Name: Bhumika Kosarkar **PRN:** 21070521022

Department: CSE **Division:** A

Symbiosis Institute of Technology

Department of Applied Science

Generative AI

CA-2

Computer Science and Engineering

Batch 2021-2025

Semester - VII

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.

Ans: Let's understand the model for Housing loan scheme in certain steps:

1. EMI Calculation:

The formula for calculating the Equated Monthly Installment (EMI) is based on the principal loan amount, the interest rate, and the tenure of the loan.

The formula used is:

$$E = P \times r \times \frac{(1+r)^n}{(1+r)^n - 1}$$

Where,

- E is the EMI
- P is the principal amount
- r is the monthly rate of interest
- n is the number of months

The EMI remains constant throughout the loan tenure, but the proportion of interest and principal in each EMI payment changes. Initially, the interest portion is higher, and as the outstanding loan balance reduces, the interest portion decreases while the principal repayment increases.

2. Reducing Balance Calculation:

After each EMI payment, the loan balance reduces. The loan scheme uses a reducing balance method, meaning the interest for each month is calculated on the remaining principal balance (the amount of loan still unpaid).

Here's how the reducing balance is calculated month by month:

- The interest for the month is calculated month by month:

$$\text{Interest} = \text{Remaining Principal} \times \frac{\text{Annual Interest Rate}}{12 \times 100}$$

- The principal component of the EMI is then calculated as:

Principal Payment = EMI - Interest

- The new remaining is:

Remaining Principal = Previous Remaining Principal - Principal Payment

3. Early Loan Closure:

If a customer decides to close the loan early (before the full tenure), the model calculates the interest that would have been paid for the remaining months if the loan had not been closed early.

To calculate the interest lost:

- After a specific number of months (say, 60 months for early closure after 5 years in a 10-year loan), the model sums up the interest that would have been paid over the remaining tenure (from month 61 to month 120).
- This represents the interest lost due to early closure.

4. Visualizing EMI vs Interest Rate:

To help users understand how different interest rates affect the EMI, a chart is generated that plots the EMI against various rates of interest. The script calculates the EMI for interest rates ranging from 5% to 15% (in increments of 0.5%) and then plots the EMI values for comparison.This visualization provides insight into how sensitive the EMI is to changes in the interest rate. For example, a slight increase in the interest rate can significantly raise the EMI, making the loan more expensive for the customer.

Python Code :

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

def calculate_emi(principal, rate_of_interest, tenure_years):
    tenure_months = tenure_years * 12
    monthly_interest_rate = rate_of_interest / (12 * 100)
    emi = principal * monthly_interest_rate * (1 + monthly_interest_rate) ** tenure_months / ((1 + monthly_interest_rate) ** tenure_months - 1)
    return emi, tenure_months

def loan_summary(principal, rate_of_interest, tenure_years, early_closure_month):
    emi, total_months = calculate_emi(principal, rate_of_interest, tenure_years)
    remaining_principal = principal
    interest_paid = 0
    balance_list = []
    emi_list = []

    for month in range(1, total_months + 1):
        interest_for_month = remaining_principal * rate_of_interest / (12 * 100)
        principal_payment = emi - interest_for_month
        remaining_principal -= principal_payment
        interest_paid += interest_for_month
        balance_list.append(remaining_principal)
        emi_list.append(emi)

        if month == early_closure_month:
            interest_lost = sum([remaining_principal * rate_of_interest / (12 * 100) for _ in range(month, total_months)])
            print(f"Interest lost due to early closure at month {early_closure_month}: {interest_lost:.2f}")
            break

    return emi_list[:early_closure_month], balance_list[:early_closure_month]
```

calculate_emi function:

- It calculates the EMI (Equated Monthly Installment) for a loan.
- Inputs: Principal amount, interest rate, and loan tenure (in years).
- Outputs: EMI amount and total number of months for repayment.

loan_summary function:

- It tracks the remaining loan balance and total interest paid over time.
- It prints how much interest would be lost if the loan is closed early (at a specified month).
- It returns lists of EMI payments and remaining balance for each month up to early closure.

