11/30/2018

Freddie Loan Data Analysis

ADS- Midterm Assignment

TEAM 5
Ankit Yadav
Ritu Agrawal
Pritam Jain

Report

The report summarizes the analysis performed on Single-family loan data, provided by Freddie Mac(http://www.freddiemac.com/news/finance/sf_loanlevel_dataset.html). The data provided by Freddie Mac consists of the following details:

Mortgages originated from January 1, 1999, through the "Origination Cutoff Date", with monthly loan performance data through the "Performance Cutoff Date," that were sold to Freddie Mac or back Freddie Mac Participation Certificates (PCs).

 Fully amortizing 15-, 20-, and 30-year fixed-rate mortgages

 Mortgages categorized as having verified or waived documentation.

We then build predictive analytics models using the datasets. The problem presented is divided into 2 sections:

Section 1: Data wrangling

- Data Download and pre-processing
- Exploratory Data analysis

Section 2: Building and evaluating models

- **Prediction** using Linear Regression, Random Forest, Neural Network KNN Algorithms
- **Classification** using Logistic Regression, Random Forest, Neural Network, SVN Algorithms

1. Part I: Data Ingestion and Wrangling

1.1 THE DATA

Single Family Loan-Level Dataset

For each calendar quarter, there is one file containing loan **origination data** and one file containing **monthly performance data** for each loan in the **origination data file**.

Freddie Mac has created a smaller dataset for those who may not require, or have the capability, to download the full Dataset. The sample dataset is a simple random sample6 of 50,000 loans selected from each full vintage year and a proportionate number of loans from each partial vintage year of the full Single-Family Loan-Level Dataset. Each vintage year has one origination data file and one corresponding monthly performance data file, containing the same loan-level data fields as those included in the full Dataset. Due to the size of the dataset, the data has been broken up and compressed as detailed below. The files are organized chronologically by year and quarter.

Dataset	File Name Format	Contents	File Type	Delimiter
Full	historical_data1_QnYYYY.zip	historical_data1_QnYYYY.txt	Origination Data	Pipe ("|")
		historical_data1_time_QnYYYY.txt	Monthly Performance Data	
Sample	sample_YYYY.zip	sample_orig_YYYY.txt	Origination Data	Pipe ("|")
		Sample_svcg_YYYY.txt	Monthly Performance Data	

Data Download and pre-processing:

The very first challenge was to programmatically download the data from Freddie Mac website (https://freddiemac.embs.com/FLoan/Data/download.php) and download and preprocess the "SAMPLE" file both for origination and performance data.

To download the file programmatically, first the user should register him/herself by creating username and password. Once logged in, the user can download all the file required for analysis. We have used the python requests library for this purpose. To store the user credential, we need to store them in the request session so that user didn't redirect back to the login page whenever he/she required to download a file from the Freddie Mac posted dataset.

```
import requests
import os
import json
from bs4 import BeautifulSoup
import urllib.request as rq
import urllib
from zipfile import ZipFile
from io import BytesIO
import lxml
import pandas as pd
import sys
from configparser import ConfigParser


# In[1]:

# Logging in to the website and downloading the sample files from 2005 onwards
def downloadSampleFiles(username, password, startYear, endYear):
    LOGIN_URL = "https://freddiemac.embs.com/FLoan/secure/auth.php"
    URL = "https://freddiemac.embs.com/FLoan/Data/download.php"
    with requests.session() as c:

        payload = {'username': username, 'password': password, 'action': 'acceptTandC', 'acceptSubmit': 'Continue',
                   'accept': 'Yes'}
```

Once, the user is logged in to the website, we will be using request session to drive the further functionality. We will be using **Beautiful Soup** package, a powerful python package for data scrapping from the Freddie Mac Website and download all the "**Sample**" files for our analysis purpose.

```
        if not year_list:
            print('Sample Files already exist!!!')
            exit(0)
        else:
            print('Starting sample files download!!', '\n')
            files_required = []
            count = 0
            for link in list_of_links:
                for year in year_list:
                    if ('sample_' + str(year)) in link:
                        count = count + 1
                    if count > 0:
                        files_required.append([link, 'sample_' + str(year)])
                    count = 0

            for file, filename in files_required:
                samplefile_response = c.get(file)
                samplefile_content = ZipFile(BytesIO(samplefile_response.content))
                samplefile_content.extractall(download_path + filename)

            print('Sample files downloaded in the path: ' + download_path, '\n')


# In[2]:

# Created a function that retrieves the link that needs to be downloaded
def find_files(response):
    soup = BeautifulSoup(response.text, "lxml")

    download_url = 'https://freddiemac.embs.com/FLoan/Data/'
    hrefs = []

    for a in soup.find_all('a'):
        hrefs.append(download_url + a['href'])

    return hrefs
```

1.2 Data Preprocessing and Cleaning

Since, these files are big in size and consist of huge amount of data, we need to preprocess these files before getting saved in our drive. These Zip file consist of two files:

Origination File:

For origination file, we first analyze the file size for all the year. Using python pandas, we create a data frame where we append all the data from the sample file for all the year. Origination file consist of 27

cred_scr	fst_paymnt_dte	fst_hmebyr_flg	maturty_dte	metro_stat_area	mort_insur_pctg	nbr_units	occu_status	orig_cmbnd_ln_to_value	orig_dbt_to_incm	...
722	200504	N	203503	0	0	1	P	80	48	...
759	200503	N	203502	0	0	1	P	25	25	...
591	200504	N	203503	39100	0	1	P	48	34	...
792	200503	N	203502	39100	0	1	P	90	33	...
725	200503	N	203502	48864	0	1	P	49	41	...
788	200503	N	203502	0	0	1	P	75	55	...
691	200503	N	203502	0	35	1	P	97	48	...
687	200503	N	203502	0	0	1	P	40	46	...
784	200503	N	203502	0	0	1	P	35	30	...
703	200503	N	203502	0	0	1	P	67	53	...
778	200503	N	203502	37460	0	1	S	53	44	...

columns which consist of various details associated with the loan originated in each year. Some of the major columns are defined below:

We have many Null values and spaces (an invalid values) which we need to handle before using these files to compute a summary report. We must make sure that our data is in proper format with same datatype. We have created following functions:

```
def orig_fillNA(orig_df):
    orig_df['cred_scr'] = orig_df['cred_scr'].fillna(0)
    orig_df['fst_hmebyr_flg'] = orig_df['fst_hmebyr_flg'].fillna('Unknown')
    orig_df['metro_stat_area'] = orig_df['metro_stat_area'].fillna(0)
    orig_df['mort_insur_pctg'] = orig_df['mort_insur_pctg'].fillna(0)
    orig_df['nbr_units'] = orig_df['nbr_units'].fillna(0)
    orig_df['occu_status'] = orig_df['occu_status'].fillna('Unknown')
    orig_df['orig_cmbnd_ln_to_value'] = orig_df['orig_cmbnd_ln_to_value'].fillna(0)
    orig_df['orig_dbt_to_incm'] = orig_df['orig_dbt_to_incm'].fillna(0)
    orig_df['orig_ln_to_value'] = orig_df['orig_ln_to_value'].fillna(0)
    orig_df['chnl'] = orig_df['chnl'].fillna('Unknown')
    orig_df['pre_pnl_mort_flg'] = orig_df['pre_pnl_mort_flg'].fillna('Unknown')
    orig_df['proptype'] = orig_df['proptype'].fillna('Unknown')
    orig_df['zipcode'] = orig_df['zipcode'].fillna(0)
    orig_df['ln_purps'] = orig_df['ln_purps'].fillna('Unknown')
    orig_df['nbr_brwrs'] = orig_df['nbr_brwrs'].fillna(0)
    orig_df['spr_confrm_flg'] = orig_df['spr_confrm_flg'].fillna('N')

    return orig_df
```

Performance File:

For performance file, we first analyze the file size for all the year. Using python pandas, we create a data frame where we append all the data for all the year. Performance file consist of 27 columns which consist of various information about the loan origination in a year. Since, the size of the file is very large, we decide to summarize the input file for all the year during its preprocessing. Some of the major columns are defined below:

ln_sq_nbr	min_current_aupb	max_current_aupb	min_curr_ln_delin_status	max_curr_ln_delin_status	min_zero_bal_cd	max_zero_bal_cd	min_mi_recoveries	max_mi_reco
F105Q1000064	0.00	62000.00	0.0	0.0	0.0	1.0	0.0	
F105Q1000076	0.00	197000.00	0.0	0.0	0.0	1.0	0.0	
F105Q1000087	0.00	100000.00	0.0	1.0	0.0	1.0	0.0	
F105Q1000130	0.00	334000.00	0.0	0.0	0.0	1.0	0.0	
F105Q1000195	0.00	125000.00	0.0	0.0	0.0	1.0	0.0	
F105Q1000217	0.00	145000.00	0.0	0.0	0.0	1.0	0.0	
F105Q1000227	61175.12	73000.00	0.0	4.0	0.0	0.0	0.0	
F105Q1000261	0.00	61000.00	0.0	0.0	0.0	1.0	0.0	
F105Q1000282	0.00	80000.00	0.0	0.0	0.0	1.0	0.0	
F105Q1000305	0.00	100000.00	0.0	0.0	0.0	1.0	0.0	
F105Q1000315	0.00	170000.00	0.0	0.0	0.0	6.0	0.0	

As we have many empty column values in our origination and preprocessing file, we need to clean those to ensure that we don't have any NAN/NA value in our data. Also, we need to take care of the data type of column. These columns will be required while creating the summary matrices.

```
def performance_fillNA(perf_df):
    perf_df['curr_ln_delin_status'] = perf_df['curr_ln_delin_status'].fillna(0)
    perf_df['repurch_flag'] = perf_df['repurch_flag'].fillna('Unknown')
    perf_df['mod_flag'] = perf_df['mod_flag'].fillna('N')
    perf_df['zero_bal_cd'] = perf_df['zero_bal_cd'].fillna(00)
    perf_df['zero_bal_eff_dt'] = perf_df['zero_bal_eff_dt'].fillna('999901')
    perf_df['current_dupb'] = perf_df['current_dupb'].fillna(0)
    perf_df['lst_pd_inst_duedt'] = perf_df['lst_pd_inst_duedt'].fillna('999901')
    perf_df['mi_recoveries'] = perf_df['mi_recoveries'].fillna(0)
    perf_df['net_sale_proceeds'] = perf_df['net_sale_proceeds'].fillna('U')
    perf_df['non_mi_recoveries'] = perf_df['non_mi_recoveries'].fillna(0)
    perf_df['expenses'] = perf_df['expenses'].fillna(0)
    perf_df['legal_costs'] = perf_df['legal_costs'].fillna(0)
    perf_df['maint_pres_costs'] = perf_df['maint_pres_costs'].fillna(0)
    perf_df['taxes_and_insur'] = perf_df['taxes_and_insur'].fillna(0)
    perf_df['misc_expenses'] = perf_df['misc_expenses'].fillna(0)
    perf_df['actual_loss_calc'] = perf_df['actual_loss_calc'].fillna(0)
    perf_df['mod_cost'] = perf_df['mod_cost'].fillna(0)

    return perf_df
```

Once, we are done with the cleaning of the performance file, we will create a summarized version of the file based on certain column which are important for us. For this step, we created a function which will get the Max/Min/Average value of the columns in our summarized performance file.

1.3 Creating Summarized CSV (Output)

Once the preprocessing and data cleaning steps are performed, we will have created our final output file, one for origination file named 'OriginationCombined' and for performance file named 'PerformanceCombined. These final files will be used for our analysis performed in part 2.

```
with open(actual_file, 'r', encoding='utf-8') as f:
    perf_df = pd.read_csv(f, sep="|",
                          names=['ln_sq_nbr', 'mon_rpt_prd', 'current_aupb', 'curr_ln_delin_status',
                                 'loan_age', 'remng_mon_to_leg_matur', 'repurch_flag', 'mod_flag',
                                 'zero_bal_cd', 'zero_bal_eff_dt', 'current_int_rte', 'current_dupb',
                                 'lst_pd_inst_duedt', 'mi_recoveries', 'net_sale_proceeds',
                                 'non_mi_recoveries', 'expenses', 'legal_costs', 'maint_pres_costs',
                                 'taxes_and_insur', 'misc_expenses', 'actual_loss_calc', 'mod_cost',
                                 'stp_mod_flg','def_pymnt_mod','est_loan_to_vlv'],
                             skipinitialspace=True,error_bad_lines=False, low_memory=False)

    perf_df['curr_ln_delin_status'] = [999 if x == 'R' else x for x in
                                       (perf_df['curr_ln_delin_status'].apply(lambda x: x))]
    perf_df['curr_ln_delin_status'] = [0 if x == 'XX' else x for x in
                                       (perf_df['curr_ln_delin_status'].apply(lambda x: x))]
    perf_df = performance_fillNA(perf_df)

    perf_df[
        ['curr_ln_delin_status', 'loan_age', 'remng_mon_to_leg_matur', 'zero_bal_cd', 'current_dupb',
         'actual_loss_calc','est_loan_to_vlv']] = perf_df[
        ['curr_ln_delin_status', 'loan_age', 'remng_mon_to_leg_matur', 'zero_bal_cd', 'current_dupb',
         'actual_loss_calc','est_loan_to_vlv']].astype('float64')

    perf_df[['mon_rpt_prd', 'zero_bal_eff_dt', 'lst_pd_inst_duedt','stp_mod_flg','def_pymnt_mod']] = perf_df[
        ['mon_rpt_prd', 'zero_bal_eff_dt', 'lst_pd_inst_duedt','stp_mod_flg','def_pymnt_mod']].astype('str')

    filtered_df = minmax(perf_df)

    filtered_df['Year_Perf'] = ['20' + x for x in (filtered_df['ln_sq_nbr'].apply(lambda x: x[2:4]))]

    if firstRun == 1:
        filtered_df.to_csv(combinedfile_name, mode='a', header=True, index=False)
        firstRun = 0
    else:
        filtered_df.to_csv(combinedfile_name, mode='a', header=False, index=False)
```

NOTE: As we are working on two different files, we need to know about the data and the relationship between the two files created. In origination file, we have **ln_sq_nbr** which is a unique loan sequence number with quarter and year of loan origination attached to it. In performance file, for a given year we have multiple rows associated with a loan number which depicts its performance. We don't have any loan number duplicated in origination file with respect to year. It is always **Unique**.

PART 2: Exploratory Data Analysis

2.1 Analysis – Jupyter Notebook

In Part 2, we were asked to write Jupyter notebook using R/Python to graphically represent different summaries of data and summarize our findings in this notebook.

We first create a pandas' data frame for the origination file and group the data on the year.

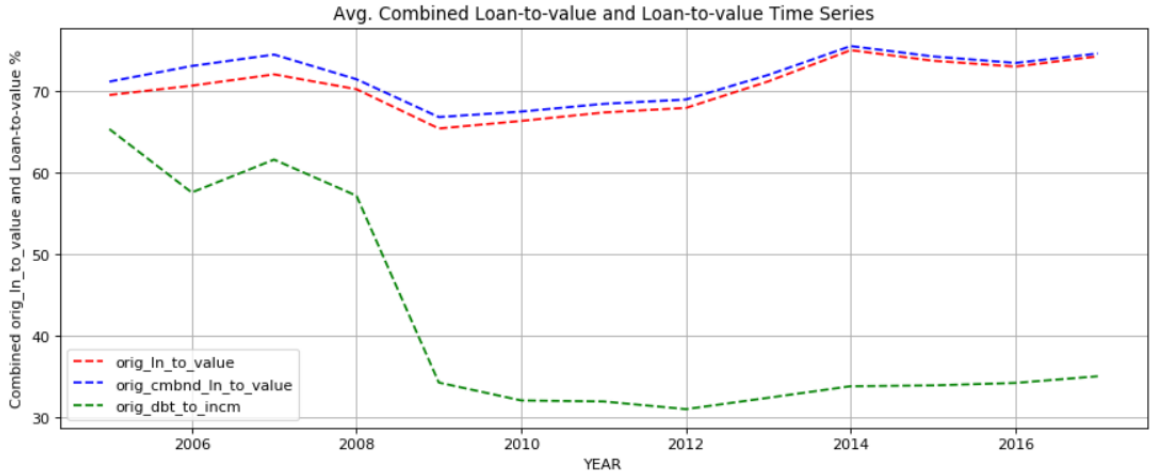
```
summ_df = pd.DataFrame()
grouped = df.groupby('Year_Orig')
summ_df = summ_df.append(grouped.aggregate(np.mean))
summ_df['loancount']=df['cred_scr'].groupby(df['Year_Orig']).count()
summ_df['year'] = summ_df.index
del summ_df['fst_paymnt_dte']
del summ_df['zipcode']
del summ_df['nbr_brwrs']
del summ_df['orig_ln_trm']
del summ_df['nbr_units']
del summ_df['metro_stat_area']
del summ_df['maturty_dte']
summ_df
```

Year_Orig	cred_scr	mort_insur_pctg	orig_cmbnd_ln_to_value	orig_dbt_to_incm	orig_upb	orig_ln_to_value	orig_intrst_rate	loancount	year
2005	728.499040	4.384360	71.216100	65.381640	170691.860000	69.568040	5.805862	50000	2005
2006	729.955000	3.476260	73.131020	57.585180	179592.580000	70.713940	6.406876	50000	2006
2007	728.461540	5.190540	74.538900	61.625620	183764.160000	72.103600	6.376952	50000	2007
2008	746.369047	4.268185	71.504450	57.184824	203978.499570	70.281966	6.057034	49999	2008
2009	762.336313	1.552389	66.853343	34.216716	213722.905542	65.449951	4.958592	50001	2009
2010	763.085820	1.776420	67.530100	32.019080	208388.820000	66.377100	4.637233	50000	2010
2011	764.000360	2.442760	68.477940	31.893660	217079.460000	67.420520	4.347664	50000	2011
2012	766.539360	3.066860	69.012900	30.950480	222671.620000	67.980500	3.609081	50000	2012
2013	758.535120	4.847660	72.036400	32.349620	217599.680000	71.241460	3.848064	50000	2013
2014	751.727660	7.074460	75.583360	33.768060	219868.100000	75.079060	4.287931	50000	2014
2015	751.740640	6.453200	74.307380	33.867120	229367.920000	73.776180	3.956772	50000	2015
2016	751.378880	6.138400	73.509260	34.171520	239712.880000	73.059320	3.776303	50000	2016
2017	748.426800	6.620933	74.671467	34.991253	236279.973333	74.320320	4.198448	37500	2017

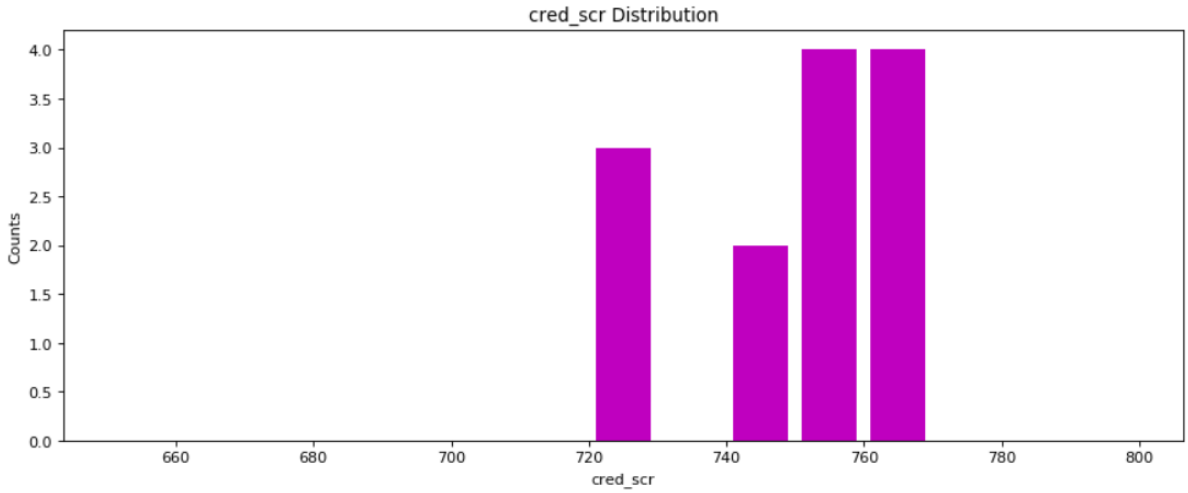
Here we show the various details associated with the origination file and the total loan count and mean of various important factor like credit score, interest rate based on year.

CLTV – LTV & DTI Comparison based on Year

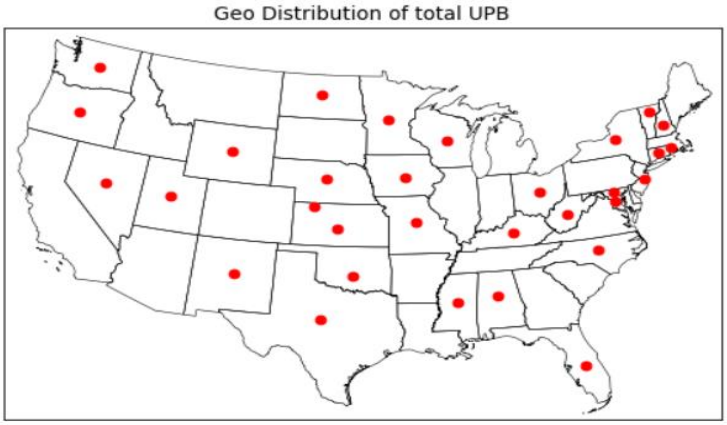
```
def plot_time_trends_1():
    orig_cmbnd_ln_to_value=summ_df['orig_cmbnd_ln_to_value']
    orig_ln_to_value=summ_df['orig_ln_to_value']
    Year_Orig=df['Year_Orig'].drop_duplicates()
    cred_scr=summ_df['cred_scr']
    orig_dbt_to_incm=summ_df['orig_dbt_to_incm']

    plt.figure(num=None, figsize=(12, 10), dpi=80, facecolor='w', edgecolor='b')

    ax1=plt.subplot(211)
    plt.plot(Year_Orig,orig_ln_to_value,'r--',Year_Orig,orig_cmbnd_ln_to_value,'b--',Year_Orig,orig_dbt_to_incm,'g--')
    plt.xlabel('YEAR')
    plt.ylabel('Combined orig_ln_to_value and Loan-to-value %')
    plt.legend(['orig_ln_to_value','orig_cmbnd_ln_to_value','orig_dbt_to_incm'])
    plt.grid(True)

    plt.title('Avg. Combined Loan-to-value and Loan-to-value Time Series')

    ax2=plt.subplot(212)
    bins=[650,660,670,680,690,700,710,720,730,740,750,760,770,780,790,800]
    plt.hist(cred_scr, bins=bins, histtype='bar', rwidth=0.8 ,color='m')
    plt.xlabel('cred_scr')
    plt.ylabel('Counts')
    plt.title('cred_scr Distribution')

    ax2.margins(0.05)
    plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0.25,
                        wspace=0.35)
    plt.show()
```



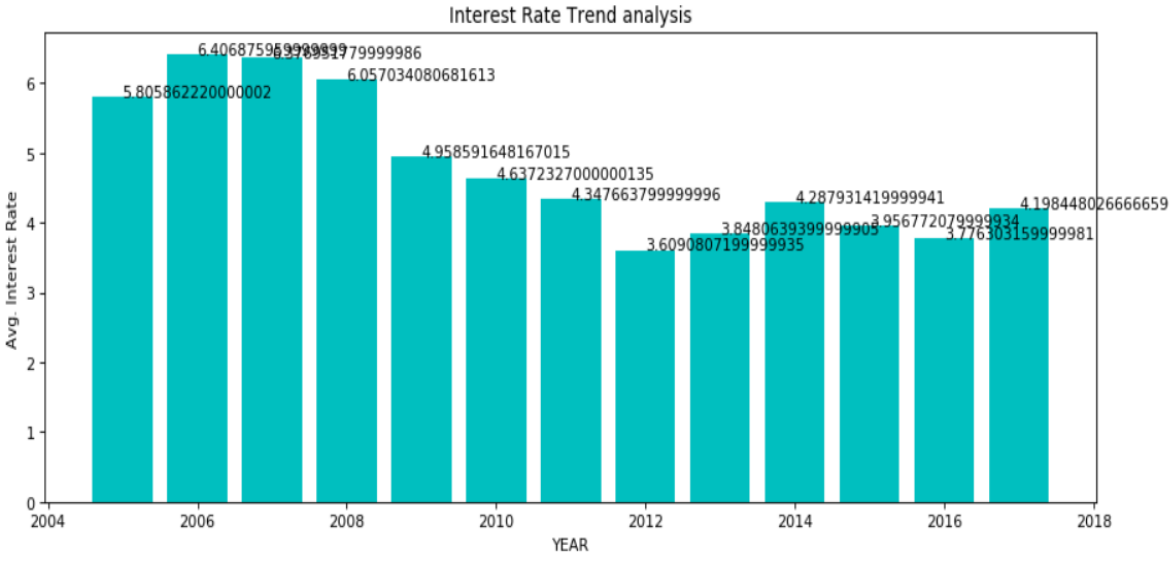
FICO Based on Count



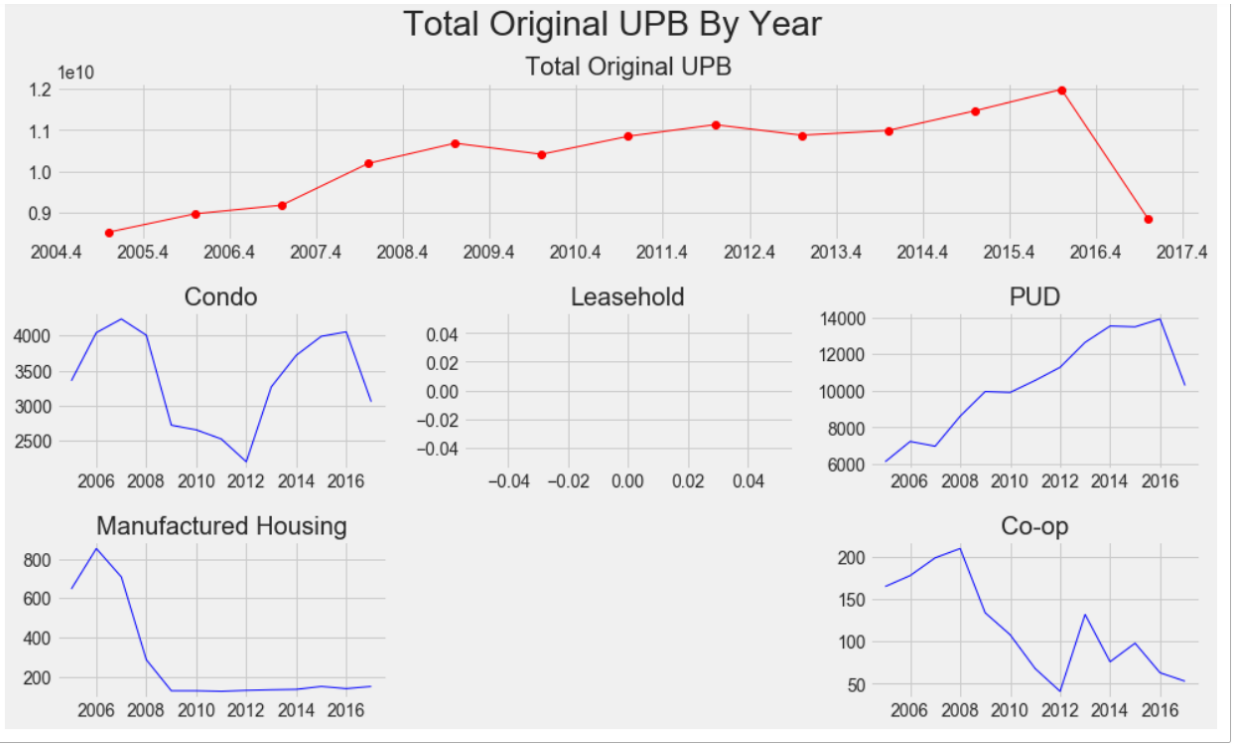
Count of Loan based on Geographical Presence



