Anay Vyawahare 21070521009

1 Housing Loan Model(Q2)

This project implements a Python-based housing loan model that calculates EMI (Equated Monthly Installment), generates amortization schedules, and visualizes loan repayment details. It also includes functionality to calculate interest lost on early loan closure.

1.1 Features

- Calculate monthly EMI for a given loan amount, interest rate, and loan term
- Generate a complete amortization schedule
- Visualize EMI breakdown with a stacked bar chart
- Calculate interest lost on early loan closure

1.2 Usage

1. Import the `HousingLoan` class from the main script:

```python
from housing_loan_model import HousingLoan

```

2. Create a loan object with your desired parameters:

```python
loan_amount = 300000
annual_interest_rate = 7.5
loan_term_years = 20

loan = HousingLoan(loan_amount, annual_interest_rate,
    loan_term_years)

```

3. Calculate and display the monthly EMI:

```python
print(f"Monthly EMI: ${loan.emi:.2f}")

```

4. Generate and display the EMI breakdown chart:

```python
loan.plot_emi_chart()

```

5. Calculate interest lost on early closure:

```python
early_closure_month = 60  # Closing after 5 years
interest_lost, remaining_months = loan.calculate_interest_lost_on_early_closure(early_closure_month)
print(f"\nInterest lost on early closure after {early_closure_month} months: ${interest_lost:.2f}")
print(f"Interest lost per remaining month: ${interest_lost / remaining_months:.2f}")

```

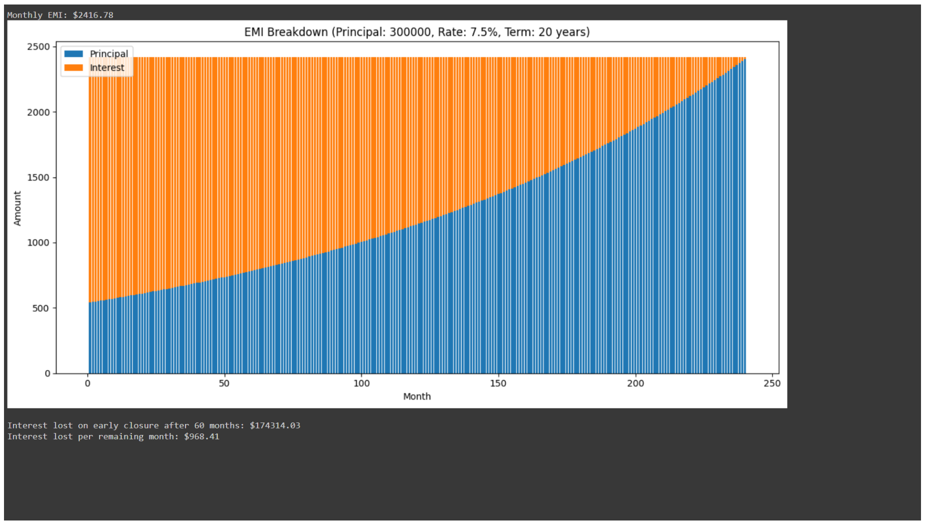
1.3 Code

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

class HousingLoan:
    def __init__(self, principal, annual_interest_rate, loan_term_years):
        self.principal = principal
        self.annual_interest_rate = annual_interest_rate
        self.loan_term_years = loan_term_years
        self.monthly_interest_rate = annual_interest_rate / 12 / 100
        self.total_payments = loan_term_years * 12
        self.emi = self.calculate_emi()

    def calculate_emi(self):
        emi = (self.principal * self.monthly_interest_rate * (1 + self.monthly_interest_rate) ** self.total_payments) / \
              ((1 + self.monthly_interest_rate) ** self.total_payments - 1)
        return emi

    def generate_amortization_schedule(self):
        balance = self.principal
        schedule = []
        for month in range(1, self.total_payments + 1):
            interest = balance * self.monthly_interest_rate
            principal = self.emi - interest
            balance -= principal
            schedule.append((month, self.emi, principal, interest, balance))
        return schedule

    def calculate_interest_lost_on_early_closure(self, closure_month):
        schedule = self.generate_amortization_schedule()
        total_interest = sum(payment[3] for payment in schedule[closure_month:])
        remaining_months = self.total_payments - closure_month
        return total_interest, remaining_months

    def plot_emi_chart(self):
        schedule = self.generate_amortization_schedule()
        months = [payment[0] for payment in schedule]
        principal_payments = [payment[2] for payment in schedule]
        interest_payments = [payment[3] for payment in schedule]

        plt.figure(figsize=(12, 6))
        plt.bar(months, principal_payments, label='Principal')
        plt.bar(months, interest_payments, bottom=principal_payments, label='Interest')
        plt.xlabel('Month')
        plt.ylabel('Amount')
        plt.title(f'EMI Breakdown (Principal: {self.principal}, Rate: {self.annual_interest_rate}%, Term: {self.loan_term_years} years)')
        plt.legend()
        plt.tight_layout()
        plt.show()

