Name - Ameya Lonare section - A
PRN - 21070521008

TITLE - Generative AI CA 2

Q:1 Generate a model in Python for representation of a bank account of type savings and balance along with transactions of deposit and withdrawals and currently create a program to generate 100 accounts with Random balance and transactions for no. of months and no. of transactions with a seed value of amount. Print all 100 accounts with the last balance and organize them by lowest to highest balance.

Solution -

```
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):
        if amount > 0:
            self.balance += amount
            self.transactions.append(f"Deposit: +{amount}")

    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
            self.transactions.append(f"Withdraw: -{amount}")

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

# Function to generate random transactions for a number of months and seed value
def generate_random_accounts(num_accounts, num_months, num_transactions, seed_value):
    random.seed(seed_value)
    accounts = []

    for i in range(num_accounts):
        initial_balance = random.randint(100, 10000)  # Random initial balance between 100 and
10,000
        account = BankAccount(account_id=i, balance=initial_balance)

        for _ in range(num_months):
```

```
        for _ in range(num_transactions):
            transaction_type = random.choice(["deposit", "withdraw"])
            amount = random.randint(10, 1000)  # Random transaction amount between 10 and
1000

            if transaction_type == "deposit":
                account.deposit(amount)
            elif transaction_type == "withdraw":
                account.withdraw(amount)

        accounts.append(account)

    return sorted(accounts, key=lambda x: x.balance)

# Generate 100 accounts with random balance and transactions
num_accounts = 100
num_months = 12
num_transactions = 10
seed_value = 42

accounts = generate_random_accounts(num_accounts, num_months, num_transactions,
seed_value)

# Print the accounts sorted by balance
for account in accounts:
    print(account)
```

Code explanation -
BankAccount Class: Represents a bank account with an account ID, balance, and a list of transactions (deposit or withdrawal).
deposit() and withdraw() Methods: Allow deposits and withdrawals. The balance updates accordingly, and each transaction is logged.
generate_random_accounts() Function: Creates 100 accounts, assigns a random initial balance, and generates a random number of transactions for each account over several months.
Random Transactions: For each account, random deposit or withdrawal transactions are made, simulating real-world banking activity.
Sorting by Balance: The accounts are sorted by final balance (from lowest to highest) and displayed at the end.

Q:2 Generate a model for an Insurance company to hold information on the insurer's vehicle, and create a chart of monthly, yearly, and qtrly premiums based on no. of years of insurance where in each year, the value of the vehicle depreciates by 7%
Solution -

```
import pandas as pd
import matplotlib.pyplot as plt

class InsurancePolicy:
    def __init__(self, policy_id, vehicle_value, base_premium):
        self.policy_id = policy_id
        self.vehicle_value = vehicle_value
        self.base_premium = base_premium
        self.depreciation_rate = 0.07
        self.premium_data = {}

    def calculate_premiums(self, num_years):
        current_value = self.vehicle_value

        for year in range(1, num_years + 1):
            current_value *= (1 - self.depreciation_rate)
            yearly_premium = self.base_premium
            quarterly_premium = yearly_premium / 4
            monthly_premium = yearly_premium / 12

            self.premium_data[year] = {
                'Vehicle Value': round(current_value, 2),
                'Yearly Premium': round(yearly_premium, 2),
                'Quarterly Premium': round(quarterly_premium, 2),
                'Monthly Premium': round(monthly_premium, 2)
            }

    def __repr__(self):
        return f"Policy ID: {self.policy_id}, Vehicle Value: {self.vehicle_value}, Base Premium:
{self.base_premium}"

# Function to generate random insurance policies
def generate_random_policies(num_policies, vehicle_value_range, base_premium_range,
num_years):
    policies = []

    for i in range(num_policies):
        vehicle_value = random.randint(*vehicle_value_range)
        base_premium = random.randint(*base_premium_range)
        policy = InsurancePolicy(policy_id=i, vehicle_value=vehicle_value,
base_premium=base_premium)
        policy.calculate_premiums(num_years)
        policies.append(policy)
```

```
    return policies

# Function to plot premium chart
def plot_premiums(policy):
    df = pd.DataFrame(policy.premium_data).T
    df.plot(kind='bar', figsize=(10, 6))
    plt.title(f"Premium Chart for Policy {policy.policy_id}")
    plt.ylabel("Premium/Vehicle Value")
    plt.xlabel("Year")
    plt.xticks(rotation=0)
    plt.show()

# Generate 100 random insurance policies and print their premium data
num_policies = 100
vehicle_value_range = (5000, 50000)
base_premium_range = (500, 5000)
num_years = 5

policies = generate_random_policies(num_policies, vehicle_value_range,
base_premium_range, num_years)

# Sort policies by final vehicle value
sorted_policies = sorted(policies, key=lambda x: x.premium_data[num_years]['Vehicle Value'])

# Print premium data for sorted policies
for policy in sorted_policies:
    print(policy)
    print(policy.premium_data)

# Example: Plot premiums for the first policy
plot_premiums(sorted_policies[0])
```

Code Explanation -

InsurancePolicy Class: Represents an insurance policy with a vehicle value, a base premium, and a depreciation rate (7% per year).

calculate_premiums() Method: Calculates the premiums (monthly, quarterly, yearly) based on the vehicle value's depreciation over a specified number of years.

generate_random_policies() Function: Creates multiple insurance policies with random vehicle values and base premiums, calculating their premiums over time.

plot_premiums() Function: Plots the vehicle value and premiums (monthly, quarterly, and yearly) for a specific policy over the years.

Sorting and Display: The policies are sorted by the final vehicle value, and the premiums are displayed for each policy, simulating how the vehicle depreciates over time and affects the insurance premiums.