Assignment 3: Midterm Case Studies- Freddie Mac "Single Family Home" Dataset

Rupesh Acharya, Niyati Maheshwari, Peter Vayda
November 29, 2018

Part I: Data Wrangling:

In this section we will download the files from the Freddie Mac website. We will be authenticating the user to download the files.

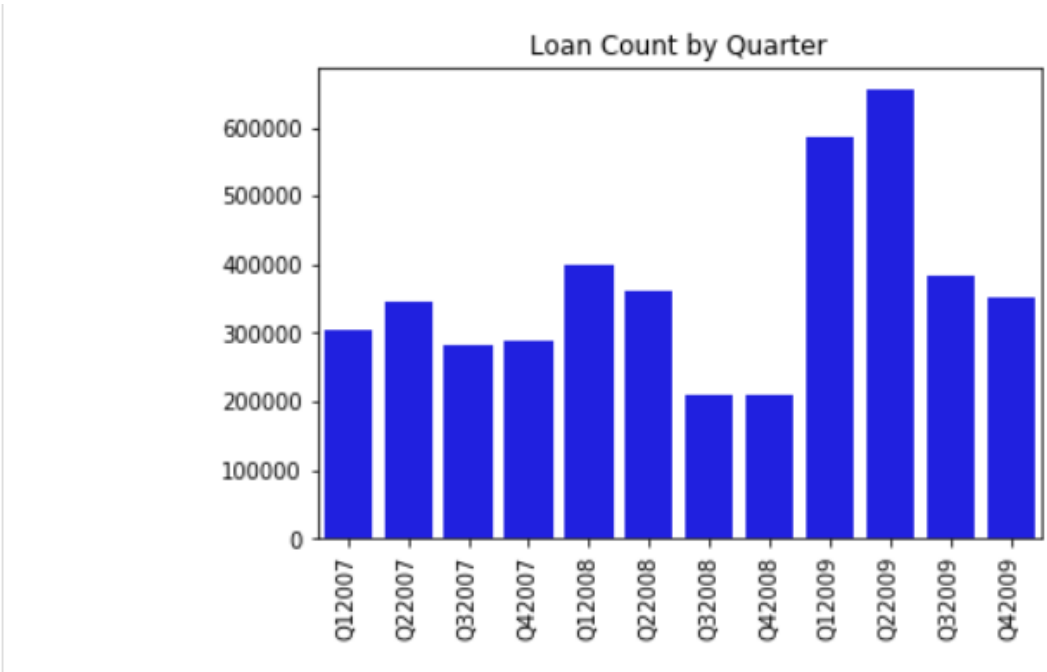
A. Data Download and Pre-processing:

The downloading and preprocessing part will programmatically enter the Freddie Mac website and scrape the samples file from 2005 onwards and store them in the system where Docker image in run and clean them and new file with the summaries will be created which is the combined for all the origination data in one file and combined file for performance in other file.

We have used BeautifulSoup for web scraping and requests library for creating and maintaining sessions.

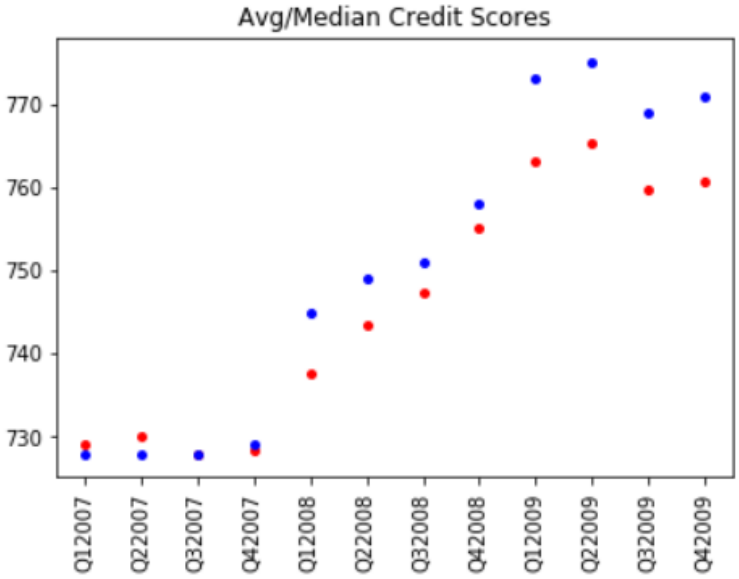
B. Exploratory Data Analysis:

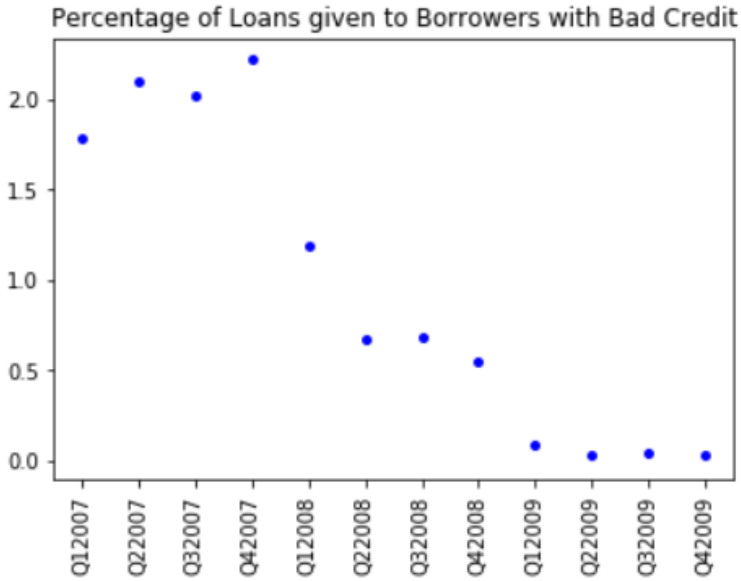
To understand our dataset, we performed an exploratory data analysis on the quarterly data from 2007-2009. The quarterly data encompassed two files: origination loan data and performance loan data. The origination loan data contained categorical attributes about the loan like (original interest rate, property state, property use, length of loan, etc). These attributes help categorize the borrower and the original loan terms. The performance loan data contains loan specific data related to borrower's performance in paying over the lifetime of the loan. The analysis evaluated numerical and categorical datatypes from both data sources over the specified timeline. The EDA Jupyter notebook contains more specific analysis. Overall, we were able to see data trends of the 2008 recession.

