Name: Raghav Birewar  
PRN: 21070521056  
GenAI CA2 Submission  
  
Q2. 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:

class HousingLoan:

def \_\_init\_\_(self, loan\_amount, annual\_interest\_rate, loan\_period\_years):

self.loan\_amount = loan\_amount

self.annual\_interest\_rate = annual\_interest\_rate

self.loan\_period\_years = loan\_period\_years

self.loan\_period\_months = loan\_period\_years \* 12

self.monthly\_interest\_rate = annual\_interest\_rate / 12 / 100

self.emi = self.calculate\_emi()

* Defines a class HousingLoan that initializes with loan\_amount, annual\_interest\_rate, and loan\_period\_years.
* Converts the loan period into months and calculates the monthly interest rate.
* Calls a method to calculate the Equated Monthly Installment (EMI) immediately upon initialization.

def calculate\_emi(self):

r = self.monthly\_interest\_rate

n = self.loan\_period\_months

P = self.loan\_amount

emi = (P \* r \* (1 + r)\*\*n) / ((1 + r)\*\*n - 1)

return emi

* This method calculates the EMI using the standard EMI formula.
* It returns the EMI value based on the principal loan amount, monthly interest rate, and total number of months.

def calculate\_outstanding\_balance(self, months\_paid):

r = self.monthly\_interest\_rate

n = self.loan\_period\_months

P = self.loan\_amount

remaining\_balance = (P \* ((1 + r)\*\*n - (1 + r)\*\*months\_paid)) / ((1 + r)\*\*n - 1)

return remaining\_balance

* This method computes the outstanding loan balance after a certain number of months have been paid.
* The formula subtracts the effects of the months paid to determine how much remains of the loan.

def calculate\_interest\_lost(self, months\_paid):

total\_paid = self.emi \* months\_paid

remaining\_balance = self.calculate\_outstanding\_balance(months\_paid)

interest\_lost = self.loan\_amount + (self.emi \* self.loan\_period\_months) - (total\_paid + remaining\_balance)

return interest\_lost / (self.loan\_period\_months - months\_paid)

* This method calculates the amount of interest lost if the loan is closed before the complete loan term.
* It finds how much interest is lost monthly over the remaining period of the loan.

def plot\_emi\_vs\_interest(loan\_amount, loan\_period\_years):

interest\_rates = np.linspace(5, 15, 100) # Interest rates from 5% to 15%

emis = []

for rate in interest\_rates:

loan = HousingLoan(loan\_amount, rate, loan\_period\_years)

emis.append(loan.emi)

plt.figure(figsize=(10, 6))

plt.plot(interest\_rates, emis, label='EMI', color='blue')

plt.xlabel('Interest Rate (%)')

plt.ylabel('EMI Amount')

plt.title('EMI Amount vs Interest Rate for Loan Period of {} Years'.format(loan\_period\_years))

plt.grid(True)

plt.show()

* This function generates and plots a graph showing the variation in EMI as a function of different interest rates (from 5% to 15%) for a given loan period.
* Uses numpy to generate a range of interest rates and iterates to calculate EMI for each rate, which is then plotted using matplotlib.

loan\_amount = 500000 # Example loan amount

loan\_period\_years = 10 # Example loan period (years)

annual\_interest\_rate = 8 # Example interest rate

loan = HousingLoan(loan\_amount, annual\_interest\_rate, loan\_period\_years)

months\_paid = 5 \* 12

outstanding\_balance = loan.calculate\_outstanding\_balance(months\_paid)

interest\_lost = loan.calculate\_interest\_lost(months\_paid)

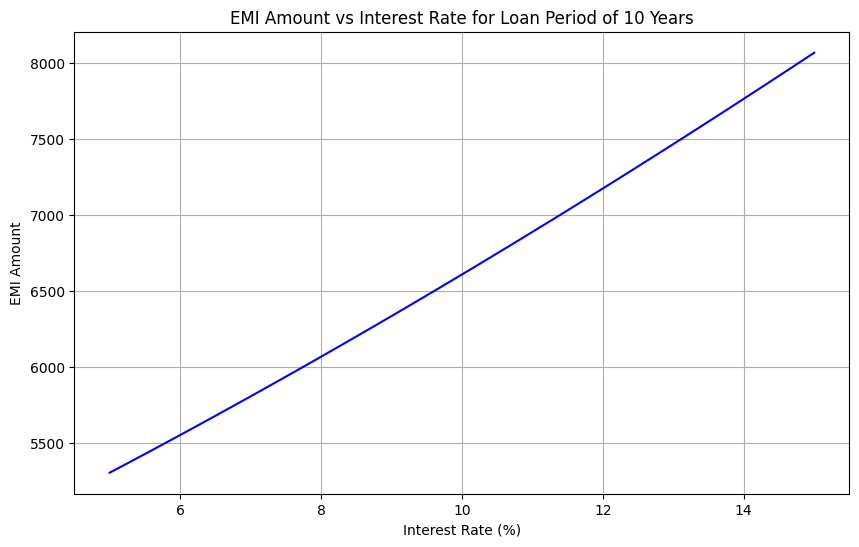
print(f"EMI for loan amount {loan\_amount} at {annual\_interest\_rate}% interest: {loan.emi:.2f}")

print(f"Outstanding balance after {months\_paid} months: {outstanding\_balance:.2f}")

print(f"Interest lost per month if loan is closed after {months\_paid} months: {interest\_lost:.2f}")

plot\_emi\_vs\_interest(loan\_amount, loan\_period\_years)

* **Initialize loan**: Create a loan object with specified loan amount, interest rate, and loan period.
* **Calculate EMI**: Print EMI, outstanding balance after 5 years (60 months), and interest lost if the loan is closed early.
* **Plot**: Plot EMI vs. interest rates using the plot\_emi\_vs\_interest function.

]

Q4. 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.

Solution:

class InterestBankAccount:

def \_\_init\_\_(self, account\_id, initial\_balance, annual\_interest\_rate):

self.account\_id = account\_id

self.initial\_balance = initial\_balance

self.annual\_interest\_rate = annual\_interest\_rate

self.transactions = [[] for \_ in range(6)] # Transactions for 6 months

self.min\_balances = [initial\_balance] \* 6 # Initialize with initial balance

* Initializes a InterestBankAccount class with account\_id, initial\_balance, and annual\_interest\_rate.
* Sets up lists to store 6 months of transactions and initialize the minimum balances for each month with the initial balance.

def add\_transaction(self, month, amount):

if 0 <= month < 6:

self.transactions[month].append(amount)

self.update\_min\_balance(month)

* Allows adding a transaction (deposit/withdrawal) for a specified month.
* After a transaction is added, it calls the update\_min\_balance method to recalculate the minimum balance for that month.

def update\_min\_balance(self, month):

current\_balance = self.initial\_balance

min\_balance = current\_balance

for amount in self.transactions[month]:

current\_balance += amount

min\_balance = min(min\_balance, current\_balance)

self.min\_balances[month] = min\_balance

* This method recalculates the minimum balance after all transactions for a given month.
* It simulates month-by-month balances and updates the minimum balance, ensuring it reflects the lowest value for that month.

def calculate\_interest\_for\_6\_months(self):

total\_min\_balance = sum(self.min\_balances)

monthly\_interest = (total\_min\_balance \* self.annual\_interest\_rate / 100) / 12

interest\_for\_6\_months = monthly\_interest \* 6

return interest\_for\_6\_months

* Computes the total minimum balances over the 6 months.
* Calculates interest based on the sum of minimum balances and the annual interest rate, and then multiplies the monthly interest by 6 to get the interest for 6 months.

def \_\_str\_\_(self):

return f"Account ID: {self.account\_id}, Min Balances: {self.min\_balances}"

* Defines a string representation of the account, displaying the account ID and the minimum balances for each month.

def simulate\_account\_interest():

account = InterestBankAccount(account\_id="ACC001", initial\_balance=50000, annual\_interest\_rate=4)

transactions = [

[-5000, 1000], # Month 1

[-2000, -3000], # Month 2

[5000], # Month 3

[-1000, 2000], # Month 4

[-1500], # Month 5

[-2500, 4000] # Month 6

]

for month, txn\_list in enumerate(transactions):

for txn in txn\_list:

account.add\_transaction(month, txn)

interest\_for\_6\_months = account.calculate\_interest\_for\_6\_months()

print(f"Minimum balances for each month: {account.min\_balances}")

print(f"Interest for 6 months: {interest\_for\_6\_months:.2f}")

* Simulates an account with an initial balance of ₹50,000 and 4% annual interest rate.
* Processes transactions for each of the 6 months and calculates the interest earned over 6 months.
* Prints the minimum balances for each month and the total interest earned.

def plot\_minimum\_balances(account):

months = ['Month 1', 'Month 2', 'Month 3', 'Month 4', 'Month 5', 'Month 6']

min\_balances = account.min\_balances

plt.figure(figsize=(10, 6))

plt.plot(months, min\_balances, marker='o', color='blue', label='Minimum Balance')

plt.xlabel('Months')

plt.ylabel('Minimum Balance (₹)')

plt.title('Minimum Balances for Each Month')

plt.grid(True)

plt.legend()

plt.show()

* Plots the minimum balances for each month using matplotlib.
* Displays the data points and the trend of minimum balances for 6 months.

if \_\_name\_\_ == "\_\_main\_\_":

simulate\_account\_interest()

account = InterestBankAccount(account\_id="ACC001", initial\_balance=50000, annual\_interest\_rate=4)

transactions = [

[-6000, 3000], # Month 1

[-4000, -1000], # Month 2

[3000], # Month 3

[-2000, 15000], # Month 4

[-2500], # Month 5

[-2300, 5300] # Month 6

]

for month, txn\_list in enumerate(transactions):

for txn in txn\_list:

account.add\_transaction(month, txn)

plot\_minimum\_balances(account)

* Simulates account interest and prints the results for one set of transactions.
* Creates a new account and processes another set of transactions for plotting minimum balances over 6 months. The minimum balances are visualized using the plot\_minimum\_balances function.

