Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

S.Y.B.Tech. **Sem: IV**

Subject: Computational Methods and Pricing Models Laboratory

Experiment 2

Name: Smayan Kulkarni **SAP ID: 60009230142**

Date:	**Experiment Title:** Mortgage Payment Analysis with Extra Contributions and Investment Growth using Rule of 72
Aim	To analyze the impact of extra principal payments on mortgage tenure and interest savings, generate an amortization schedule, visualize loan balance reduction, and conduct a parametric study on different extra payment strategies. Additionally, apply the Rule of 72 to estimate the time required for an investment to double under different interest rate models (simple and compound interest) and compounding frequencies.
Software	Python on Google Colab

Theory	A mortgage is a loan secured by real estate, where borrowers make regular payments comprising principal and interest. The key factors influencing mortgage repayment include: • **Principal**: The amount borrowed from the lender. • **Interest Rate**: The cost of borrowing money, expressed as an annual percentage. • **Loan Tenure**: The total repayment period, typically measured in months. • **Monthly Payment (EMI)**: A fixed payment consisting of principal and interest, calculated using the mortgage payment formula: $EMI = \frac{P \times r \times (1+r)^n}{(1+r)^n - 1}$ Where: • P = Loan Amount • r = Monthly Interest Rate (Annual Rate / 12 / 100) • n = Total Number of Payments (Term in Months) The principal and interest components change over time, with a higher portion of the early payments going toward interest. The outstanding loan balance reduces over time as principal payments increase. Impact of Extra Principal Payments Borrowers can make additional payments towards the principal to reduce the loan tenure and total interest paid. The benefits include: • **Shorter Loan Duration**: Extra payments directly reduce the outstanding balance, leading to fewer months required to repay the loan. • **Interest Savings**: Since interest is calculated on the remaining balance, paying extra reduces the total interest expense. • **Flexibility in Repayment**: Borrowers can choose to make fixed extra payments every month or occasional lump-sum payments. The formula for the updated balance after an extra payment is: Where: • **B_m** = Outstanding loan balance after month m • **EMI** = Regular monthly payment • **Interest** = Monthly interest charge • **ExtraPayment** = Additional amount paid towards principal The **Rule of 72:** The Rule of 72 is a quick, useful formula that is popularly used to

estimate the number of years required to double the invested money at a given annual rate of return. Alternatively, it can compute the annual rate of compounded return from an investment, given how many years it will take to double the investment.

While calculators and spreadsheet programs like Microsoft Excel have functions to accurately calculate the precise time required to double the invested money, the Rule of 72 comes in handy for mental calculations to quickly gauge an approximate value.

Formula:

where T is the approximate doubling time in years and r is the annual interest rate (as a percentage).

To compare the results with the actual doubling time using the standard formulas for

- **Simple Interest:**

$$A = P(1 + rt)$$

- **Compound Interest:**

$$A = P(1 + \frac{r}{n})^{nt}$$

where n is the compounding frequency.

Implementation

Part 1: Mortgage Payment Analysis with Extra Contributions

Step 1: Calculate EMI for a given loan

- Define the following loan parameters:
  - Loan Amount = Rs. 50 lakhs
  - Interest Rate = 8% annually
  - Term = 20 years (240 months)
- Compute EMI using the mortgage payment formula.

Step 2: Generate the amortization schedule with extra payments

- Compute interest and principal breakdown for each month.
- Deduct an additional principal payment of Rs. 5000 per month.
- Adjust the remaining loan balance iteratively.
- Display the first and last 10 rows of the amortization schedule.

Step 3: Visualize loan balance reduction

- Plot a line graph showing loan balance vs. months.
- Compare scenarios with and without extra payments.

Step 4: Conduct a parametric study

- Vary the extra payment amount (e.g., Rs. 3000, Rs. 5000, Rs. 10000 per month).
- Analyze and visualize the effects on:
  - Loan tenure reduction
  - Total interest savings
  - Total amount paid

Step 5: Compare Lump-Sum vs. Regular Extra Payments

- Compare the effect of a one-time lump-sum payment (e.g., Rs. 2 lakhs) versus monthly extra payments.
- Determine which strategy provides greater interest savings and a faster loan payoff.

Use different graphs to illustrate the findings and discuss conclusions.

Part 2: Investment Growth Analysis using the Rule of 72

Step 6: Compute Doubling Time Using Simple and Compound Interest

- Implement the **Rule of 72** formula
- Compare the results with the actual doubling time using the standard formulas for: Simple Interest and Compound Interest

Step 7: Compute Doubling Period for Different Compounding Frequencies

- Calculate the actual time required for an investment to double when compounded:
  - Annually

	◦ Semi-annually ◦ Quarterly ◦ Monthly • Compare theoretical Rule of 72 estimates with exact values. Step 8: Visualize Investment Growth • Plot investment growth curves for different compounding frequencies. • Compare simple vs. compound interest using a line graph.