Average Credit Score based on Year

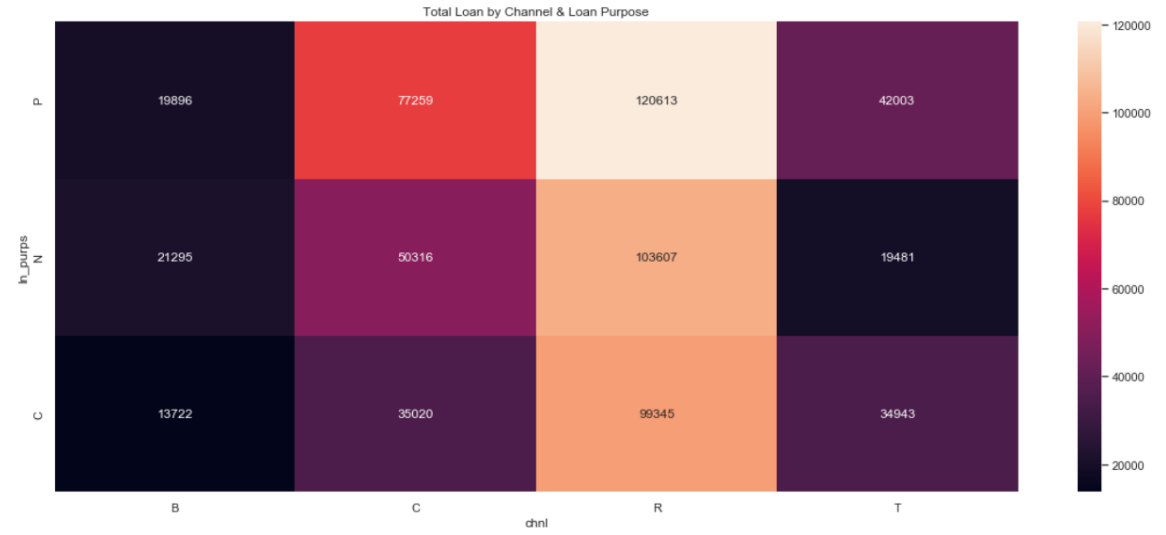


Average UPB based on Year



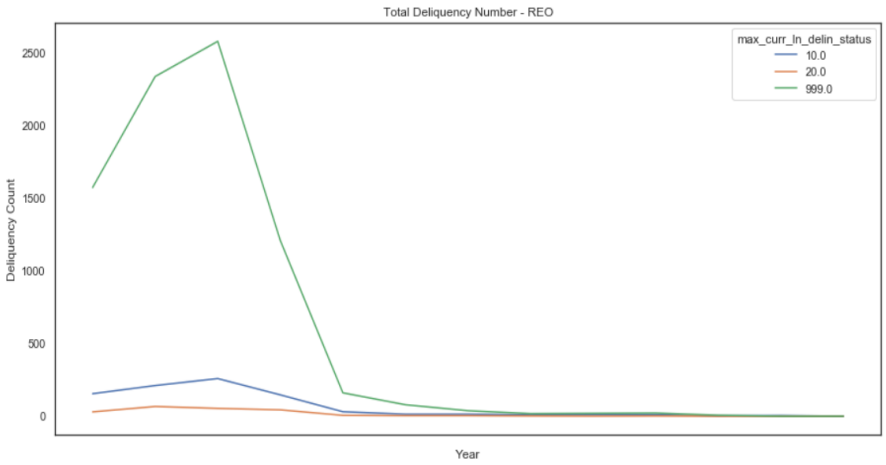
Total Loan Count by Purpose

```
import seaborn as sns
sns.set(style='white')

plt.figure(figsize=(20, 8))
plt.title('Total Loan by Channel & Loan Purpose')
ax =sns.heatmap(loan.T,mask= loan.T.isnull(),annot=True,fmt='g');
ax.invert_yaxis()
```

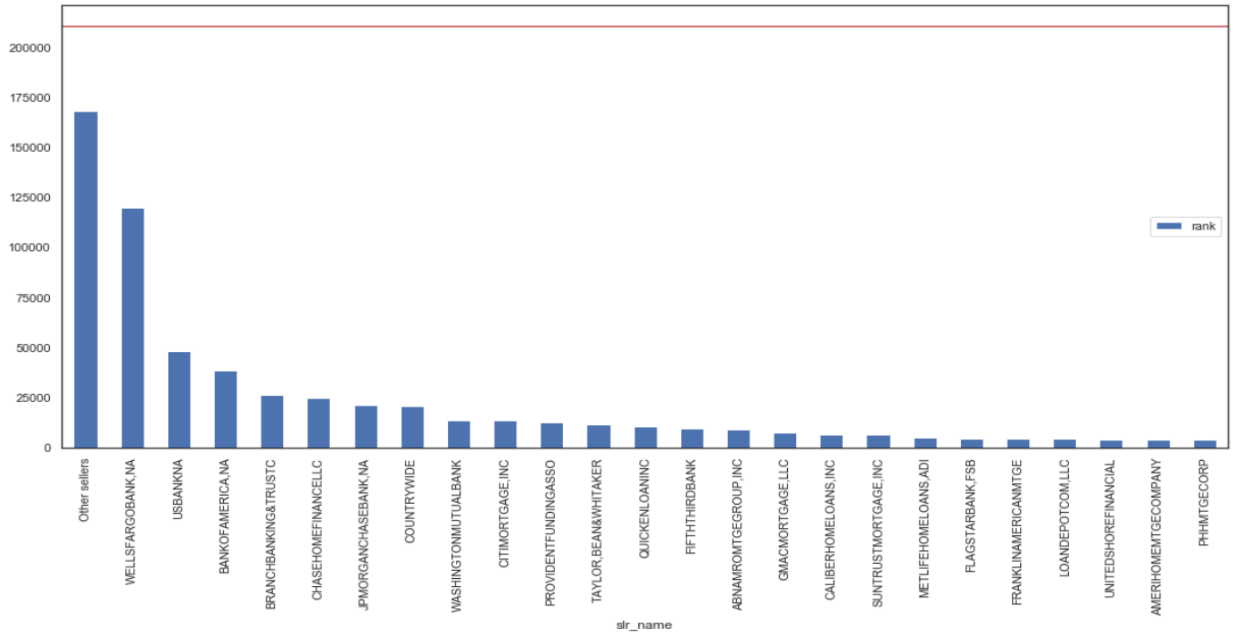


Total Deposition by Year

```
user[[10, 20,999]].plot(figsize=(15,7))
plt.title('Total Deliquency Number - REO')
plt.ylabel('Deliquency Count')
plt.show()
```



Top 25 Seller



2. Part II: Building and evaluating models

PART A- Prediction

Here, we are trying to perform Predictive Analytics on the Freddie Mac Single Family Loan Dataset. Freddie Mac is making this Dataset available at the direction of its regulator, the Federal Housing Finance Agency (FHFA) as part of a larger effort to increase transparency and help investors build more accurate credit performance models in support of ongoing and future credit risk-sharing transactions highlighted in FHFA's 2017 Conservatorship Scorecard.

This part of the project consists of the following parts.

1. Downloading the data by scraping the web
   Here, we perform web scraping to download data from the credentials provided. The credentials are provided using the configuration file.

```
--------------------------------------------------------------------------------
C:\Users\ritua/HistoricalInputFiles
https://freddiemac.embs.com/FLoan/Data/download.php?f=historical_data1_Q12005&s=292380058
C:\Users\ritua/HistoricalInputFiles
https://freddiemac.embs.com/FLoan/Data/download.php?f=historical_data1_Q22005&s=330868332

Downloading https://freddiemac.embs.com/FLoan/Data/download.php?f=historical_data1_Q22005&s=330868332: 100%|| 2/2 [07:10<00:00, 210.86s/it]
```

2. Preprocessing the downloaded data
   The data is preprocessed by filling the nulls, creating dummies.

Out[67]:

	fico	dt_first_pi	flag_fthb	dt_matr	cd_msa	mi_pct	cnt_units	occpy_sts	cltv	dti	...	st	prop_type	zipcode	id_loan	loan_purpose	orig_loan_te
0	699	200505	N	203504	39300.0	0	1	P	56	42	...	RI	SF	2800.0	F105Q1000001	C	
1	691	200504	N	203503	36420.0	25	1	P	90	36	...	OK	SF	73000.0	F105Q1000002	N	
2	713	200503	N	203502	28740.0	0	1	P	72	45	...	NY	SF	12500.0	F105Q1000003	P	
3	719	200505	N	203504	NaN	0	1	S	85	47	...	MO	CO	65000.0	F105Q1000004	P	
4	656	200503	N	203502	40340.0	0	1	P	68	30	...	MN	SF	55900.0	F105Q1000005	C	

5 rows × 26 columns

In [68]:
```
df_test.head()
```

Out[68]:

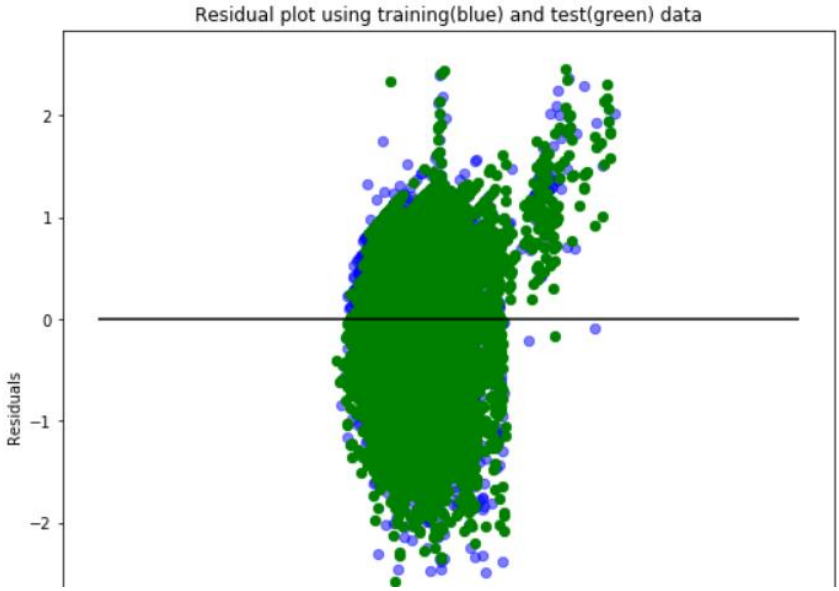
	fico	dt_first_pi	flag_fthb	dt_matr	cd_msa	mi_pct	cnt_units	occpy_sts	cltv	dti	...	st	prop_type	zipcode	id_loan	loan_purpose	orig_loan_te
0	715	200508	N	203507	33700.0	0	1	P	58	41	...	CA	SF	95300.0	F105Q2000001	C	
1	743	200508	N	203507	NaN	0	1	P	80	36	...	IL	SF	62400.0	F105Q2000002	N	
2	772	200508	N	203507	37860.0	0	1	P	80	34	...	FL	PU	32500.0	F105Q2000003	P	
3	773	200507	N	203506	NaN	0	1	P	64	60	...	MN	SF	56300.0	F105Q2000004	N	
4	647	200508	N	203507	NaN	30	1	P	91	28	...	MI	SF	49200.0	F105Q2000005	N	

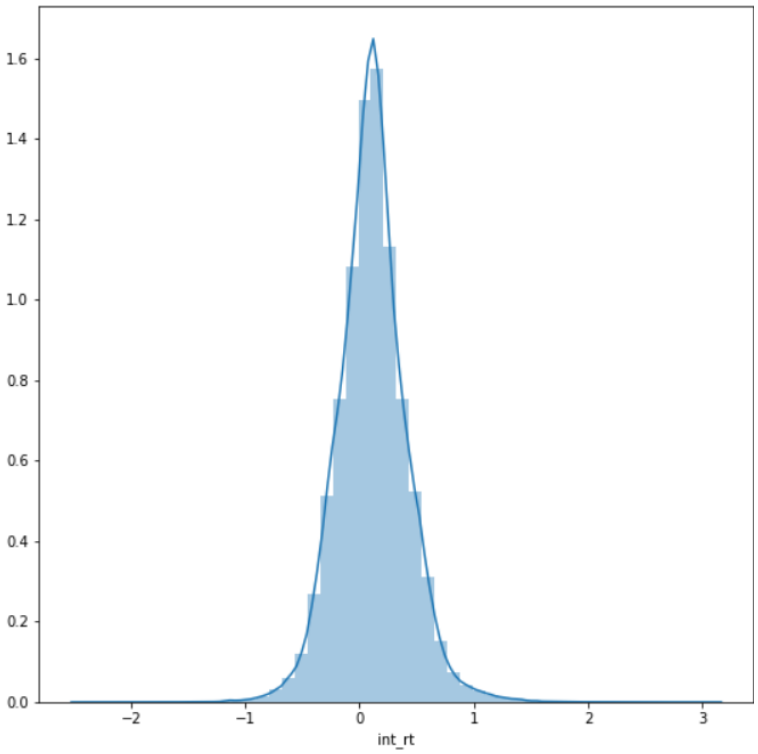
5 rows × 26 columns

3. Builds a Regression model for the interest rate using Q12005 data as training data (col 13)

   Here, the Linear Regression is used to perform predictive analytics.

```
linear regression:
0.37824718717748795
Linear Regression---------
Training Data
MAE:0.21357986375983826
RMSE:0.28998513312818
Median Absolute Error:0.16145840185246652
Testing Data:
MAE:0.2478038471138706
RMSE:0.3248587101740713
Median Absolute Error:0.19642159208547838
Linear Regression Peformed
```





4. Variable selection

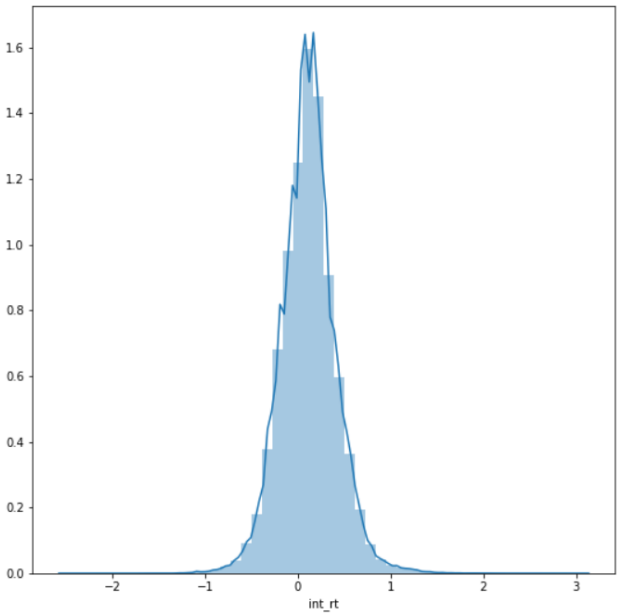
In [ ]:
```
def selectKBest(kb,x,y):
    b=SelectKBest(f_regression,k=20)
    b.fit(x,y)
    X_train = b.fit_transform(x,y)
    kb.fit(X_train,y)
    score = kb.score(X_train, y)
    pred=kb.predict(X_train)
    sc=r2_score(y,pred)
    print("select k best:")
    print(sc)
    return kb
```

In [ ]:
```
def FeatureSelectionTechniques(q1):
    org=lm
    print("Training Data")
    Data1=trainModel(q1)
    y_train1=Data1.int_rt
    Data1.drop('int_rt',axis=1,inplace=True)
    x_train1=Data1
    selectKBest(org,x_train1,y_train1)
```

We have also done feature selection using mlxtend. In this, we have implemented forward, backward and exhaustive search techniques.

5. Repeated this using Random Forest & Neural Network algorithms.
   We have performed Random Forest and Neural Network and computed the performance matrix. Here, we found that the Random Forest model performed the best.

```
Random Forest-------------
Training Data:
MAE:0.2045007111093127
RMSE:0.2762591519308622
Median Absolute Error:0.15721243794753192
Testing Data:
MAE:0.2430911425385264
RMSE:0.31595224213111467
Median Absolute Error:0.1928715650773416
Random Forest Regression Peformed
```



```
In [76]: calc_error_metric('Neural Network', nn_model, x_train1, y_train1, x_test1, y_test1)
```

Out[76]:

	Model	mae_test	mae_train	mape_test	mape_train	r2_test	r2_train	rms_test	rms_train
0	Linear Regression	0.247804	0.213580	4.226464	3.779000	0.141400	0.378247	0.324859	0.289985
0	Random Forest Regressor	0.243345	0.204917	4.153833	3.628222	0.186015	0.434457	0.316306	0.276566
0	Neural Network	1.468431	1.494523	25.636405	26.572565	-24.394059	-22.894344	1.766709	1.797688

