculating common 'Purely Variable Rate' benchmark...")
    variable_repayments = []
    for _ in range(NUM_SIMULATIONS):
        result = simulator.
↪calculate_variable_rate_scenario(INITIAL_REPO_RATE_X, FIXED_MARGIN_Y)
        variable_repayments.append(result['total_repayment'])
    average_variable_repayment = sum(variable_repayments) / NUM_SIMULATIONS
    final_results["Scenario 2 (Pure Variable)"]["Common Benchmark"] =␣
↪average_variable_repayment
    print("Done.\n")


    # --- Main Loop to Iterate Through Each Bank ---
    for bank in bank_data:
```

```
        print(f"Processing Bank: {bank['name']}...")

        # --- SCENARIO 1: PURELY FIXED RATE ---
        if bank['fixed_full_tenure_rate'] is not None:
            fixed_result = simulator.calculate_fixed_rate_scenario(
                fixed_annual_rate=bank['fixed_full_tenure_rate']
            )
            final_results["Scenario 1 (Full Fixed)"][bank['name']] =␣
↪fixed_result['total_repayment']

        # --- SCENARIO 3: HYBRID (FIXED + VARIABLE) RATE ---
        if bank['hybrid_fixed_rate'] is not None:
            hybrid_repayments = []
            for _ in range(NUM_SIMULATIONS):
                hybrid_result = simulator.calculate_hybrid_scenario(
                    fixed_rate=bank['hybrid_fixed_rate'],
                    fixed_tenure_months=bank['hybrid_fixed_months'],
                    initial_repo_rate_x=INITIAL_REPO_RATE_X,
                    fixed_margin_y=FIXED_MARGIN_Y
                )
                hybrid_repayments.append(hybrid_result['total_repayment'])

            average_hybrid_repayment = sum(hybrid_repayments) / NUM_SIMULATIONS
            final_results["Scenario 3 (Hybrid)"][bank['name']] =␣
↪average_hybrid_repayment

    print("\nSimulations Complete. Analyzing results...")
    print("="*60 + "\n")


    # --- Analyze and Print Final Recommendations ---
    for scenario, bank_repayments in final_results.items():
        print(f"--- {scenario} ---")
        if not bank_repayments:
            print("  No data available for this scenario.\n")
            continue

        # Print all results for the scenario
        for bank_name, amount in bank_repayments.items():
            print(f"  - {bank_name:<18}:  {amount:,.2f}")

        # Find and print the winner
        if len(bank_repayments) > 1:
            winner_bank = min(bank_repayments, key=bank_repayments.get)
            lowest_amount = bank_repayments[winner_bank]
            print(f"  \n  RECOMMENDATION: {winner_bank} has the lowest␣
↪estimated repayment.\n")
```

```python
        else:
            print()
```

```
============================================================
Running Loan Repayment Simulations for Multiple Banks…
Number of simulations per bank for variable scenarios: 1000
============================================================
Calculating common 'Purely Variable Rate' benchmark…
Done.

Processing Bank: SBI…
Processing Bank: PNB…
Processing Bank: HDFC Bank…
Processing Bank: Axis Bank…
Processing Bank: Bank of Baroda…

Simulations Complete. Analyzing results…
============================================================

--- Scenario 1 (Full Fixed) ---
  - SBI              :  4,396,160.80
  - PNB              :  4,474,229.70
  - HDFC Bank        :  4,226,510.03
  - Axis Bank        :  5,968,899.89
  - Bank of Baroda   :  4,474,229.70

  RECOMMENDATION: HDFC Bank has the lowest estimated repayment.

--- Scenario 2 (Pure Variable) ---
  - Common Benchmark  :  4,162,306.06

--- Scenario 3 (Hybrid) ---
  - SBI              :  4,196,834.65
  - PNB              :  4,222,044.44
  - HDFC Bank        :  4,185,391.76
  - Axis Bank        :  4,171,594.84
  - Bank of Baroda   :  4,229,210.85

  RECOMMENDATION: Axis Bank has the lowest estimated repayment.
```