CECL INTRODUCTON AND IMPLEMENTATION

With Personal Project on CECL

FEBRUARY 25, 2019

DEEPAK KUMAR TIWARI

Financial Mathematics Graduate Student, NCSU

CECL: CECL stands for Current Expected Credit Losses

- Requires to be implemented by institution issuing credit, including banks, savings institutions, credit unions and holding companies filing under GAAP accounting standards. The CECL standards will be effective starting Dec. 15, 2019, for public business entities that are US SEC filers.
- Change the way institutions have been estimating Allowance for Loan and Lease Losses
- Estimate expected loss over the life of the loans using historical data, current conditions and reasonable and supportable forecasts
- Lesser the data more would be the assumptions to estimate expected loss. More will be the data, more supportable would be the forecasts and lesser would be questions from regulators.
- Loss would be calculated on a loan level amortized cost calculation basis
  Amortized cost = Current Balance +/- deferred fees/costs +/- Premium/Discounts + Accrued Interests
- Risk Based pools can be made to determine allowance for losses in loans having similar risk parameters. Most important Risk factors are:
  1. FICO Score
  2. Debt to Income Ratio
  3. Loan to Value Ratio
  4. Disposable Income
  5. Employment type
  6. Property type
  7.Loan Purpose
  8.Geographic Locations
- Already matured or closed loans can be segregated based on above risk profiles and see how loans behaved in the past
- Adjust those results based on various **qualitative and quantitative factors**. These factors can be:
  1.Unemployment Rate
  2.Consumer Price Index
  3.Interest Rate (Yield curve)
  4.House Price Index
  5.Gross Domestic Product
  6.Per Capita Income
  7.Taxes
  These factors are forecasted by many industries or organizations so, often CECL company wouldn't require forecasting these data.
- Most Convenient and exact ways to compute allowance under CECL can be **Vintage and PD/LGD model**

1.Most common Method to find Allowance for loan losses: Vintage Model

1.Vintage Model calculates losses for similar risky loans based on origination date and their historical performances

2. Works with loans which are homogeneous and follow some predictive behavior

3.Loans with similar risk parameters or type are segmented according to origination years

4.Can be applied to portfolio of loans where underwriting standard, behavior and loan terms are same.

5.Loss amount are calculated based on some linear rate or other based on conditions and expectations

6.Steps to find loss allowance in vintage loans

- Suppose its 2016 and we want to forecast loss allowance for next 5 years till 2021.
- We will stratify loans based on loan type like ARM and Fixed according to origination years
- We will come up with the way how we expect to receive the payment. Below is the table with loan amounts in **thousands** originated in each year and how payment is expected to be received.

Vintage		Principal Collections										
Year	Originations	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
2011	40,000	6,667	6,125	6,548	6,978	7,152	6,530	-	-	-	-	-
2012	42,400	-	7,067	6,941	7,397	7,581	6,922	6,493	-	-	-	-
2013	44,944	-	-	7,491	7,840	8,036	7,337	6,882	7,357	-	-	-
2014	47,641	-	-	-	7,940	8,518	7,778	7,295	7,799	8,311	-	-
2015	50,499	-	-	-	-	8,417	8,244	7,733	8,267	8,810	9,029	-
2016	53,529	-	-	-	-	-	8,922	8,197	8,763	9,338	9,571	8,739
	Totals	6,667	13,192	20,980	30,155	39,704	45,734	36,599	32,186	26,459	18,600	8,739
	Period End Loan Balances	33,333	62,542	86,506	103,991	114,787	122,582	85,983	53,798	27,339	8,739	-

- Suppose we have history of loans till 2016 and we want to forecast loss for loans originated in year 2017. Above is the payment schedule for loans originated in a particular year. Like 40000 loans in 2011, 42400 loans in 2012 and so on.
- In 2011, of these 40000 loans bank expect to receive 6667 in 1st year, 6125 in 2nd year, 6548 in 3rd year and so on. Similarly, in 2016, out of 53529 8922 is expected to be received in 1st year, 8197 in 2nd year and so on.
- Below are the charge offs. These are loans in which payments have not been made for 180 days usually. And Bank recognize them as loss and doesn't expect to receive. It goes as charge offs on Balance sheet.