Here we can see that random forest is the best

6. Using AutoML tools- TPOT, H20.Ai and AutoSKLearn AutoML algorithms

H2O

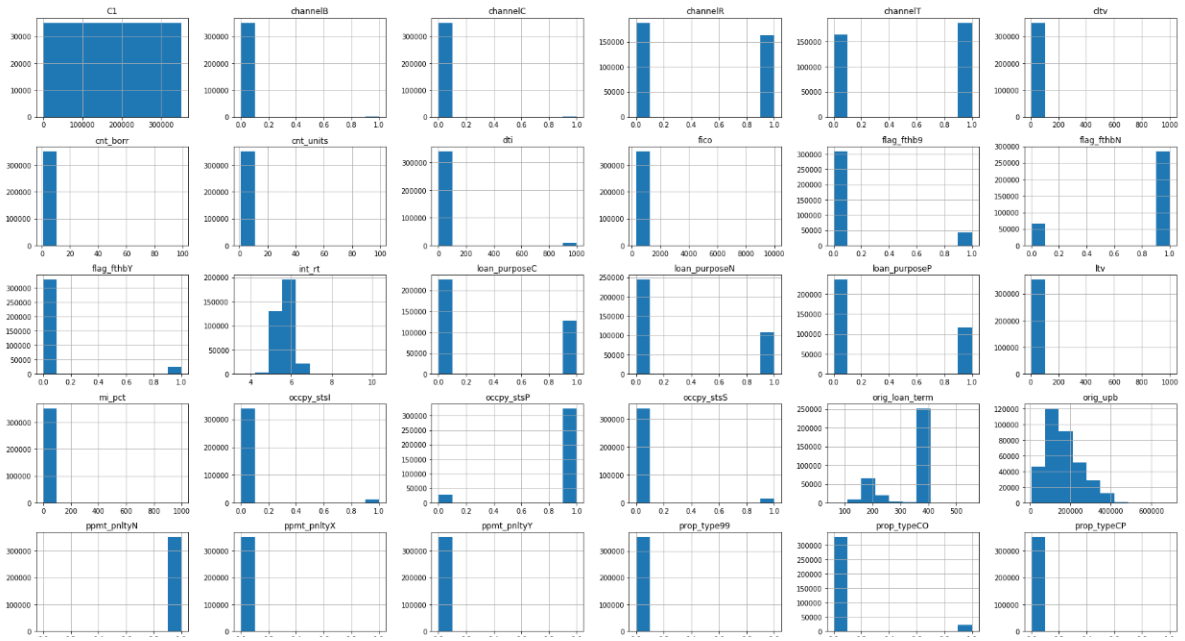
```
In [71]: import h2o
```

```
In [72]: h2o.init(nthreads = -1, max_mem_size = 8)
```

```
Checking whether there is an H2O instance running at http://localhost:54321..... not found.
Attempting to start a local H2O server...
; Java HotSpot(TM) 64-Bit Server VM (build 25.191-b12, mixed mode)
  Starting server from C:\Users\ritua\Anaconda3\lib\site-packages\h2o\backend\bin\h2o.jar
  Ice root: C:\Users\ritua\AppData\Local\Temp\tmpzg6mos4b
  JVM stdout: C:\Users\ritua\AppData\Local\Temp\tmpzg6mos4b\h2o_ritua_started_from_python.out
  JVM stderr: C:\Users\ritua\AppData\Local\Temp\tmpzg6mos4b\h2o_ritua_started_from_python.err
  Server is running at http://127.0.0.1:54321
Connecting to H2O server at http://127.0.0.1:54321... successful.
```

H2O cluster uptime:	01 secs
H2O cluster timezone:	America/New_York

```
[103]: import pylab as pl
data.as_data_frame().hist(figsize=(30,20))
pl.show()
```



```
n [108]: h2o_glm
```

```
Model Details
=============
H2OGeneralizedLinearEstimator :  Generalized Linear Modeling
Model Key:  GLM_model_python_1543465412803_1


ModelMetricsRegressionGLM: glm
** Reported on train data. **

MSE: 0.08387168932294989
RMSE: 0.28960609338021515
MAE: 0.21324998202673287
RMSLE: 0.043169994889018015
R^2: 0.37987150965378935
Mean Residual Deviance: 0.08387168932294989
Null degrees of freedom: 351633
Residual degrees of freedom: 351608
Null deviance: 47558.10781555675
Residual deviance: 29492.13760338616
AIC: 126434.7444267603

ModelMetricsRegressionGLM: glm
** Reported on cross-validation data. **

MSE: 0.08391938881159304
RMSE: 0.2896884340314488
MAE: 0.21327786933114193
RMSLE: 0.04318020160705125
R^2: 0.3795188303155429
Mean Residual Deviance: 0.08391938881159304
Null degrees of freedom: 351633
Residual degrees of freedom: 351607
Null deviance: 47558.583652359026
```

We have also used tpot for AutoML.

What-if Analysis: -

The what-if analysis uses the already trained models to predict the odd behaviors in the return in investments of share market. For example the year 1999 and 2013 saw economic boom as we saw it had 21% and 32% return on investments whereas there were years which saw financial crisis.

According to research on the economy and return on investments, presidential elections have also been a major reason for fluctuation of economy generally in a positive way.

This could be seen as we train the model with a data from some other year and try to predict the behavior of the data in the year that faced any such incident.

PART B- Classification

In this part 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- Data Download and cleansing

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.

For downloading data first we run the script and save after unzipping the data.

```
def getFilesFromFreddieMacPerQuarter(payload, quarter, testquarter):
    with requests.Session() as s:
        preUrl = s.post(url, data=payload)
        payload2 = {'accept': 'Yes', 'acceptSubmit': 'Continue', 'action': 'acceptTandC'}
        finalUrl = s.post(postUrl, payload2)
        linkhtml = finalUrl.text
        allzipfiles = BeautifulSoup(linkhtml, "html.parser")
        ziplist = allzipfiles.find_all('td')
        sampledata = []
        historicaldata = []
        count = 0
        # q =quarter[2:6]
        # t =testquarter[2:6]
        for li in ziplist:
            zipatags = li.findAll('a')
            for zipa in zipatags:
                fetchFile = 'historical_data1_'
                if (quarter in zipa.text):
                    if (fetchFile in zipa.text):
                        link = zipa.get('href')
                        foldername = 'HistoricalInputFiles'
                        Historicalpath = str(os.getcwd()) + "/" + foldername
                        assure_path_exists(Historicalpath)
                        finallink = 'https://freddiemac.embs.com/FLoan/Data/' + link
                        print(finallink)
                        historicaldata.append(finallink)
                elif (testquarter in zipa.text):
                    if (fetchFile in zipa.text):
                        link = zipa.get('href')
                        foldername = 'HistoricalInputFiles'
                        Historicalpath = str(os.getcwd()) + "/" + foldername
                        assure_path_exists(Historicalpath)
                        finallink = 'https://freddiemac.embs.com/FLoan/Data/' + link
                        print(finallink)
                        historicaldata.append(finallink)
        extracrtZip(s, historicaldata, Historicalpath)
```

Since the data file is very big. We upload it in chunks in the pandas dataframe.

```
def  getTrainData(trainQ):
    print("Starting train data download")
    downloadPath='./HistoricalInputFiles/historical_data1_time_'+trainQ+'.txt'

    c_size = 2500000
    df = pd.read_csv('./head.txt', sep="|",
                     names=['LOAN_SEQ_NO', 'MONTHLY_REPORTING_PERIOD', 'CURRENT_ACTUAL_UPB', 'CURR_LOAN_DEL_STATUS',
                            'LOAN_AGE', 'REM_MTH_LEGAL_MATURITY', 'REPURCHASE_FLAG', 'MODIFICATION_FLAG',
                            'ZERO_BALANCE_CODE', 'ZERO_BALANCE_EFF_DATE', 'CURRENT_INTEREST_DATE',
                            'CURRENT_DEFERRED_UPB', 'DUE_DATE_LAST_PAID_INST', 'MI_RECOVERIES',
                            'NET_SALES_PROCEEDS',
                            'NON_MI_RECOVERIES', 'EXPENSES', 'LEGAL_COSTS', 'MAIN_PRES_COSTS',
                            'TAXES_INSURANCE', 'MISC_EXPENSES', 'ACTUAL_LOSS', 'MODIFICATION_COST', 'STEP_MOD_FLAG',
                            'DEFERRED_PAYMENT_MODI', 'EST_LOAN_TO_VALUE'],
                     skipinitialspace=True, error_bad_lines=False, index_col=False, dtype='unicode')

    for gm_chunk in pd.read_csv(downloadPath, sep="|",
                                names=['LOAN_SEQ_NO', 'MONTHLY_REPORTING_PERIOD', 'CURRENT_ACTUAL_UPB',
                                       'CURR_LOAN_DEL_STATUS',
                                       'LOAN_AGE', 'REM_MTH_LEGAL_MATURITY', 'REPURCHASE_FLAG', 'MODIFICATION_FLAG',
                                       'ZERO_BALANCE_CODE', 'ZERO_BALANCE_EFF_DATE', 'CURRENT_INTEREST_DATE',
                                       'CURRENT_DEFERRED_UPB', 'DUE_DATE_LAST_PAID_INST', 'MI_RECOVERIES',
                                       'NET_SALES_PROCEEDS',
                                       'NON_MI_RECOVERIES', 'EXPENSES', 'LEGAL_COSTS', 'MAIN_PRES_COSTS',
                                       'TAXES_INSURANCE', 'MISC_EXPENSES', 'ACTUAL_LOSS', 'MODIFICATION_COST',
                                       'STEP_MOD_FLAG',
                                       'DEFERRED_PAYMENT_MODI', 'EST_LOAN_TO_VALUE'],
                                skipinitialspace=True, error_bad_lines=False, index_col=False, dtype='unicode',
                                chunksize=c_size):
        frames = [df, gm_chunk]
        df = pd.concat(frames)
        print(df.shape)
```

After downloading the data we check for missing values and get the values from following function

```
rowcnt = len(df.index)
# Example data
features = []
missngpercnt = []
#error = np.random.rand(len(features))
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
```

Depending on the following output, we get to know that for top 19 columns around 99 % data is missing. So, we can not use these columns for our classification algorithm.