# Example usage
loan_amount = 300000
annual_interest_rate = 7.5
loan_term_years = 20

loan = HousingLoan(loan_amount, annual_interest_rate, loan_term_years)
print(f"Monthly EMI: ${loan.emi:.2f}")

# Plot EMI chart
loan.plot_emi_chart()

# Calculate interest lost on early closure
early_closure_month = 60  # Closing after 5 years
interest_lost, remaining_months = loan.calculate_interest_lost_on_early_closure(early_closure_month)
print(f"\nInterest lost on early closure after {early_closure_month} months: ${interest_lost:.2f}")
print(f"Interest lost per remaining month: ${interest_lost / remaining_months:.2f}")
```

1.4 Example Output

When you run the script, you'll see:

1. The calculated monthly EMI printed to the console.
2. A stacked bar chart showing the EMI breakdown over the loan term.
3. The total interest lost if closing the loan early and the interest lost per remaining month.



```
Monthly EMI: $2416.78
```

```
Interest lost on early closure after 60 months: $174314.03
Interest lost per remaining month: $968.41
```

Figure 1: Example output of the Housing Loan Model

1.5 Customization

You can easily modify the loan parameters in the script to model different scenarios:

- Change the `loan_amount` to model different loan principals.
- Adjust the `annual_interest_rate` to see how interest rates affect the EMI and total interest paid.
- Modify the `loan_term_years` to model shorter or longer loan terms.
- Change the `early_closure_month` to calculate interest lost for different early closure scenarios.

2 Insurance Company Vehicle Model(Q3)

This project implements a Python-based model for an insurance company to manage information on insured vehicles. It calculates monthly, yearly, and quarterly premiums based on the number of years of insurance, taking into account an annual vehicle depreciation.

2.1 Features

- Store and manage vehicle information (make, model, year, initial value)
- Calculate vehicle depreciation over time

- Compute premiums on monthly, quarterly, and yearly bases
- Generate a comprehensive premium chart using pandas
- Visualize premium changes and vehicle value depreciation over time

2.2 Usage

1. Import the necessary classes and functions:

```python
from insurance_model import Vehicle, InsurancePolicy,
    generate_premium_chart, plot_premium_chart
from datetime import datetime

```

2. Create a Vehicle object:

```python
vehicle = Vehicle("Toyota", "Camry", 2022, 25000)

```

3. Create an InsurancePolicy object:

```python
policy = InsurancePolicy(vehicle, datetime(2022, 1, 1))

```

4. Generate the premium chart:

```python
premium_chart = generate_premium_chart(policy)

```

5. Display the premium chart:

```python
print("Premium Chart:")
print(premium_chart.to_string(index=False))

```

6. Visualize the premiums and vehicle value over time:

```python
plot_premium_chart(premium_chart)

```

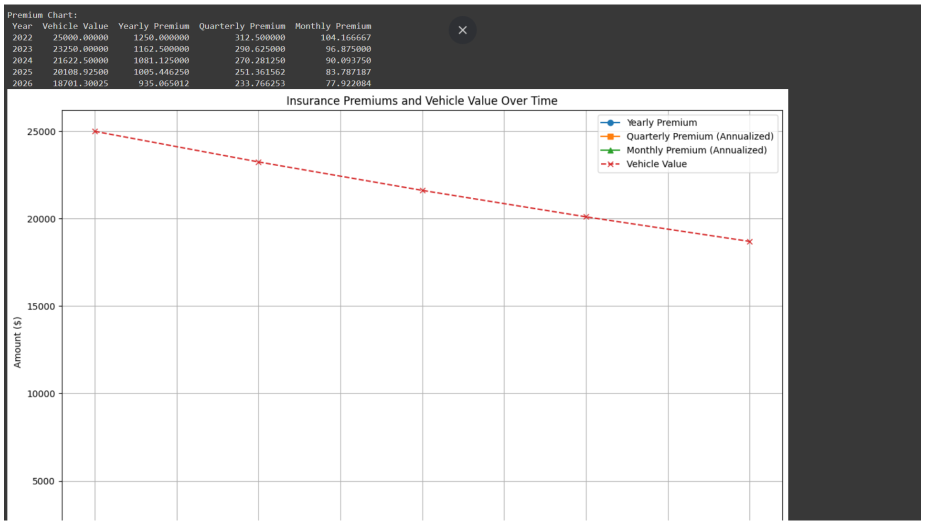
2.3 Code

```python
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime

class Vehicle:
    def __init__(self, make, model, year, initial_value):
        self.make = make
        self.model = model
        self.year = year
        self.initial_value = initial_value

```

```python
class InsurancePolicy:
    def __init__(self, vehicle, start_date, base_premium_rate=0.05, depreciation_rate=0.07):
        self.vehicle = vehicle
        self.start_date = start_date
        self.base_premium_rate = base_premium_rate
        self.depreciation_rate = depreciation_rate

    def calculate_current_value(self, current_year):
        years_passed = current_year - self.start_date.year
        depreciation_factor = (1 - self.depreciation_rate) ** years_passed
        return self.vehicle.initial_value * depreciation_factor

    def calculate_premium(self, current_year, frequency='yearly'):
        current_value = self.calculate_current_value(current_year)
        yearly_premium = current_value * self.base_premium_rate

        if frequency == 'yearly':
            return yearly_premium
        elif frequency == 'quarterly':
            return yearly_premium / 4
        elif frequency == 'monthly':
            return yearly_premium / 12
        else:
            raise ValueError("Invalid frequency. Choose 'yearly', 'quarterly', or 'monthly'.")

def generate_premium_chart(policy, years=5):
    data = []
    current_year = policy.start_date.year

    for year in range(years):
        yearly_premium = policy.calculate_premium(current_year + year, 'yearly')
        quarterly_premium = policy.calculate_premium(current_year + year, 'quarterly')
        monthly_premium = policy.calculate_premium(current_year + year, 'monthly')
        vehicle_value = policy.calculate_current_value(current_year + year)

        data.append({
            'Year': current_year + year,
            'Vehicle Value': vehicle_value,
            'Yearly Premium': yearly_premium,
            'Quarterly Premium': quarterly_premium,
            'Monthly Premium': monthly_premium
        })

    return pd.DataFrame(data)

def plot_premium_chart(premium_chart):
    plt.figure(figsize=(12, 8))

    plt.plot(premium_chart['Year'], premium_chart['Yearly Premium'], label='Yearly Premium', marker='o')
    plt.plot(premium_chart['Year'], premium_chart['Quarterly Premium'] * 4, label='Quarterly Premium (Annualized)', marker='s')
    plt.plot(premium_chart['Year'], premium_chart['Monthly Premium'] * 12, label='Monthly Premium (Annualized)', marker='^')
    plt.plot(premium_chart['Year'], premium_chart['Vehicle Value'], label='Vehicle Value', linestyle='--', marker='x')

    plt.title('Insurance Premiums and Vehicle Value Over Time')
    plt.xlabel('Year')
    plt.ylabel('Amount ($)')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()

# Example usage
vehicle = Vehicle("Toyota", "Camry", 2022, 25000)
policy = InsurancePolicy(vehicle, datetime(2022, 1, 1))
premium_chart = generate_premium_chart(policy)

print("Premium Chart:")
print(premium_chart.to_string(index=False))

plot_premium_chart(premium_chart)
```

2.4 Example Output

When you run the script, you'll see:



```
Premium Chart:
 Year  Vehicle Value  Yearly Premium  Quarterly Premium  Monthly Premium
 2022    25000.00000     1250.000000         312.500000       104.166667
 2023    23250.00000     1162.500000         290.625000        96.875000
 2024    21622.50000     1081.125000         270.281250        90.093750
 2025    20108.92500     1005.446250         251.361562        83.787187
 2026    18701.30025      935.065012         233.766253        77.922084
```

Insurance Premiums and Vehicle Value Over Time
Amount ($)
25000
20000
15000
10000
5000
Yearly Premium
Quarterly Premium (Annualized)
Monthly Premium (Annualized)
Vehicle Value

Figure 2: Example output of the Insurance Company Vehicle Model

1. A pandas DataFrame printed to the console, showing:
    - Year
    - Vehicle Value
    - Yearly Premium
    - Quarterly Premium
    - Monthly Premium
2. A chart displaying:
    - Yearly Premium
    - Quarterly Premium (Annualized)
    - Monthly Premium (Annualized)
    - Vehicle Value

The chart will show how these values change over the insurance period, accounting for vehicle depreciation.

2.5 Customization

You can easily modify the model parameters to simulate different scenarios:

- Change the vehicle details in the `Vehicle` constructor.
- Adjust the policy start date in the `InsurancePolicy` constructor.
- Modify the `base_premium_rate` and `depreciation_rate` in the `InsurancePolicy` constructor.
- Change the number of years in the `generate_premium_chart` function call.