Conclusion	Additional principal payments dramatically lower mortgage tenure and interest expense, as evident from the amortization schedule. Visualization of loan balance verifies the payoff acceleration effect. The Rule of 72 offers a rough estimate of investment doubling time in different interest models and compounding frequencies.

kchxknixi

February 21, 2025

Google Colab Link : https://colab.research.google.com/github/SmayanKulkarni/AI-and-ML-Course/blob/master/D100%20CMPM/exp-2.ipynb

[2]:
```
def calc_mor_emi(p,r,n):
    mor_emi= (amt * r * (1+r)**n) / ((1+r)**n - 1)
    return mor_emi
```

[3]:
```
def calc_interest(bal, r):
    interest = bal * r
    return interest
```

[4]:
```
def calc_principal(emi, interest):
    princ = emi - interest
    return princ
```

[5]:
```
def calc_update_bal(bal,p,ep):
    newbal = bal - (p + ep)
    return newbal
```

1 Part 1

[6]:
```
amt = 5000000
r = 0.08 / 12
n = 240
emi = calc_mor_emi(amt, r,n)
print("The EMI for the details is: ", emi)
```

```
The EMI for the details is:  41822.00344967332
```

2 Part 2

[7]:
```
ep = 5000
bal = 5000000
intrest_breakdown = []
principal_breakdown = []
bal_breakdown = []
for i in range(n):
```

```
    temp_int = calc_interest(bal,r)
    temp_princ= calc_principal(emi, temp_int)
    bal = calc_update_bal(bal, temp_princ,ep)
    intrest_breakdown.append(temp_int)
    principal_breakdown.append(amt)
    if(bal<0):
        bal_breakdown.append(0)
        break
    bal_breakdown.append(bal)
    amt = amt - temp_princ
```

[8]:
```
bal_breakdown[:10]
```

[8]:
```
[4986511.32988366,
 4972932.735299878,
 4959263.616752204,
 4945503.370747545,
 4931651.389769522,
 4917707.062251645,
 4903669.7725503165,
 4889538.900917646,
 4875313.82347409,
 4860993.91218091]
```

[9]:
```
len(bal_breakdown)
```

[9]: 188

[10]:
```
import pandas as pd
import numpy as np
```

[11]:
```
df = pd.DataFrame()
```

[12]:
```
df['Balances'] = np.array(bal_breakdown)
```

[13]:
```
df['Intrest'] = np.array(intrest_breakdown)
```

[14]:
```
df['Princial'] = np.array(principal_breakdown)
```

[15]:
```
df.head(10)
```

[15]:

	Balances	Intrest	Princial
0	4.986511e+06	33333.333333	5.000000e+06
1	4.972933e+06	33243.408866	4.991511e+06
2	4.959264e+06	33152.884902	4.982933e+06
3	4.945503e+06	33061.757445	4.974264e+06
4	4.931651e+06	32970.022472	4.965503e+06

```
5  4.917707e+06  32877.675932  4.956651e+06
6  4.903670e+06  32784.713748  4.947707e+06
7  4.889539e+06  32691.131817  4.938670e+06
8  4.875314e+06  32596.926006  4.929539e+06
9  4.860994e+06  32502.092156  4.920314e+06
```

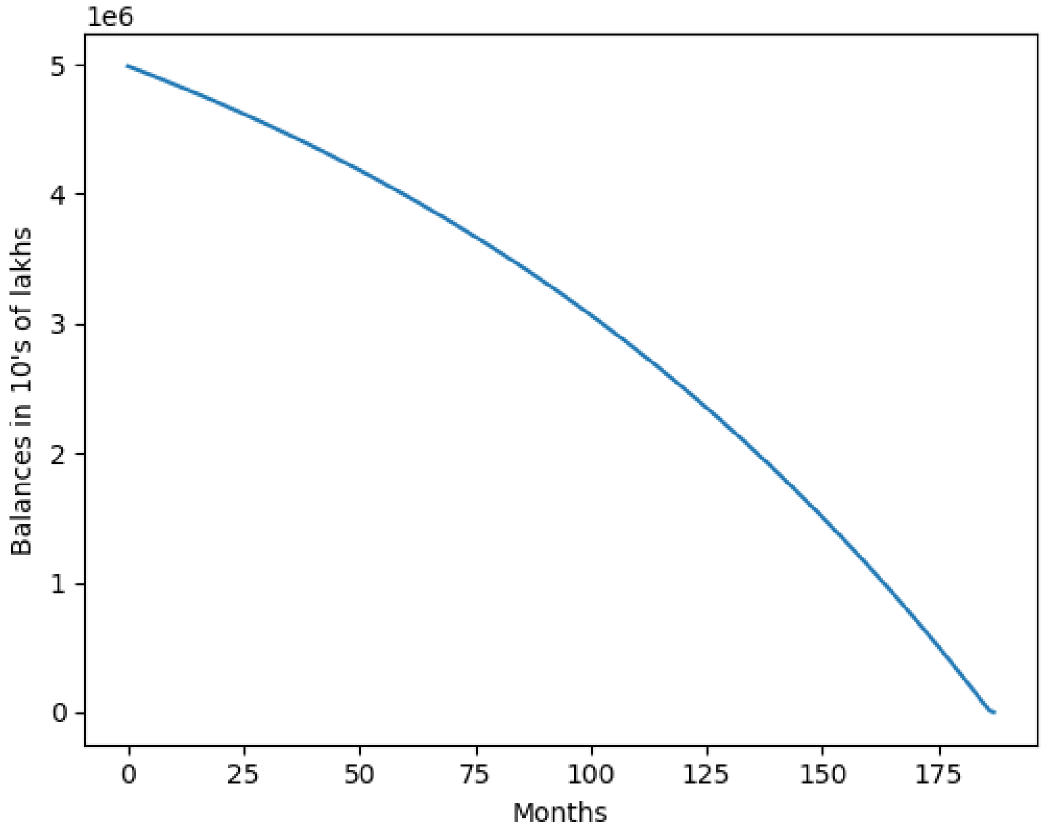
[16]:
```python
df.tail(10)
```

[16]:
```
          Balances      Intrest      Princial
178  376715.150153  2804.881812  1.310732e+06
179  332404.581037  2511.434334  1.271715e+06
180  287798.608128  2216.030540  1.232405e+06
181  242895.262066  1918.657388  1.192799e+06
182  197692.560363  1619.301747  1.152895e+06
183  152188.507316  1317.950402  1.112693e+06
184  106381.093915  1014.590049  1.072189e+06
185   60268.297758   709.207293  1.031381e+06
186   13848.082960   401.788652  9.902683e+05
187       0.000000    92.320553  9.488481e+05
```

[17]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
```

[18]:
```python
sns.lineplot(df['Balances'])
plt.xlabel("Months")
plt.ylabel("Balances in 10's of lakhs")
```

[18]:
```
Text(0, 0.5, "Balances in 10's of lakhs")
```



[19]:
```
amt = 5000000
r = 0.08 / 12
n = 240
emi = calc_mor_emi(amt, r,n)
ep = 0
bal = 5000000
intrest_breakdown2 = []
principal_breakdown2 = []
bal_breakdown2 = []
for i in range(n):
    temp_int = calc_interest(bal,r)
    temp_princ= calc_principal(emi, temp_int)
    bal = calc_update_bal(bal, temp_princ,ep)
    intrest_breakdown2.append(temp_int)
    principal_breakdown2.append(amt)
    if(bal<0):
        bal_breakdown2.append(0)
        break
    bal_breakdown2.append(bal)
    amt = amt - temp_princ
```

[20]:
```
df2 = pd.DataFrame()
df2['Balances'] = np.array(bal_breakdown2)
df2['Intrest'] = np.array(intrest_breakdown2)
df2['Princial'] = np.array(principal_breakdown2)
```

[21]:
```
df2.head(10)
```

[21]:

	Balances	Intrest	Princial
0	4.991511e+06	33333.333333	5.000000e+06
1	4.982966e+06	33276.742199	4.991511e+06
2	4.974364e+06	33219.773791	4.982966e+06
3	4.965704e+06	33162.425593	4.974364e+06
4	4.956987e+06	33104.695074	4.965704e+06
5	4.948212e+06	33046.579685	4.956987e+06
6	4.939378e+06	32988.076860	4.948212e+06
7	4.930485e+06	32929.184016	4.939378e+06
8	4.921533e+06	32869.898553	4.930485e+06
9	4.912521e+06	32810.217854	4.921533e+06

[22]:
```
df2.tail(10)
```

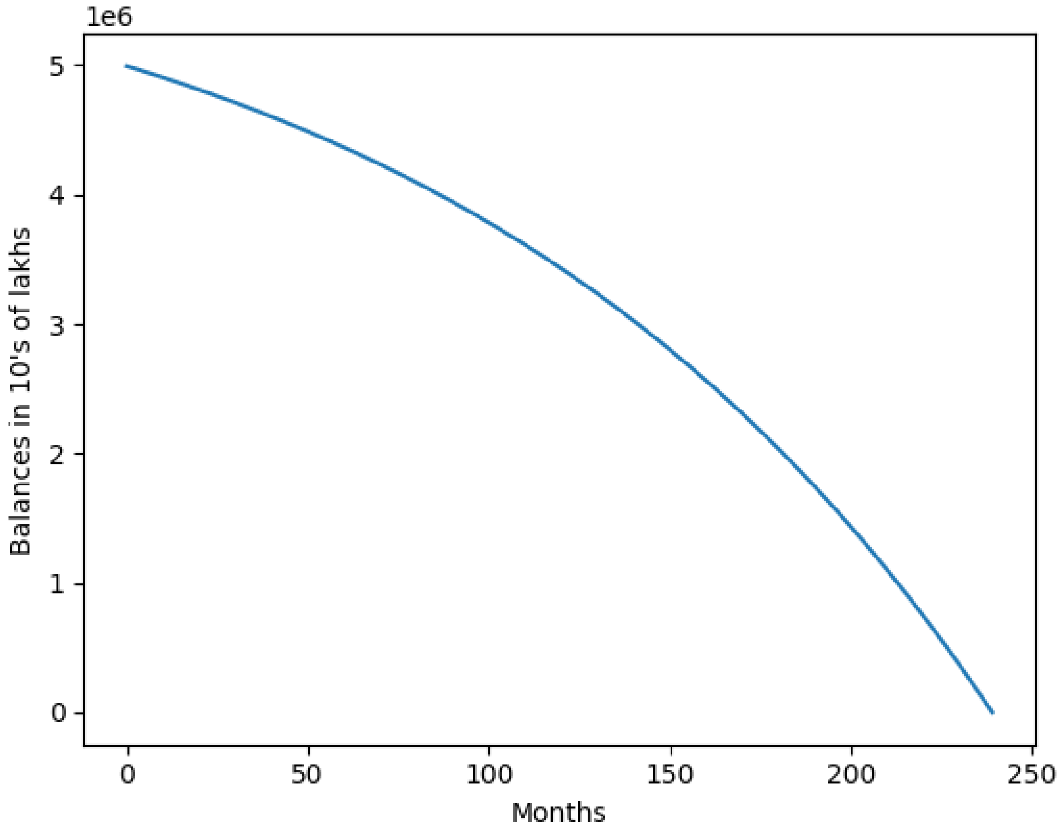
[22]:

	Balances	Intrest	Princial
230	364152.095500	2688.570192	403285.528758
231	324757.772687	2427.680637	364152.095500
232	285100.821056	2165.051818	324757.772687
233	245179.489746	1900.672140	285100.821056
234	204992.016228	1634.529932	245179.489746
235	164536.626220	1366.613442	204992.016228
236	123811.533612	1096.910841	164536.626220
237	82814.940386	825.410224	123811.533612
238	41545.036539	552.099603	82814.940386
239	0.000000	276.966910	41545.036539

3 Part 3

[23]:
```
sns.lineplot(df2['Balances'])
plt.xlabel("Months")
plt.ylabel("Balances in 10's of lakhs")
```

[23]: Text(0, 0.5, "Balances in 10's of lakhs")



It is seen here that with extra payment the balance is cleared out in 188 months but without th extra paymnets it takes 238 months to clear out the balance.

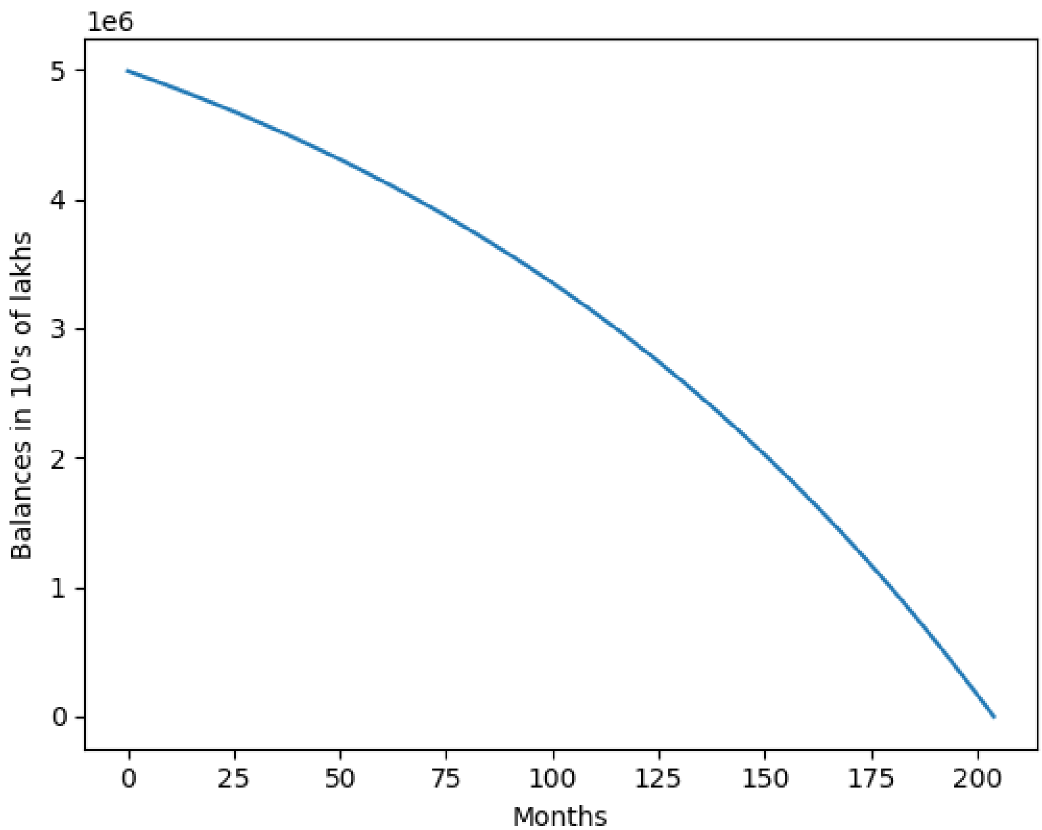
4 Part 4

[24]:
```
amt = 5000000
r = 0.08 / 12
n = 240
emi = calc_mor_emi(amt, r,n)
ep = 3000
bal = 5000000
intrest_breakdown2 = []
principal_breakdown2 = []
bal_breakdown2 = []
for i in range(n):
    temp_int = calc_interest(bal,r)
    temp_princ= calc_principal(emi, temp_int)
    bal = calc_update_bal(bal, temp_princ,ep)
    intrest_breakdown2.append(temp_int)
```

```
    principal_breakdown2.append(amt)
    if(bal<0):
        bal_breakdown2.append(0)
        break
    bal_breakdown2.append(bal)
    amt = amt - temp_princ
df2 = pd.DataFrame()
df2['Balances'] = np.array(bal_breakdown2)
df2['Intrest'] = np.array(intrest_breakdown2)
df2['Princial'] = np.array(principal_breakdown2)
print(df2.head(10))
print(df2.tail(10))
# Part 3
sns.lineplot(df2['Balances'])
plt.xlabel("Months")
plt.ylabel("Balances in 10's of lakhs")
```

```
       Balances        Intrest      Princial
0  4.988511e+06  33333.333333  5.000000e+06
1  4.976946e+06  33256.742199  4.991511e+06
2  4.965304e+06  33179.640458  4.982946e+06
3  4.953584e+06  33102.024704  4.974304e+06
4  4.941786e+06  33023.891513  4.965584e+06
5  4.929909e+06  32945.237433  4.956786e+06
6  4.917953e+06  32866.058993  4.947909e+06
7  4.905917e+06  32786.352697  4.938953e+06
8  4.893801e+06  32706.115025  4.929917e+06
9  4.881605e+06  32625.342435  4.920801e+06
          Balances      Intrest      Princial
195  385246.839668  2848.138034  1.012221e+06
196  342993.148482  2568.312264  9.732468e+05
197  300457.766023  2286.620990  9.339931e+05
198  257638.814346  2003.051773  8.944578e+05
199  214534.402992  1717.592096  8.546388e+05
200  171142.628896  1430.229353  8.145344e+05
201  127461.576306  1140.950859  7.741426e+05
202   83489.316698   849.743842  7.334616e+05
203   39223.908693   556.595445  6.924893e+05
204       0.000000   261.492725  6.512239e+05
```

[24]:
```
Text(0, 0.5, "Balances in 10's of lakhs")
```

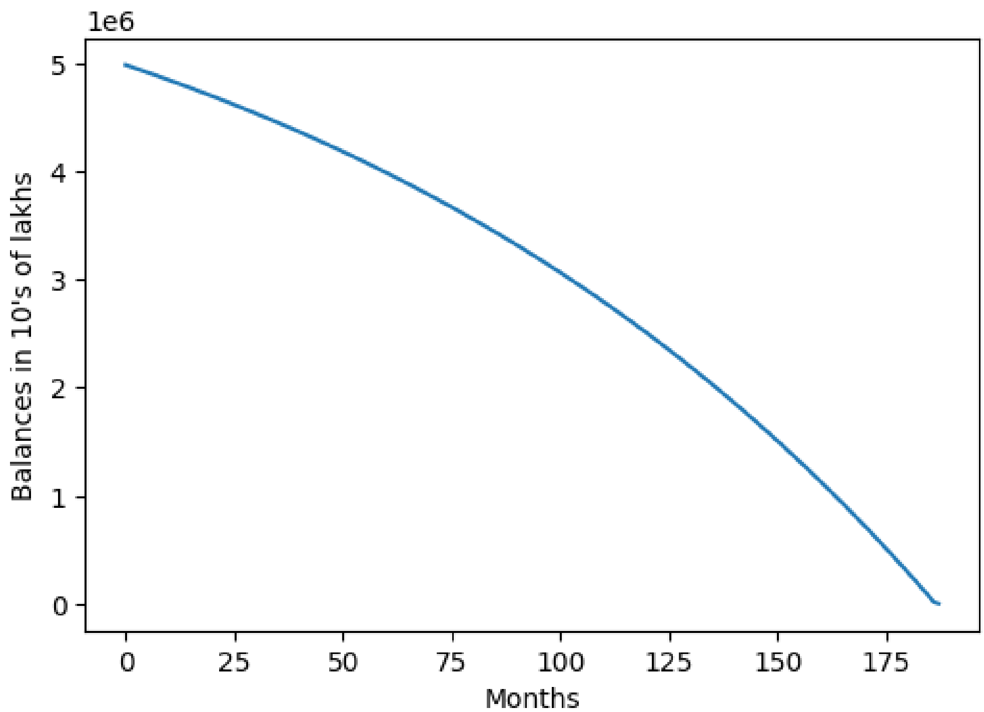


[25]:
```
amt = 5000000
r = 0.08 / 12
n = 240
emi = calc_mor_emi(amt, r,n)
ep = 5000
bal = 5000000
intrest_breakdown2 = []
principal_breakdown2 = []
bal_breakdown2 = []
for i in range(n):
    temp_int = calc_interest(bal,r)
    temp_princ= calc_principal(emi, temp_int)
    bal = calc_update_bal(bal, temp_princ,ep)
    intrest_breakdown2.append(temp_int)
    principal_breakdown2.append(amt)
    if(bal<0):
        bal_breakdown2.append(0)
        break
    bal_breakdown2.append(bal)
    amt = amt - temp_princ
```