Out[6]:

	Features	MissingPercentage
24	DEFERRED_PAYMENT_MODI	99.95132
23	STEP_MOD_FLAG	99.95132
7	MODIFICATION_FLAG	99.95132
13	MI_RECOVERIES	99.93172
20	MISC_EXPENSES	99.93172
15	NON_MI_RECOVERIES	99.93172
14	NET_SALES_PROCEEDS	99.93172
18	MAIN_PRES_COSTS	99.93172
19	TAXES_INSURANCE	99.93172
16	EXPENSES	99.93172
17	LEGAL_COSTS	99.93172
21	ACTUAL_LOSS	99.93172
12	DUE_DATE_LAST_PAID_INST	99.88900
9	ZERO_BALANCE_EFF_DATE	98.81684
8	ZERO_BALANCE_CODE	98.81684
6	REPURCHASE_FLAG	98.81652
22	MODIFICATION_COST	98.67084
25	EST_LOAN_TO_VALUE	98.51928
1	MONTHLY_REPORTING_PERIOD	0.00000
11	CURRENT_DEFERRED_UPB	0.00000
10	CURRENT_INTEREST_DATE	0.00000
5	REM_MTH_LEGAL_MATURITY	0.00000
4	LOAN_AGE	0.00000
3	CURR_LOAN_DEL_STATUS	0.00000
2	CURRENT_ACTUAL_UPB	0.00000
0	LOAN_SEQ_NO	0.00000

```
df = df.drop(columns=['REPURCHASE_FLAG', 'MODIFICATION_FLAG', 'ZERO_BALANCE_CODE', 'ZERO_BALANCE_EFF_DATE',
                      'DUE_DATE_LAST_PAID_INST', 'MI_RECOVERIES',
                      'NET_SALES_PROCEEDS', 'NON_MI_RECOVERIES', 'EXPENSES', 'LEGAL_COSTS', 'MAIN_PRES_COSTS',
                      'TAXES_INSURANCE',
                      'MISC_EXPENSES', 'ACTUAL_LOSS', 'MODIFICATION_COST', 'STEP_MOD_FLAG', 'DEFERRED_PAYMENT_MODI',
                      'EST_LOAN_TO_VALUE'])
```

After dropping the unnecessary columns, we clean the remaining data.

First we change the data types of the columns depending upon the values.

```
df.CURRENT_ACTUAL_UPB = df.CURRENT_ACTUAL_UPB.astype('float64')
df.CURRENT_DEFERRED_UPB = df.CURRENT_DEFERRED_UPB.astype('float64')
df.CURRENT_INTEREST_DATE = df.CURRENT_INTEREST_DATE.astype('float64')
df[['MONTHLY_REPORTING_PERIOD', 'LOAN_AGE', 'REM_MTH_LEGAL_MATURITY']] = df[
    ['MONTHLY_REPORTING_PERIOD', 'LOAN_AGE', 'REM_MTH_LEGAL_MATURITY']].astype('int64')
df[['LOAN_SEQ_NO', 'CURR_LOAN_DEL_STATUS']] = df[['LOAN_SEQ_NO', 'CURR_LOAN_DEL_STATUS']].astype('str')
df['CURR_LOAN_DEL_STATUS'] = [999 if x == 'R' else x for x in (df['CURR_LOAN_DEL_STATUS'].apply(lambda x: x))]
df['CURR_LOAN_DEL_STATUS'] = [0 if x == 'XX' else x for x in (df['CURR_LOAN_DEL_STATUS'].apply(lambda x: x))]
```

Then we add two new columns for year and quarter for future use in classification matrix.

```
df['YEAR'] = ['19' + x if x == '99' else '20' + x for x in (df['LOAN_SEQ_NO'].apply(lambda x: x[2:4]))]
df['QUARTER'] = df['LOAN_SEQ_NO'].apply(lambda x: x[4:6])
```

Then we add anew column as DELINQUENT depending upon the values from column - CURRENT LOAN DEL STATUS and then drop this column as this is not required in the classification data.

```
df[['CURR_LOAN_DEL_STATUS']] = df[['CURR_LOAN_DEL_STATUS']].astype('int64')
df['DELINQUENT'] = (df.CURR_LOAN_DEL_STATUS > 0).astype(int)
```

Now, we have clean data and we are ready for our classification algorithms.

We do the same for testing quarter data and save both in Train_DF and Test_DF.

We take column DELINQUENT as our Y variable.

In [5]:
```
df=pd.read_csv('trainData_' + trainQ + '.csv')
traincols=['MONTHLY_REPORTING_PERIOD','CURRENT_ACTUAL_UPB','LOAN_AGE',
           'REM_MTH_LEGAL_MATURITY','CURRENT_INTEREST_DATE','CURRENT_DEFERRED_UPB']
y_train=df['DELINQUENT']
Train_DF=df[traincols]
Train_DF.head()
```

Out[5]:

	MONTHLY_REPORTING_PERIOD	CURRENT_ACTUAL_UPB	LOAN_AGE	REM_MTH_LEGAL_MATURITY	CURRENT_INTEREST_DATE	CURRENT_DEFERRED_UPB
0	200504	190000.0	0	360	5.625	0.0
1	200505	190000.0	1	359	5.625	0.0
2	200506	190000.0	2	358	5.625	0.0
3	200507	189000.0	3	357	5.625	0.0
4	200508	189000.0	4	356	5.625	0.0

In [6]:
```
test_df=pd.read_csv('testData_' + testQ + '.csv')
testcols = ['MONTHLY_REPORTING_PERIOD', 'CURRENT_ACTUAL_UPB', 'LOAN_AGE',
            'REM_MTH_LEGAL_MATURITY', 'CURRENT_INTEREST_DATE', 'CURRENT_DEFERRED_UPB']
y_test = test_df['DELINQUENT']
Test_DF = test_df[testcols]
Test_DF.head()
```

Out[6]:

	MONTHLY_REPORTING_PERIOD	CURRENT_ACTUAL_UPB	LOAN_AGE	REM_MTH_LEGAL_MATURITY	CURRENT_INTEREST_DATE	CURRENT_DEFERRED_UPB
0	200507	214000.0	0	360	5.75	0.0
1	200508	214000.0	1	359	5.75	0.0
2	200509	214000.0	2	358	5.75	0.0
3	200510	213000.0	3	357	5.75	0.0
4	200511	213000.0	4	356	5.75	0.0

Now we run classification algorithms on these.

Part 2- Running Algorithms

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.

Logistic Regression

In [7]:
```
model = LogisticRegression()
mod_fit = model.fit(Train_DF,y_train)
```

Accuracy of Train

In [8]:
```
print(model.score(Train_DF,y_train))
```

```
0.9568720224673917
```

In [9]:
```
pred = mod_fit.predict(Test_DF)
```

Accuracy of Test

In [10]:
```
metrics.accuracy_score(y_test,pred)
```

Out[10]:
```
0.9523774930957131
```

Confusion Matrix

In [11]:
```
print (metrics.confusion_matrix(y_test,pred))
```

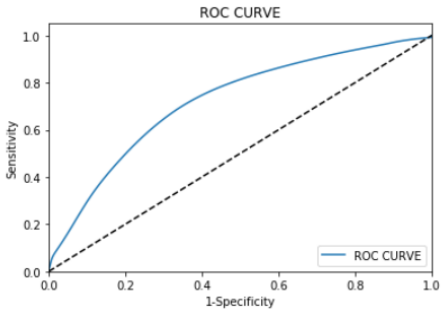
```
[[28448700      179]
 [ 1422367       50]]
```

In [12]:
```
preds = model.predict_proba(Test_DF)[:,1]
fpr,tpr, _ = roc_curve(y_test,preds)
```

ROC Curve

The ROC curve based on this algorithms is:

In [13]:
```
import matplotlib.pylab as plt

plt.figure()
plt.plot(fpr,tpr,label= "ROC CURVE")
plt.plot([0,1],[0,1],'k--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('1-Specificity')
plt.ylabel('Sensitivity')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()
```



Also we used logit for detailed analysis.

In [15]:
```
logit = sm.Logit(y_train, Train_DF[traincols])
logregression_result = logit.fit()
print(logregression_result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.165377
         Iterations 8
                           Logit Regression Results
==============================================================================
Dep. Variable:             DELINQUENT   No. Observations:             26508106
Model:                          Logit   Df Residuals:                 26508100
Method:                           MLE   Df Model:                            5
Date:                Thu, 29 Nov 2018   Pseudo R-squ.:                0.06963
Time:                        13:25:38   Log-Likelihood:           -4.3838e+06
converged:                       True   LL-Null:                  -4.7119e+06
                                        LLR p-value:                     0.000
================================================================================================
                                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------------
MONTHLY_REPORTING_PERIOD -2.979e-05   5.36e-08   -555.545      0.000   -2.99e-05   -2.97e-05
CURRENT_ACTUAL_UPB        1.811e-06   1.28e-08    141.005      0.000    1.79e-06    1.84e-06
LOAN_AGE                     0.0204   2.58e-05    792.562      0.000       0.020       0.020
REM_MTH_LEGAL_MATURITY       0.0043   1.32e-05    327.702      0.000       0.004       0.004
CURRENT_INTEREST_DATE        0.0356      0.002     22.355      0.000       0.033       0.039
CURRENT_DEFERRED_UPB     -8.431e-06    1.9e-07    -44.489      0.000    -8.8e-06   -8.06e-06
================================================================================================
```

Finally, the values of other coefficients:

In [16]:
```
print(logregression_result.conf_int())
```

```
                                0         1
MONTHLY_REPORTING_PERIOD -0.000030 -0.000030
CURRENT_ACTUAL_UPB        0.000002  0.000002
LOAN_AGE                  0.020375  0.020476
REM_MTH_LEGAL_MATURITY    0.004304  0.004356
CURRENT_INTEREST_DATE     0.032503  0.038750
CURRENT_DEFERRED_UPB     -0.000009 -0.000008
```

In [17]:
```
model.coef_
```

Out[17]:
```
array([[-2.92169587e-05,  1.78156846e-06,  2.09367968e-02,
         4.51494852e-03, -3.67804723e-05, -1.06894916e-05]])
```

Random Forest Algorithms:

We chose random forest as best algorithm due to its accuracy.