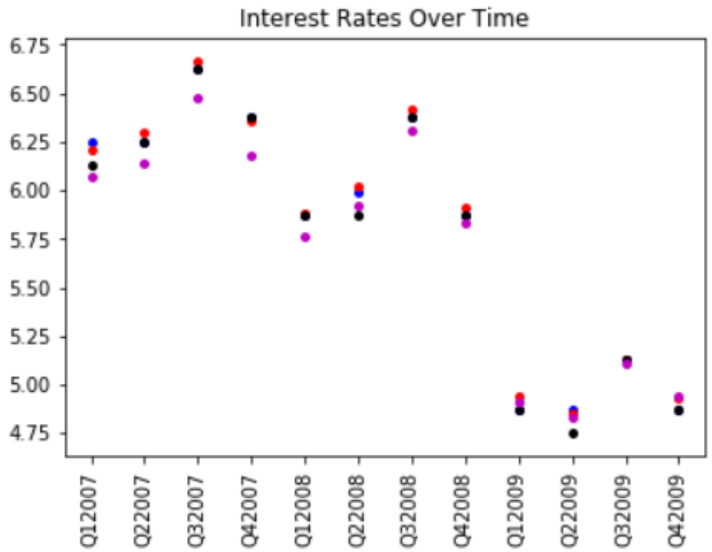


Evaluating the datasets enabled the team to assess quality and completeness as well as observe temporal and location specific trends. Overall the dataset was well prepared with documentation from Freddie Mac. User guide was very helpful in extracting required data for analysis. Upon first glance in the loan performance files, it appeared as though data categories were missing in their entirety. On closer examination, there were a variety of fields that were situation specific (i.e. a borrower paid off the loan in full or Freddie Mac sold the loan to another lender).

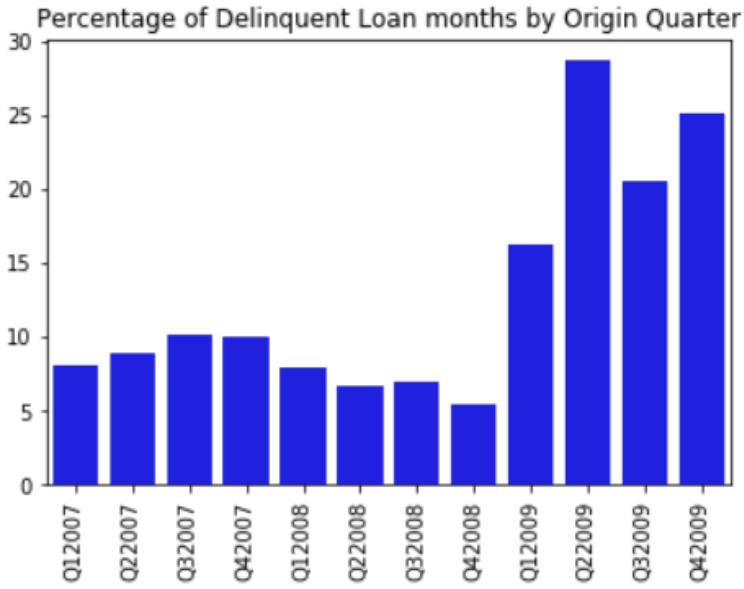
Trends over time were observed in loan count, credit score, bad credit borrower population, interest rates, delinquent loans, actual loss calculation, loans sold, and current actual unpaid principal balance (UPB).

