Generative AI CA – 2

Submitted by –

Name – Anjali Singh

PRN - 21070521012

Q1: Bank Account Model with Savings and Transactions

In this problem, I developed a model in Python to represent a basic savings account with functionality to handle deposits and withdrawals. Each account starts with an initial balance, and a series of random transactions is performed over a period. The aim is to simulate real-world banking operations, where users may deposit or withdraw funds at any time.

Attributes:

- **Savings Account**: This type of account holds the balance of the customer, allowing them to perform transactions like deposits and withdrawals. In this model, we are not considering the interest earned on the savings.
- **Transactions**:
    - **Deposit**: When money is added to the account, the balance increases accordingly.
    - **Withdrawal**: When money is withdrawn from the account, the balance decreases, provided the account has sufficient funds to cover the withdrawal.
- **Random Data Generation**: To mimic real-world variability, I generated 100 accounts with random balances and transactions. The number of transactions and the amounts are also randomized to simulate multiple months of activity.
- **Sorting by Balance**: After processing all transactions, the accounts are sorted by their final balance, from the lowest to the highest. This helps in visualizing which accounts have the smallest and largest balances at the end of the simulation.

Implementation:

```
import random

class BankAccount:
    def __init__(self, account_id, balance=0):
        self.account_id = account_id
        self.balance = balance
        self.transactions = []

    def deposit(self, amount):
        self.balance += amount
        self.transactions.append(f'Deposit: {amount}')

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            self.transactions.append(f'Withdraw: {amount}')
        else:
            print(f"Insufficient balance for account {self.account_id}")

    def __repr__(self):
        return f"Account {self.account_id}: Balance {self.balance}"

random.seed(42)
accounts = [BankAccount(i, random.randint(1000, 10000)) for i in range(1, 101)]

for account in accounts:
    for _ in range(random.randint(1, 10)):
        if random.choice([True, False]):
            account.deposit(random.randint(100, 5000))
        else:
            account.withdraw(random.randint(100, 5000))

accounts.sort(key=lambda acc: acc.balance)
for acc in accounts:
    print(acc)
```

Output:

The final output of this model is a list of 100 accounts, each with its last balance displayed in ascending order. This allows us to see at a glance which accounts have the least and most money after the transactions have been processed.

```
Insufficient balance for account 77
Insufficient balance for account 77
Insufficient balance for account 83
Insufficient balance for account 87
Insufficient balance for account 89
Insufficient balance for account 94
Insufficient balance for account 96
Insufficient balance for account 96
Insufficient balance for account 97
Insufficient balance for account 97
Account 71: Balance 6
Account 83: Balance 7
Account 81: Balance 142
Account 6: Balance 163
Account 40: Balance 215
Account 12: Balance 295
Account 70: Balance 349
Account 14: Balance 547
Account 68: Balance 601
Account 82: Balance 731
Account 52: Balance 800
Account 51: Balance 877
Account 80: Balance 908
Account 36: Balance 929
Account 25: Balance 945
Account 77: Balance 1102
Account 94: Balance 1107
Account 89: Balance 1116
Account 96: Balance 1373
Account 75: Balance 1497
Account 97: Balance 1502
Account 88: Balance 1528
Account 56: Balance 1541
Account 10: Balance 1794
Account 32: Balance 1917
Account 4: Balance 2106
```

Q2: Housing Loan Scheme and EMI Calculation

In this task, I created a Python model to represent a **housing loan scheme**, focusing on calculating the **Equated Monthly Installments (EMI)** and tracking the loan's reducing balance

over time. This is a crucial aspect of loan repayment, as the borrower needs to know how much they'll need to pay each month and how their outstanding loan amount will decrease over time.

Key Concepts:

- **EMI (Equated Monthly Installment)**: EMI is the fixed monthly amount the borrower pays over a specified period to repay the loan. It's calculated using the formula:

EMI=P×r×(1+r)n(1+r)n−1EMI = \frac{P \times r \times (1 + r)^n}{(1 + r)^n - 1}EMI=(1+r)n−1P×r×(1+r)n

Where:

- PPP represents the loan amount (principal)
- rrr is the monthly interest rate
- nnn is the total number of months for repayment
- **Reducing Balance Method**: This method recalculates the interest amount each month based on the outstanding principal. This is beneficial for the borrower, as the amount of interest decreases as the loan is repaid.
- **Early Loan Closure**: If the borrower decides to close the loan earlier than the full term, they save on the interest payments for the remaining period. I added functionality to calculate the interest that would have been paid if the loan was completed over the full term, showing how much interest is saved by closing early.

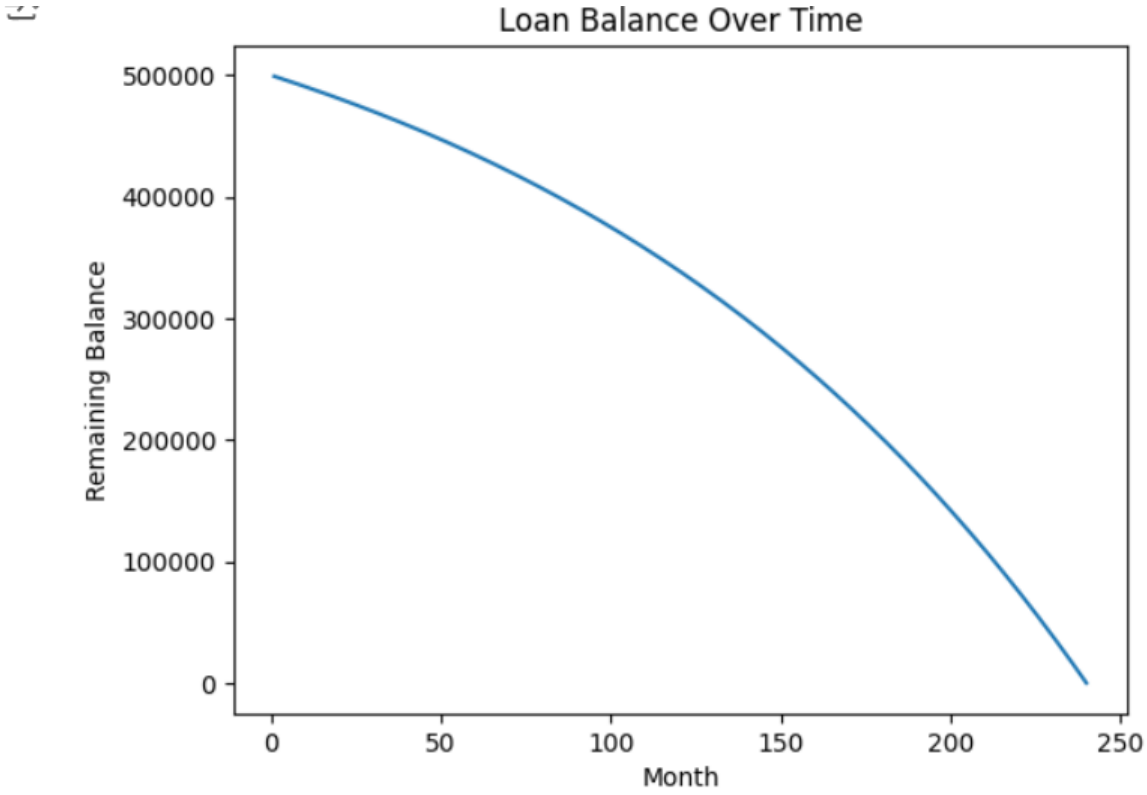
EMI Chart:

I used Python's **matplotlib** library to generate a chart that visualizes the reducing balance over time. The chart shows how the outstanding loan amount decreases month by month as payments are made. This visual representation makes it easier to understand the progression of the loan repayment.

Implementation:

```python
import matplotlib.pyplot as plt

def calculate_emi(principal, annual_rate, months):
    monthly_rate = annual_rate / (12 * 100)
    emi = principal * monthly_rate * (1 + monthly_rate)**months / ((1 + monthly_rate)**months - 1)
    return emi

def generate_emi_chart(principal, annual_rate, months):
    emi = calculate_emi(principal, annual_rate, months)
    balances = []
    total_payment = 0
    for month in range(1, months + 1):
        interest = (principal * annual_rate) / 1200
        total_payment += emi
        principal -= (emi - interest)
        balances.append(principal)

    plt.plot(range(1, months + 1), balances)
    plt.xlabel('Month')
    plt.ylabel('Remaining Balance')
    plt.title('Loan Balance Over Time')
    plt.show()

# Example usage:
principal = 500000  # Loan amount
annual_rate = 7.5   # Annual interest rate
months = 240        # 20 years loan
generate_emi_chart(principal, annual_rate, months)
```

Output:



The output of this is shown above, which illustrates the loan balance over time as the borrower makes regular monthly payments, or EMIs. The graph provides a clear representation of how the outstanding loan amount reduces during the repayment period.

Analysis of the Output:

- The x-axis shows the number of months, which corresponds to the total tenure of the loan.
- The y-axis represents the remaining loan balance at any given point during the loan term.
- The curve starts at the highest point, which is the initial loan amount (principal), and gradually decreases over time. In the beginning, most of the EMI goes towards the interest, so the balance decreases slowly. As the loan progresses, more of the EMI goes towards paying off the principal, leading to a sharper decline in the loan balance.

This graph effectively demonstrates the reducing balance method, where the principal reduces more rapidly as the loan term progresses. The visual helps in understanding how the outstanding balance diminishes with each payment, aligning with the concept of EMI and interest calculation.

This output provides a good way to visualize the repayment schedule and highlights the benefits of the reducing balance approach, where the borrower saves more on interest payments as the principal reduces.