Homework 3

February 4, 2020

1 Customer Lifetime Value - Simple & General Retention Models

1.1 Submitted by: Kristyan Dimitrov, Srividya Ganapathi, Shreyashi Ganguly, Greesham Simon, Joe Zhang

2/3/2020

1.1.1 Problem 1

a)

[1]:
```
import pandas as pd
```

[30]:
```
m = [10, 15, 20]
r=[.51, .595, .68]
d = .12
```

[31]:
```
def CLV(m, r, d): # Using Simple Retention model with constant retention rate = r
    return m*r/(1+d-r) # Assuming first payment comes at end of year 1
```

[32]:
```
CLV(m[0],r[0],d) # Pessimistic
```

[32]: 8.360655737704915

[33]:
```
CLV(m[1],r[1],d) # Average
```

[33]: 16.999999999999993

[34]:
```
CLV(m[2],r[2],d) # Optimistic
```

[34]: 30.909090909090907

b) Given that the acquisition costs are between 30 and 55, and the CLV of a customer in the optimistic scenario barely exceed 30, it would appear the acquisition strategy is not profitable (under the Simple Retention Model)

1.1.2 Problem 2

[35]:
```
n = 3_290_000
cost = 117_000_000
```

[36]:
```
'${:,.2f}'.format(CLV(m[1],r[1],d) * n) # Customer Equity under Average␣
↪assumptions
```

[36]: '$55,930,000.00'

[37]:
```
'${:,.2f}'.format(CLV(m[1],r[2],d) * n) # Customer Equity under Optimistic␣
↪assumptions
```

[37]: '$76,268,181.82'

[38]:
```
'${:,.2f}'.format(CLV(m[1],.8,d) * n) # Customer Equity under Optimistic␣
↪assumptions
```

[38]: '$123,375,000.00'

[26]:
```
from sympy.solvers import solve
from sympy import Symbol
```

[40]:
```
r = Symbol('r')
solve(CLV(m[1], r, d)*n - cost, r) # The below is the minimum retention rate␣
↪that Bertelsmann would have to achieve to 'break-even'
```

[40]: [0.787736699729486]

It would appear Bertelsmann paid too much, unless we are in the Optimistic scenario.

1.1.3 Problem 3

[43]:
```
r = .82
m = 60
```

[123]:
```
def CLV_immediate(m, r, d): # Using Simple Retention model with constant␣
↪retention rate = r
    return m*(1+d)/(1+d-r) # Assuming first payment comes at beginning of year␣
↪1
```

[125]:
```
def P(t, r):
    return (1-r)*(r**(t-1))
```

a) Expected value of T (time to attrition in months)

[126]:
```
1 / (1-r) # Theoretical result for Expected value of T i.e. E[T]
```

[126]: 5.5555555555555545

[127]:
```python
import numpy as np
from statistics import mean
from statistics import median
Probabilities = np.array(list(map(P, range(1, 10000), [r]*10000)))
Times = np.array(range(1,10000))
sum(Probabilities * Times) # Empirical result for expected value of T i.e. E[T]
```

[127]: 5.555555555555552

In any case, when rounding to number of months, the answer in both cases is 6 months.

Finding the Median

[128]:
```python
i = 1
while sum(Probabilities[0:i]) < .5: # The median is the point at which P(T<=t)␣
↪> .5
    i+=1
print(i) # This is the Median!
# (need to substract 1, because we start from 0 indexing, but the last index is␣
↪not included in Probabilities[0:i], so it balances out)
```

4

b) Expected Lifetime Value

[290]:
```python
'${:,.2f}'.format(CLV_immediate(60, .82, .01))
```

[290]: '$318.95'

c) Customer Equity

[131]:
```python
'${:,.2f}'.format(CLV_immediate(60, .82, .01) * 1000)
```

[131]: '$318,947.37'

d) Unobserved Heterogeneity

[135]:
```python
'${:,.2f}'.format(CLV_immediate(60, .92, .01)) # Loyalist
```

[135]: '$673.33'

[134]:
```python
'${:,.2f}'.format(CLV_immediate(60, .72, .01)) # Non-Loyalist
```