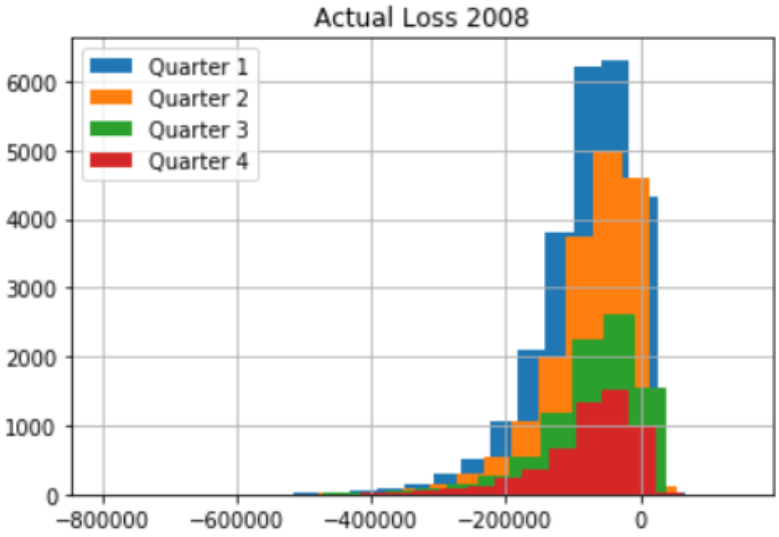




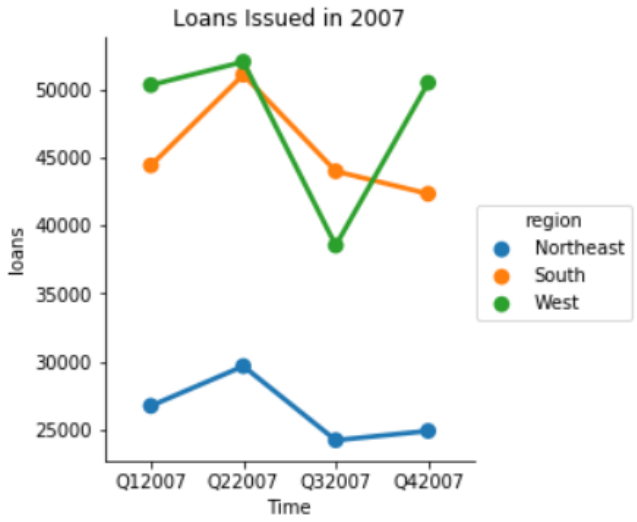


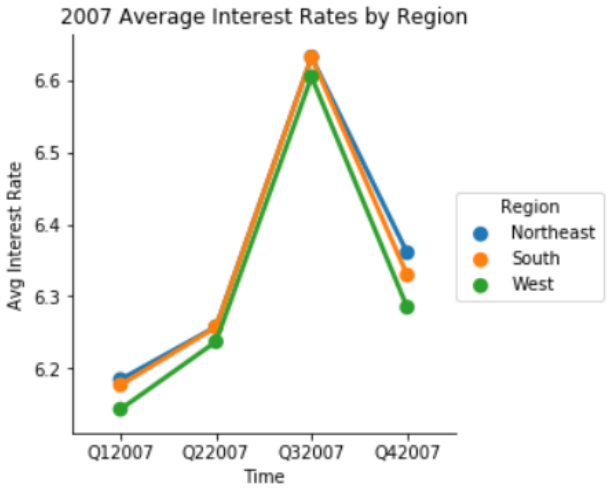
Loan count was mostly consistent with a drop and spike in issued loans near the end and beginning of 2008 and 2009, respectfully. With credit scores and interest rate data, we were able to observe an increase in the quality of borrowers (higher credit scores) and a decrease in the number of borrowers with bad credit over the timespan. In the same timeframe, we saw interest rates were reduced. These phenomena are most likely related in part to higher quality borrowers enabling a lower interest rate (less risk in the transaction). The percentage of delinquent loans had very interesting behavior. They rose in 2007, fell in 2008, then skyrocketed in 2009 well past double their initial percentage from 2007. This might be explained by a delayed borrowers reaction to the recession. By evaluating actual loss data, we were able to deduce that the number of loan sold consistently decreased over the 2007-2009 timespan. Finally, current UPB average rose consistently over the course of the 3 years.





Regional specific trends were observed in loan count and interest rate. Regions were broken out into Northeast, South, and West. Based on these regions issued loans and interest rates were evaluated over the year 2007. Most notably, the Northeast region had the highest regional interest rate while the West region had the lowest.





Part II: Building and Evaluating Models:

In this section, we will be evaluating the dataset against various models. We will be doing feature selection and best model prediction.

A. Prediction:

We will first download the Q1 and Q2 files for year 2005 programmatically. We will be taking the input from the user as the username, password, startyear and end year.

```
user = input("Enter Username: ")
passwd = input("Enter password: ")
trainQ = input("Enter Quarter for trainQ: ")
testQ = input("Enter Test Quarter: ")

print("USERNAME=" + user)
print("PASSWORD=" + passwd)
print("TRAINQUARTER=" + (trainQ))
print("TESTQUARTER=" + (testQ))

payload = payloadCreation(user, passwd)
getFilesFromFreddieMacPerQuarter(payload, trainQ, testQ)
```

We then built the regression model for the provided test and train data.

In [2]:
```
from sklearn import preprocessing
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error


lm = LinearRegression()
lm.fit(x_train, y_train)

data_estimate_train = lm.predict(x_train)
data_estimate_test = lm.predict(x_test)

rms_train = mean_squared_error(y_train, data_estimate_train)
rms_test = mean_squared_error(y_test, data_estimate_test)

mae_train = mean_absolute_error(y_train, data_estimate_train)
mae_test = mean_absolute_error(y_test, data_estimate_test)


mape_train = np.mean(np.abs((y_train - data_estimate_train) / y_train)) * 100
mape_test = np.mean(np.abs((y_test - data_estimate_test) / y_test)) * 100

#data_estimate_test
#y_test

print("rms_train = ",rms_train,"   rms_test = ",rms_test )
print("mae_train = " ,mae_train ,"   mae_test = ",mae_test )
print("mape_train = " ,float(mape_train),"   mape_test = ",float(mape_test))
```

```
rms_train =  0.0891450216954988    rms_test =  9.728717030786936e+23
mae_train =  0.21899348712457417    mae_test =  986198126028.8126
mape_train =  3.8707227290838286    mape_test =  17091148895280.447
```

