Name: Veera Bhosale  
PRN: 21070521093  
GenAI CA 2  
  
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 which 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 is used to calculate 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 will compute 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 the loan remains.

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

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)

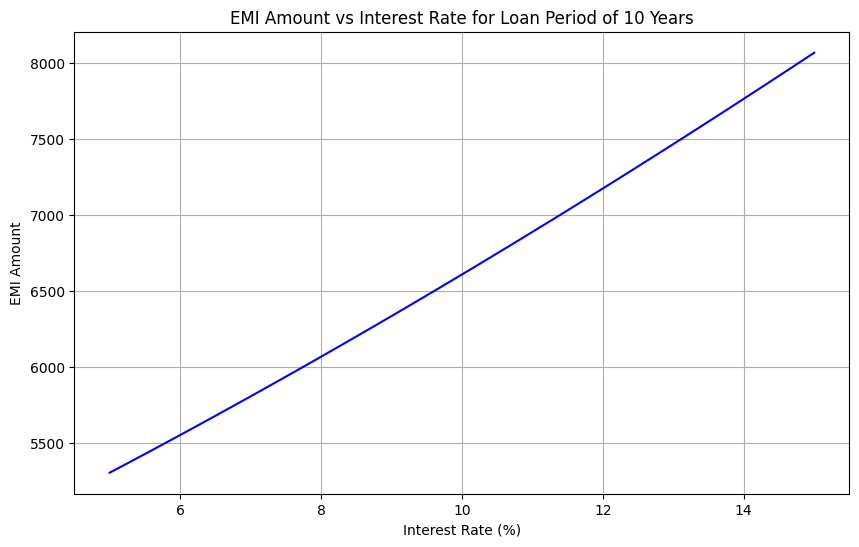
print(f"EMI for loan amount {loan\_amount} at {annual\_interest\_rate}% interest: {loan.emi:.2f}")

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.

]

Q:3 Generate a model for an Insurance company to hold information on the insurer's vehicle,

and create a chart of monthly, yearly, and quarterly premiums based on no. of years of insurance

where in each year, the value of the vehicle depreciates by 7%.

class VehicleInsurance:

def \_\_init\_\_(self, vehicle\_value, initial\_premium\_rate, insurance\_period\_years):

self.vehicle\_value = vehicle\_value

self.initial\_premium\_rate = initial\_premium\_rate # Premium rate as a percentage of vehicle value

self.insurance\_period\_years = insurance\_period\_years

self.depreciation\_rate = 0.07 # 7% depreciation per year

Initializes the VehicleInsurance class with:

* vehicle\_value: initial value of the vehicle.
* initial\_premium\_rate: percentage of vehicle value used to calculate the insurance premium.
* insurance\_period\_years: number of years the policy is valid.
* depreciation\_rate: set at 7% annually, to account for vehicle depreciation.

def calculate\_premium(self, year):

depreciated\_value = self.vehicle\_value \* ((1 - self.depreciation\_rate) \*\* year)

annual\_premium = depreciated\_value \* self.initial\_premium\_rate / 100

quarterly\_premium = annual\_premium / 4

monthly\_premium = annual\_premium / 12

return monthly\_premium, quarterly\_premium, annual\_premium

Calculates the premium for a specific year:

* Depreciates the vehicle value by 7% each year.
* Uses the depreciated value to calculate annual, quarterly, and monthly premiums based on the initial premium rate.
* Returns monthly, quarterly, and annual premiums.

def calculate\_premiums\_over\_years(self):

monthly\_premiums = []

quarterly\_premiums = []

annual\_premiums = []

for year in range(1, self.insurance\_period\_years + 1):

monthly, quarterly, annual = self.calculate\_premium(year)

monthly\_premiums.append(monthly)

quarterly\_premiums.append(quarterly)

annual\_premiums.append(annual)

return monthly\_premiums, quarterly\_premiums, annual\_premiums

Calculates premiums for every year in the insurance period:

* Iterates over each year of the policy (from year 1 to insurance\_period\_years).
* For each year, calls the calculate\_premium method to compute the monthly, quarterly, and annual premiums.
* Stores these premiums in respective lists and returns the lists of premiums for all years.

def plot\_premiums(insurance\_period\_years, monthly\_premiums, quarterly\_premiums, annual\_premiums):

years = list(range(1, insurance\_period\_years + 1))

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

plt.plot(years, monthly\_premiums, label='Monthly Premium', marker='o', color='blue')

plt.plot(years, quarterly\_premiums, label='Quarterly Premium', marker='o', color='green')

plt.plot(years, annual\_premiums, label='Annual Premium', marker='o', color='red')

plt.xlabel('Years of Insurance')

plt.ylabel('Premium Amount (in ₹)')

plt.title('Vehicle Insurance Premiums Over Time')

plt.grid(True)

plt.legend()

plt.show()

Plots the monthly, quarterly, and annual premiums over the entire insurance period:

* The x-axis represents the number of years the insurance policy has been running.
* The y-axis shows the premium amounts.
* It plots three lines: monthly, quarterly, and annual premiums, using different colors and labels.
* The graph includes a title, grid, and legend.

vehicle\_value = 1000000 # Initial value of the vehicle (₹1,000,000)

initial\_premium\_rate = 2 # 2% of vehicle value as the annual premium

insurance\_period\_years = 10 # Number of years the insurance policy will run

# Create a vehicle insurance object

insurance = VehicleInsurance(vehicle\_value, initial\_premium\_rate, insurance\_period\_years)

# Calculate the premiums over the insurance period

monthly\_premiums, quarterly\_premiums, annual\_premiums = insurance.calculate\_premiums\_over\_years()

# Plot the premiums

plot\_premiums(insurance\_period\_years, monthly\_premiums, quarterly\_premiums, annual\_premiums)

* Defines the vehicle\_value, initial\_premium\_rate, and insurance\_period\_years for the insurance policy.
* Creates a VehicleInsurance object using the defined values.
* Calls the calculate\_premiums\_over\_years method to get lists of monthly, quarterly, and annual premiums over the policy period.
* Calls the plot\_premiums function to visualize the premium data over the entire insurance period.