[134]: '$208.97'

e) Acquiring 1000 customers

```
[136]: '${:,.2f}'.format(CLV_immediate(60, .92, .01)*500 + CLV_immediate(60, .72, .
       ↪01)*500)
```

```
[136]: '$441,149.43'
```

We see that acquiring 500 loyalists + 500 non-loyalists is better than acquiring 1000 semi-loyalists

1.1.4 Problem 4

```
[235]: import operator
       import functools
       def Prob(t, r):
           if t == 1:
               return 1-r[0]
           else:
               return (1-r[t-1])*functools.reduce(operator.mul, r[:t-1]) # Returns the␣
       ↪product of all elements in r up until the one at position t, multiplied by␣
       ↪1-r[t]
```

```
[208]: r = [.95] * 4 +[.7, .6] + [.9] * 1000 # Creating the retention rates array and␣
       ↪looking at it to make sure it's ok
       r[:10]
```

```
[208]: [0.95, 0.95, 0.95, 0.95, 0.7, 0.6, 0.9, 0.9, 0.9, 0.9]
```

a) Probability of Default in Month 8

```
[236]: "{:.2%}".format(Prob(8,r))
```

```
[236]: '3.08%'
```

b) Probability of Default after 1 year

```
[237]: "{:.2%}".format(Prob(12,r))
```

```
[237]: '2.02%'
```

c) Probability of Default after 2 year

```
[238]: "{:.2%}".format(Prob(24,r))
```

```
[238]: '0.57%'
```

d) Probability of Cancelling before end of first year

```
[323]: def CDF(t, r):
           if t==1:
               return Prob(1,r)
```

```
    else:
        Probabilities = list(map(Prob, range(1,t), [r for _ in range(t)])) #␣
↪Calculate probabilities for all months up to month t
        return functools.reduce(operator.add, Probabilities) # Add up all the␣
↪probabilities up to month t
```

[324]:
```
"{:.2%}".format(CDF(12,r))
```

[324]: '79.80%'

e) Expected Month of Attrition

[277]:
```
Probabilities = np.array(list(map(Prob, range(1,1000), [r for _ in␣
↪range(999)]))) # Calculating the first 1000 probabilities
Probabilities[:14] # Taking a look at them to make sure they are ok
```

[277]:
```
array([0.05      , 0.0475    , 0.045125  , 0.04286875, 0.24435187,
       0.22806175, 0.03420926, 0.03078834, 0.0277095 , 0.02493855,
       0.0224447 , 0.02020023, 0.0181802 , 0.01636218])
```

[287]:
```
Times = np.arange(1,1000) # Declaring the first 1000 time periods
Times[:14] # Taking a look at them to make sure they are ok
```

[287]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

[294]:
```
import math
math.floor(sum(Times * Probabilities))
```

[294]: 8

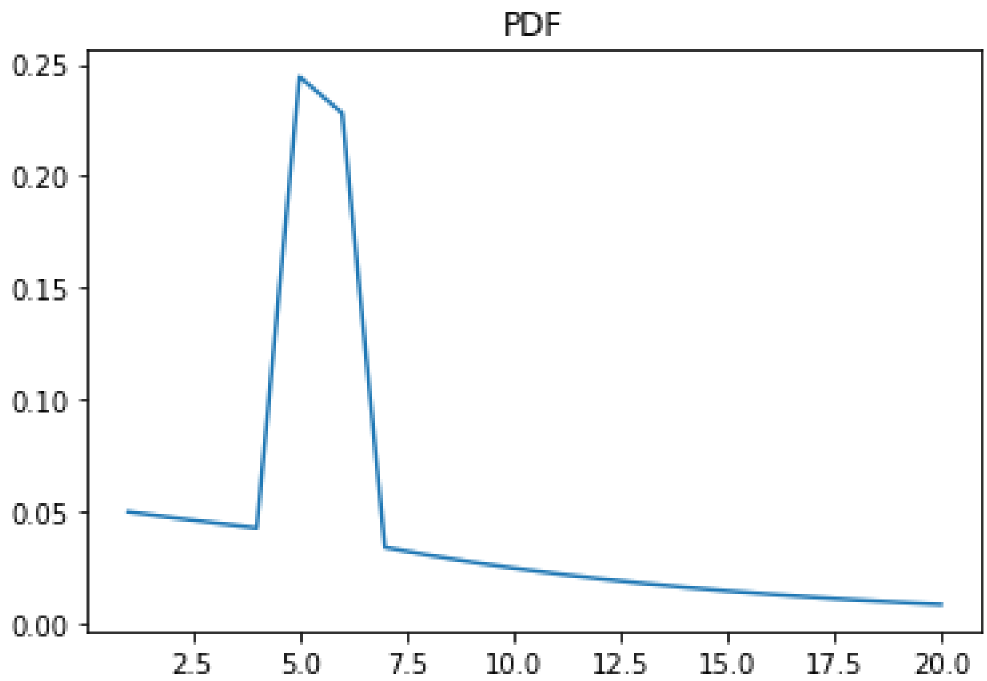
The expected month of attrition is 8 i.e. on average, we would expect a customer to leave us after 8 months