We also did the feature selection using forward, backward and exhaustive search techniques.

```
----Selected Features from FWD Search----
('fico', 'dt_first_pi', 'flag_fthb', 'dt_matr', 'mi_pct', 'cnt_units', 'orig_upb', 'ltv', 'channel', 'loan_purpose', 'orig_loan
_term')
-Training Metrics-
R Squared: 0.8456364123444997
MAE: 0.1023524021511848
RMS: 0.14170788122554664
MAPE: 1.648424219686985
-Testing Metrics-
R Squared: 0.8456364123444997
MAE: 0.1023524021511848
RMS: 0.14170788122554664
MAPE: 1.648424219686985
```

```
----Selected Features from Exhaustive Search----
-Training Metrics-
R Squared: 0.8300393154859349
MAE: 0.10549713961889888
RMS: 0.14869480940205662
MAPE: 1.6987067994948768
-Testing Metrics-
R Squared: 0.8300393154859349
MAE: 0.10549713961889888
RMS: 0.14869480940205662
MAPE: 1.6987067994948768
```

Then, we tried out different models:

1. **RandomForestRegressor:**

In [3]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor

# y_train = pd.DataFrame(data_chunk_1['int_rt'].astype('float64'))

# y1 =  pd.DataFrame(data=y_train[1:,1:], index=y_train[1:,0],columns=y_train[0,1:])


rf = RandomForestRegressor(n_estimators=15)
rf.fit(x_train, y_train.values.ravel())

data_estimate_train = rf.predict(x_train)
data_estimate_test = rf.predict(x_test)


rms_train = mean_squared_error(y_train, data_estimate_train)
rms_test = mean_squared_error(y_test, data_estimate_test)

mae_train = mean_absolute_error(y_train, data_estimate_train)
mae_test = mean_absolute_error(y_test, data_estimate_test)

mape_train = np.mean(np.abs((y_train - data_estimate_train) / y_train)) * 100
mape_test = np.mean(np.abs((y_test - data_estimate_test) / y_test)) * 100


print("rms_train = ",rms_train,"   rms_test = ",rms_test )
print("mae_train = " ,mae_train ,"   mae_test = ",mae_test )
print("mape_train = " ,float(mape_train),"   mape_test = ",float(mape_test))
```

```
rms_train =  0.01539326990865208    rms_test =  0.11577747799682632
mae_train =  0.08870962278189018    mae_test =  0.25880206025743163
mape_train =  1.5688467656909961    mape_test =  4.476056436875322
```

2. **Neural Networks MLPRegressor**:

In [4]:
```
import operator
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.neural_network import MLPRegressor


nn = MLPRegressor()
nn.fit(x_train, y_train)

data_estimate_train = nn.predict(x_train)
data_estimate_test = nn.predict(x_test)

rms_train = mean_squared_error(y_train, data_estimate_train)
rms_test = mean_squared_error(y_test, data_estimate_test)

mae_train = mean_absolute_error(y_train, data_estimate_train)
mae_test = mean_absolute_error(y_test, data_estimate_test)


mape_train = np.mean(np.abs((y_train - data_estimate_train) / y_train)) * 100
mape_test = np.mean(np.abs((y_test - data_estimate_test) / y_test)) * 100

print("rms_train = ",rms_train,"   rms_test = ",rms_test )
print("mae_train = " ,mae_train ,"   mae_test = ",mae_test )
print("mape_train = " ,mape_train,"   mape_test = ",mape_test )
```

```
rms_train =  0.0852448158285891    rms_test =  0.6681216883842699
mae_train =  0.218341918593569    mae_test =  0.758721505861952
mape_train =  3.9034492368859364    mape_test =  12.901851477033214
```

Best model for us was RandomForestRegressor.
We also tried auto generating model techniques:
1. **TPOT**:

In [7]:
```
from tpot import TPOTRegressor
```

In [8]:
```
tpot = TPOTRegressor(generations=5, population_size=30,
                     offspring_size=None,
                     mutation_rate=0.9,
                     verbosity=3,cv=2,n_jobs=-1)
```

In [9]:
```
tpot.fit(x_train,y_train)
```

```
C:\Users\achar\Anaconda3\lib\importlib\_bootstrap.py:219: ImportWarning: can't resolve package from __spec__ or __package__, fa
lling back on __name__ and __path__
  return f(*args, **kwds)

Warning: xgboost.XGBRegressor is not available and will not be used by TPOT.
28 operators have been imported by TPOT.

Skipped pipeline #6 due to time out. Continuing to the next pipeline.
Skipped pipeline #9 due to time out. Continuing to the next pipeline.
Skipped pipeline #11 due to time out. Continuing to the next pipeline.
Skipped pipeline #15 due to time out. Continuing to the next pipeline.
Skipped pipeline #21 due to time out. Continuing to the next pipeline.
Skipped pipeline #29 due to time out. Continuing to the next pipeline.
Skipped pipeline #32 due to time out. Continuing to the next pipeline.
Skipped pipeline #35 due to time out. Continuing to the next pipeline.
_pre_test decorator: _random_mutation_operator: num_test=0 Expected n_neighbors <= n_samples,  but n_samples = 50, n_neighbors
= 62
_pre_test decorator: _random_mutation_operator: num_test=0 Unsupported set of arguments: The combination of penalty='l2' and lo
ss='epsilon_insensitive' are not supported when dual=False, Parameters: penalty='l2', loss='epsilon_insensitive', dual=False
_pre_test decorator: _random_mutation_operator: num_test=0 Unsupported set of arguments: The combination of penalty='l2' and lo
ss='epsilon_insensitive' are not supported when dual=False, Parameters: penalty='l2', loss='epsilon_insensitive', dual=False
_pre_test decorator: _random_mutation_operator: num_test=0 Found array with 0 feature(s) (shape=(50, 0)) while a minimum of 1 i
s required by MaxAbsScaler.
Pipeline encountered that has previously been evaluated during the optimization process. Using the score from the previous eval
```

2. **H2o AutoML:**

```
In [40]: aml.leaderboard.as_data_frame()
```

Out[40]:

	model_id	mean_residual_deviance	rmse	mse	mae	rmsle
0	DRF_1_AutoML_20181129_004622	0.020985	0.144861	0.020985	0.103502	0.020998
1	XRT_1_AutoML_20181129_004622	0.022624	0.150413	0.022624	0.108672	0.021833
2	GBM_grid_1_AutoML_20181129_004622_model_2	0.030812	0.175534	0.030812	0.131149	0.025499
3	StackedEnsemble_BestOfFamily_AutoML_20181129_0...	0.041197	0.202971	0.041197	0.147829	0.029397
4	GBM_4_AutoML_20181129_004622	0.041550	0.203838	0.041550	0.148709	0.029514
5	GBM_grid_1_AutoML_20181129_004622_model_9	0.041628	0.204029	0.041628	0.148562	0.029505
6	GBM_3_AutoML_20181129_004622	0.044823	0.211714	0.044823	0.154491	0.030654
7	GBM_grid_1_AutoML_20181129_004622_model_12	0.045371	0.213006	0.045371	0.155251	0.030909
8	GBM_grid_1_AutoML_20181129_004622_model_10	0.045497	0.213301	0.045497	0.155555	0.030915
9	GBM_grid_1_AutoML_20181129_004622_model_11	0.045931	0.214316	0.045931	0.155382	0.031014
10	GBM_grid_1_AutoML_20181129_004622_model_17	0.046257	0.215074	0.046257	0.155422	0.031092
11	GBM_1_AutoML_20181129_004622	0.046438	0.215495	0.046438	0.157484	0.031279
12	GBM_grid_1_AutoML_20181129_004622_model_16	0.046743	0.216200	0.046743	0.156435	0.031271
13	GBM_2_AutoML_20181129_004622	0.047174	0.217196	0.047174	0.157988	0.031452
14	GBM_grid_1_AutoML_20181129_004622_model_1	0.048815	0.220940	0.048815	0.159361	0.031993
15	StackedEnsemble_AllModels_AutoML_20181129_004622	0.048905	0.221144	0.048905	0.161477	0.032051
16	GBM_grid_1_AutoML_20181129_004622_model_4	0.050692	0.225148	0.050692	0.163058	0.032566
17	GBM_5_AutoML_20181129_004622	0.051099	0.226052	0.051099	0.163306	0.032694
18	GBM_grid_1_AutoML_20181129_004622_model_15	0.051642	0.227248	0.051642	0.165800	0.032941
19	DeepLearning_1_AutoML_20181129_004622	0.055004	0.234529	0.055004	0.169897	0.033989
20	DeepLearning_grid_1_AutoML_20181129_004622_mod...	0.055265	0.235085	0.055265	0.170337	0.034062

We assumed that, the best model fit is RandomForestRegressor because the results from above methods also states the similar model.
For financial crisis we ran our algorithm for the four rolling quarters and interpreted the results:

When the housing bubble of 2001-2007 burst, it caused a mortgage security meltdown. This contributed to a general credit crisis, which evolved into a worldwide financial crisis. Many critics have held the United States Congress - and its unwillingness to rein in Fannie Mae and Freddie Mac - responsible for the credit crisis.

In the fall of 2007, Freddie Mac shocked the market by announcing large credit-related loses, fueling the fire for the argument that the two companies pose a tremendous risk to the entire financial system. (http://www.investopedia.com/articles/economics/08/fannie-mae-freddie-mac-credit-crisis.asp)

The Federal Home Loan Mortgage Corporation (Freddie Mac) announced that it will no longer buy the most risky subprime mortgages and mortgage-related securities.

In July 24, 2007 Countrywide Financial Corporation warned of “difficult conditions.” This is evident from the Q32007 Testing measures as the difference between Training and Testing RMSE increased substantially by around 16%.

In November 1, 2007 financial market pressures intensified, reflected in diminished liquidity in interbank funding markets. This is evident in Q42007 Testing measures as the difference between Training and Testing RMSE increased substantially by around 22%.

In [50]:
```
##For Data Q12007 and Q22007

from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor

# y_train = pd.DataFrame(data_chunk_1['int_rt'].astype('float64'))

# y1 =  pd.DataFrame(data=y_train[1:,1:], index=y_train[1:,0],columns=y_train[0,1:])


rf = RandomForestRegressor(n_estimators=15)
rf.fit(x_train, y_train.values.ravel())

data_estimate_train = rf.predict(x_train)
data_estimate_test = rf.predict(x_test)


rms_train = mean_squared_error(y_train, data_estimate_train)
rms_test = mean_squared_error(y_test, data_estimate_test)

mae_train = mean_absolute_error(y_train, data_estimate_train)
mae_test = mean_absolute_error(y_test, data_estimate_test)

mape_train = np.mean(np.abs((y_train - data_estimate_train) / y_train)) * 100
mape_test = np.mean(np.abs((y_test - data_estimate_test) / y_test)) * 100


print("rms_train = ",rms_train,"   rms_test = ",rms_test )
print("mae_train = " ,mae_train ,"   mae_test = ",mae_test )
print("mape_train = " ,float(mape_train),"   mape_test = ",float(mape_test))
```

```
rms_train =  0.020004349442008802    rms_test =  0.15671855464617357
mae_train =  0.10213132143050459    mae_test =  0.30884401201951217
mape_train =  1.6445330461060808    mape_test =  4.949835307831991
```

In [52]:
```
##For Data Q22007 and Q32007

from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor

# y_train = pd.DataFrame(data_chunk_1['int_rt'].astype('float64'))

# y1 =  pd.DataFrame(data=y_train[1:,1:], index=y_train[1:,0],columns=y_train[0,1:])


rf = RandomForestRegressor(n_estimators=15)
rf.fit(x_train, y_train.values.ravel())

data_estimate_train = rf.predict(x_train)
data_estimate_test = rf.predict(x_test)


rms_train = mean_squared_error(y_train, data_estimate_train)
rms_test = mean_squared_error(y_test, data_estimate_test)

mae_train = mean_absolute_error(y_train, data_estimate_train)
mae_test = mean_absolute_error(y_test, data_estimate_test)

mape_train = np.mean(np.abs((y_train - data_estimate_train) / y_train)) * 100
mape_test = np.mean(np.abs((y_test - data_estimate_test) / y_test)) * 100


print("rms_train = ",rms_train,"   rms_test = ",rms_test )
print("mae_train = " ,mae_train ,"   mae_test = ",mae_test )
print("mape_train = " ,float(mape_train),"   mape_test = ",float(mape_test))
```

```
rms_train =  0.02218409865087728    rms_test =  1.5896536658495548
mae_train =  0.10782338528130112    mae_test =  1.1053679126710956
mape_train =  1.7101652225091704    mape_test =  16.476452425442922
```

In [54]:
```
##For Data Q32007 and Q42007

from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor

# y_train = pd.DataFrame(data_chunk_1['int_rt'].astype('float64'))

# y1 =  pd.DataFrame(data=y_train[1:,1:], index=y_train[1:,0],columns=y_train[0,1:])


rf = RandomForestRegressor(n_estimators=15)
rf.fit(x_train, y_train.values.ravel())

data_estimate_train = rf.predict(x_train)
data_estimate_test = rf.predict(x_test)


rms_train = mean_squared_error(y_train, data_estimate_train)
rms_test = mean_squared_error(y_test, data_estimate_test)

mae_train = mean_absolute_error(y_train, data_estimate_train)
mae_test = mean_absolute_error(y_test, data_estimate_test)

mape_train = np.mean(np.abs((y_train - data_estimate_train) / y_train)) * 100
mape_test = np.mean(np.abs((y_test - data_estimate_test) / y_test)) * 100


print("rms_train = ",rms_train,"   rms_test = ",rms_test )
print("mae_train = " ,mae_train ,"   mae_test = ",mae_test )
print("mape_train = " ,float(mape_train),"   mape_test = ",float(mape_test))
```

```
rms_train =  0.021504977943394836    rms_test =  0.26858666270797615
mae_train =  0.10621768328190262    mae_test =  0.4229598555525624
mape_train =  1.6002248272482211    mape_test =  6.857160179949282
```

In [57]:
```
##For Data Q42007 and Q12008

from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor

# y_train = pd.DataFrame(data_chunk_1['int_rt'].astype('float64'))

# y1 =  pd.DataFrame(data=y_train[1:,1:], index=y_train[1:,0],columns=y_train[0,1:])


rf = RandomForestRegressor(n_estimators=15)
rf.fit(x_train, y_train.values.ravel())

data_estimate_train = rf.predict(x_train)
data_estimate_test = rf.predict(x_test)


rms_train = mean_squared_error(y_train, data_estimate_train)
rms_test = mean_squared_error(y_test, data_estimate_test)

mae_train = mean_absolute_error(y_train, data_estimate_train)
mae_test = mean_absolute_error(y_test, data_estimate_test)

mape_train = np.mean(np.abs((y_train - data_estimate_train) / y_train)) * 100
mape_test = np.mean(np.abs((y_test - data_estimate_test) / y_test)) * 100


print("rms_train = ",rms_train,"   rms_test = ",rms_test )
print("mae_train = " ,mae_train ,"   mae_test = ",mae_test )
print("mape_train = " ,float(mape_train),"   mape_test = ",float(mape_test))
```

```
rms_train =  0.025342854178966888    rms_test =  0.45380985569808596
mae_train =  0.11562012563287243    mae_test =  0.573147945969758
mape_train =  1.8184349750590787    mape_test =  10.176160913262445
```

B. Classification:

In Loan Performance file, we have a column name delq_sts on which we should predict the Loan Delinquency Status by training the data on the quarter provided and predict the result for the next quarter.

For the classification we first read the data quarterly from historical data and then played with missing values and outliers.

In [24]:
```
rowcnt = len(df.index)
# Example data
features = []
missngpercnt = []
if rowcnt > 0:
    for x in df:
        features.append(x)
        prcnt = float(len(df[df[x].isnull()])) / float(rowcnt) * 100.0
        missngpercnt.append(prcnt)
else:
    features = df.columns
    missngpercnt = [0] * len(features)


missingprcntdata = pd.DataFrame({'Features':features,'MissingPercentage':missngpercnt})
missingprcntdata = missingprcntdata.sort_values('MissingPercentage',ascending =False)
missingprcntdata
```

Out[24]:

	Features	MissingPercentage
24	def_py_mod	99.952333
23	step_mod_flag	99.952333
7	flag_mod	99.952333
13	mi_recoveries	99.932333
20	misc_costs	99.932333
15	non_mi_recoveries	99.932333
14	net_sale_proceeds	99.932333
18	maint_pres_costs	99.932333
19	taxes_ins_costs	99.932333
16	expenses	99.932333
17	legal_costs	99.932333
21	actual_loss	99.932333
12	dt_lst_pi	99.888000
9	dt_zero_bal	98.825067
8	cd_zero_bal	98.825067
6	repch_flag	98.824733
22	modcost	98.673800
25	eltv	98.457933
1	svcg_cycle	0.000000
11	non_int_brng_upb	0.000000
10	current_int_rt	0.000000
5	mths_remng	0.000000
4	loan_age	0.000000
3	delq_sts	0.000000
2	current_upb	0.000000
0	id_loan	0.000000

As we can see there are many zeros in maximum columns so we decided to remove them for future purpose.

In [66]:
```
test_df.columns = ['id_loan', 'svcg_cycle', 'current_upb', 'delq_sts', 'loan_age', 'mths_remng',
                   'repch_flag', 'flag_mod', 'cd_zero_bal',
                   'dt_zero_bal', 'current_int_rt', 'non_int_brng_upb', 'dt_lst_pi', 'mi_recoveries',
                   'net_sale_proceeds', 'non_mi_recoveries', 'expenses', 'legal_costs',
                   'maint_pres_costs', 'taxes_ins_costs', 'misc_costs', 'actual_loss', 'modcost','step_mod_flag',
                    'def_py_mod', 'eltv']
```

In [67]:
```
test_df.current_upb =test_df.current_upb.astype('float64')
test_df.non_int_brng_upb =test_df.non_int_brng_upb.astype('float64')
test_df.current_int_rt =test_df.current_int_rt.astype('float64')

test_df[['svcg_cycle', 'loan_age', 'mths_remng']] = test_df[
       ['svcg_cycle', 'loan_age', 'mths_remng']].astype('int64')

test_df[['id_loan', 'delq_sts']] = test_df[['id_loan', 'delq_sts']].astype('str')

test_df['delq_sts'] = [999 if x=='R' else x for x in (test_df['delq_sts'].apply(lambda x: x))]
test_df['delq_sts'] = [0 if x=='XX' else x for x in (test_df['delq_sts'].apply(lambda x: x))]

test_df[['delq_sts']] = test_df[['delq_sts']].astype('int64')
test_df['new_delinq'] = (test_df.delq_sts > 0).astype(int)

test_df.drop('delq_sts', axis = 1,inplace=True)
```

In [68]:
```
testcols = ['svcg_cycle','current_upb','loan_age',
            'mths_remng','current_int_rt','non_int_brng_upb']
y_test = test_df['new_delinq']
Test_DF = test_df[testcols]
```

In this, we converted the datatypes and removed the delinquent row from the x variable and added it to the y variable as we will be predicting delinquency.

We then ran our data through various models and computed the confusion matrix for each one.

1. **Logistic Regression:**

Accuracy and Confusion Metrics:

In [69]:
```
model = LogisticRegression()
mod_fit = model.fit(Train_DF, y_train)
pred = mod_fit.predict(Test_DF)
metrics.accuracy_score(y_test, pred)
print(metrics.accuracy_score(y_test, pred))
```

```
0.954674
```

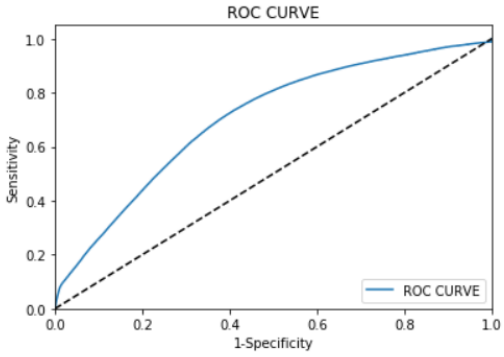
In [71]:
```
cf = confusion_matrix(y_test, pred, labels=None, sample_weight=None)
numDelinqProper = cf[1][1]
numnondelinqimproper = cf[0][1]
numRecordsInDataset = y_test.count()
numPredictedDelinq = cf[1][0] + cf[1][1]
numActualDelinq = y_test[y_test == 1].count()

record = str(numActualDelinq) + "," + str(numPredictedDelinq) + "," + str(numRecordsInDataset) + "," + str(numDelinqProper) + ","

print(record)
```

```
22618,22618,500000,19,64
```

In [72]:
```
print (metrics.confusion_matrix(y_test,pred))
```

```
[[477318     64]
 [ 22599     19]]
```

ROC curve:



2. **RandomForestClassifier:**
Accuracy and Confusion Metrics:

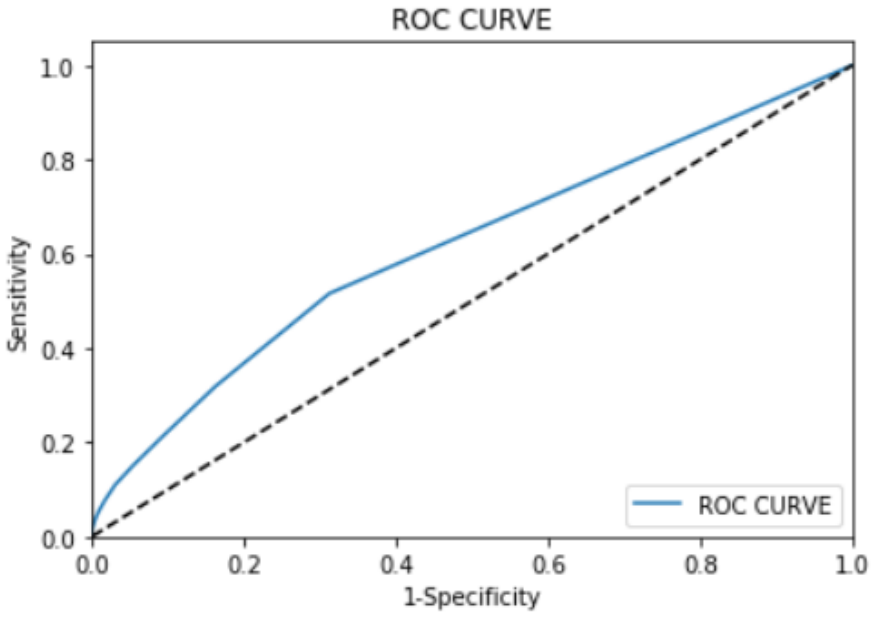
In [76]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
mod_fit = rf.fit(Train_DF, y_train)
pred = mod_fit.predict(Test_DF)
metrics.accuracy_score(y_test, pred)
print(metrics.accuracy_score(y_test, pred))
```

```
0.941926
```

In [77]:
```
cf = confusion_matrix(y_test, pred, labels=None, sample_weight=None)
numDelinqProper = cf[1][1]
numnondelinqimproper = cf[0][1]
numRecordsInDataset = y_test.count()
numPredictedDelinq = cf[1][0] + cf[1][1]
numActualDelinq = y_test[y_test == 1].count()

record = str(numActualDelinq) + "," + str(numPredictedDelinq) + "," + str(nu

print(record)

print (metrics.confusion_matrix(y_test,pred))
```

```
22618,22618,500000,1731,8150
[[469232   8150]
 [ 20887   1731]]
```

ROC Curve:



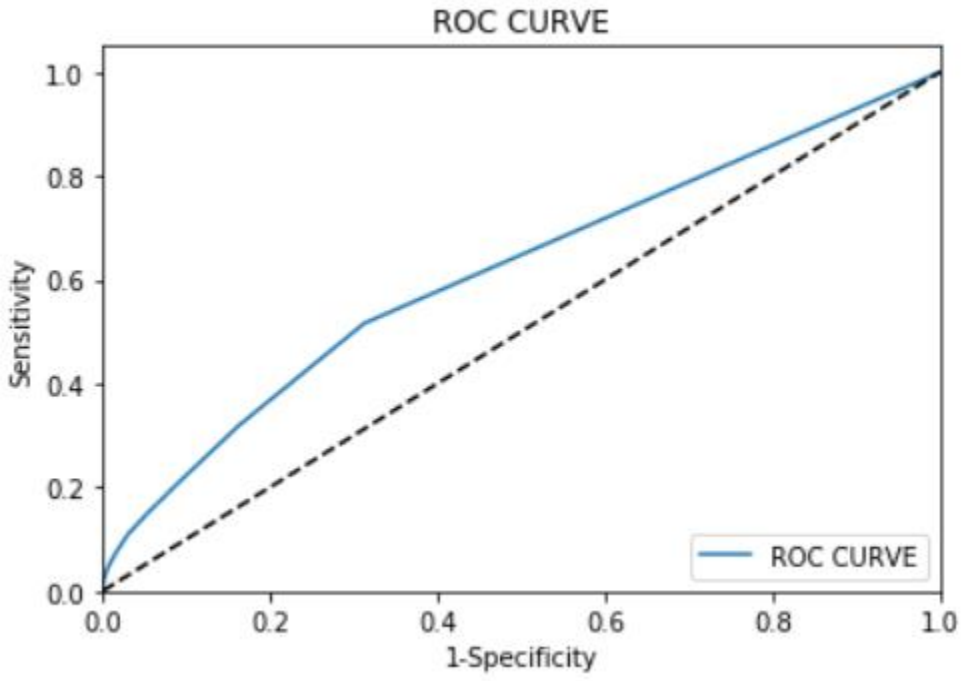
3. **Neural Network:**

Accuracy and Confusion Metrics:

```
In [80]: from sklearn.neural_network import MLPClassifier
         nn = MLPClassifier()
         mod_fit = nn.fit(Train_DF, y_train)
         pred = mod_fit.predict(Test_DF)
         metrics.accuracy_score(y_test, pred)
         print(metrics.accuracy_score(y_test, pred))

         0.954764
```

```
In [81]: cf = confusion_matrix(y_test, pred, labels=None, sample_weight=None)
         numDelinqProper = cf[1][1]
         numnondelinqimproper = cf[0][1]
         numRecordsInDataset = y_test.count()
         numPredictedDelinq = cf[1][0] + cf[1][1]
         numActualDelinq = y_test[y_test == 1].count()

         record = str(numActualDelinq) + "," + str(numPredictedDelinq) + "," + str(numRecor

         print(record)

         print (metrics.confusion_matrix(y_test,pred))

         22618,22618,500000,0,0
         [[477382      0]
          [ 22618      0]]
```

ROC Curve:



We also ran the TPOT and H2o automl classifiers to get the best model.

Comment on the quality of the model and it's outputs. What can you do to do better? Would you recommend using this model to predict delinquents in the next quarter? Justify your answers:

According to us, the model fir for the data would be random forest, because it is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the class's output by individual trees.

Advantages of Random Forest:

- Honestly, not requiring cross-validation alone for model selection significantly speeds training by 10x-100x or more.
- Resistance to over training.
- It can handle data without preprocessing
- It can handle missing values by itself.

Contributions by each team member: