Name - Paankhi Jain sec - A
PRN - 21070521049

Title - Generative AI CA 2

Q1.Generate a model in Python to represent a Housing loan scheme and create a chart to display the Emi based on rate of interest and reducing balance for a given period. If a customer wishes to close the loan earlier, print the interest lost distributed over the remaining no. Of months. Assume suitable data and inputs as necessary.

Solution -

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

class HousingLoan:
    def __init__(self, loan_amount, interest_rate, tenure_years):
        self.loan_amount = loan_amount
        self.interest_rate = interest_rate / 100  # Convert percentage to decimal
        self.tenure_years = tenure_years
        self.tenure_months = tenure_years * 12
        self.emi = self.calculate_emi()
        self.emi_data = {}

    def calculate_emi(self):
        """ Calculate EMI using the formula for reducing balance method. """
        monthly_rate = self.interest_rate / 12
        emi = self.loan_amount * monthly_rate * (1 + monthly_rate) ** self.tenure_months / ((1 +
monthly_rate) ** self.tenure_months - 1)
        return emi

    def generate_emi_schedule(self):
        """ Generate EMI schedule for each month with reducing balance. """
        outstanding_balance = self.loan_amount
        total_interest_paid = 0

        for month in range(1, self.tenure_months + 1):
            monthly_interest = outstanding_balance * (self.interest_rate / 12)
            principal_paid = self.emi - monthly_interest
            outstanding_balance -= principal_paid
            total_interest_paid += monthly_interest

            self.emi_data[month] = {
                'EMI': round(self.emi, 2),
                'Principal Paid': round(principal_paid, 2),
```

```
            'Interest Paid': round(monthly_interest, 2),
            'Outstanding Balance': round(outstanding_balance, 2)
        }

    return total_interest_paid

def early_closure(self, months_paid):
    """ Calculate interest lost if loan is closed early. """
    total_interest_paid_till_now = sum([self.emi_data[m]['Interest Paid'] for m in range
```

Code Explanation -
HousingLoan Class: Represents a housing loan with attributes such as loan amount, interest rate, and tenure (in years).
calculate_emi() Method: Calculates the EMI using the reducing balance method, which considers the outstanding principal reducing every month as part of the EMI goes toward repaying the principal.
generate_emi_schedule() Method: Creates an EMI schedule for each month, showing how much of the EMI goes toward interest and principal, along with the outstanding balance after each month.
early_closure() Method: Calculates the interest lost if the loan is closed earlier than the original tenure by comparing the total interest scheduled and the interest paid till the early closure.
plot_emi_schedule() Function: Displays a chart of the EMI schedule, showing the breakdown of principal, interest paid, and outstanding balance over time.

Q2. Generate a model to represent interest calculations of a Bank account where the process of calculating interest for 6 months is a. Find minimum balance for each month b. Make a total of all minimum balances c. Calculate interest based on interest rate d. Divide interest by 12 to find one-month interest e. Multiply interest by 6 to show interest in the account. Generate a model to represent transactions and interest calculations for 6 months.

Code -

```
class BankAccount:
    def __init__(self, initial_balance, interest_rate):
        self.balance = initial_balance
        self.interest_rate = interest_rate
        self.transactions = []
        self.monthly_min_balances = []

    def deposit(self, amount, month):
        self.balance += amount
        self.transactions.append((month, "Deposit", amount))

    def withdraw(self, amount, month):
        if self.balance >= amount:
```

```
            self.balance -= amount
            self.transactions.append((month, "Withdrawal", -amount))
        else:
            print(f"Insufficient funds for withdrawal in month {month}")

    def calculate_monthly_minimum_balance(self, month):
        month_transactions = [t for t in self.transactions if t[0] == month]
        min_balance = self.balance
        current_balance = self.balance

        for _, transaction_type, amount in month_transactions:
            if transaction_type == "Deposit":
                current_balance += amount
            else:
                current_balance -= amount
            min_balance = min(min_balance, current_balance)

        self.monthly_min_balances.append(min_balance)

    def calculate_interest(self):
        total_min_balance = sum(self.monthly_min_balances)
        annual_interest = total_min_balance * (self.interest_rate / 100)
        monthly_interest = annual_interest / 12
        six_month_interest = monthly_interest * 6
        return six_month_interest

    def simulate_six_months(self):
        for month in range(1, 7):
            self.calculate_monthly_minimum_balance(month)

        interest = self.calculate_interest()
        self.balance += interest
        return interest

# Example usage
account = BankAccount(initial_balance=1000, interest_rate=5)

# Simulate some transactions
account.deposit(500, month=1)
account.withdraw(200, month=2)
account.deposit(1000, month=3)
account.withdraw(300, month=4)
account.deposit(200, month=5)
account.withdraw(100, month=6)
```

```
# Calculate and display the interest
interest_earned = account.simulate_six_months()
print(f"Interest earned over 6 months: ${interest_earned:.2f}")
print(f"Final balance: ${account.balance:.2f}")
```

Code explanation -

BankAccount Class: Represents a bank account with attributes for balance, interest rate, transactions, and monthly minimum balances.

deposit() and withdraw() Methods: Allow for depositing and withdrawing money, updating the balance and recording each transaction.

calculate_monthly_minimum_balance() Method: Determines the minimum balance for each month by simulating daily balance fluctuations based on transactions.

calculate_interest() Method: Implements the interest calculation. It sums the minimum balances, calculates the annual interest, divides it by 12 to get the monthly interest, and then multiplies by 6 for six months of interest.

simulate_six_months() Method: Simulates six months of banking activity, including deposits and withdrawals, calculates the minimum balance for each month, and then calculates and adds the interest to the balance.