f) Plotting Survival, PDF, and Hazard Rate

[304]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

[313]:
```
_ = plt.plot(Times[:20],Probabilities[:20])
plt.title('PDF')
```

[313]: Text(0.5, 1.0, 'PDF')

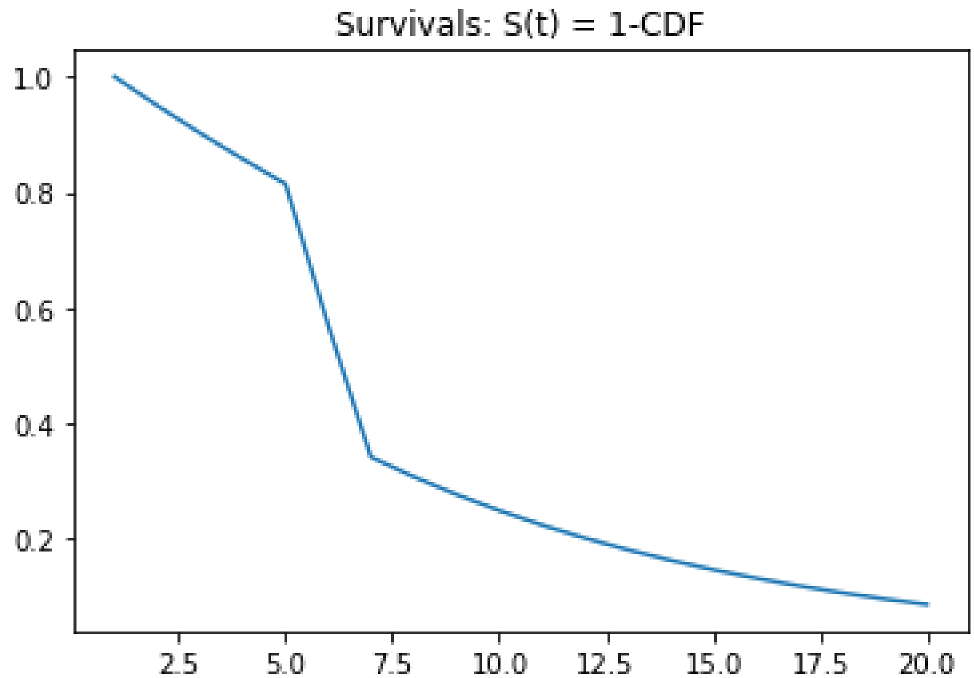


[351]:
```
Survivals = 1-np.array([0]+list(map(CDF, range(2,100), [r for _ in␣
↪range(98)]))) # S(t) = 1-CDF(t)
Survivals[:15]
```

[351]:
```
array([1.        , 0.95      , 0.9025    , 0.857375  , 0.81450625,
       0.57015437, 0.34209262, 0.30788336, 0.27709503, 0.24938552,
       0.22444697, 0.20200227, 0.18180205, 0.16362184, 0.14725966])
```

[352]:
```
_ = plt.plot(Times[:20],Survivals[:20])
plt.title('Survivals: S(t) = 1-CDF')
```

[352]:
```
Text(0.5, 1.0, 'Survivals: S(t) = 1-CDF')
```

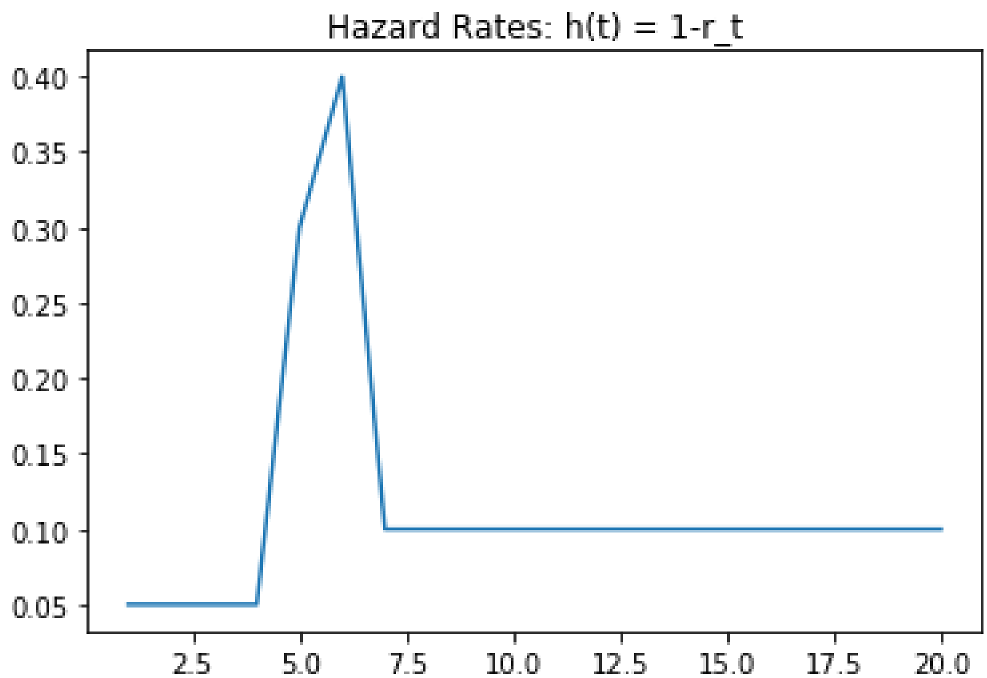


[353]:
```
Hazards = 1-np.array(r)
Hazards[:15]
```

[353]:
```
array([0.05, 0.05, 0.05, 0.05, 0.3 , 0.4 , 0.1 , 0.1 , 0.1 , 0.1 , 0.1 ,
       0.1 , 0.1 , 0.1 , 0.1 ])
```

[354]:
```
_ = plt.plot(Times[:20],Hazards[:20])
plt.title('Hazard Rates: h(t) = 1-r_t')
```

[354]:
```
Text(0.5, 1.0, 'Hazard Rates: h(t) = 1-r_t')
```



g) Expected CLV

[355]:
```
d = .12
m = np.array([29.99]*6+[60.48]*1000) # Defining Non-discounted payments
m[:15] # Taking a look at them to make sure they are ok
```

[355]:
```
array([29.99, 29.99, 29.99, 29.99, 29.99, 29.99, 60.48, 60.48, 60.48,
       60.48, 60.48, 60.48, 60.48, 60.48, 60.48])
```

[358]:
```
mDiscounted = np.array(list(map(lambda t, m, d: m / ((1+d)**t)  , range(99), m␣
 ↪,[d]*1000))) # Discounting the payments for Present Value
mDiscounted[:15] # Taking a look at them to make sure they are ok
```

[358]:
```
array([29.99      , 26.77678571, 23.90784439, 21.34628963, 19.05918717,
       17.0171314 , 30.64105021, 27.35808054, 24.42685763, 21.80969431,
       19.47294135, 17.38655478, 15.52370962, 13.86045502, 12.37540627])
```

[362]:
```
'CLV = ${:,.2f}'.format(sum(mDiscounted * Survivals))
```

[362]:
```
'CLV = $173.90'
```