```
df2 = pd.DataFrame()
df2['Balances'] = np.array(bal_breakdown2)
df2['Intrest'] = np.array(intrest_breakdown2)
df2['Princial'] = np.array(principal_breakdown2)
print(df2.head(10))
print(df2.tail(10))
# Part 3
plt.figure(figsize=(6,4))
sns.lineplot(df2['Balances'])
plt.xlabel("Months")
plt.ylabel("Balances in 10's of lakhs")
```

```
       Balances       Intrest      Princial
0  4.986511e+06  33333.333333  5.000000e+06
1  4.972933e+06  33243.408866  4.991511e+06
2  4.959264e+06  33152.884902  4.982933e+06
3  4.945503e+06  33061.757445  4.974264e+06
4  4.931651e+06  32970.022472  4.965503e+06
5  4.917707e+06  32877.675932  4.956651e+06
6  4.903670e+06  32784.713748  4.947707e+06
7  4.889539e+06  32691.131817  4.938670e+06
8  4.875314e+06  32596.926006  4.929539e+06
9  4.860994e+06  32502.092156  4.920314e+06
          Balances      Intrest      Princial
178  376715.150153  2804.881812  1.310732e+06
179  332404.581037  2511.434334  1.271715e+06
180  287798.608128  2216.030540  1.232405e+06
181  242895.262066  1918.657388  1.192799e+06
182  197692.560363  1619.301747  1.152895e+06
183  152188.507316  1317.950402  1.112693e+06
184  106381.093915  1014.590049  1.072189e+06
185   60268.297758   709.207293  1.031381e+06
186   13848.082960   401.788652  9.902683e+05
187       0.000000    92.320553  9.488481e+05
```

[25]: Text(0, 0.5, "Balances in 10's of lakhs")

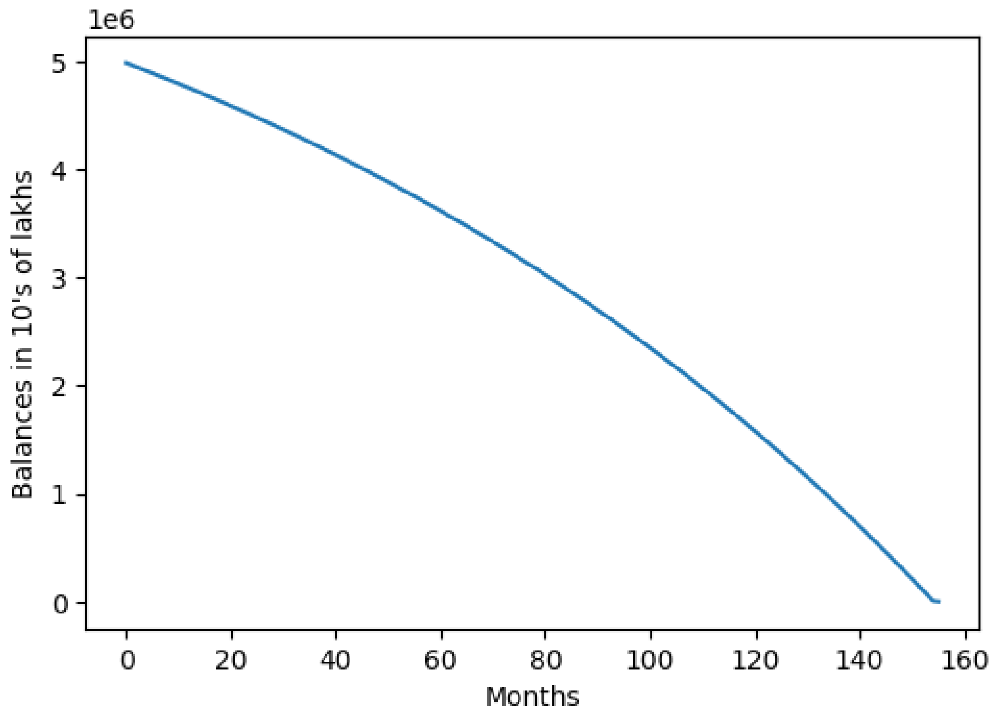


5 Part 5

[26]:
```
amt = 5000000
r = 0.08 / 12
n = 240
emi = calc_mor_emi(amt, r,n)
ep = 10000
bal = 5000000
intrest_breakdown2 = []
principal_breakdown2 = []
bal_breakdown2 = []
for i in range(n):
    temp_int = calc_interest(bal,r)
    temp_princ= calc_principal(emi, temp_int)
    bal = calc_update_bal(bal, temp_princ,ep)
    intrest_breakdown2.append(temp_int)
    principal_breakdown2.append(amt)
    if(bal<0):
        bal_breakdown2.append(0)
        break
    bal_breakdown2.append(bal)
    amt = amt - temp_princ
df2 = pd.DataFrame()
```

```python
df2['Balances'] = np.array(bal_breakdown2)
df2['Intrest'] = np.array(intrest_breakdown2)
df2['Princial'] = np.array(principal_breakdown2)

# Part 3
plt.figure(figsize=(6,4))
sns.lineplot(df2['Balances'])
plt.xlabel("Months")
plt.ylabel("Balances in 10's of lakhs")
```

[26]: Text(0, 0.5, "Balances in 10's of lakhs")



[27]:
```python
df2.head(10)
```

[27]:

	Balances	Intrest	Princial
0	4.981511e+06	33333.333333	5.000000e+06
1	4.962899e+06	33210.075533	4.991511e+06
2	4.944163e+06	33085.996013	4.982899e+06
3	4.925302e+06	32961.089297	4.974163e+06
4	4.906316e+06	32835.349869	4.965302e+06
5	4.887203e+06	32708.772179	4.956316e+06
6	4.867962e+06	32581.350637	4.947203e+06
7	4.848593e+06	32453.079618	4.937962e+06

```
8  4.829095e+06  32323.953459  4.928593e+06
9  4.809467e+06  32193.966459  4.919095e+06
```

[28]:
```
df2.tail(10)
```

[28]:

	Balances	Intrest	Princial
146	407955.362081	3044.883215	1.916732e+06
147	358853.061045	2719.702414	1.877955e+06
148	309423.411336	2392.353740	1.838853e+06
149	259664.230628	2062.822742	1.799423e+06
150	209573.322050	1731.094871	1.759664e+06
151	159148.474080	1397.155480	1.719573e+06
152	108387.460458	1060.989827	1.679148e+06
153	57288.040078	722.583070	1.638387e+06
154	5847.956895	381.920267	1.597288e+06
155	0.000000	38.986379	1.555848e+06

6 Part 6

[29]:
```
def rule_of_72(rate):
    return 72 / rate
r = 6
t = rule_of_72(r)
print("Time to double (in years) =",t)
```

```
Time to double (in years) = 12.0
```

[30]:
```
p = 50000
r = 6/100
si = p
t = (si)/(p*r)
print("Time to double (in years) using SI =",t)
```

```
Time to double (in years) using SI = 16.666666666666668
```

7 Part 7

[31]:
```
import math
p = 50000
r = 6/100
a = 2*p
t1 = math.log(a/p)/math.log(1+r)
print("Annually :")
print("Time to double (in years) using CI =",t1)
```

```
Annually :
Time to double (in years) using CI = 11.895661045941875
```

[32]:
```
p = 50000
r = 6/100
a = 2*p
t2 = math.log(a/p)/(2*math.log(1+(r/2)))
print("Semi-annualy:")
print("Time to double (in years) using CI =",t2)
```

```
Semi-annualy:
Time to double (in years) using CI = 11.724886125218868
```

[33]:
```
p = 50000
r = 6/100
a = 2*p
t3 = math.log(a/p)/(4*math.log(1+(r/4)))
print("Quarterly:")
print("Time to double (in years) using CI =",t3)
```

```
Quarterly:
Time to double (in years) using CI = 11.638881407701545
```

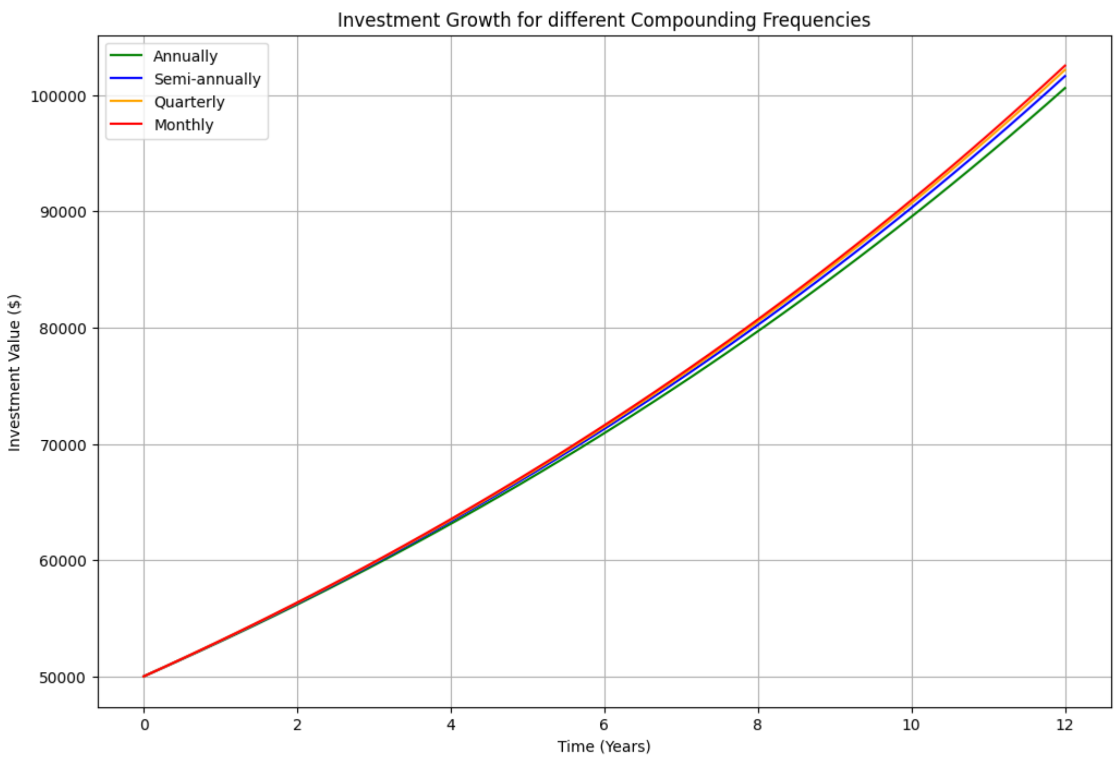
[34]:
```
p = 50000
r = 6/100
a = 2*p
t4 = math.log(a/p)/(12*math.log(1+(r/12)))
print("Monthly:")
print("Time to double (in years) using CI =",t4)
```

```
Monthly:
Time to double (in years) using CI = 11.581310134224728
```

8 Part 8

[35]:
```
t = np.linspace(0, 12)

P = 50000
r = 0.06
n_a = 1
n_s = 2
n_q = 4
n_m = 12

compound_annual = P * (1 + r / n_a) ** (n_a * t)
compound_semi_annual = P * (1 + r / n_s) ** (n_s * t)
compound_quarterly = P * (1 + r / n_q) ** (n_q * t)
```

```python
compound_monthly = P * (1 + r / n_m) ** (n_m * t)

plt.figure(figsize=(12, 8))
plt.plot(t, compound_annual, label="Annually", color='green')
plt.plot(t, compound_semi_annual, label="Semi-annually", color='blue')
plt.plot(t, compound_quarterly, label="Quarterly", color='orange')
plt.plot(t, compound_monthly, label="Monthly", color='red')

plt.title("Investment Growth for different Compounding Frequencies")
plt.xlabel("Time (Years)")
plt.ylabel("Investment Value ($)")
plt.legend()
plt.grid(True)
plt.show()
```



[36]:
```python
P = 50000
r = 0.06

simple_interest = P * (1 + (r * t))

compound_interest = P * ((1 + r) ** t)

plt.figure(figsize=(12,8))
```

```python
plt.plot(t, simple_interest, label="Simple Interest", color='blue', linestyle='--')
plt.plot(t, compound_interest, label="Compound Interest (Annually)", color='green')

plt.title("Simple Interest vs Compound Interest Over Time")
plt.xlabel("Time (Years)")
plt.ylabel("Investment Value ($)")
plt.legend()
plt.grid(True)
plt.show()
```