Charge-offs:					
Commercial LHFI	**(131)**	(167)	(287)	(117)	(128)
Consumer LHFI	**(322)**	(324)	(304)	(353)	(479)
Total charge-offs	**(453)**	(491)	(591)	(470)	(607)

- Among 6667 amounts expected 52 were recognized as charge off. Similarly, in 2nd year 84 were recognized as charge offs and 36 in 6th year. We have this known data for 2011 loans. But we have only 5 years known charge offs for 2012 loans, 4 years for 2012 loans and so on and only 1 year known charge offs for 2016 loans and none known for 2017 loans.

Origination	Charge-offs by Origination Year ($)													
Year	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	Total	
2011	52	84	112	124	64	36	-	-	-	-	-	-	472	1.18%
2012	-	51	85	114	127	64	38	-	-	-	-	-	479	1.13%
2013	-	-	49	85	117	130	59	39	-	-	-	-	480	1.07%
2014	-	-	-	48	86	119	107	61	42	-	-	-	463	0.97%
2015	-	-	-	-	45	86	90	112	66	47	-	-	446	0.88%
2016	-	-	-	-	-	43	63	94	120	73	56	-	450	0.84%
Totals	52	135	246	372	439	478	357	306	228	120	56	-	2,789	

- Total charge offs for 2011 loans is 472 which is 1.18% of total of 40000 loans originated in 2011. We can do some kind of regression to find how loans have been behaving over years and project charge offs over years for loan which we don't have data yet.
- Alternative way is to use qualitative factor also known as Q-factor to predict loss amounts. These Q-factors can be interest rate, unemployment rate, Consumer Price Index, GDP etc. depending on which loss relates most. That correlation can be found by data visualizations

Loss Rates by Vintage						
Origination	Y1	Y2	Y3	Y4	Y5	Y6
2011	0.13%	0.21%	0.28%	0.31%	0.16%	0.09%
2012	0.12%	0.20%	0.27%	0.30%	0.15%	
2013	0.11%	0.19%	0.26%	0.29%		
2014	0.10%	0.18%	0.25%			
2015	0.09%	0.17%				
2016	0.08%					
Average	0.11%	0.19%	0.27%	0.30%	0.16%	0.09%
Loss/Q factor	1.96%	3.74%	5.67%	7.14%	4.13%	2.81%

Q Factor by Vintage - Eg. MA Unemployment						
Origination	Y1	Y2	Y3	Y4	Y5	Y6
2011	6.70%	6.70%	6.10%	5.10%	4.30%	3.20%
2012	6.70%	6.10%	5.10%	4.30%	3.20%	
2013	6.10%	5.10%	4.30%	3.20%		
2014	5.10%	4.30%	3.20%			
2015	4.30%	3.20%				
2016	3.20%					
Average	5.35%	5.08%	4.68%	4.20%	3.75%	3.20%

- On the left, these loss rates are found by calculating charge offs in a particular year to total loans originated in that year. Like for 2011 loans, 1st year charge offs were 52 which is 0.13% of total of 40000 loans. For year 2nd, 84 was 0.21% of total of 40000 loans. All the known loss rates can be found in a same manner.

- We average out all the loss rates by averaging only those loss rates which are available to us.
- Q-factor Unemployment rate similarly is known for all these years which can be averaged out in a similar way.
- Then we can calculate Loss/Q factor ratio by simply dividing Average loss rates by average Q factor
- Blue part of Q-factor needs to be predicted by regression analysis or directly getting those predictions from market as there are many industries which predict those macroeconomic factors.

Loss Rates by Vintage						
Origination	Y1	Y2	Y3	Y4	Y5	Y6
2011	0.13%	0.21%	0.28%	0.31%	0.16%	0.09%
2012	0.12%	0.20%	0.27%	0.30%	0.15%	0.09%
2013	0.11%	0.19%	0.26%	0.29%	0.13%	0.09%
2014	0.10%	0.18%	0.25%	0.23%	0.13%	0.09%
2015	0.09%	0.17%	0.18%	0.22%	0.13%	0.09%
2016	0.08%	0.12%	0.18%	0.23%	0.14%	0.11%
Average	0.11%	0.18%	0.24%	0.26%	0.14%	0.09%
Loss/Q factor	1.96%	3.74%	5.67%	7.14%	4.13%	2.81%

Reasonable Supportable Forecast						
Origination	Y1	Y2	Y3	Y4	Y5	Y6
2011	6.70%	6.70%	6.10%	5.10%	4.30%	3.20%
2012	6.70%	6.10%	5.10%	4.30%	3.20%	3.15%
2013	6.10%	5.10%	4.30%	3.20%	3.15%	3.10%
2014	5.10%	4.30%	3.20%	3.15%	3.10%	3.15%
2015	4.30%	3.20%	3.15%	3.10%	3.15%	3.30%
2016	3.20%	3.15%	3.10%	3.15%	3.30%	3.75%
Average	5.35%	4.76%	4.16%	3.67%	3.37%	3.28%

- 2nd step is to find new average of Q-factor by including predicted values and then recalculating average of loss rates. Please note that Loss/Q factor ratio would be same in long run because of behavior of loss amount with respect to Q factors
- Once new average is found for each year, we can find individual unknown loss rates as
    1. Loss / Q factor * New average of Q factor
       1.96% * 5.35%=0.178 or 0.18%
    2. 0.21%+0.20%+0.19%+0.18%+0.17%+x(unknown)=0.18%*6
    3. X=0.13%
- Similarly, if two or three or whatever loss rates would be there, we can assume it would be equal and solve for unknown.
- Drawback is that we loss rates for same year of different origination years are equal.
- Bunch of Q factor can also be taken instead of one and averaged out and calculated similarly as above

PD/LGD/EAD Method: Probability of Default/ Loss Given Default/ Exposure at Default:

Probability of Default: This is the percentage of loan going default in any pool over certain period. However, this probability and time period is subjective and can be calculated in a different way by institutions
Exposure at default: Total amount at risk at any point during the life of a loan
Loss given Default: Percentage of total amount at Risk that can be lost in event of default

Loss Rate= PD*LGD
Loss rate can be adjusted according to some qualitative or quantitative factors based

Allowance of loan losses and lease (ALLL)= Loss Rate*EAD

Steps of implementing PD/LGD/EAD model:

1. Get all historical loan data for at least 10 years. It can be more too, larger the data more will be the accuracy, but it should be made sure that economic conditions haven't change substantially since loan of which data are being used
2. Make pool of loans based on Geographic location. It is important to do so because we can know which state is riskier to do business in, consumer behavior and finally it can help to hedge risk by lending more in states where default is less or some other measures
3. Make further sub-pools of loans based on risk factors like DTI, LTV, Employment type, Disposable income, Loan purpose.

4. We can then apply Markov Chain Transition model to see how loans have been behaving in same risk pools. We can segregate each sub-pool in a following way:

   1. Prepayments
   2. Paid before late charge is accrued (Usually within 45th or 46th days of last payment)
   3. After 15th days of due date but within 60 days
   4. Paid after loan is delinquent (after 60 days but before 90 days)
   5. Paid within 90 days and 120 days
   6. Finally Default Loans

   In Markov Chains Prepayment and Default are terminal states as once loan has been prepaid or defaulted it can not move to other states and are no longer performing and accrual loans for Bank

Markov Chain State Transition

State 1 can be prepaid so the borrower can make prepayment anytime
State 2 can be within 45 days of past payment. He can either pre-pay the loan or move to 46-59 days due pool
Similarly, loans can move to prepayment or next past due dates
Once loan moves to the default state, it cannot move further to any bucket

1. Once these all probabilities are found, we can use regression method like logistic regression to find Probability for new or current to move from one state to another
2. LGD would be outstanding loan balance at each state after deducting possible recoveries
3. Loss rates can be calculated by multiplying PD and LGD
4. Finally, Exposure at default would be total loan amount at stake at each sub-pool

Logistic Regression
With defined delinquency status, we can define the transition Zit of delinquency status between month t and month t −1 for loan i. This transition is the target variable of logistic regression. Then Zit

is a categorical variable takes values?

Zit can be defined as below:
0−1 from performance to prepayment
00 from performance to performance
01 from performance to delinquency less than 30 days
10 from delinquency less than 30 days to performance
01 from delinquency less than 30 days to delinquency less than 60 days
21 from delinquency less than 60 days to delinquency less than 30 days
23 from delinquency less than 60 days to delinquency less than 90 days
32 from delinquency less than 90 days to delinquency less than 60 days
34 from delinquency less than 90 days to default

Partition the dataset into training and testing two sets. The training sets are 70% of the total data. The testing set contains the rest of it. This regression method would predict PD for new loans in a certain time frame.

CECL PD/LGD Model Implementation Using Markov Chain and Logistic Regression Model in Python using Data Analysis and Machine Learning

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import seaborn as sb

from functools import reduce
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import classification_report
import statsmodels.formula.api as sm
from sklearn.metrics import confusion_matrix
```

In [2]:
```
# Read Data

acquisition = ['LoanID','Channel','SellerName','OrigInterestRate','OrigUnpPrinc',
               'OrigDate','FirstPayment','OrigLTV','OrigCLTV','NumBorrow','DTIRat
               'FTHomeBuyer','LoanPurpose','PropertyType','NumUnits','OccStatus',
               'Zip','MortInsPerc','ProductType','CoCreditScore','MortInsType','R

performance = ['LoanID','MonthRep','Servicer','LAST_RT','LAST_UPB',
               'LoanAge','MonthsToMaturity','AdMonthsToMaturity','MaturityDate','
               'CLDS','ModFlag','ZB_Code','ZB_Date','LPI_Date',
               'FCC_Date','DISP_Date','FCC_COST','PP_COST','AR_COST',
               'IE_COST', 'TAX_COST','NS_PROCS','CE_PROCS','RMW_PROCS','O_PROCS',
              'PRIN_FORG_UPB_FHFA','REPCH_FLAG','PRIN_FORG_UPB_OTH','TRANSFER_FLA

a1 = pd.read_csv('/Users/xy4/Documents/CECL/Acquisition_2007Q1.txt', sep='|', nam
p1 = pd.read_csv('/Users/xy4/Documents/CECL/Performance_2007Q1.txt', sep='|', nam
```

In [3]:
```
keep_var_acq = ['LoanID','OrigInterestRate','OrigUnpPrinc','OrigLTV','CreditScore
keep_var_per = ['LoanID','MonthRep','LAST_RT','LAST_UPB',
               'MonthsToMaturity','CLDS',
               'ZB_Code','ZB_Date','LPI_Date','FCC_Date','DISP_Date',
               'FCC_COST','PP_COST','AR_COST','IE_COST','TAX_COST','NS_PROCS',
               'O_PROCS','NON_INT_UPB']

a1_sub = a1[keep_var_acq].reset_index(drop=True)
p1_sub = p1[keep_var_per].reset_index(drop=True)

a1 = a1_sub.dropna(subset = ['CreditScore'])
```

In [4]:
```
def_info = p1_sub.loc[p1_sub['ZB_Code'].isin([2.0,3.0,6.0,9.0,15,16])]
```

In [5]:
```
prepay_info = p1_sub.loc[p1_sub['ZB_Code'].isin([1.0])]
```

In [6]:
```
sample_id = a1[~a1['LoanID'].isin(def_info['LoanID'])].sample(n=def_info.shape[0]
```

In [7]:
```
sample_a1 = a1.loc[a1['LoanID'].isin(sample_id['LoanID']) | a1['LoanID'].isin(def
```

In [8]:
```
sample_p1 = p1_sub.loc[p1['LoanID'].isin(sample_a1['LoanID'])]
```

In [9]:
```
sample_p1 = sample_p1.assign(**{'prepay_flag':np.nan})
```

In [10]:
```
sample_p1.loc[sample_p1['LoanID'].isin(prepay_info['LoanID']),'prepay_flag']=1
```

In [11]:
```
def f_delq(row):
    if row['prepay_flag'] == 1:
        if row['CLDS'] == '0':
            return 0
        elif row['ZB_Code'] == 1.0:
            return -1
    else:
        if row['CLDS'] == '0':
            return 0
        elif row['CLDS'] == '1':
            return 1
        elif row['CLDS'] == '2':
            return 2
        elif row['CLDS'] == '3':
            return 3
        elif row['ZB_Code'] in [2.0,3.0,6.0,9.0,15,16]:
            return 4
```

In [12]:
```
sample_p1 = sample_p1.assign(DELQ_STAT= sample_p1.apply(f_delq, axis=1))
```

In [13]:
```
# Define Transitions
```

In [14]:
```
sample_p1 = sample_p1.dropna(subset=["DELQ_STAT"])
```

In [15]:
```
first_row_index = sample_p1.groupby(['LoanID'], as_index=False).apply(lambda g: g
```

In [16]:
```
sample_p1.loc[first_row_index,'firstrow'] = 1
```

In [17]:
```
sample_p1['delq_shift'] = sample_p1.DELQ_STAT.shift()
```

In [18]:
```
def f_tran(row): # transition
    if row['firstrow'] != 1:
        if row['DELQ_STAT'] == 0 and row['delq_shift'] == 0:
            return '00'
        elif row['DELQ_STAT'] == 1 and row['delq_shift'] == 0:
            return '10'
     #   elif row['DELQ_STAT'] == 1 and row['delq_shift'] == 1:
      #      return '11'
        elif row['DELQ_STAT'] == 0 and row['delq_shift'] == 1:
            return '01'
        elif row['DELQ_STAT'] == 2 and row['delq_shift'] == 1:
            return '12'
        elif row['DELQ_STAT'] == 1 and row['delq_shift'] == 2:
            return '21'
    #    elif row['DELQ_STAT'] == 2 and row['delq_shift'] == 2:
     #       return '22'
        elif row['DELQ_STAT'] == 3 and row['delq_shift'] == 2:
            return '23'
        elif row['DELQ_STAT'] == 2 and row['delq_shift'] == 3:
            return '32'
    #    elif row['DELQ_STAT'] == 3 and row['delq_shift'] == 3:
     #       return '33'
        elif row['DELQ_STAT'] == 4 and row['delq_shift'] == 3:
            return '34'
        elif row['DELQ_STAT'] == -1 and row['delq_shift'] == 0:
            return '0-1'
```

In [19]:
```
sample_p1 = sample_p1.assign(tran= sample_p1.apply(f_tran, axis=1))
```

In [20]:
```
# Read Historical Macro Economic Variables (MEV)
unemployment = pd.read_csv('/Users/xy4/Documents/CECL/UNRATE.csv', sep=',', names
unemployment['DATE'] = pd.to_datetime(unemployment.DATE)
unemployment['DATE'] = unemployment['DATE'].dt.strftime('%m/%d/%Y')
hpi = pd.read_csv('/Users/xy4/Documents/CECL/HPI.csv', sep=',', names=['DATE','HP
hpi['DATE'] = pd.to_datetime(hpi.DATE)
hpi['DATE'] = hpi['DATE'].dt.strftime('%m/%d/%Y')
cpi = pd.read_csv('/Users/xy4/Documents/CECL/CPALTT01USM661S.csv', sep=',', names
cpi['DATE'] = pd.to_datetime(cpi.DATE)
cpi['DATE'] = cpi['DATE'].dt.strftime('%m/%d/%Y')
gs3m = pd.read_csv('/Users/xy4/Documents/CECL/GS3M.csv', sep=',', names=['DATE','
gs3m['DATE'] = pd.to_datetime(gs3m.DATE)
gs3m['DATE'] = gs3m['DATE'].dt.strftime('%m/%d/%Y')
gs5 = pd.read_csv('/Users/xy4/Documents/CECL/GS5.csv', sep=',', names=['DATE','GS
gs5['DATE'] = pd.to_datetime(gs5.DATE)
gs5['DATE'] = gs5['DATE'].dt.strftime('%m/%d/%Y')
gs10 = pd.read_csv('/Users/xy4/Documents/CECL/GS5.csv', sep=',', names=['DATE','G
gs10['DATE'] = pd.to_datetime(gs10.DATE)
gs10['DATE'] = gs10['DATE'].dt.strftime('%m/%d/%Y')
# Make one table
temp = [unemployment,hpi,cpi,gs3m,gs5,gs10]
MEV = reduce(lambda left,right:pd.merge(left,right,on='DATE'),temp)
```

In [21]:
```
MEV.set_index('DATE',inplace=True)
```

In [22]:
```
MEV = MEV.pct_change()
```

In [23]:
```
mQ1 = pd.merge(sample_a1,sample_p1,how="outer", on=['LoanID'])
```

In [24]:
```
mQ1 = pd.merge(mQ1, MEV, how = 'left', left_on = "MonthRep", right_on="DATE")
```

In [25]:
```
mQ1 = mQ1.dropna(subset=['tran'])
```

In [26]:
```
mQ1_00 = mQ1.loc[mQ1['tran'] == '00']
```

In [27]:
```
mQ1_000 = mQ1.loc[mQ1['tran'] != '00']
```

In [28]:
```
mQ1_00_sample = mQ1_00.sample(frac=0.01,random_state=1)
```

In [29]:
```
mQ1 = pd.concat([mQ1_000, mQ1_00_sample], join="inner")
```

In [30]:
```
MEVs = ['UNEMPLOY','HPI','GS3M','CreditScore']
```

In [31]:
```
# Regression for state 0
mQ1_0 = mQ1.loc[mQ1['tran'].isin(['00','01','0-1'])]
X = pd.DataFrame(mQ1_0,columns=MEVs).values
Y = mQ1_0['tran'].values
```

In [32]:
```
X_train0, X_test0, Y_train0, Y_test0 = train_test_split(X,Y, test_size = 0.3, ran
LogReg0 = LogisticRegression()
state_0 = LogReg0.fit(X_train0,Y_train0)
Y_pred0 = LogReg0.predict(X_test0)
print(classification_report(Y_test0,Y_pred0))
```

```
             precision    recall  f1-score   support

        0-1       0.46      0.58      0.51      5711
         00       0.38      0.07      0.12      5882
         01       0.54      0.78      0.64      7487

avg / total       0.47      0.50      0.44     19080
```

In [33]:
```
# Regression for state 1
mQ1_1 = mQ1.loc[mQ1['tran'].isin(['10','12'])]
X = pd.DataFrame(mQ1_1,columns=MEVs).values
Y = mQ1_1['tran'].values
```

In [34]:
```
X_train1, X_test1, Y_train1, Y_test1 = train_test_split(X,Y, test_size = 0.3, ran
LogReg1 = LogisticRegression()
state_1 = LogReg1.fit(X_train1,Y_train1)
Y_pred1 = LogReg1.predict(X_test1)
print(classification_report(Y_test1,Y_pred1))
```

```
             precision    recall  f1-score   support

         10       0.59      1.00      0.74     16455
         12       0.00      0.00      0.00     11268

avg / total       0.35      0.59      0.44     27723


C:\Users\xy4\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:113
5: UndefinedMetricWarning: Precision and F-score are ill-defined and being set
to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

In [35]:
```
mQ1_2 = mQ1.loc[mQ1['tran'].isin(['21','23'])]
X = pd.DataFrame(mQ1_2,columns=MEVs).values
Y = mQ1_2['tran'].values
```

In [36]:
```
X_train2, X_test2, Y_train2, Y_test2 = train_test_split(X,Y, test_size = 0.3, ran
LogReg2 = LogisticRegression()
state_2 = LogReg2.fit(X_train2,Y_train2)
Y_pred2 = LogReg2.predict(X_test2)
print(classification_report(Y_test2,Y_pred2))
```

```
             precision    recall  f1-score   support

         21       0.00      0.00      0.00      1579
         23       0.85      1.00      0.92      9133

avg / total       0.73      0.85      0.78     10712


C:\Users\xy4\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:113
5: UndefinedMetricWarning: Precision and F-score are ill-defined and being set
to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

In [37]:
```
mQ1_3 = mQ1.loc[mQ1['tran'].isin(['32','34'])]
X = pd.DataFrame(mQ1_3,columns=MEVs).values
Y = mQ1_3['tran'].values
```

In [38]:
```
X_train3, X_test3, Y_train3, Y_test3 = train_test_split(X,Y, test_size = 0.3, ran
LogReg3 = LogisticRegression()
state_3 = LogReg3.fit(X_train3,Y_train3)
Y_pred3 = LogReg3.predict(X_test3)
print(classification_report(Y_test3,Y_pred3))
```

```
             precision    recall  f1-score   support

         32       0.00      0.00      0.00       636
         34       0.91      1.00      0.95      6079

avg / total       0.82      0.91      0.86      6715


C:\Users\xy4\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:113
5: UndefinedMetricWarning: Precision and F-score are ill-defined and being set
to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

In [39]:
```
# LGD
```

In [40]:
```
lgd_data = pd.merge(a1.loc[a1['LoanID'].isin(def_info['LoanID'])],
                    p1.loc[p1['ZB_Code'].isin([2.0,3.0,6.0,9.0,15,16])],
                           how="outer", on=['LoanID'])
```

In [41]:
```
lgd_data = lgd_data.dropna(subset=['CreditScore'])
```

In [42]:
```
lgd_data = pd.merge(lgd_data,MEV,how = 'left', left_on = "MonthRep", right_on="DA
```

In [43]:
```
lgd_data['Expenses'] =  lgd_data[['FCC_COST','PP_COST','AR_COST',
               'IE_COST']].sum(axis=1)
```

In [44]:
```
lgd_data['Proceeds'] = lgd_data[['NS_PROCS','CE_PROCS','RMW_PROCS','O_PROCS']].su
```

In [45]:
```
lgd_data['Loss'] = lgd_data['LAST_UPB'] + lgd_data['Expenses']-lgd_data['Proceeds
```

In [46]:
```
lgd_data.loc[lgd_data['Loss']<0]=0
```

In [47]:
```
lgd_data['Y'] = lgd_data['Loss']/lgd_data['LAST_UPB']
```

In [208]:
```
lgd_result = sm.ols(formula="Y ~ CreditScore + HPI + UNEMPLOY", data=lgd_data).fi
```

In [50]:
```
last_row_indicator = p1.groupby(['LoanID'], as_index=False).apply(lambda g: g.ind
```

In [131]:
```
p1.loc[last_row_indicator,'lastrow'] = 1
```

In [132]:
```
pd_data = p1.loc[p1['lastrow']==1]
```

In [133]:
```
pd_data = pd_data[~pd_data['ZB_Code'].isin([1.0,2.0,3.0,6.0,9.0,15,16])]
```

In [134]:
```
pd_data = pd_data.dropna(axis='columns')
```

In [135]:
```
pd_data = pd_data.drop(columns = ['lastrow','TRANSFER_FLAG','ModFlag','MSA'])
```

In [163]:
```
pd_mev = pd.merge(pd_data, MEV, how = 'left', left_on = "MonthRep", right_on="DAT
```

In [144]:
```
cresco = a1[['LoanID','CreditScore']]
```

In [164]:
```
pd_mev = pd.merge(pd_mev, cresco, how = 'left', left_on = "LoanID", right_on="Loa
```

In [165]:
```
pd_mev = pd_mev.dropna(subset=['CreditScore'])
```

In [166]:
```
pd_mev = pd_mev.reset_index(drop=True)
```

In [170]:
```
pred0 = state_0.predict_proba(pd_mev[MEVs].values)
pred1 = state_1.predict_proba(pd_mev[MEVs].values)
pred2 = state_2.predict_proba(pd_mev[MEVs].values)
pred3 = state_3.predict_proba(pd_mev[MEVs].values)
pd_mev['DefaultF'] = 0
pd_mev['DefaultP'] = 0
pd_mev['DefaultS'] = 0
```

In [172]:
```
for i, row in pd_mev.iterrows():
    a = pred0[i]
    b = pred1[i]
    c = pred2[i]
    d = pred3[i]
    TPM = np.array([[1,0,0,0,0,0],
                   [a[0],a[1],a[2],0,0,0],
                   [0,b[0],0,b[1],0,0,],
                   [0,0,c[0],0,c[1],0],
                   [0,0,0,d[0],0,d[1]],
                   [0,0,0,0,0,1]])
    j = 1
    P = TPM
    if pd_mev.iloc[i,7] == '0':
        k = 1
    elif pd_mev.iloc[i,7] == "1":
            K = 2
    elif pd_mev.iloc[i,7] == "2":
                k = 3
    elif pd_mev.iloc[i,7] == "3":
                    k = 4
    else:
        k = 5
    while k!= 0 and k!= 5 and j < pd_mev.iloc[i,5]:
        next_state = np.where(TPM[k]==max(TPM[k]))
        k = next_state[0][0]
        P = P*TPM
        j = j + 1
    pd_mev.iloc[i,-3] = j
    pd_mev.iloc[i,-2] = P[4,5]
    pd_mev.iloc[i,-1] = k
```

In [173]:
```
pd_mev.head()
```

Out[173]:

	LoanID	MonthRep	LAST_RT	LAST_UPB	LoanAge	MonthsToMaturity	MaturityDate	CLD
0	100036401006	09/01/2017	5.750	163704.94	127	233	02/2037	
1	100048597640	09/01/2017	6.250	208724.20	126	234	03/2037	
2	100122275573	09/01/2017	6.125	148843.11	127	233	02/2037	
3	100134033573	09/01/2017	6.750	57125.92	128	52	01/2022	
4	100252434914	09/01/2017	5.375	18858.39	128	52	01/2022	

In [240]:
```
lgd_pd_temp = pd_mev[pd_mev['DefaultF'] != pd_mev['MonthsToMaturity'] ]
```

In [241]:
```
lgd_pd = lgd_pd_temp[~lgd_pd_temp['DefaultS'].isin([0,1])]
```

In [283]:
```
lgd_pd = lgd_pd.reset_index(drop=True)
```

In [247]:
```
lgd_pd['lgd'] = lgd_result.predict(lgd_pd)
```

In [288]:
```
lgd_pd['MP'] = lgd_pd['LAST_RT']/100/12*lgd_pd['LAST_UPB']*(1+lgd_pd['LAST_RT']/1(
```

In [289]:
```
lgd_pd['MP1'] = lgd_pd['MP'] *(1+lgd_pd['LAST_RT']/100/12)
```

In [295]:
```
lgd_pd['ead'] = lgd_pd['LAST_UPB']*(1+lgd_pd['LAST_RT']/100/12)**lgd_pd['DefaultF
```

In [297]:
```
lgd_pd['cecl'] = lgd_pd['ead']*lgd_pd['DefaultP']*lgd_pd['lgd']/((1+lgd_pd['LAST_
```

In [304]:
```
result = sum(lgd_pd['cecl'])
```

In [305]:
```
print(result)
```

```
163592404.47582096
```

In [303]:
```
result / sum(pd_mev['LAST_UPB'])
```

Out[303]:
```
0.042254805047209715
```

In [306]:
```
sum(pd_mev['LAST_UPB'])
```

Out[306]:
```
3871569263.969986
```

In [307]:
```
lgd_result.summary()
```

Out[307]:

OLS Regression Results

Dep. Variable:	Y	**R-squared:**	0.017
Model:	OLS	**Adj. R-squared:**	0.017
Method:	Least Squares	**F-statistic:**	111.6
Date:	Wed, 05 Dec 2018	**Prob (F-statistic):**	1.30e-71
Time:	12:40:32	**Log-Likelihood:**	-3102.6
No. Observations:	18992	**AIC:**	6213.
Df Residuals:	18988	**BIC:**	6245.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	0.8267	0.025	32.976	0.000	0.778	0.876
CreditScore	-0.0005	3.61e-05	-13.472	0.000	-0.001	-0.000
HPI	4.3948	0.379	11.593	0.000	3.652	5.138
UNEMPLOY	0.3829	0.096	3.970	0.000	0.194	0.572

Omnibus:	1029.423	**Durbin-Watson:**	2.041
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	791.170
Skew:	0.409	**Prob(JB):**	1.58e-172
Kurtosis:	2.426	**Cond. No.**	1.27e+05

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.27e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [ ]: