Midterm Project

Prediction & Classification Modelling

on

Freddie Mac's Single Family Loans Dataset

A project by

Team 1

Vishal Satam

Manasi Dalvi

At Northeastern University under the guidance of

Professor Srikanth Krishnamurthy

Introduction

This project aims to perform a detailed analysis on the Single Family Home Mortgages Dataset by Freddie Mac and build a model to predict interest rates and classify loans that will default. The dataset being used is the Single Family Loan-Level Dataset which was released by Freddie Mac to increase transparency and enable investors build more accurate predictive models. We will login automatically based on the access keys that have been passed, scrape the links and perform exploratory data analysis on sample dataset from the years 2005-2016. We will summarize the dataset based on the sample datasets available at the API. We will begin with Exploratory data analysis to understand the nature of the data and understand the different features with correlations between them. We use predictive modelling techniques using Linear Regression, Random Forest and Neural Network algorithms to predict the interest rate for a loan based on different attributes of a loan. We will look at Lasso Regression, recursive feature elimination, F regression and PCA for performing feature extraction and selection. We analyze the impact of the market crash in 2008 and how the interest rate prediction algorithm performs in certain timeframes. Classification of a loan as delinquent or non-delinquent has been performed using machine learning algorithms. We have explored Logistic Regression, Random Forest and Neural Network algorithms to understand which one gives the best results. This algorithm has been tested over the period of 1999-2013. We use one quarter's data to train the model in Prediction as well as Classification and then apply this model to predict and classify interest rate and delinquents respectively for the consecutive quarter. A comparative analysis has been performed for how well the prediction and classification algorithms performs for different quarters.

Dataset

The Single Family Loans dataset provided by Freddie Mac is used for this project. The dataset is divided into 2 parts. The first part is the origination data for different loans. The second part is the monthly performance file in which the performance of every loan is tracked per monthly reporting period. The dataset has many interesting parameters about every loan which we will explore more in the Exploratory Data Analysis section. It contains data from first quarter of 1999 till 2nd Quarter of 2016. Freddie Mac also provides sample datasets as a part of their API. These sample datasets contain data from the main files which is sampled based on Simple Random Sampling.

http://www.freddiemac.com/research/datasets/sf_loanlevel_dataset.html

Docker and Execution Instructions

The entire project is available on github and has been put into a docker container. You can visit the below github link for more details. **Minimum memory requirements for docker virtual machine– 6 GB RAM**

Docker image on docker hub → vishalsatam1988/midterm

Github Link for the Project

https://github.com/vishalsatam/PredictiveModellingOnFreddieMacLoans

Part 1 Exploratory Data Analysis

Part 1.1 Download

The dataset is very large and it is not practical to perform data analysis based on the entire dataset. For this reason, we are performing the exploratory data analysis on the sample files provided by Freddie Mac's API. We are using the BeautifulSoup library of Python to scrape the web page for all the anchor tags. We then identify the links that are relevant for this task and get all the links sample from 2005 – 2016.

We then download the data which is a zip file, extract it and then clean it.

```
opcookie=downloaderfunctions.getSession(AUTHURL,username,password)
searchList=list(range(startyear,endyear))
origcombinefilelist=[]
perfcombinefilelist=[]
```

```
def getDownloadLinksFrom(filetype,searchList,downloadurl,username,password,authcookie):
    params = {"username":username,"password":password,"action":"acceptTandC","accept":"Yes","acceptSubmit":"Continue"}
    page = requests.post(downloadurl, data=params,cookies=authcookie)
    soup = BeautifulSoup(page.content, 'html.parser')
    downloadlinks=soup.find_all('a')
    downlink=""
    downfilename=""
    downfilesDict={}
    for link in downloadlinks:
        for filename in searchList:
            if (str(filename) in str(link.get_text())) & (filetype in str(link.get_text())):
                print("download from = "+link.attrs.get('href'))
                downlink=str(link.attrs.get('href'))
                downfilename=link.get_text()
                downfilesDict[downfilename]=downlink

    return downfilesDict
```

After this, the first step is summarization.

Part 1.2 Summarization

We login into the dataset automatically using the access credentials provided and programmatically download the data. The data is downloaded as a zip file, automatically extracted and cleaned. Summarization is performed on these clean files to reduce the size further to facilitate data analysis.

Data Wrangling and Cleaning

We are performing data wrangling in 3 phases.

1. Replace Blank spaces

   We have through missing data analysis that in the performance file, there are many columns for which we have missing values. In the origination file, we have only one column.
   Origination File

	Features	MissingPercentage
25	SUPCONFORMFLG	100.000
4	MSA	17.342
2	FIRSTTIMEHMBUYFLG	6.994
14	PPM	0.226
9	DTI	0.114
22	NUMBORR	0.046
8	CLTV	0.004
11	ORIGLTV	0.004
16	PROPSTATE	0.000
24	SERVICERNAME	0.000
23	SELLERNAME	0.000
21	ORIGTERM	0.000
20	PURPOSE	0.000
19	LOANSEQNUM	0.000
18	PROPZIP	0.000
17	PROPTYPE	0.000
0	CREDITSCORE	0.000
15	PRODTYPE	0.000
1	FIRSTPYMNTDATE	0.000
12	ORIGINTRATE	0.000
10	ORIGUPB	0.000
7	OCCUSTAT	0.000
6	NUMUNITS	0.000
5	MI	0.000
3	MATURITYDATE	0.000
13	CHANNEL	0.000

Performance Flag

	Features	MissingPercentage
7	MODIFYFG	99.858153
15	NONMIREVOVERIES	99.825609
21	ACTLOSSCALC	99.825609
20	MISCEXPENSES	99.825609
19	TAXANDINSU	99.825609
18	MAINTPRESERVCOST	99.825609
17	LEGALCOST	99.825609
16	EXPENSES	99.825609
13	MIRECOVERIES	99.825609
14	NETSALESPROCEEDS	99.825609
12	DDLPI	99.744607
8	ZEROBALCODE	98.234025
9	ZEROBALEFFDATE	98.234025
6	REPURFLG	98.232006
22	MODIFICATIONCOST	97.991958
0	LOANSEQNUM	0.000000
1	MONTHLYREPORTPERIOD	0.000000
10	CURRINTRATE	0.000000
5	REMMONTHSTOMAT	0.000000
4	AGE	0.000000
3	CURRDELSTAT	0.000000
2	CURRACTUPB	0.000000
11	CURRDEFUPB	0.000000

2. Remove Missing Columns
   We have removed columns for which there was more than 90% missing data.

3. Convert Data Types
   The data types of the columns that we select are not proper. We will have to convert the datatypes to correspond to the correct ones.

Summarizing sample files

For origination files, since these are smaller in size, we are only performing data wrangling on them. But the performance files are large, hence we first perform data wrangling and then summarize the information present in the different columns as Min, Max for each column.

```
def SUMM_CUR_ACT_UPB(groupdf):
    return {'MIN_CUR_ACT_UPB': groupdf.min(), 'MAX_CUR_ACT_UPB': groupdf.max()}
def SUMM_CUR_LOAN_DELQ_STAT(groupdf):
    return {'MIN_CUR_LOAN_DELQ_STAT': groupdf.min(), 'MAX_CUR_LOAN_DELQ_STAT': groupdf.max()}
def SUMM_MONTHLY_REPORT_PERIOD(groupdf):
    return {'MIN_MONTHLY_REPORT_PERIOD': groupdf.min(), 'MAX_MONTHLY_REPORT_PERIOD': groupdf.max()}
def SUMM_LOAN_AGE(groupdf):
    return {'MIN_LOAN_AGE': groupdf.min(), 'MAX_LOAN_AGE': groupdf.max()}
def SUMM_MONTHS_LEGAL_MATURITY(groupdf):
    return {'MIN_MONTHS_LEGAL_MATURITY': groupdf.min(), 'MAX_MONTHS_LEGAL_MATURITY': groupdf.max()}
def SUMM_REPURCHASE_FLAG(group):
    if 'Y' in group.unique():
        return {'REPURCHASED':'Y'}
    else:
        return {'REPURCHASED':'N'}
def SUMM_ZERO_BAL_CODE(groupdf):
    return {'MIN_ZERO_BAL_CODE': groupdf.min(), 'MAX_ZERO_BAL_CODE': groupdf.max()}
def SUMM_CURR_INTERESTRATE(groupdf):
    return {'MIN_CURR_INTERESTRATE': groupdf.min(), 'MAX_CURR_INTERESTRATE': groupdf.max()}
def SUMM_CURR_DEF_UPB(groupdf):
    return {'MIN_CURR_DEF_UPB': groupdf.min(), 'MAX_CURR_DEF_UPB': groupdf.max()}
def SUMM_MI_RECOVERIES(groupdf):
    return {'MIN_MI_RECOVERIES': groupdf.min(), 'MAX_MI_RECOVERIES': groupdf.max()}
def SUMM_NON_MI_RECOV(groupdf):
    return {'MIN_NON_MI_RECOV': groupdf.min(), 'MAX_NON_MI_RECOV': groupdf.max()}
def SUMM_EXPENSES(groupdf):
    return {'MIN_EXPENSES': groupdf.min(), 'MAX_EXPENSES': groupdf.max()}
def SUMM_LEGAL_COSTS(groupdf):
    return {'MIN_LEGAL_COSTS': groupdf.min(), 'MAX_LEGAL_COSTS': groupdf.max()}
def SUMM_TAX_INSUR(groupdf):
    return {'MIN_TAX_INSUR': groupdf.min(), 'MAX_TAX_INSUR': groupdf.max()}
def SUMM_ACT_LOSS_CALC(groupdf):
    return {'MIN_ACT_LOSS_CALC': groupdf.min(), 'MAX_ACT_LOSS_CALC': groupdf.max()}
def SUMM_MOD_COST(groupdf):
    return {'MIN_MOD_COST': groupdf.min(), 'MAX_MOD_COST': groupdf.max()}
```

```
def summarizeperformancefiles(df_summary):
    summary_df=pd.DataFrame()
    summary_df['LOAN_SEQ_NO']=df_summary['LOAN_SEQ_NO'].drop_duplicates()
    summary_df=summary_df.join((df_summary['CUR_ACT_UPB'].groupby(df_summary['LOAN_SEQ_NO']).apply(SUMM_CUR_ACT_UPB).unstack()),on='LOAN_SEQ_NO')
    summary_df=summary_df.join((df_summary['CUR_LOAN_DELQ_STAT'].groupby(df_summary['LOAN_SEQ_NO']).apply(SUMM_CUR_LOAN_DELQ_STAT).unstack()),on='LOAN_SEQ_NO')
    summary_df=summary_df.join((df_summary['CURR_INTERESTRATE'].groupby(df_summary['LOAN_SEQ_NO']).apply(SUMM_CURR_INTERESTRATE).unstack()),on='LOAN_SEQ_NO')
    summary_df=summary_df.join((df_summary['MONTHLY_REPORT_PERIOD'].groupby(df_summary['LOAN_SEQ_NO']).apply(SUMM_MONTHLY_REPORT_PERIOD).unstack()),on='LOAN_SEQ_NO')
    summary_df=summary_df.join((df_summary['LOAN_AGE'].groupby(df_summary['LOAN_SEQ_NO']).apply(SUMM_LOAN_AGE).unstack()),on='LOAN_SEQ_NO')
    summary_df=summary_df.join((df_summary['MONTHS_LEGAL_MATURITY'].groupby(df_summary['LOAN_SEQ_NO']).apply(SUMM_MONTHS_LEGAL_MATURITY).unstack()),on='LOAN_SEQ_NO')
    summary_df=summary_df.join((df_summary['REPURCHASE_FLAG'].groupby(df_summary['LOAN_SEQ_NO']).apply(SUMM_REPURCHASE_FLAG).unstack()),on='LOAN_SEQ_NO')
    summary_df=summary_df.join((df_summary['ZERO_BAL_CODE'].groupby(df_summary['LOAN_SEQ_NO']).apply(SUMM_ZERO_BAL_CODE).unstack()),on='LOAN_SEQ_NO')
    summary_df=summary_df.join((df_summary['CURR_DEF_UPB'].groupby(df_summary['LOAN_SEQ_NO']).apply(SUMM_CURR_DEF_UPB).unstack()),on='LOAN_SEQ_NO')
    summary_df=summary_df.join((df_summary['MI_RECOVERIES'].groupby(df_summary['LOAN_SEQ_NO']).apply(SUMM_MI_RECOVERIES).unstack()),on='LOAN_SEQ_NO')
    summary_df=summary_df.join((df_summary['NON_MI_RECOV'].groupby(df_summary['LOAN_SEQ_NO']).apply(SUMM_NON_MI_RECOV).unstack()),on='LOAN_SEQ_NO')
    summary_df=summary_df.join((df_summary['EXPENSES'].groupby(df_summary['LOAN_SEQ_NO']).apply(SUMM_EXPENSES).unstack()),on='LOAN_SEQ_NO')
    summary_df=summary_df.join((df_summary['LEGAL_COSTS'].groupby(df_summary['LOAN_SEQ_NO']).apply(SUMM_LEGAL_COSTS).unstack()),on='LOAN_SEQ_NO')
    summary_df=summary_df.join((df_summary['TAX_INSUR'].groupby(df_summary['LOAN_SEQ_NO']).apply(SUMM_TAX_INSUR).unstack()),on='LOAN_SEQ_NO')
    summary_df=summary_df.join((df_summary['ACT_LOSS_CALC'].groupby(df_summary['LOAN_SEQ_NO']).apply(SUMM_ACT_LOSS_CALC).unstack()),on='LOAN_SEQ_NO')
    summary_df=summary_df.join((df_summary['MOD_COST'].groupby(df_summary['LOAN_SEQ_NO']).apply(SUMM_MOD_COST).unstack()),on='LOAN_SEQ_NO')
    return summary_df
```

We are also adding a new field called Quarter and Year so that we will be able to segregate the loans easily.

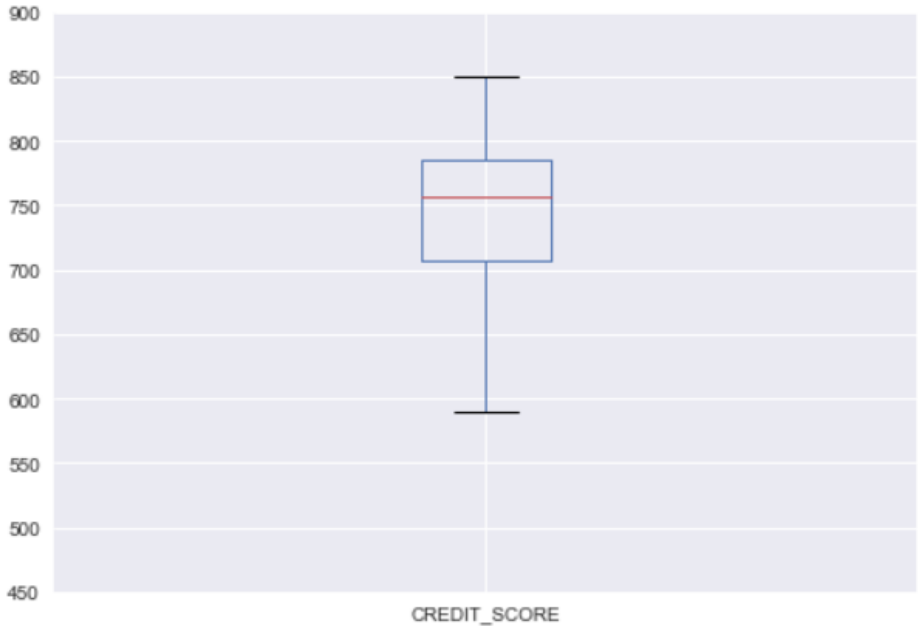
```
def cleanAndMergeOrig(downpath,origcombinefilelist):
    header=True
    with open(downpath, 'w',encoding='utf-8',newline="") as file:
        for filepath in origcombinefilelist:
            st = datetime.datetime.fromtimestamp(time.time()).strftime('%Y-%m-%d %H:%M:%S.%f')
            print("Summarizing : "+st + ":"+filepath)
            logging.info("Summarizing : "+st + ":"+filepath)
            df_orig=cleanOrig(filepath)
            df_orig['OG_YEAR'] = ['19'+x if x=='99' else '20'+x for x in (df_orig['LOAN_SEQ_NO'].apply(lambda x: x[2:4]))]
            df_orig['OG_QUARTER'] = [x for x in (df_orig['LOAN_SEQ_NO'].apply(lambda x: x[4:6]))]
            if header is True:
                df_orig.to_csv(file, mode='a', header=True,index=False)
                header = False
            else:
                df_orig.to_csv(file, mode='a', header=False,index=False)
            st = datetime.datetime.fromtimestamp(time.time()).strftime('%Y-%m-%d %H:%M:%S.%f')
            print("End of Summarizing : "+st + ":"+filepath)
            os.remove(filepath)
```

Part 1.3 Exploratory Data Analysis

We are performing the Exploratory data analysis on this dataset for the range 2005 – 2011. We have been asked to perform data analysis for the data between 2007-2009 but we have expanded the range by 2 years on either side so that we can get a better picture of the history and future of what happened around the market crash in 2008. We have an interesting dataset with one file containing cross-sectional data and the other file containing time-series data.

Credit Scores

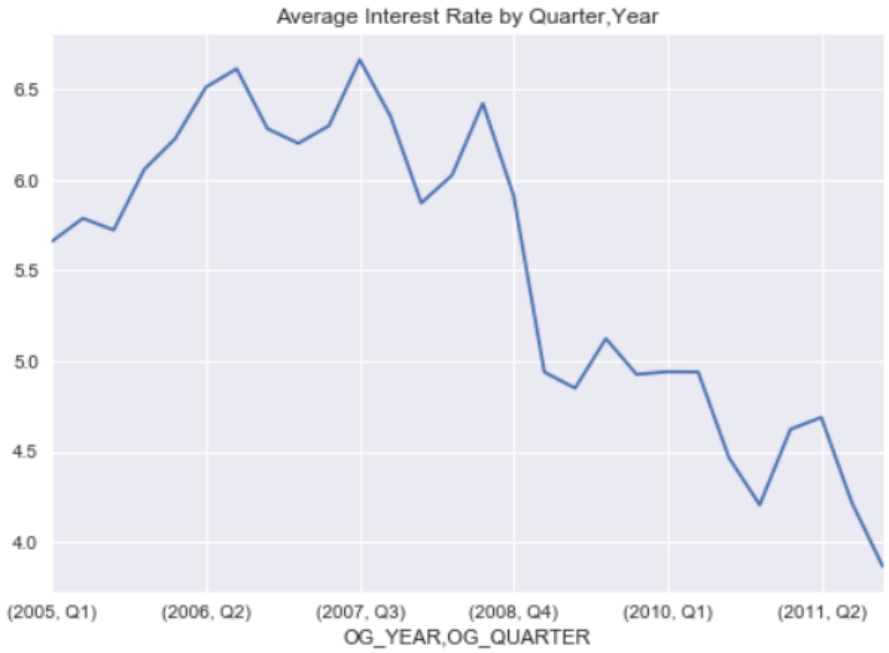
The following plot shows how the credit scores have been distributed. As we can see, Freddie Mac has been very careful in selecting the type of borrowers. Because majority of mortgages have been provided to people with good credit scores (between 700-830)



Average Interest Rate per Quarter

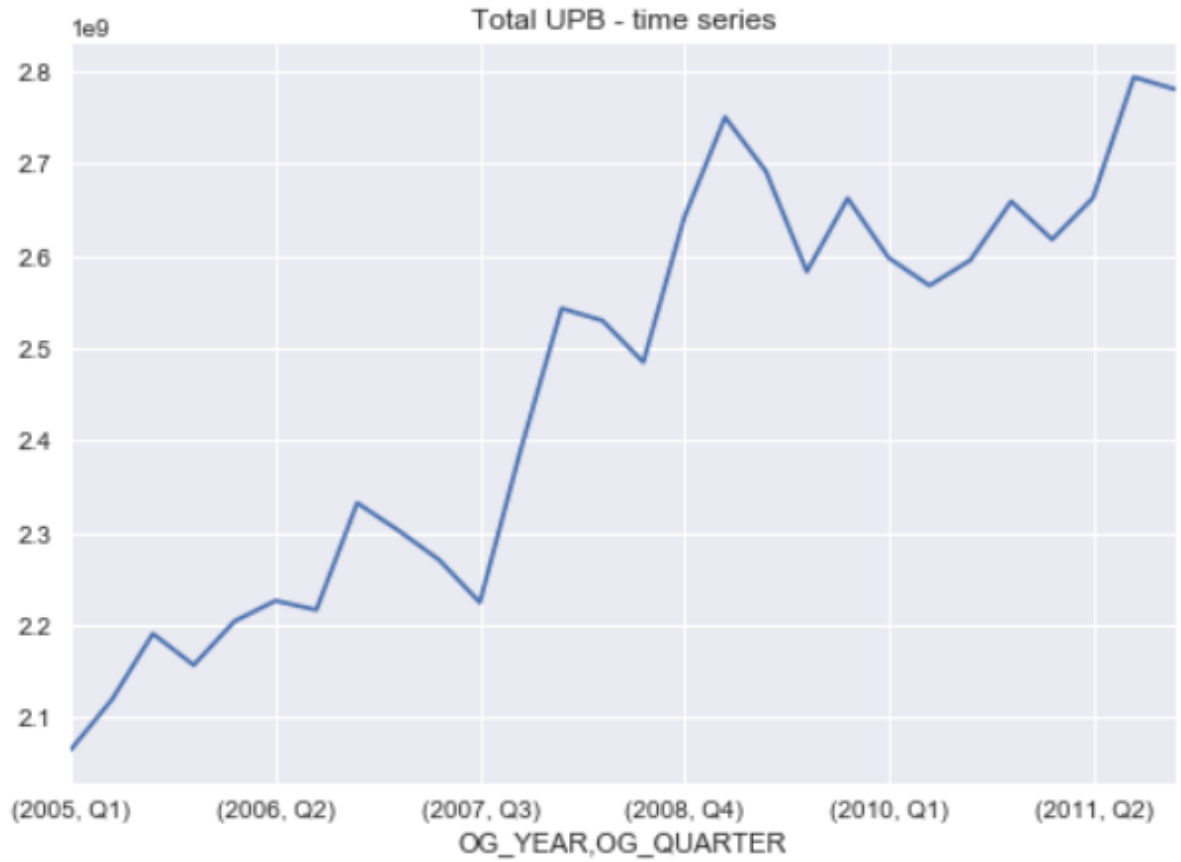
The average interest rate as we can see from the below plot, the average interest rate per quarter is fairly constant between 6.25 and 6.5 but we can also see that during 2008, during the market crash the interest rates fell by about 1 percent. Historically, Because the market had crashed, mortgage interest rates had dropped to attract more customers.

But we can see that the Average LTV, CLTV, DTI has remained fairly unaffected between 2005 and 2011.

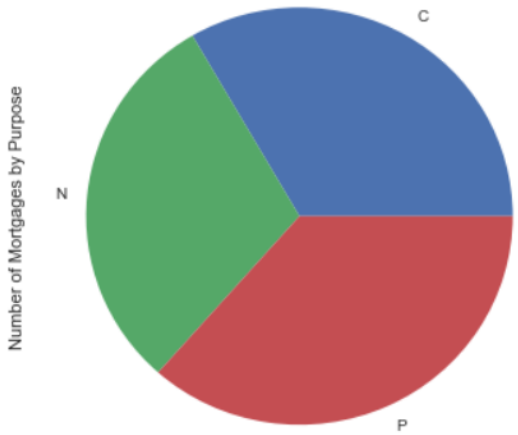


Total Unpaid Principal Balance per quarter

The below graph shows that the total UPB grew over time. This means that more loans were given out every quarter. The spike began from 2007. This is an indicator of how the housing market might have crashed in 2008. The total unpaid principal balance kept rising sharply.

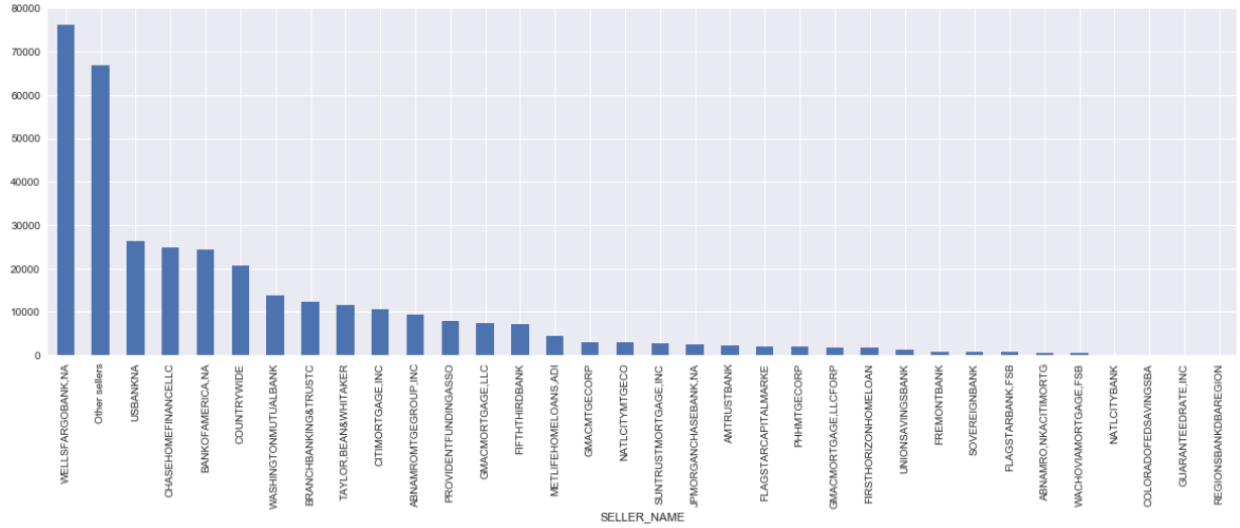


The number of loans are fairly evenly distributed between Purchased, Cash out for refinance and No cash out for refinance loans.



Top Sellers

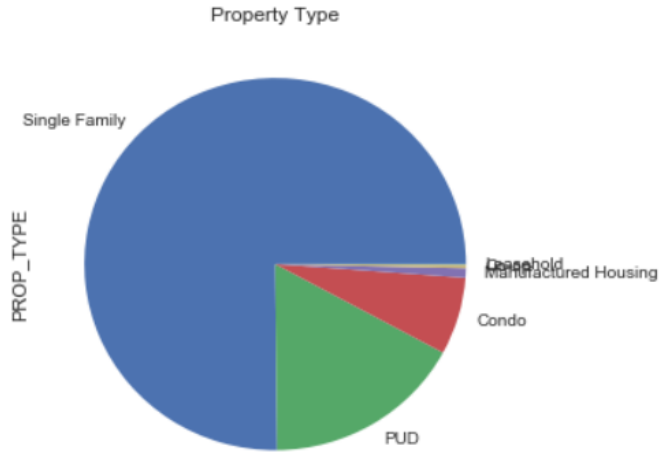
The top seller during 2005 – 2011 was WellsFargoBank, followed by USBank, Chase, BankOfAmerica

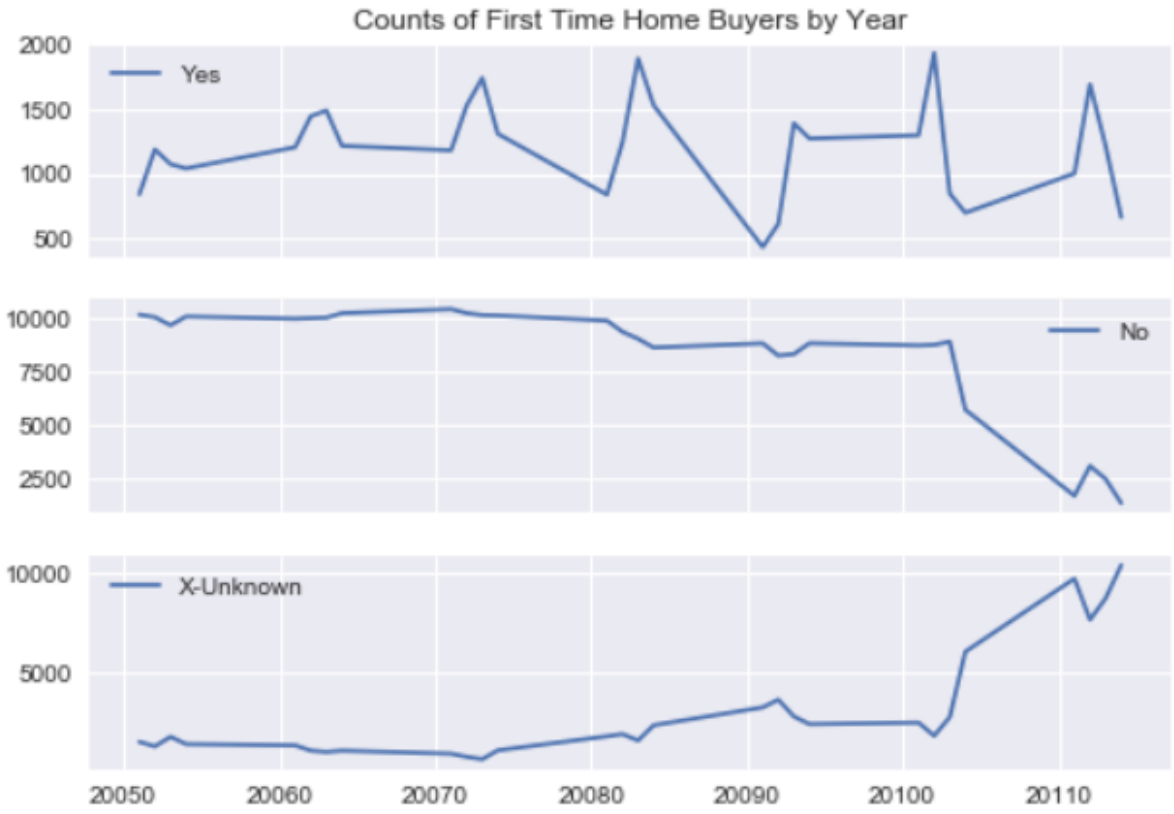


Total UPB by State

California had the highest Unpaid Principal Balance during 2005 – 2011. The other larger states for which the UPB was high was Florida and Texas.

As expected, this dataset contains maximum number of Single Family loans. There are other major types of properties for which Freddie Mac has given out loans. These include Condos and Planned Unit dev.





Total UPB by type of property

The investments made in Condos and Manufactured Units dropped during 2008 market crash. That is why the prices of homes plummeted.


Variability of Sum of UPB
1e8
2.2
2.0
1.8
1.6
1.4
1.2
Condo
1e7
1.4
1.2
1.0
0.8
0.6
0.4
0.2
Co-Op
3000000
2000000
1000000
0
Lease Household
1e7
2.5
2.0
1.5
1.0
0.5
Manufactured Household
1e8
7
6
5
4
3
1e9
2.0
1.9
1.8
1.7
1.6
Single Family
20050
20060
20070
20080
20090
20100
20110

Delinquent Analysis

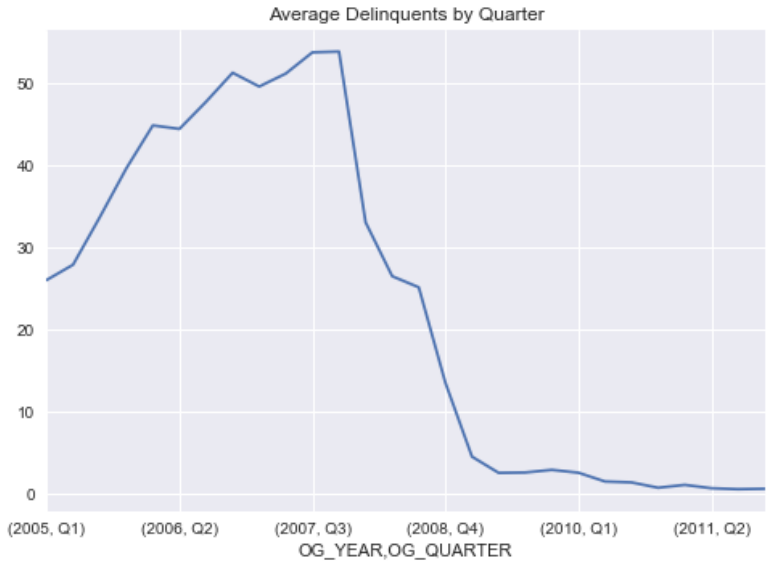
Average Delinquents by Quarter

The number of mortgages that went delinquent peaked during the market crash. This peak of delinquent loans was the main cause for the market crash. The delinquencies started rising from 2005 till 2007. More delinquencies resulted in heavy losses incurred by mortgage companies and eventually the entire economy.

Aftermath : Freddie Mac appears to have made changes to their policies. After the market recovery started in 2008 because as we can see the number of delinquents has been near 0 after 2008 till 2011.

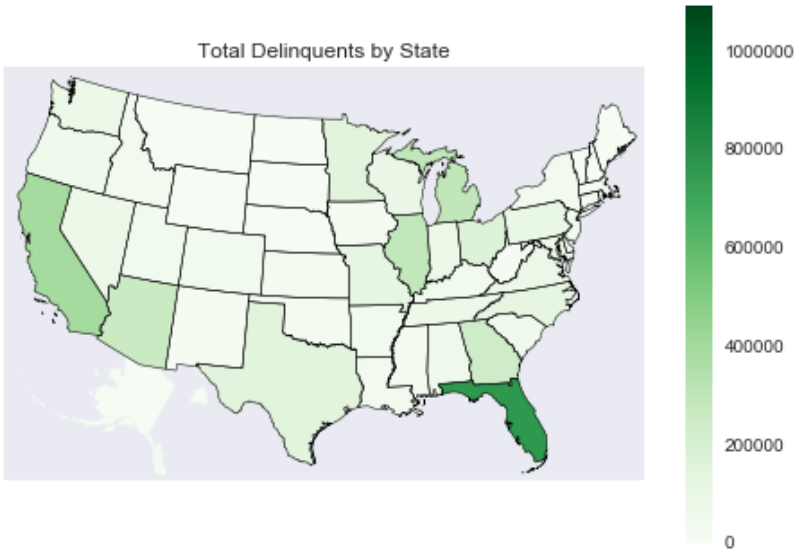
We can understand from the below link that during the 2nd Quarter of 2007, S&P started putting a credit watch on the a lot of mortgage backed securities which might have led to increased scrutiny. Also, Freddie Mac stopped buying the risky subprime mortgages during the first quarter of 2007

https://www.stlouisfed.org/financial-crisis/full-timeline



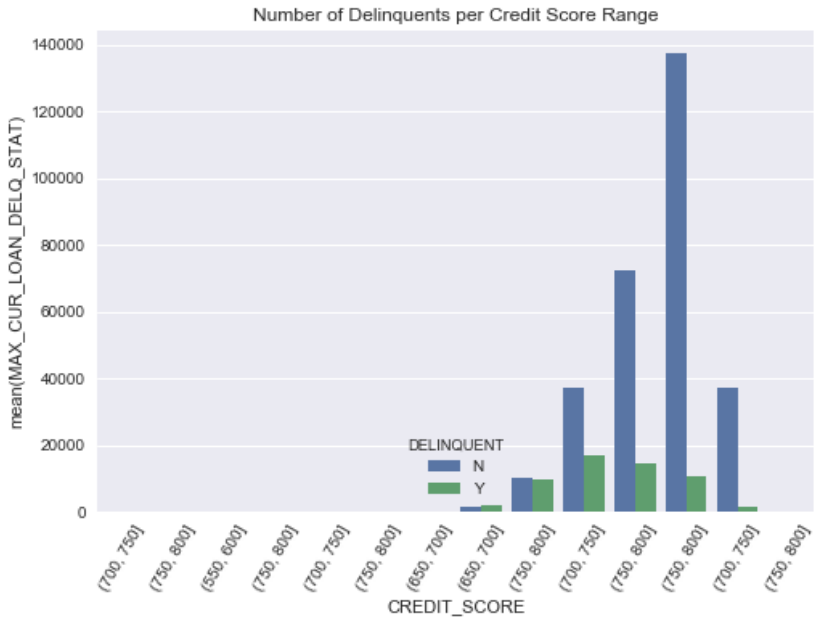
Total Delinquents By State

If we see historically, Florida was a key player in the mortgage defaults that occurred which eventually led to the market crash. This is clearly evident in our data. Florida has the highest number of delinquents.



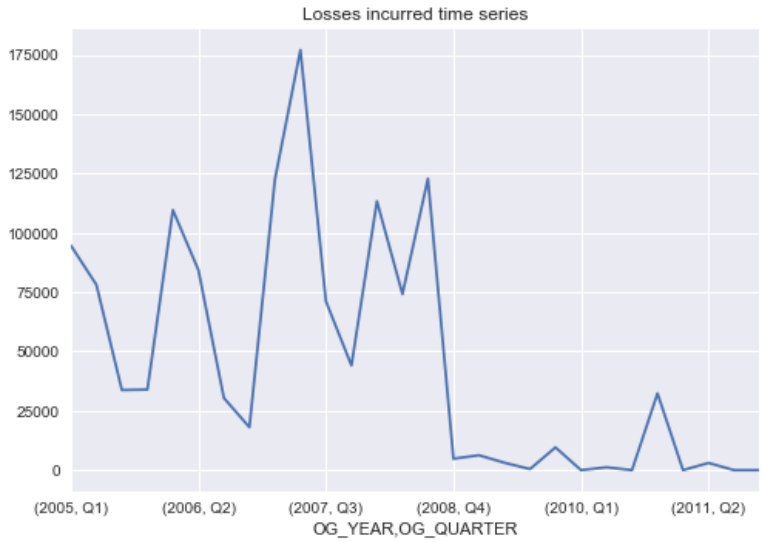
Number of Delinquents per credit score range.

If we bin the data in ranges of 50 and plot the total number of delinquents, we can see that the number of delinquents starts dropping with higher credit scores and the number of non-delinquents increases with people with higher credit scores.

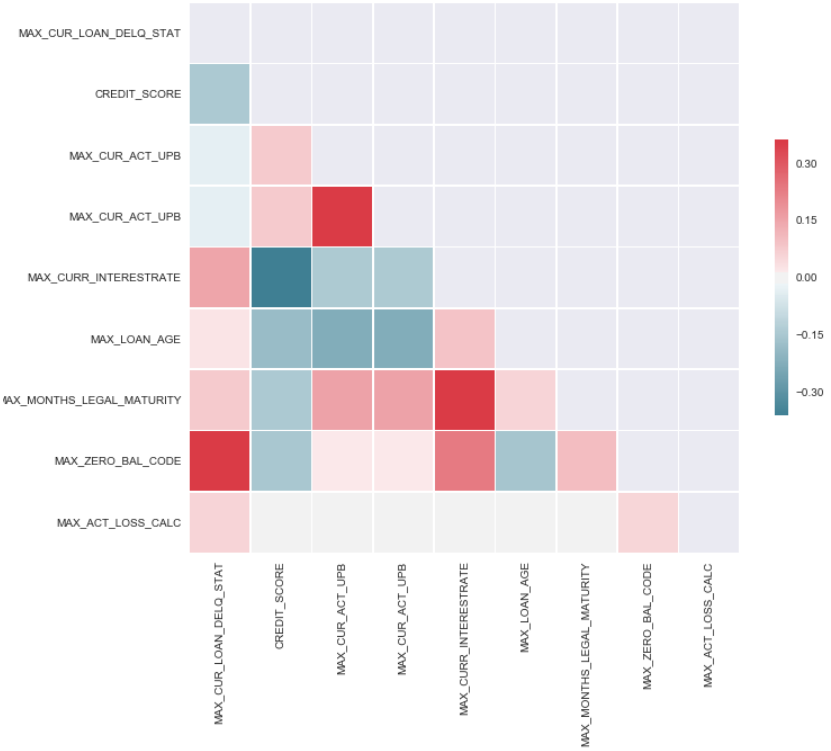


Losses Incurred by Freddie Mac

Freddie Mac took heavy losses during the 2007-2008 period. The losses appear to be cyclic in nature with spikes in the data.

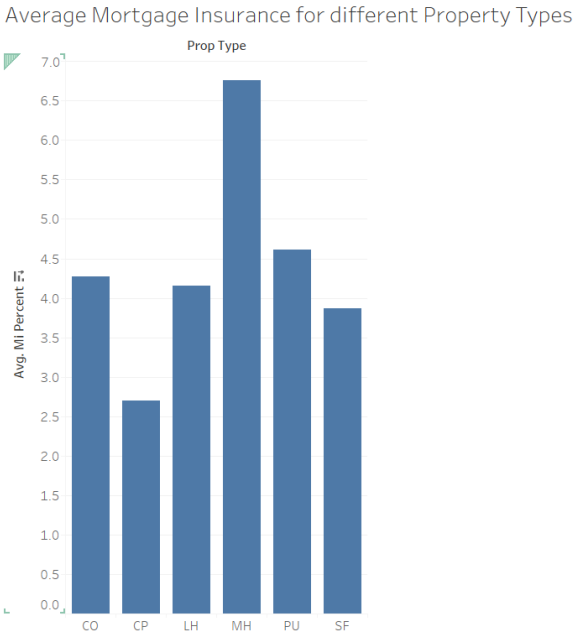


Correlation Matrices for the Origination and Performance File.

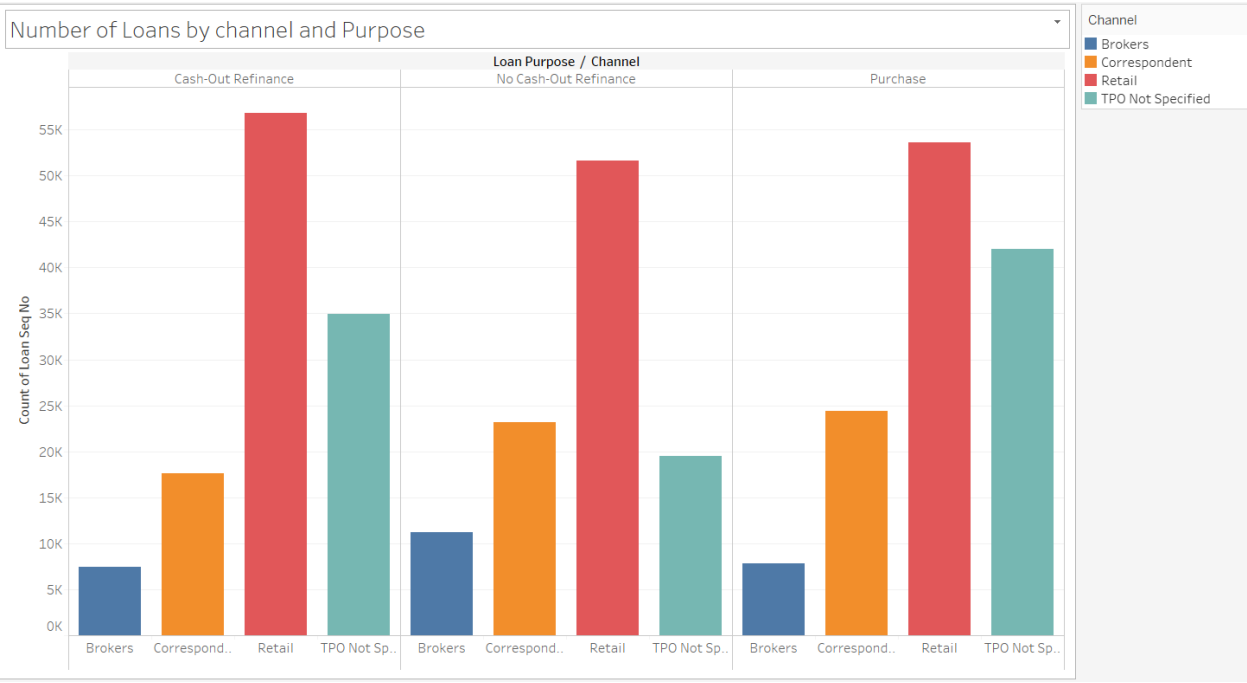


Analysis on Tableau

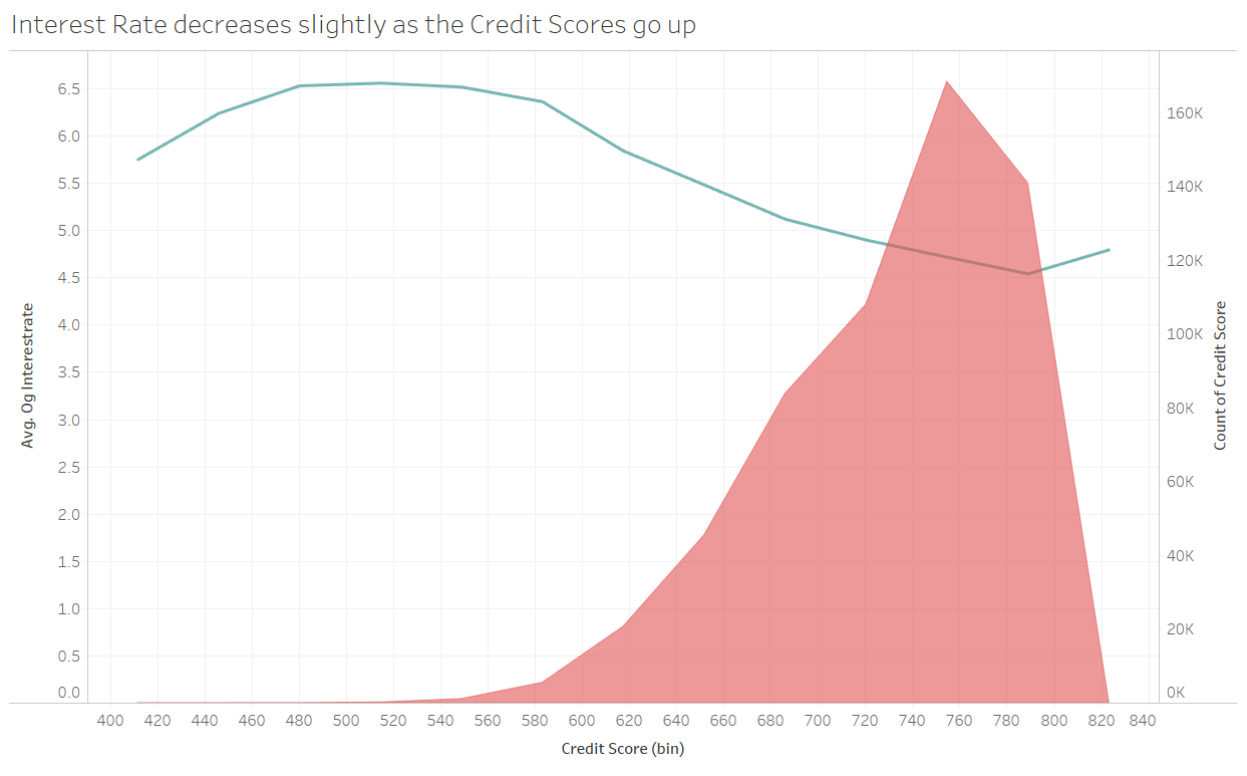
Mortgage insurance companies are mostly interested in Manufactured Housing according to this data. This might be the case because there is another comany associated with covering a part of the losses in case the mortgage might be defaulted.s



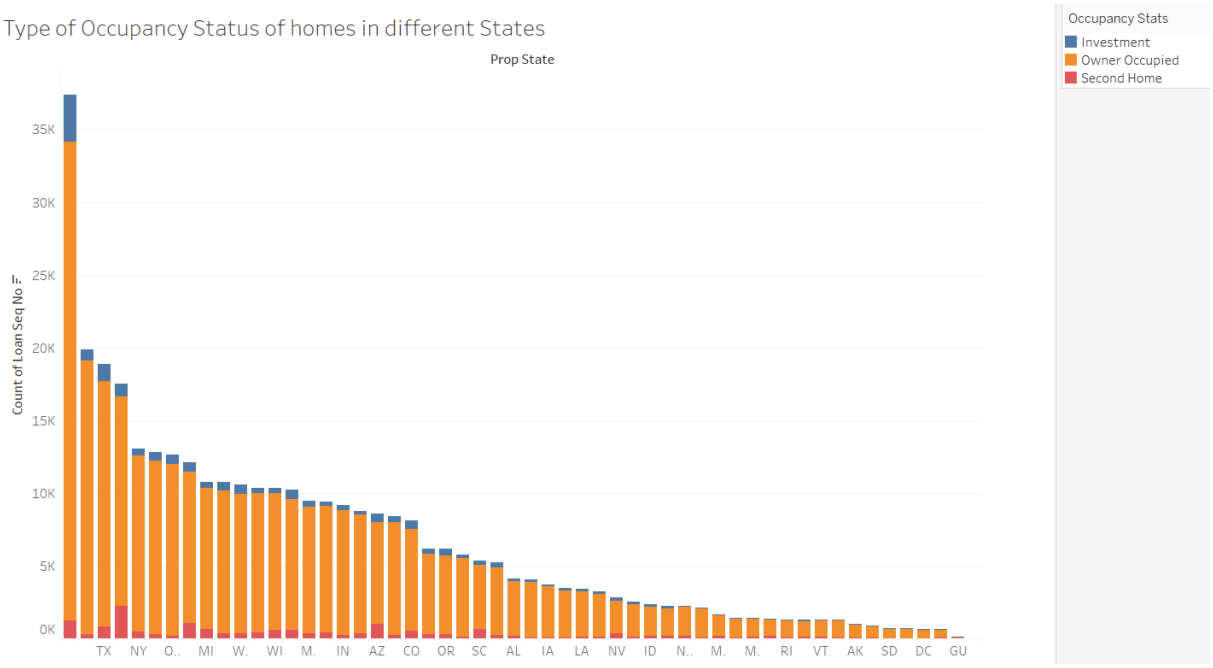
Freddie Mac deals mostly with mortgages that have originated as directly with them as retail loans. The dependency on brokers for selling mortgages to clients is minimum.

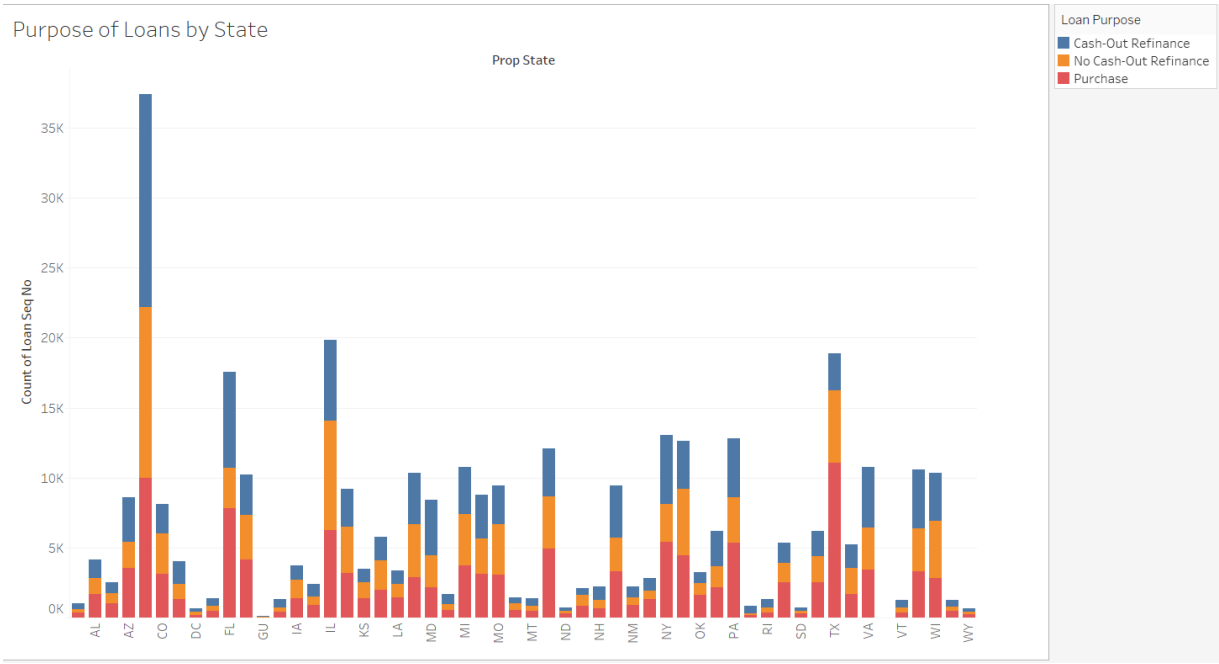


As expected, applicants with higher credit scores receive lower interest rates by a factor of 0.5-1.0 for their mortgages.



California ranks as the state with the highest number of home mortgages. Bulk of them are Owner occupied throughout the state and also sees most number of investments (% wise).





Summary for Exploratory Data Analysis

We have analyzed the different attributes of Freddie Mac. We have seen how the number of delinquencies led to the mortgage crisis of 2008. This was reflected in the dataset. We have also analyzed the different patterns in the data and explored the correlation to be able to select features better during predictive modelling.

Part 2 - Prediction

Overview

We will create predictive models based on the datasets that we have. The approach that we will follow will be that of rolling quarters. This means that we will treat one quarter as the train data and the following quarter as the test data on which we will run our algorithm to predict certain aspects of the data. We are interested in predicting the interest rate that a particular loan might get assigned based on the model that we will build here. We are also interested in determining whether a loan will default based on classification modelling using the monthly performance data of loans in the previous quarter.

We have created one function that downloads the data for the training quarter (parameterized) as well as the consecutive quarter to be used for prediction/classification. We are calling this function iteratively and have provided 2 separate scripts. One that performs analysis on a single quarter and the other which performs the analysis on multiple (Rolling) quarters. At the end of the modelling, prediction, classification and testing the function outputs 2 performance metrics.

- A CSV file containing the Error metrics of the prediction modelling based on every quarter.
- A CSV file containing the Confusion matrix for the quarter that we run on.

These output CSV files containing the metrics can be found at the following paths in the Docker image. We have also uploaded these to the github repository under the data folder.

/src/midterm/data/ ClassificationMetrics.csv

/src/midterm/data/ RegressionMetrics.csv

2.1 Predictive Modelling

In this section we will explore three types of algorithms, Random Forest, Linear Regression and Neural Network for predicting the interest rate for the next quarter based on a given quarter. The aim is to take a quarter as input, pre-process the data, run it through the selected algorithm and get predictions for the next quarter with minimum error.

Part 1 – Downloading the data

```
---
   st = datetime.datetime.fromtimestamp(time.time()).strftime('%Y-%m-%d %H:%M:%S.%f')
   logging.info("Download Start Time : "+st)
   print("Download Start Time : "+st)
   for key, value in downlinksdict.items():
        downlist=downloaderfunctions.downloadExtractRemove(BASEURL+value,DOWNPATH,key,opcookie)
        #downlist=['historical_data1_Q12005.txt','historical_data1_time_Q12005.txt']
        for downfilename in downlist:
            #self.filelist.append(DOWNPATH+"/"+downfilename)
            oldfile=downfilename
            newfile=""
            if 'time' in downfilename:
                l = 7000000
                size=os.path.getsize(DOWNPATH+"/"+downfilename)
                numchunks=round(size/400000000)
                if numchunks>=2:
                    with open(DOWNPATH+"/"+downfilename) as bigfile:
                        cnt=0
                        for i, lines in enumerate(chunks(bigfile, l)):
                            cnt+=1
                            if cnt<=1:
                                file_split = '{}.{}'.format(DOWNPATH+"/"+downfilename, i)
                                with open(file_split, 'w') as f:
                                    f.writelines(lines)
                    newfile=downfilename+".0"
                if newfile!="":
                    downfilename=newfile
                temp=pd.DataFrame()
                temp=cleaningandmergefunctions.cleanPerfClassification(DOWNPATH+"/"+downfilename)
                temp.to_csv(DOWNPATH+"/"+"clean"+"_"+downfilename,index=False)
                if trainquarter in downfilename:
                    perftrainlist.append(DOWNPATH+"/"+"clean"+"_"+downfilename)
                elif testquarter[0] in downfilename:
                    perftestlist.append(DOWNPATH+"/"+"clean"+"_"+downfilename)
                if newfile!="":
                    os.remove(DOWNPATH+"/"+oldfile)
                os.remove(DOWNPATH+"/"+downfilename)
            else:
                temp=pd.DataFrame()
                temp=cleaningandmergefunctions.cleanOrigPrediction(DOWNPATH+"/"+downfilename)
                temp.to_csv(DOWNPATH+"/"+"clean"+"_"+downfilename,index=False)
                if trainquarter in downfilename:
                    origtrainlist.append(DOWNPATH+"/"+"clean"+"_"+downfilename)
                elif testquarter[0] in downfilename:
                    origtestlist.append(DOWNPATH+"/"+"clean"+"_"+downfilename)

                os.remove(DOWNPATH+"/"+downfilename)
```

The above script downloads the historical data for the inputted quarters and forwards it to the cleaning script. The cleaning scripts fills the missing values, converts the datatypes and forwards it to the data preprocessing step required for data modelling. The following script performs feature transformation on categorical variables and converts in into numeric. LabelEncoder() conversion is used. It assigns a numeric value to each level in a feature.

```
def dummyvar(df):
    dumvar=df
    dumvar.select_dtypes(include=['object']).copy()
    #dumvar=pd.get_dummies(dumvar, columns=["PPM_FLAG"], prefix=["ppm"])
    lb_make = LabelEncoder()
    dumvar["PPM_FLAG_CODE"] = lb_make.fit_transform(dumvar["PPM_FLAG"])
    dumvar["LOAN_PURPOSE_CODE"] = lb_make.fit_transform(dumvar["LOAN_PURPOSE"])
    dumvar["OCCUPANCY_STATS_CODE"] = lb_make.fit_transform(dumvar["OCCUPANCY_STATS"])
    dumvar["PROP_TYPE_CODE"] = lb_make.fit_transform(dumvar["PROP_TYPE"])
    dumvar["FIRST_HOME_BUYER_FLAG_CODE"] = lb_make.fit_transform(dumvar["FIRST_HOME_BUYER_FLAG"])
    dumvar["PROP_STATE_CODE"] = lb_make.fit_transform(dumvar["PROP_STATE"])
    dumvar["CHANNEL_CODE"] = lb_make.fit_transform(dumvar["CHANNEL"])
    dumvar["SELLER_NAME_CODE"] = lb_make.fit_transform(dumvar["SELLER_NAME"])
    dumvar["SERVICE_NAME_CODE"] = lb_make.fit_transform(dumvar["SERVICE_NAME"])
    return dumvar
```

The numeric columns are selected from the output dataframe received from the above script, the target column is separated and the models are applied.

```
a1 =a.select_dtypes(include=['number'])
b1 =b.select_dtypes(include=['number'])
a1.info()
```

```
X_train=a1.drop('OG_IR',axis=1)
Y_train=a1.OG_IR
X_test=b1.drop('OG_IR',axis=1)
Y_test=b1.OG_IR
```

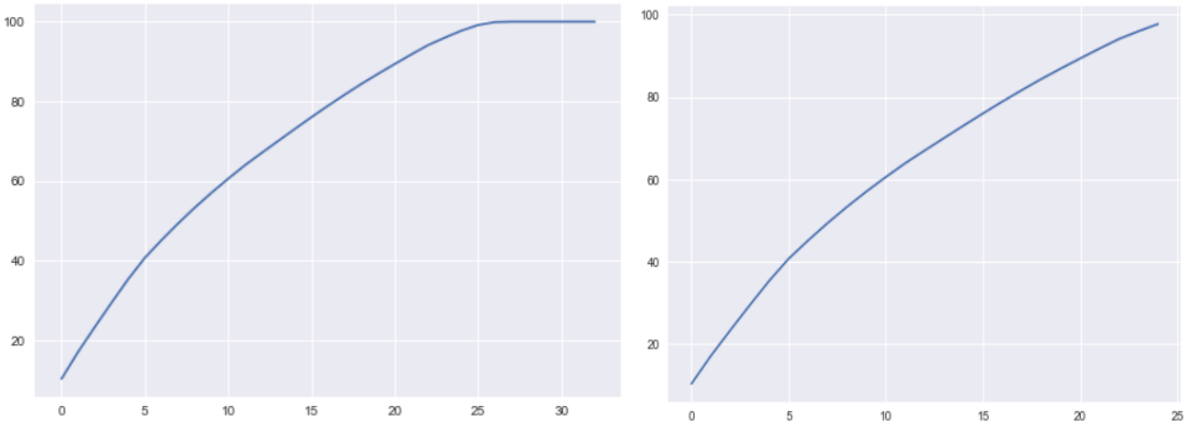
Part 2- Feature Selection:

The key step for creating a good model is to select the features that contribute the most to the target value to be predicted.

Principal Component Analysis (PCA)

PCA is used to overcome features redundancy in a data set. These features are low dimensional in nature. These features a.k.a components are a resultant of normalized linear combination of original predictor variables. These components aim to capture as much information as possible with high explained variance. The first component has the highest variance followed by second, third and so on.

WE can see the graph changes at 25, so we will set the number of components as 25 and reduced the original 36 features to first 25 features based on the variance obtained in the graph




```
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
X=scale(X4)
pca=PCA()
pca.fit(X)
#The amount of variance that each PC explains
var= pca.explained_variance_ratio_
#Cumulative Variance explains
var1=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)
plt.plot(var1)
plt.show()
```

```
X=scale(X4)
pca=PCA(n_components=25)
pca.fit(X)
#The amount of variance that each PC explains
```

Lasso (least absolute shrinkage and selection operator)

Lasso is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces.

```
# Create a function called lasso,
def lasso(alphas):
    df = pd.DataFrame()
    df['Feature Name'] = names

       for alpha in alphas:
        lasso = Lasso(alpha=alpha)
        # Fit the lasso regression
        lasso.fit(X, Y)
        # Create a column name for that alpha value
        column_name = 'Alpha = %f' % alpha
        # Create a column of coefficient values
        df[column_name] = lasso.coef_
    # Return the datafram
    return df
# Run the function called, Lasso
lasso([.0001, .5, 10])
```

Lasso conducts regularization process, where in the features in a model are penalized only to narrow down to only the most important features. The alpha parameter value decides the coefficient values of the features, as alpha increases the features are forced towards a zero coefficient.

	Feature Name	Alpha = 0.000100	Alpha = 0.500000	Alpha = 10.000000
0	Unnamed: 0	-0.061290	-0.0	-0.0
1	CREDIT_SCORE	-0.039544	-0.0	-0.0
2	MSA	0.001777	-0.0	-0.0
3	MI_PERCENT	0.039913	0.0	0.0
4	N_UNITS	0.012942	0.0	0.0
5	OG_CLTV	0.000000	0.0	0.0
6	OG_DTI	-0.002972	0.0	0.0
7	OG_UPB	-0.070587	-0.0	-0.0
8	OG_LTV	0.021723	0.0	0.0
9	POSTALCODE	-0.009702	-0.0	-0.0
10	OG_LOANTERM	0.102139	0.0	0.0
11	NO_BORROWERS	-0.004936	-0.0	-0.0
12	PPM_FLAG_CODE	0.012759	0.0	0.0
13	LOAN_PURPOSE_CODE	-0.026983	0.0	0.0
14	OCCUPANCY_STATS_CODE	-0.030935	-0.0	-0.0
15	PROP_TYPE_CODE	-0.010039	-0.0	-0.0

Recursive Feature Elimination

The Recursive Feature Elimination (RFE) method is a feature selection approach. It works by recursively removing attributes and building a model on those attributes that remain. It uses the model accuracy to identify which attributes (and combination of attributes) contribute the most to predicting the target

attribute. It is based on the idea to repeatedly construct a model and choose either the best or worst performing feature (for example based on coefficients), setting the feature aside and then repeating the process with the rest of the features. This process is applied until all features in the dataset are exhausted. Features are then ranked according to when they were eliminated. It is a greedy optimization for finding the best performing subset of features.

```
# Recursive feature elimination
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
#use linear regression as the model
lr = LinearRegression()
#rank all features, i.e continue the elimination until the last one
rfe = RFE(lr, n_features_to_select=10)
rfe.fit(X_train, Y_train)
print ("Features sorted by their rank:")
print (sorted(zip(map(lambda x: round(x, 4), rfe.ranking_), X3.columns)))
#rfe_pred=rfe.predict(X4)
```

F regression:

Quick linear model for testing the effect of a single regressor, sequentially for many regressors.

```
# f regression
from sklearn.feature_selection import SelectPercentile
from sklearn.feature_selection import f_regression
Selector_f = SelectPercentile(f_regression, percentile=25)
r=Selector_f.fit(X,X1)
for n,s in zip(X_train.columns,Selector_f.scores_):
    print ('F-score: %3.2ft for feature %s' % (s,n))
```

```
F-score: 282.59t for feature MSA
F-score: 18098.76t for feature MI_PERCENT
F-score: 412.28t for feature N_UNITS
F-score: 14651.86t for feature OG_CLTV
F-score: 777.00t for feature OG_DTI
F-score: 8334.21t for feature OG_UPB
F-score: 16604.64t for feature OG_LTV
F-score: 552.37t for feature OG_IR
F-score: 64682.43t for feature POSTALCODE
F-score: 2314.05t for feature OG_LOANTERM
F-score: 614.80t for feature NO_BORROWERS
F-score: 376.92t for feature PPM_FLAG_CODE
F-score: 4354.26t for feature LOAN_PURPOSE_CODE
F-score: 417.34t for feature OCCUPANCY_STATS_CODE
F-score: 1072.80t for feature PROP_TYPE_CODE
F-score: 6.73t for feature FIRST_HOME_BUYER_FLAG_CODE
F-score: 2942.26t for feature PROP_STATE_CODE
F-score: 431.20t for feature CHANNEL_CODE
F-score: 1089.15t for feature SELLER_NAME_CODE
```

The results obtained from these algorithms were compared with Forward/Backward regression techniques for feature selection and had similar results. Based on the results the following features were selected for actual modelling.

```
X_train=a1
[['CREDIT_SCORE','OG_UPB','OG_DTI','CHANNEL_CODE','OG_LOANTERM','OG_LTV','MI_PERCENT','PPM_FLAG_CODE','NO_BORROWERS','O
G_CLTV','N_UNITS','PROP_STATE_CODE','FIRST_HOME_BUYER_FLAG_CODE','PROP_TYPE_CODE','OCCUPANCY_STATS_CODE','LOAN_PURPOSE_
CODE','OG_IR']]
```

Part 3 - Algorithms

Linear Regression

Linear regression is an approach for modeling the relationship between a scalar dependent variable *y* and one or more explanatory variables (or independent variables) denoted *X*. The case of one explanatory variable is called *simple linear regression*. For more than one explanatory variable, the process is called *multiple linear regression.*

```
#Liner Regression
regr = linear_model.LinearRegression()
regr.fit(X_train, Y_train)
reg_coef=pd.DataFrame(list(zip(X_train.columns,regr.coef_)) ,columns=['feat','estimatedcoeff'])
y_pred = regr.predict(X_test)
```

```
def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

lr_r2 = r2_score(Y_test, y_pred)
lr_mse = mean_squared_error(Y_test, y_pred)
lr_rmse = sqrt(mean_squared_error(Y_test, y_pred))
lr_mae = mean_absolute_error(Y_test, y_pred)
lr_mape = mean_absolute_percentage_error(Y_test, y_pred)
```

```
print(lr_r2)
print(lr_mse)
print(lr_rmse)
print(lr_mae)
print(lr_mape)
```

```
0.191697507492
0.0940108758753
0.30661193041904866
0.23043467101
4.0760883380531
```

Random Forest Regression:

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators=100,n_jobs=10)
regressor.fit(X_train, Y_train)

rf_predict=regressor.predict(X_test)
```

```
def mean_absolute_percentage_error(y_true, rf_predict):
    return np.mean(np.abs((y_true - rf_predict) / y_true)) * 100

rf_r2 = r2_score(Y_test, rf_predict)
rf_mse = mean_squared_error(Y_test, rf_predict)
rf_rmse = sqrt(mean_squared_error(Y_test, rf_predict))
rf_mae = mean_absolute_error(Y_test, rf_predict)
rf_mape = mean_absolute_percentage_error(Y_test, rf_predict)
```

```
print(rf_r2)
print(rf_mse)
print(rf_rmse)
print(rf_mae)
print(rf_mape)
```

```
0.255216307795
0.086623223225
0.29431823461171636
0.22372879921
3.9576141633041177
```

Neural Network:

Neural Network or Artifical neural network is a *"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."* ANNs are processing devices (algorithms or actual hardware) that are loosely modeled after the neuronal structure of the mamalian cerebral cortex but on much smaller scales. Neural networks are typically organized in layers. Layers are made up of a number of interconnected 'nodes' which contain an 'activation function'. Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted connections.

```
from sklearn.neural_network import MLPRegressor

#reg = MLPRegressor(hidden_layer_sizes=(50,), algorithm="l-bfgs")

reg = MLPRegressor(hidden_layer_sizes=(1,3) ,learning_rate_init=0.001)

reg.fit(X_train,Y_train)

predict=reg.predict(X_test)
```

```
def mean_absolute_percentage_error(y_true, predict):
    return np.mean(np.abs((y_true - predict) / y_true)) * 100

mlp_r2 = r2_score(Y_test, predict)
mlp_mse = mean_squared_error(Y_test, predict)
mlp_rmse = sqrt(mean_squared_error(Y_test, predict))
mlp_mae = mean_absolute_error(Y_test, predict)
mlp_mape = mean_absolute_percentage_error(Y_test, predict)
```

```
print(mlp_r2)
print(mlp_mse)
print(mlp_rmse)
print(mlp_mae)
print(mlp_mape)
```

```
0.189612158945
0.0942534155746
0.30700719140530097
0.230123785313
4.053501237100889
```

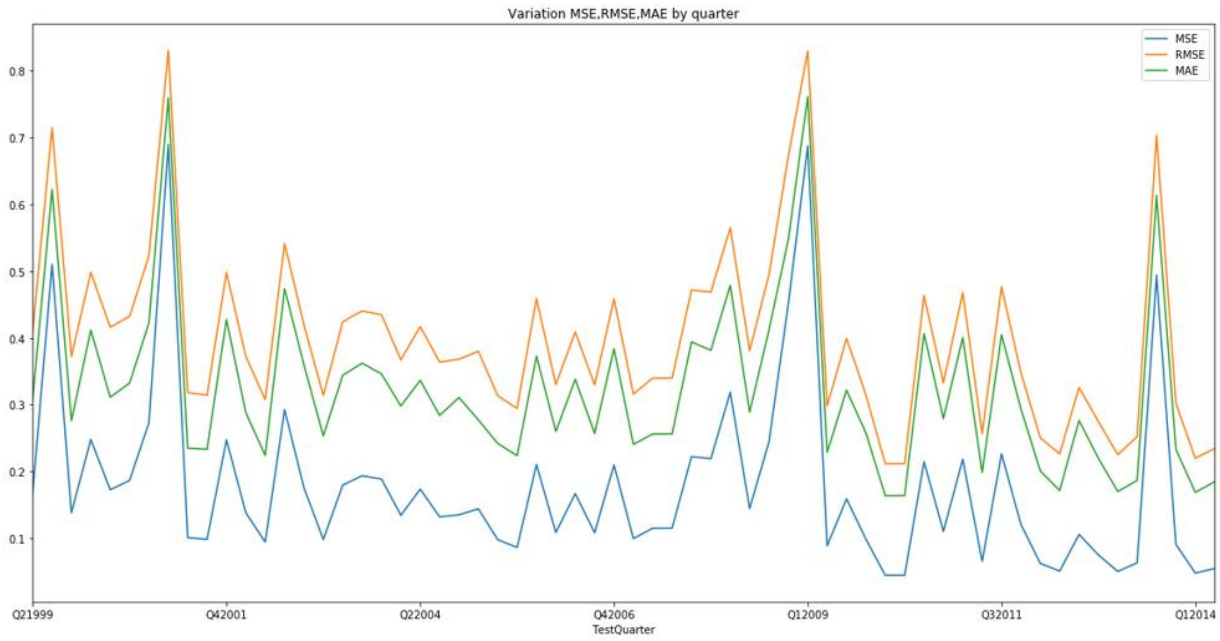
The results obtained from the above training sets shows a high value for R-squared and low for MAPE in the Random Forests Algorithm. These were trained and cross validated on all the quarters of historical data of the year 2005. The model selected is Random Forest, trained on the data for the year 2005.

What-if analysis:

Financial Crisis:

The 2007 financial crisis is the breakdown of trust within the financial system. It was caused by the subprime mortgage crisis, which itself was caused by the use of derivatives. Resale single-family home sales peaked in February 2007, when they were at an annual rate of 5.88 million. By September, single-family home resales had declined 25 percent to 4.38 million.

The financial crisis was sparked by loan companies supplying easy, expensive home loans by borrowers who had a poor credit history, or could not prove their incomes, and held a greater risk of loan default than prime borrowers - these risky transactions were known as subprime loans. From 1997 until 2007 it was easy to get a loan or a mortgage. New loans were made attractive to borrowers and many people re-mortgaged their homes due to low interest rates.



The above graph shows the variation of the prediction error for interest rate for the years 1999 to 2014.

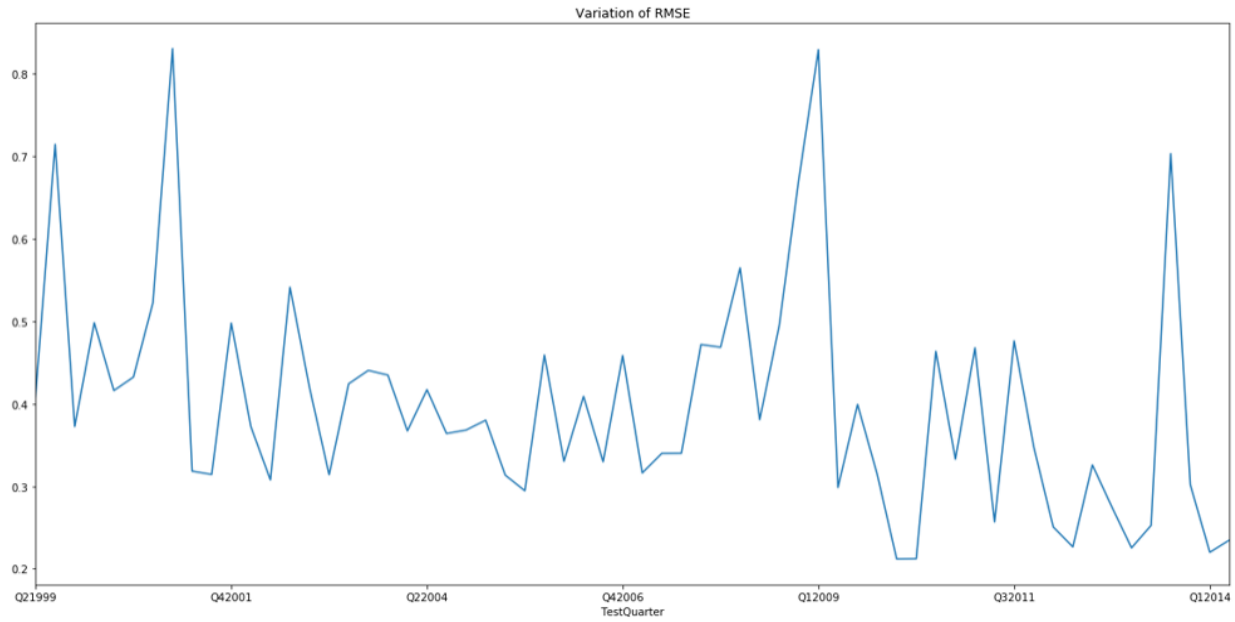
This period is distinctively marked by economic boom during 1999 -2001, the financial crisis period from 2007 to 2008-2009. We will analyze the periods separately. The error metric is Root Mean Square Error(rmse) as the scale of the interest rate does not vary as much, as it a better indicator than mse.

Economic Boom: (1999-2005)

[Ref Link: http://mortgage-x.com/general/national_monthly_average.asp?y=1993]

The average mortgage interest rate during the economic boom years 1999 to 2002 were approximately 6 to 8 percent, which are considerably high. Also the stock market returns varied between 20% to 25%..

As the interest rates were higher, our model outputs a sharp peak for those years as the model was trained on average interests of 4% to 5%.

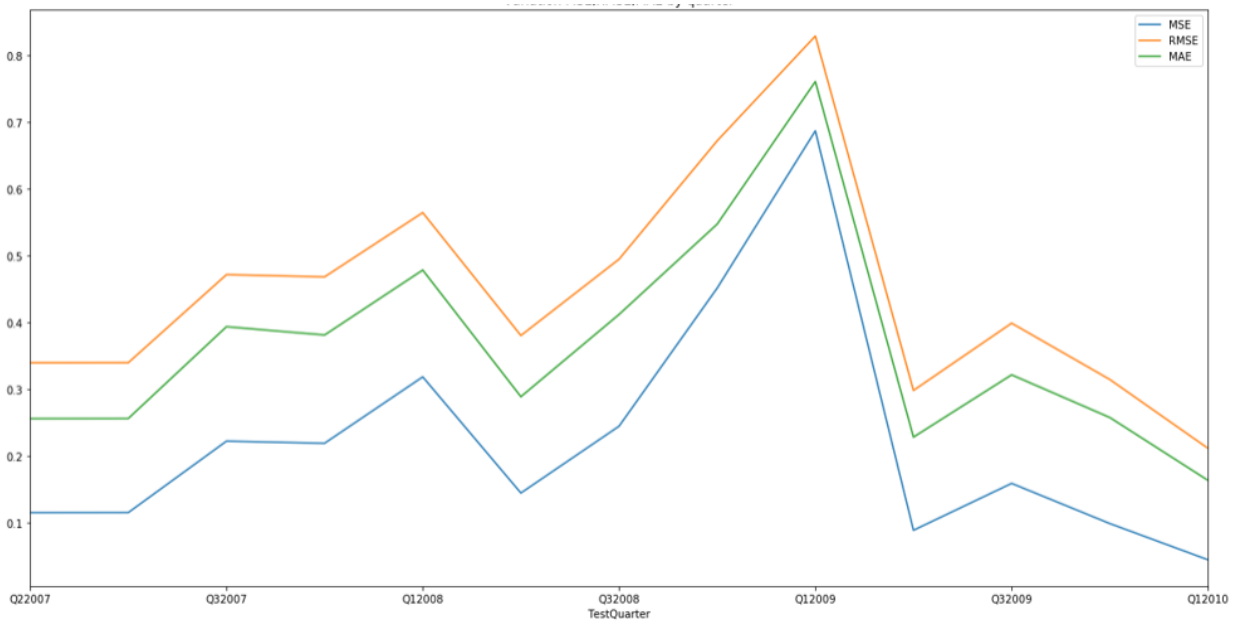


The interest rates varied between 5% to 8% till it dropped in the year 2003 to an average of 3%, when loan companies supplying easy, expensive home loans by borrowers who had a poor credit history, or could not prove their incomes, and held a greater risk of loan default than prime borrowers - these risky transactions were known as subprime loans. From 1997 until 2007 it was easy to get a loan or a mortgage. New loans were made attractive to borrowers and many people re-mortgaged their homes due to low interest rates.

Financial Crisis(2007-2009)

In 2007 the price of houses began to fall and the housing bubble began to collapse.

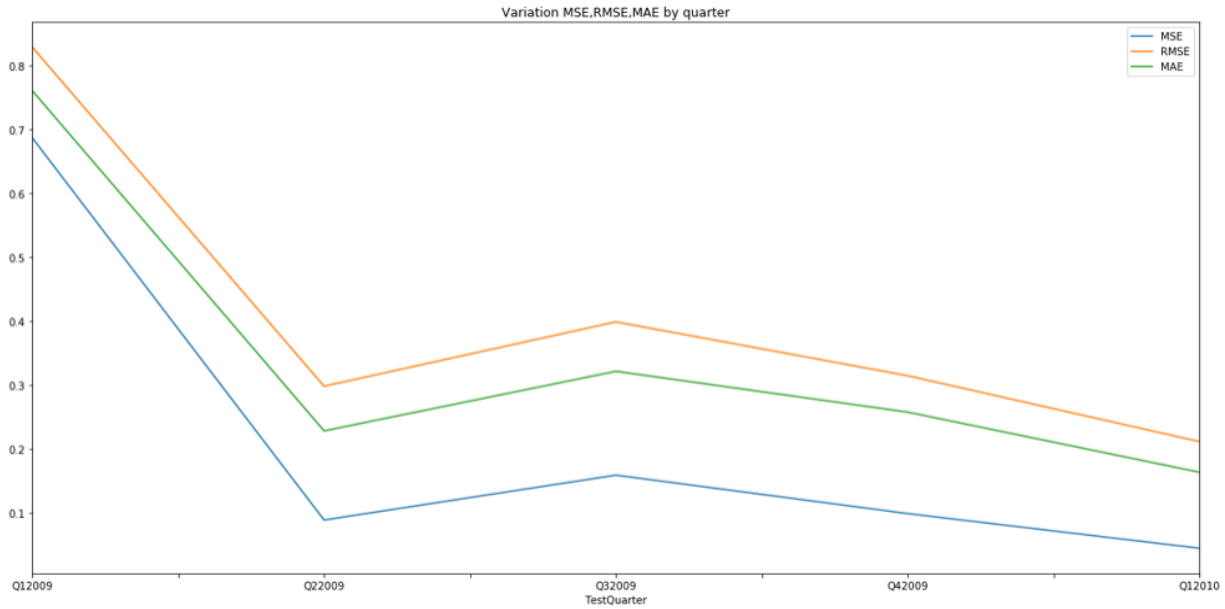
Interest rates rose, and numerous subprime mortgage borrowers began to default on their loans. Subprime borrowers found that the value of many homes dropped below the value of the remaining mortgage debt (negative equity). In 2008 US government was forced to take over 'Fannie Mae' (Federal National Mortgage Association) and 'Freddie Mac' (Federal Home Loan Mortgage Corporation) firms that had guaranteed thousands of sub-prime mortgages, fearing that a systemic global financial crisis would prompt the biggest depression since the 1930s.



Based on the variations in rmse we can the spike caused in 2008-2009 due to peaked interest rate. The prediction error increased during this period as the model being trained on average rates fails to adapt to the sudden spike.

Year 2009

The market started recovering by the year 2009 and the interest rates went back to varying between 4 to 5 percent. Hence, the rmse values have reduced considerably as compared to the previous two years.



The graph shows rmse values stable with no sharp peaks.

Prediction for next quarter. (2014)

Hypothesis: the model can be used for next quarter prediction if the interest rate values do not increase of decrease suddenly, ie if they remain in the similar range of values

Actual Values:

We run our model for the next two quarters Q1 2014 and Q2 2014. We can see that the values for rmse are low with an increased R-squared value.

TrainQuarter	TestQuarter	R-squared	MSE	RMSE	MAE	MAPE
Q32013	Q42013	0.596104	0.091073	0.301782	0.23318	5.358781
Q42013	Q12014	0.764327	0.048338	0.219858	0.168905	3.823659
Q12014	Q22014	0.723152	0.055069	0.234668	0.184929	4.377289

Conclusion: The model can be used for stable interest rates for any future quarters.

What can be done better:

- As we can see, the model does not adapt well to sudden changes in the target variable, it can be modified to do so, for better predictions.
- Feature analysis and selection can be improved by further fine tuning.
- Additional algorithms such as KNN or SVM can be tried to test model performance.

2.2 Classification

In this part we will explore different algorithms to classify loans as delinquent or non-delinquent based on the features existing in the historical performance dataset.

Part 1 – Downloading the data

For this part, we first begin with downloading the data. For downloading data, we have created a python script which accepts the train quarter as the input and downloads data for that quarter as well as the consecutive quarter. Since the performance file is very big (almost 1.5 GB) we have reduced the size of the data and are limiting it to about the initial 400 MB so that we will be able to train the dataset a bit faster while performing iterative algorithms such as Random Forest and Neural Network.

```
    st = datetime.datetime.fromtimestamp(time.time()).strftime('%Y-%m-%d %H:%M:%S.%f')
    logging.info("Download Start Time : "+st)
    print("Download Start Time : "+st)
    for key, value in downlinksdict.items():
        downlist=downloaderfunctions.downloadExtractRemove(BASEURL+value,DOWNPATH,key,opcookie)
        #downlist=['historical_data1_Q12005.txt','historical_data1_time_Q12005.txt']
        for downfilename in downlist:
            #self.filelist.append(DOWNPATH+"/"+downfilename)
            oldfile=downfilename
            newfile=""
            if 'time' in downfilename:
                l = 7000000
                size=os.path.getsize(DOWNPATH+"/"+downfilename)
                numchunks=round(size/400000000)
                if numchunks>=2:
                    with open(DOWNPATH+"/"+downfilename) as bigfile:
                        cnt=0
                        for i, lines in enumerate(chunks(bigfile, l)):
                            cnt+=1
                            if cnt<=1:
                                file_split = '{}.{}'.format(DOWNPATH+"/"+downfilename, i)
                                with open(file_split, 'w') as f:
                                    f.writelines(lines)
                    newfile=downfilename+".0"
                if newfile!="":
                    downfilename=newfile
                temp=pd.DataFrame()
                temp=cleaningandmergefunctions.cleanPerfClassification(DOWNPATH+"/"+downfilename)
                temp.to_csv(DOWNPATH+"/"+"clean"+"_"+downfilename,index=False)
                if trainquarter in downfilename:
                    perftrainlist.append(DOWNPATH+"/"+"clean"+"_"+downfilename)
                elif testquarter[0] in downfilename:
                    perftestlist.append(DOWNPATH+"/"+"clean"+"_"+downfilename)
                if newfile!="":
                    os.remove(DOWNPATH+"/"+oldfile)
                os.remove(DOWNPATH+"/"+downfilename)
            else:
                temp=pd.DataFrame()
                temp=cleaningandmergefunctions.cleanOrigPrediction(DOWNPATH+"/"+downfilename)
                temp.to_csv(DOWNPATH+"/"+"clean"+"_"+downfilename,index=False)
                if trainquarter in downfilename:
                    origtrainlist.append(DOWNPATH+"/"+"clean"+"_"+downfilename)
                elif testquarter[0] in downfilename:
                    origtestlist.append(DOWNPATH+"/"+"clean"+"_"+downfilename)

                os.remove(DOWNPATH+"/"+downfilename)
```

The function then proceeds with cleaning and wrangling the data followed by performing classification and regression on the downloaded data and then removes the space.

We have chosen the 4th column in the dataset as our y variable. (CURRENT LOAN DELINQUENCY STATUS) as the dependent variable. Since this is a continuous value and represents the number of days since the loan was last delinquent, we create an additional column “DELINQUENT”. This column will have 0 or 1 as the value with 0 indicating not Delinquent (False) and 1 indicating Delinquent (True).

A classification algorithm runs on the training dataset and creates the model. We then use this model to perform classification of the data present in the consecutive (test) quarter to determine if a loan would become delinquent.

```
def classification(trainpath,testpath,trainquarter,testquarter):
    #perform regression
    print(trainpath+"\t"+testpath+"\t"+trainquarter+"\t"+testquarter)

    Train_DF = pd.read_csv(trainpath,index_col=None)
    traincols=['MONTHLY_REPORT_PERIOD','CUR_ACT_UPB','LOAN_AGE','MONTHS_LEGAL_MATURITY','CURR_INTERESTRATE','CURR_DEF_UPB']

    y_train=Train_DF['DELINQUENT']
    Train_DF=Train_DF[traincols]

    Test_DF=pd.read_csv(testpath,index_col=None)

    testcols=['MONTHLY_REPORT_PERIOD','CUR_ACT_UPB','CUR_LOAN_DELQ_STAT','LOAN_AGE','MONTHS_LEGAL_MATURITY','CURR_INTERESTRATE','CURR_DEF_UPB','DELINQUENT']
    testcols=['MONTHLY_REPORT_PERIOD','CUR_ACT_UPB','LOAN_AGE','MONTHS_LEGAL_MATURITY','CURR_INTERESTRATE','CURR_DEF_UPB']

    y_test=Test_DF['DELINQUENT']
    Test_DF=Test_DF[testcols]

    clf = RandomForestClassifier(n_estimators=20,verbose =1,min_samples_split=10)
    clf = clf.fit(Train_DF, y_train)

    pred = clf.predict(Test_DF)

    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(testquarter)+","+str(numActualDelinq)+","+str(numPredictedDelinq)+","+str(numRecordsInDataset)+","+str(numDelinqProper)+","+str(numnondelinqimproper)
    checkFile = Path(FINALCSVPATH)
    if checkFile.is_file():
        with open(FINALCSVPATH,"a") as fil:
            fil.write(record)
            fil.write("\n")
    else:
        with open(FINALCSVPATH,"a") as fil:
            fil.write("Quarter,NumActualDelinquents,NumOfPredictedDelinquents,NumRecordsInDataset,NumDelinquentsProperlyClassified,NumNonDelinquentsImproperlyClassified")
            fil.write("\n")
            fil.write(record)
            fil.write("\n")
```

Feature Selection

Since there are more than 90% missing values for most fields, we have chosen only a few fields to derive our model for classifying the delinquent loans.

MONTHLY_REPORT_PERIOD','CUR_ACT_UPB','LOAN_AGE','MONTHS_LEGAL_MATURITY','CURR_INTERESTRATE','CURR_DEF_UPB

Algorithm chosen = Random Forest Classification algorithm

Train Accuracy

```
In [43]:  1 print(clf.score(Train_DF,y_train))

[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed:  1.2min finished
0.971850857143
```

Test Accuracy

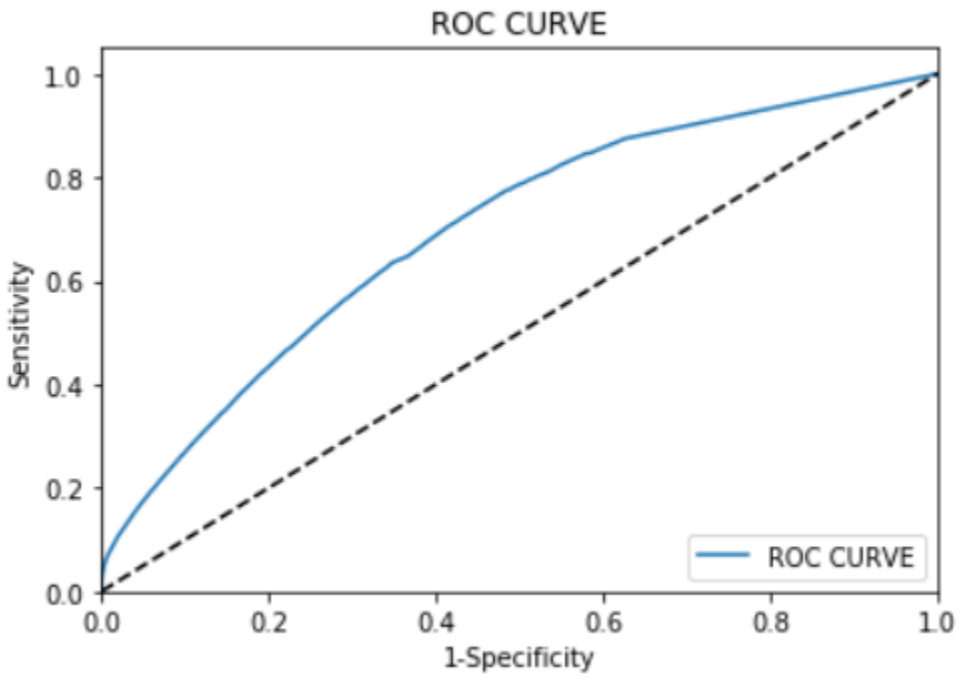
```
In [33]:  1 pred = clf.predict(Test_DF)
[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed:   40.1s finished

In [34]:  1 metrics.accuracy_score(y_test,pred)
Out[34]: 0.94709971428571427
```

Confusion Matrix

```
1 #pd.crosstab(pred, y_test,rownames=['pred'
2 confusion_matrix(y_test, pred, labels=None,
```

```
array([[6606972,   41808],
       [ 328494,   22726]])
```



Other Classification algorithms implemented for comparison :

Apart from the Random Forest Classification, we have tried Logistic Regression and Neural Network algorithm. We find that the Random Forest algorithm has better results especially based on the confusion matrix.

Logistic Regression

We did not select this model because although we got similar results for the accuracy, we got very different results for the confusion matrix. The number of true positives was lower by a significant amount.

Accuracy of Train

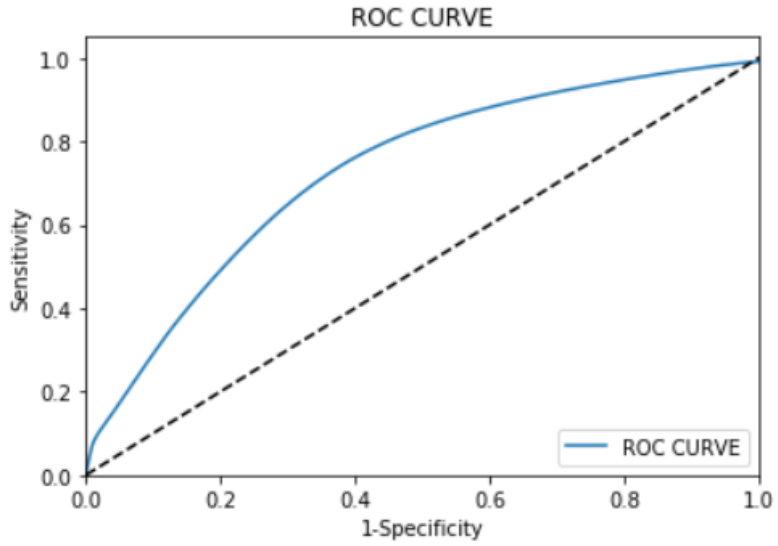
```
In [12]:  1 print(model.score(Train_DF,y_train))
          0.950684714286
```

Accuracy of Test

```
In [14]:  1 metrics.accuracy_score(y_test,pred)
Out[14]: 0.94952385714285714
```

Confusion Matrix

```
  1 print (metrics.confusio
[[6644873    3907]
 [ 349426    1794]]
```



In [19]:
```
logit = sm.Logit(y_train, Train_DF[traincols])
logregression_result = logit.fit()a
print(logregression_result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.182535
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:              DELINQUENT   No. Observations:              7000000
Model:                           Logit   Df Residuals:                  6999994
Method:                            MLE   Df Model:                            5
Date:                Thu, 20 Jul 2017   Pseudo R-squ.:                 0.06513
Time:                        20:40:18   Log-Likelihood:            -1.2777e+06
converged:                       True   LL-Null:                   -1.3668e+06
                                        LLR p-value:                     0.000
================================================================================================
                               coef    std err          z      P>|z|      [95.0% Conf. Int.]
------------------------------------------------------------------------------------------------
MONTHLY_REPORT_PERIOD -3.017e-05   1.63e-07   -184.599      0.000     -3.05e-05 -2.99e-05
CUR_ACT_UPB            1.911e-06   2.31e-08     82.792      0.000      1.87e-06  1.96e-06
LOAN_AGE                  0.0231    6.4e-05    361.691      0.000         0.023     0.023
MONTHS_LEGAL_MATURITY     0.0048   5.01e-05     95.723      0.000         0.005     0.005
CURR_INTERESTRATE        -0.0071      0.003     -2.093      0.036        -0.014    -0.000
CURR_DEF_UPB          -1.16e-05   3.76e-07    -30.841      0.000     -1.23e-05 -1.09e-05
================================================================================================
```

Neural Network

We tried running this algorithm with different tuning parameters which we tweaked to get results. We tried changing the number of hidden layers and the number of neurons with different learning rates.

We are getting similar accuracy as the one we got from Random Forest and Logistic Regression, but the Confusion matrix shows very poor results.

Neural Network is not able to identify the delinquent loans and is predicting everything as non-delinquent. That is why we have discarded this algorithm.

Train Accuracy

]:
```
pred = clf.predict(Test_DF)
```

]:
```
print(clf.score(Train_DF,y_train)
```

```
0.951103714286
```

Test Accuracy

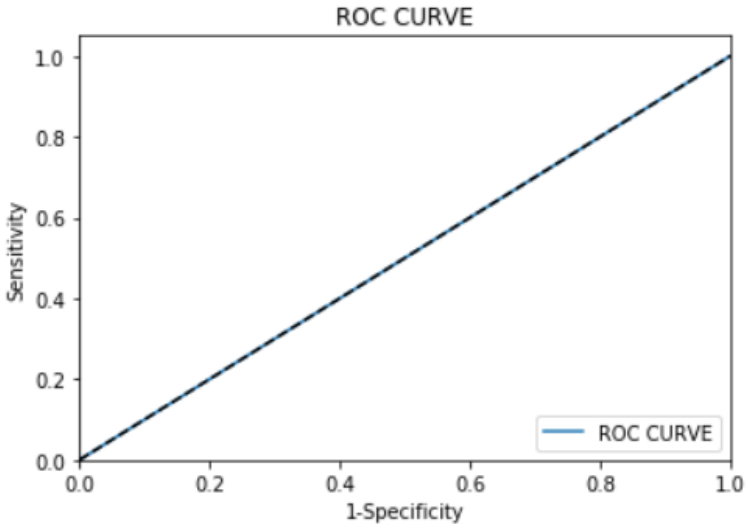
```
metrics.accuracy_score(y_test,pred)
```

```
0.94982571428571427
```

Confusion Matrix

```
1 pd.crosstab(y_test, pred,rowname
```

ytest	0
pred	
0	6648780
1	351220



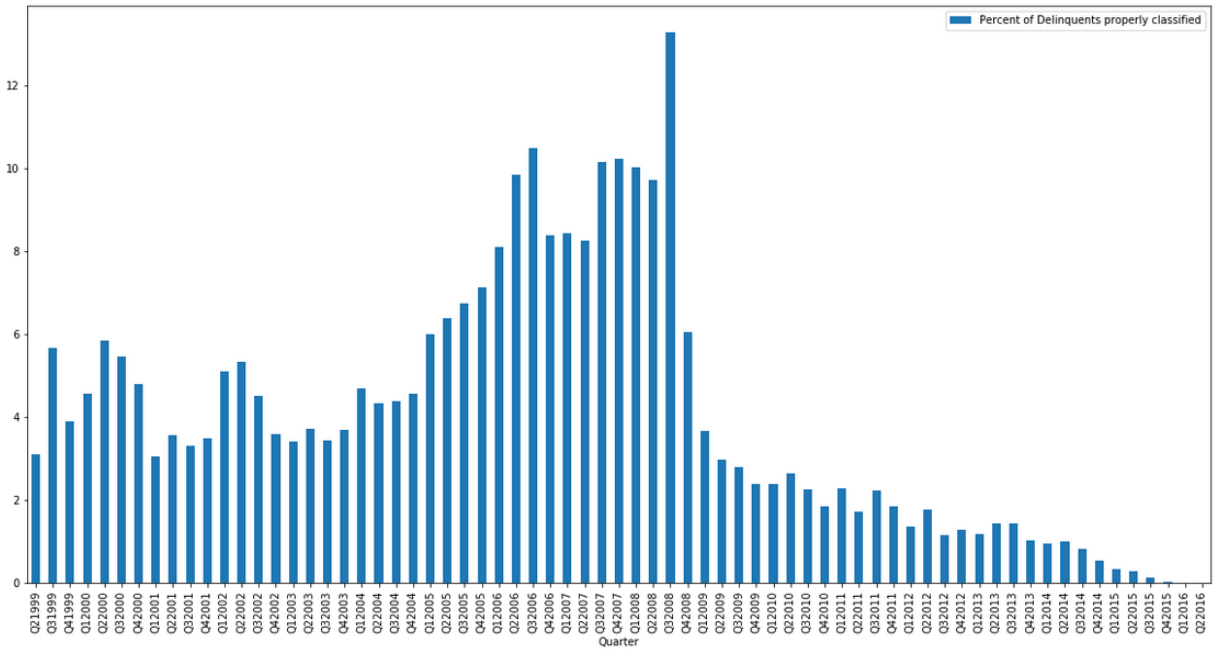
Evaluation Matrix

We ran the classification function iteratively for all the quarters from Q11999 – Q22016. The results of every computation are appended in a CSV file. The evaluation matrix that was automatically generated is given below.

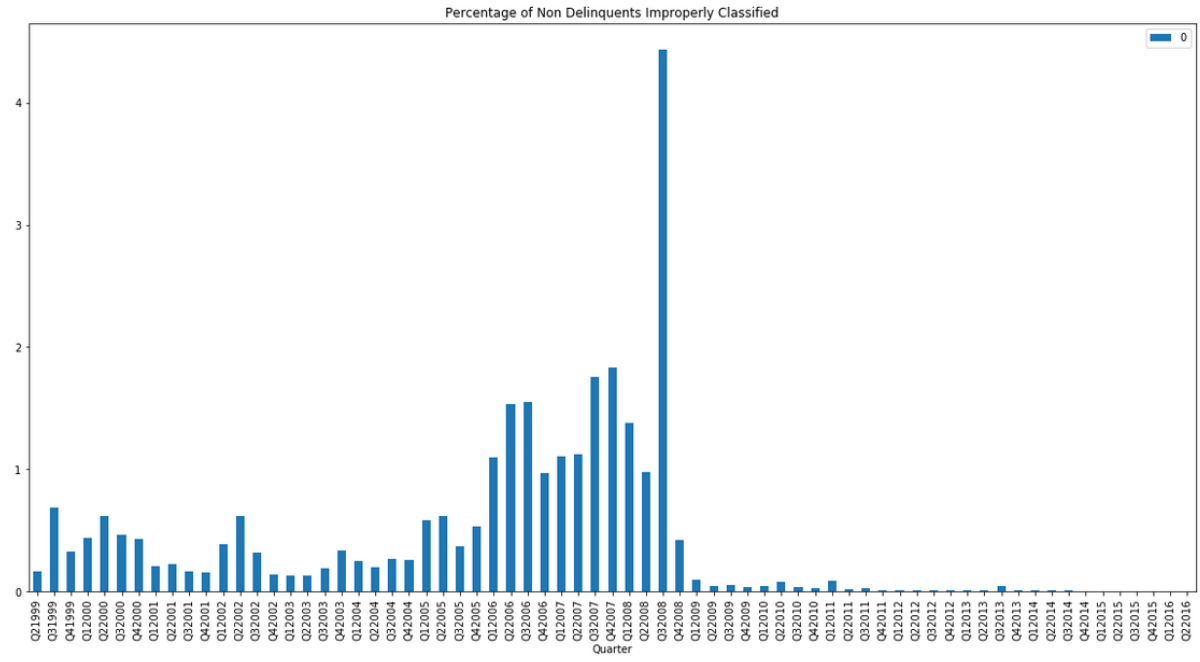
Quarter	NumActualDelinquents	NumOfPredictedDelinquents	NumRecordsInDataset	NumDelinquentsProperlyClassified	NumNonDelinquentsImproperlyClassified
Q21999	202868	202868	7000000	6300	11388
Q31999	334974	334974	8797803	19027	57699
Q41999	270684	270684	6133986	10562	19293
Q12000	185631	185631	3939565	8486	16418
Q22000	255207	255207	5056705	14922	29630
Q32000	275979	275979	5570045	15075	24623
Q42000	318650	318650	6105572	15328	24931
Q12001	239303	239303	7000000	7287	13813
Q22001	245489	245489	7000000	8728	14932
Q32001	246858	246858	7000000	8152	11068
Q42001	215040	215040	7000000	7523	10842
Q12002	315441	315441	7000000	16082	26056
Q22002	380429	380429	7000000	20265	40693
Q32002	290579	290579	7000000	13114	21562
Q42002	189361	189361	7000000	6812	9692
Q12003	172801	172801	7000000	5880	8648
Q22003	158763	158763	7000000	5907	8645
Q32003	147551	147551	7000000	5078	13061
Q42003	257707	257707	7000000	9508	22365
Q12004	254357	254357	7000000	11934	16738
Q22004	233285	233285	7000000	10134	13569
Q32004	279508	279508	7000000	12270	17661
Q42004	326638	326638	7000000	14932	17191
Q12005	342274	342274	7000000	20505	38770
Q22005	351220	351220	7000000	22404	40888
Q32005	352606	352606	7000000	23821	24787
Q42005	435169	435169	7000000	31036	34841

Q12006	531992	531992	7000000	43117	70901
Q22006	549735	549735	7000000	54173	98858
Q32006	576537	576537	7000000	60501	99753
Q42006	572178	572178	7000000	48034	62263
Q12007	576766	576766	7000000	48693	71271
Q22007	602498	602498	7000000	49732	71940
Q32007	715395	715395	7000000	72581	110414
Q42007	784369	784369	7000000	80352	113974
Q12008	582373	582373	7000000	58447	88349
Q22008	496119	496119	7000000	48255	63560
Q32008	556464	556464	8194537	73857	338362
Q42008	382666	382666	8305023	23213	33427
Q12009	120973	120973	7000000	4454	6399
Q22009	74511	74511	7000000	2223	2984
Q32009	82341	82341	7000000	2301	3717
Q42009	85354	85354	7000000	2031	2199
Q12010	82064	82064	7000000	1970	3076
Q22010	78221	78221	7000000	2079	5464
Q32010	54735	54735	7000000	1238	2220
Q42010	40943	40943	7000000	761	1808
Q12011	61963	61963	9294018	1424	8142
Q22011	61740	61740	7821671	1069	1601
Q32011	47977	47977	7000000	1067	1626
Q42011	31999	31999	7000000	589	858
Q12012	24428	24428	7000000	333	535
Q22012	28103	28103	7000000	495	805
Q32012	24278	24278	7000000	281	567
Q42012	23559	23559	7000000	305	508
Q12013	22184	22184	7000000	260	803
Q22013	26905	26905	7000000	388	760
Q32013	38675	38675	8849684	561	4056
Q42013	36988	36988	6922092	381	479
Q12014	25928	25928	4888611	246	465
Q22014	33321	33321	6588603	337	407
Q32014	33870	33870	7155941	276	828
Q42014	25968	25968	5957699	138	289
Q12015	20336	20336	6614842	67	209
Q22015	20241	20241	6582845	57	279
Q32015	16054	16054	4910107	20	95
Q42015	11307	11307	3734175	3	87
Q12016	6586	6586	2648429	1	23
Q22016	6026	6026	2689166	0	7

Percentage of number of Delinquents properly classified



Percentage of Non-Delinquents Improperly classified



We have plotted the above values to visualize the performance of the algorithm.

- The classification algorithm was developed based on the Q12005 data
- We can see that it performed the best during Q32008. The number of delinquents as we have analyzed before are the most around this quarter. The algorithm would have got more data with

delinquent mortgages to train on and that's why it performs better for this quarter and the true positive rate increases.

- The algorithm performs poorly on Q42010. The number of delinquents fell sharply during this period after the market recovered. Because of this, the algorithm got lesser delinquent data to train on and the True positive rate dropped.
- The algorithm performs poorly after the stock market crisis. This is possible due to lesser number of delinquents in data. Since the data does not contain adequate number of delinquents, the algorithm is not able to classify delinquents correctly.

Classification Summary

Justification on the chosen classification model

- Similar score for all the algorithms was obtained.
- Random forest gives the best results in terms of the confusion matrix.
- Random Forest takes lesser time to train as compared to the Neural Network.
- Although the area under the ROC curve for Logistic regression is more, we have chosen the Random Forest algorithm because the number of True positives is significantly higher.

What can be done better

- Although we can see that the random forest provides reasonable outputs for this type of data. We can perform further analysis with more computational power to arrive at better results.
- The dataset is a biased one meaning that there are significantly more number of N values for delinquents as compared Y values. Because of which, the algorithm has a lesser True positive rate.
- We can see from the evaluation that, the algorithm performs better when trained with more delinquent data. We can re-develop this algorithm and fine tune the tree parameters for the Q12007-Q42008 period which might enable us to create an algorithm with higher accuracy as far as the confusion matrix is concerned.
- We can explore more algorithms for classification such as KNN or SVM to check the accuracy.
- There might be an optimum parameter for Neural Network that exists which we haven't come across in our analysis.

Summary

We have analyzed the data given in the Freddie Mac Single Family Loans Dataset. We have seen different parameters related to the origination and performance files. These have been summarized to perform EDA on the data. Data has been automatically downloaded using HTML scraping techniques. We have performed Lasso Regression, recursive feature elimination, F regression and PCA for performing feature extraction and selection. We have analyzed different algorithms such as Regression, Random Forest and Neural Networks to perform Predictive and Classification modelling and tested the algorithm on Quarters from 1999-2016 in a rolling quarters manner. According to our analysis, Random Forest was the algorithm of choice for Prediction as well as Classification. We have done extensive analysis for the period 2007-2009 to analyze different aspects of the market crash.