Symbiosis Institute of Technology, Nagpur

Subject: GenAI
CA – II (Assignment)
Name - Akanksha gangwani
PRN - 21070521006
Section- A

Q:2 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.

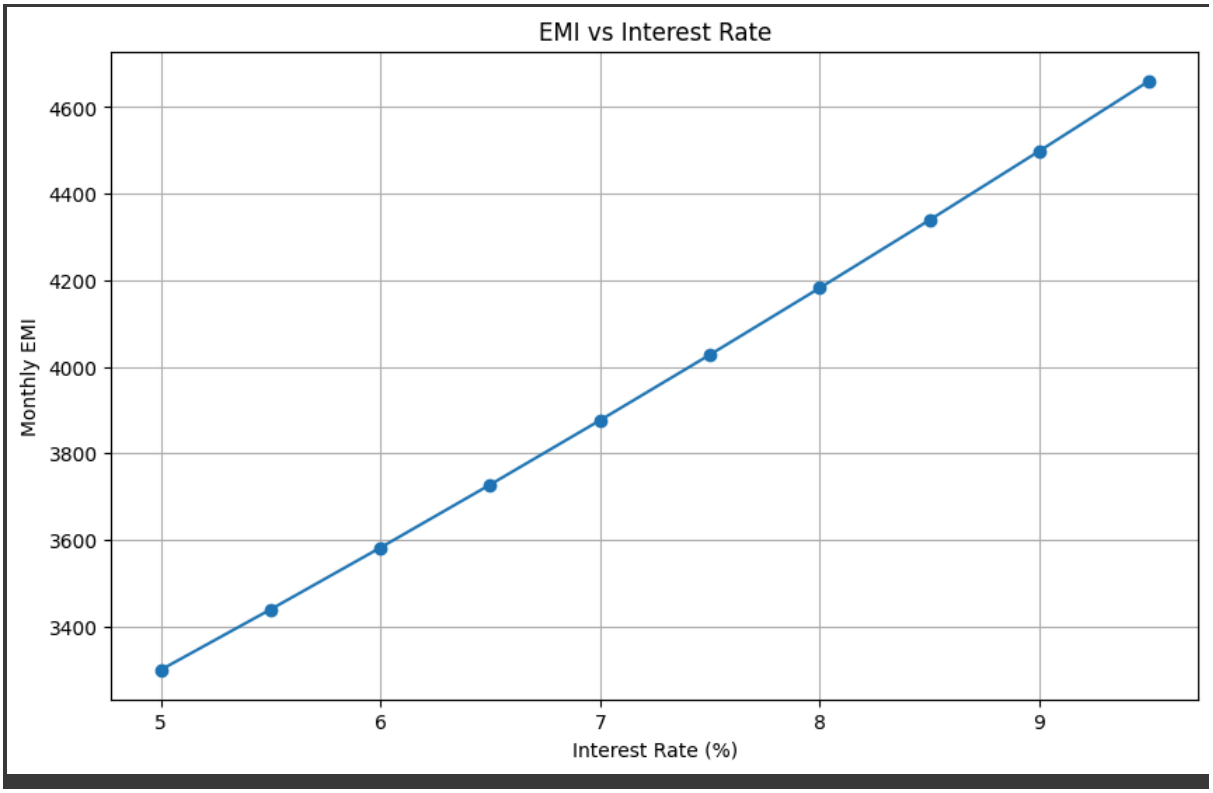
PYTHON MODEL:-

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

class HousingLoanScheme:
    def __init__(self, principal, annual_rate, tenure_years):
        self.principal = principal
        self.annual_rate = annual_rate
        self.tenure_months = tenure_years * 12
        self.monthly_rate = annual_rate / 12 / 100

    def calculate_emi(self):
        r = self.monthly_rate
        n = self.tenure_months
        P = self.principal
        emi = P * r * ((1 + r) ** n) / (((1 + r) ** n) - 1)
        return emi

    def total_interest(self):
        emi = self.calculate_emi()
        total_payment = emi * self.tenure_months
        total_interest = total_payment - self.principal
        return total_interest

    def early_closure_interest_loss(self, months_paid):
```

```
        total_interest = self.total_interest()
        remaining_months = self.tenure_months - months_paid
        emi = self.calculate_emi()
        paid_interest = (emi * months_paid) - (self.principal *
months_paid / self.tenure_months)
        interest_loss = total_interest - paid_interest
        return interest_loss, remaining_months

    def emi_for_different_rates(self, rates):
        emi_values = []
        for rate in rates:
            self.annual_rate = rate
            self.monthly_rate = rate / 12 / 100
            emi_values.append(self.calculate_emi())
        return emi_values

# Example usage
loan_amount = 500000  # 500,000
tenure_years = 20     # 20 years

loan = HousingLoanScheme(loan_amount, 6.5, tenure_years)

emi = loan.calculate_emi()
total_interest = loan.total_interest()
early_closure_loss, remaining_months =
loan.early_closure_interest_loss(120)

print(f"Monthly EMI: {emi:.2f}")
print(f"Total Interest: {total_interest:.2f}")
print(f"Interest lost if closed after 120 months:
{early_closure_loss:.2f}")
print(f"Remaining months after early closure: {remaining_months}")

interest_rates = np.arange(5.0, 10.0, 0.5)
emi_values = loan.emi_for_different_rates(interest_rates)

plt.figure(figsize=(10, 6))
plt.plot(interest_rates, emi_values, marker='o')
plt.title('EMI vs Interest Rate')
plt.xlabel('Interest Rate (%)')
plt.ylabel('Monthly EMI')
plt.grid(True)
plt.show()
```

OUTPUT:-

Monthly EMI: 3727.87
Total Interest: 394687.76
Interest lost if closed after 120 months: 197343.88
Remaining months after early closure: 120



Q:6 Generate a model to represent a mathematical equation, write a program to parse the equation, and ask for input for each parameter.

FOR EASE, I AM TAKING THIS MATHEMATICAL EQUATION:

Quadratic equation in standard form

$$ax^2 + bx + c = 0$$

Quadratic Formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```python
import math

class QuadraticEquationSolver:
    def __init__(self):
        self.equation = "ax^2 + bx + c = 0"

    def get_user_input(self):
        print("For the quadratic equation: ax^2 + bx + c = 0")
        a = float(input("Enter the value of a: "))
        b = float(input("Enter the value of b: "))
        c = float(input("Enter the value of c: "))
        return a, b, c

    def solve(self, a, b, c):
        discriminant = b**2 - 4*a*c

        if discriminant > 0:
            root1 = (-b + math.sqrt(discriminant)) / (2*a)
            root2 = (-b - math.sqrt(discriminant)) / (2*a)
            return f"The solutions are: x1 = {root1}, x2 = {root2}"
        elif discriminant == 0:
            root = -b / (2*a)
            return f"The solution is: x = {root}"
        else:
            real_part = -b / (2*a)
            imaginary_part = math.sqrt(-discriminant) / (2*a)
            return f"The solutions are: x1 = {real_part} +
{imaginary_part}i, x2 = {real_part} - {imaginary_part}i"

solver = QuadraticEquationSolver()
```

```
a, b, c = solver.get_user_input()

result = solver.solve(a, b, c)

print(result)
```

OUTPUT:

```
For the quadratic equation: ax^2 + bx + c = 0
Enter the value of a: 2
Enter the value of b: -4
Enter the value of c: 2
The solution is: x = 1.0
```