```
    return emi_list[:early_closure_month], balance_list[:early_closure_month]

def plot_emi_vs_roi(principal, tenure_years):
    rates_of_interest = np.arange(5, 15, 0.5)
    emis = []

    for rate in rates_of_interest:
        emi, _ = calculate_emi(principal, rate, tenure_years)
        emis.append(emi)

    plt.figure(figsize=(10, 6))
    plt.plot(rates_of_interest, emis, marker='o')
    plt.title('EMI vs Rate of Interest')
    plt.xlabel('Rate of Interest (%)')
    plt.ylabel('EMI (₹)')
    plt.grid(True)
    plt.show()

def main():
    principal = 1000000
    rate_of_interest = 10
    tenure_years = 10
    early_closure_month = 60

    print(f"Loan Amount: ₹{principal}, Interest Rate: {rate_of_interest}%, Tenure: {tenure_years} years\n")

    emi_list, balance_list = loan_summary(principal, rate_of_interest, tenure_years, early_closure_month)
    plot_emi_vs_roi(principal, tenure_years)

if __name__ == "__main__":
    main()
```

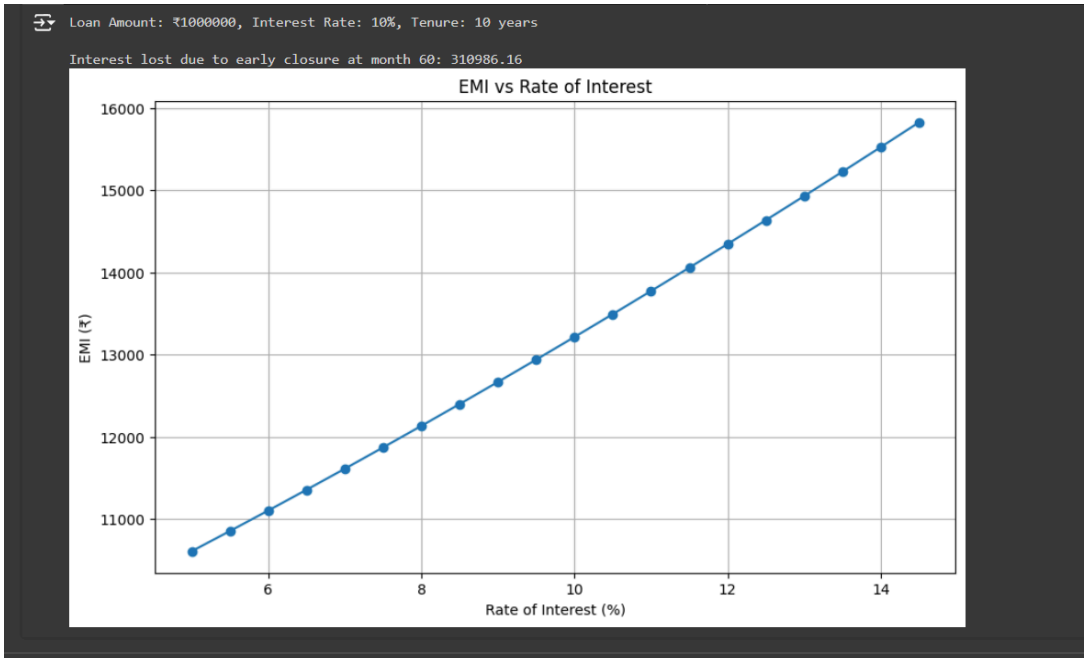
plot_emi_vs_roi function:

- It plots a graph showing how EMI changes with different interest rates (between 5% to 15%).
- The x-axis represents the rate of interest, and the y-axis represents the EMI.

main function:

- Sets loan parameters like principal amount, interest rate, and loan tenure.
- Calls the loan_summary function to calculate and print EMI and balance details.
- Calls plot_emi_vs_roi to show a graph of EMI vs rate of interest.

Output:



Q:5 Generate a model for Covid 19 with symptoms of fever, cold, shivering, weight loss, generate 100 model data with random values for each parameter and order by parameter lowest to highest in display based on the input parameter.

Ans: Lets understand the model for Covid 19 in certain steps:

1. **Data Generation:**

The function generate_data() is used to create 100 random data points. Each data point is a dictionary with the following parameters:

- fever: A random floating-point number between 97°F and 105°F, rounded to one decimal place, representing body temperature.
- cold: A random integer between 1 and 10, representing the severity of cold symptoms.
- shivering: A boolean value (True or False), indicating whether the person experiences shivering.
- weight loss: A random floating-point number between 0 and 15 pounds, rounded to one decimal place, representing weight loss.

The generated data is stored in a list, with each dictionary representing one individual's COVID-19 symptoms.

2. Sorting the Data:

The function sort_data() takes the generated data and a sorting parameter as inputs. It uses Python's built-in sorted() function to sort the data based on the given parameter. The sorting

happens in ascending order. For instance, if the user chooses to sort by fever, the data will be ordered from the lowest to the highest temperature.

3. User Input:
In the main() function, the program prompts the user to specify which parameter they want to sort the data by. The allowed parameters are:

- fever
- cold
- weight_loss

If the user provides an invalid parameter (e.g., shivering or a misspelled parameter), the program outputs an error message.

4. Displaying the Results:
After sorting the data based on the input parameter, the program iterates over the sorted list and prints each individual record in a readable format.

Python Code:

```python
def generate_covid_data():
    data = []
    for _ in range(100):
        fever = round(random.uniform(97, 105), 1)
        cold = random.randint(1, 10)
        shivering = random.choice([True, False])
        weight_loss = round(random.uniform(0, 15), 1)
        data.append({
            "fever": fever,
            "cold": cold,
            "shivering": shivering,
            "weight_loss": weight_loss
        })
    return data

def sort_covid_data(data, param):
    if param == "shivering":
        return sorted(data, key=lambda x: x[param], reverse=False)
    else:
        return sorted(data, key=lambda x: x[param])

def main():
    data = generate_covid_data()
    print("Enter the parameter to sort by (fever, cold, shivering, weight_loss): ")
    param = input().strip()

    if param in ["fever", "cold", "shivering", "weight_loss"]:
        sorted_data = sort_covid_data(data, param)
        for record in sorted_data:
            print(record)
    else:
        print("Invalid parameter. Please enter 'fever', 'cold', 'shivering', or 'weight_loss'.")

if __name__ == "__main__":
    main()
```

generate_data function:

- This function generates 100 random records, each representing a person's health data.
- Each record contains:
    - fever: A random temperature between 97°F and 105°F (rounded to one decimal place).
    - cold: A random integer between 1 and 10, representing the severity of cold symptoms.
    - shivering: A random boolean value (True or False) indicating whether the person has shivering.
    - weight_loss: A random weight loss value between 0 and 15 pounds (rounded to one decimal place).
- The function returns a list of dictionaries, where each dictionary holds the above data for one person.

sort_data function:

- This function sorts the generated data based on a specified parameter (e.g., fever, cold, weight loss).
- Inputs:
    - data: The list of dictionaries (health data).
    - param: The parameter by which the data should be sorted.
- The sorted list is returned.

main function:

- The program generates the random health data by calling the generate_data function.
- It then asks the user for a parameter (fever, cold, or weight_loss) to sort the data by.
- If the user enters a valid parameter, it sorts the data using the sort_data function and prints the sorted records.
- If the user enters an invalid parameter (e.g., "shivering"), it prints an error message.

Output:

```
Enter the parameter to sort by (fever, cold, shivering, weight_loss):
cold
{'fever': 101.9, 'cold': 1, 'shivering': False, 'weight_loss': 13.4}
{'fever': 102.7, 'cold': 1, 'shivering': True, 'weight_loss': 4.6}
{'fever': 99.6, 'cold': 1, 'shivering': True, 'weight_loss': 6.8}
{'fever': 98.6, 'cold': 1, 'shivering': False, 'weight_loss': 6.2}
{'fever': 103.5, 'cold': 1, 'shivering': False, 'weight_loss': 4.3}
{'fever': 99.0, 'cold': 1, 'shivering': False, 'weight_loss': 13.9}
{'fever': 102.3, 'cold': 1, 'shivering': True, 'weight_loss': 6.9}
{'fever': 98.5, 'cold': 1, 'shivering': False, 'weight_loss': 13.9}
{'fever': 101.4, 'cold': 1, 'shivering': False, 'weight_loss': 9.7}
{'fever': 98.6, 'cold': 2, 'shivering': True, 'weight_loss': 3.7}
{'fever': 97.5, 'cold': 2, 'shivering': True, 'weight_loss': 8.0}
{'fever': 100.4, 'cold': 2, 'shivering': False, 'weight_loss': 10.5}
{'fever': 101.1, 'cold': 2, 'shivering': True, 'weight_loss': 3.0}
{'fever': 97.8, 'cold': 2, 'shivering': False, 'weight_loss': 10.9}
{'fever': 99.2, 'cold': 2, 'shivering': False, 'weight_loss': 6.2}
{'fever': 100.8, 'cold': 2, 'shivering': True, 'weight_loss': 6.7}
{'fever': 104.0, 'cold': 2, 'shivering': False, 'weight_loss': 6.9}
{'fever': 100.6, 'cold': 3, 'shivering': False, 'weight_loss': 5.6}
{'fever': 99.4, 'cold': 3, 'shivering': True, 'weight_loss': 0.7}
{'fever': 100.0, 'cold': 3, 'shivering': True, 'weight_loss': 6.9}
{'fever': 104.5, 'cold': 3, 'shivering': True, 'weight_loss': 14.4}
```