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

```
[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed:  9.0min finished
```

Train Accuracy

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

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

```
0.971850857143
```

Test Accuracy

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

```
[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed:   40.1s finished
```

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

```
0.94709971428571427
```

Confusion Matrix

```
#pd.crosstab(pred, y_test,rownames=['pred'],colnames=['ytest'])
confusion_matrix(y_test, pred, labels=None, sample_weight=None)
```

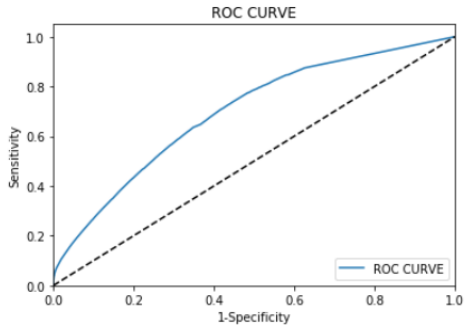
```
array([[6606972,   41808],
       [ 328494,   22726]])
```

```
preds = clf.predict_proba(Test_DF)[:,1]
fpr,tpr, _ = roc_curve(y_test,preds)
```

```
[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed:   39.9s finished
```

ROC Curve

```
import matplotlib.pylab as plt

plt.figure()
plt.plot(fpr,tpr,label= "ROC CURVE")
plt.plot([0,1],[0,1],'k--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('1-Specificity')
plt.ylabel('Sensitivity')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()
```



Neural Networks:

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)
```

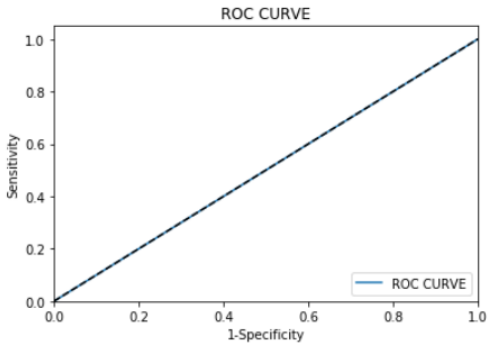
```
0.94982571428571427
```

```
pd.crosstab(y_test, pred,rownames=['pred'],colnames=['ytest'])
```

ytest	0
pred	
0	6648780
1	351220

ROC Curve

```
import matplotlib.pylab as plt

plt.figure()
plt.plot(fpr,tpr,label= "ROC CURVE")
plt.plot([0,1],[0,1],'k--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('1-Specificity')
plt.ylabel('Sensitivity')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()
```



Confusion Matrix

In [42]:
```
pd.crosstab(y_test, pred,rownames=['pred'],colnames=['ytest'])
```

Out[42]:

ytest	0
pred	
0	6648780
1	351220

In [37]:
```
preds = clf.predict_proba(Test_DF)[:,1]
fpr,tpr, _ = roc_curve(y_test,preds)
```

Evaluation Matrix

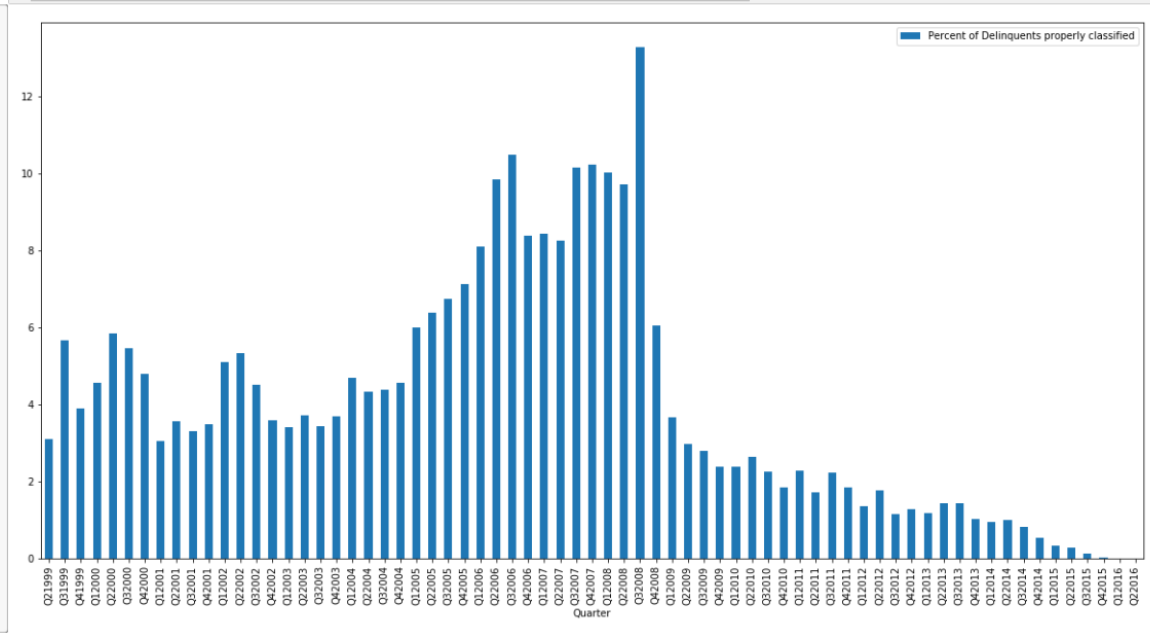
We downloaded the data from Q11999 to Q22016 and saved the evaluation matrix based on delinquents.

```
In [4]: evalmatrix_df=pd.read_csv(MATRIXPATH)
        evalmatrix_df.index=evalmatrix_df['Quarter']
        evalmatrix_df
```

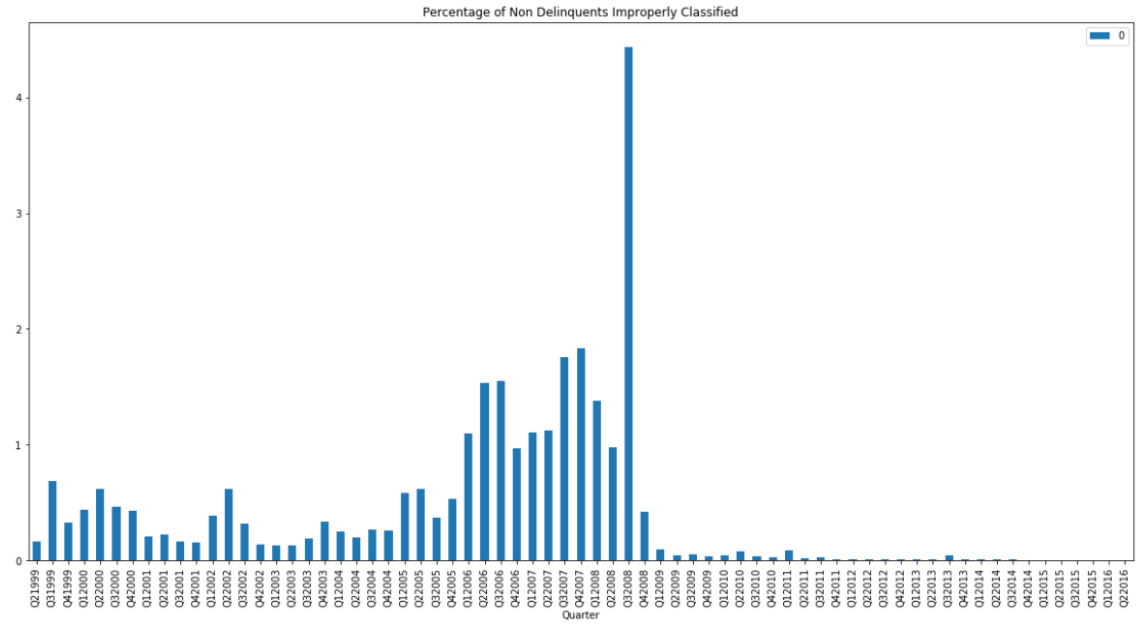
Out[4]:

Quarter	Quarter	NumActualDelinquents	NumOfPredictedDelinquents	NumRecordsInDataset	NumDelinquentsProperlyClassified	NumNonDelinquentsImproperlyCl
Q21999	Q21999	202868	202868	7000000	6300	
Q31999	Q31999	334974	334974	8797803	19027	
Q41999	Q41999	270684	270684	6133986	10562	
Q12000	Q12000	185631	185631	3939565	8486	
Q22000	Q22000	255207	255207	5056705	14922	
Q32000	Q32000	275979	275979	5570045	15075	
Q42000	Q42000	318650	318650	6105572	15328	
Q12001	Q12001	239303	239303	7000000	7287	
Q22001	Q22001	245489	245489	7000000	8728	
Q32001	Q32001	246858	246858	7000000	8152	
Q42001	Q42001	215040	215040	7000000	7523	
Q12002	Q12002	315441	315441	7000000	16082	
Q22002	Q22002	380429	380429	7000000	20265	
Q32002	Q32002	290579	290579	7000000	13114	
Q42002	Q42002	189361	189361	7000000	6812	
Q12003	Q12003	172801	172801	7000000	5880	
Q22003	Q22003	158763	158763	7000000	5907	
Q32003	Q32003	147551	147551	7000000	5078	
Q42003	Q42003	257707	257707	7000000	9508	
Q12004	Q12004	254357	254357	7000000	11934	
Q22004	Q22004	233285	233285	7000000	10134	
Q32004	Q32004	279508	279508	7000000	12270	
Q42004	Q42004	326638	326638	7000000	14932	
Q12005	Q12005	342274	342274	7000000	20505	

In [5]:
```
pd.DataFrame(evalmatrix_df['NumDelinquentsProperlyClassified']/evalmatrix_df['NumOfPredictedDelinquents']*100,columns=['Percent o
plt.show()
```

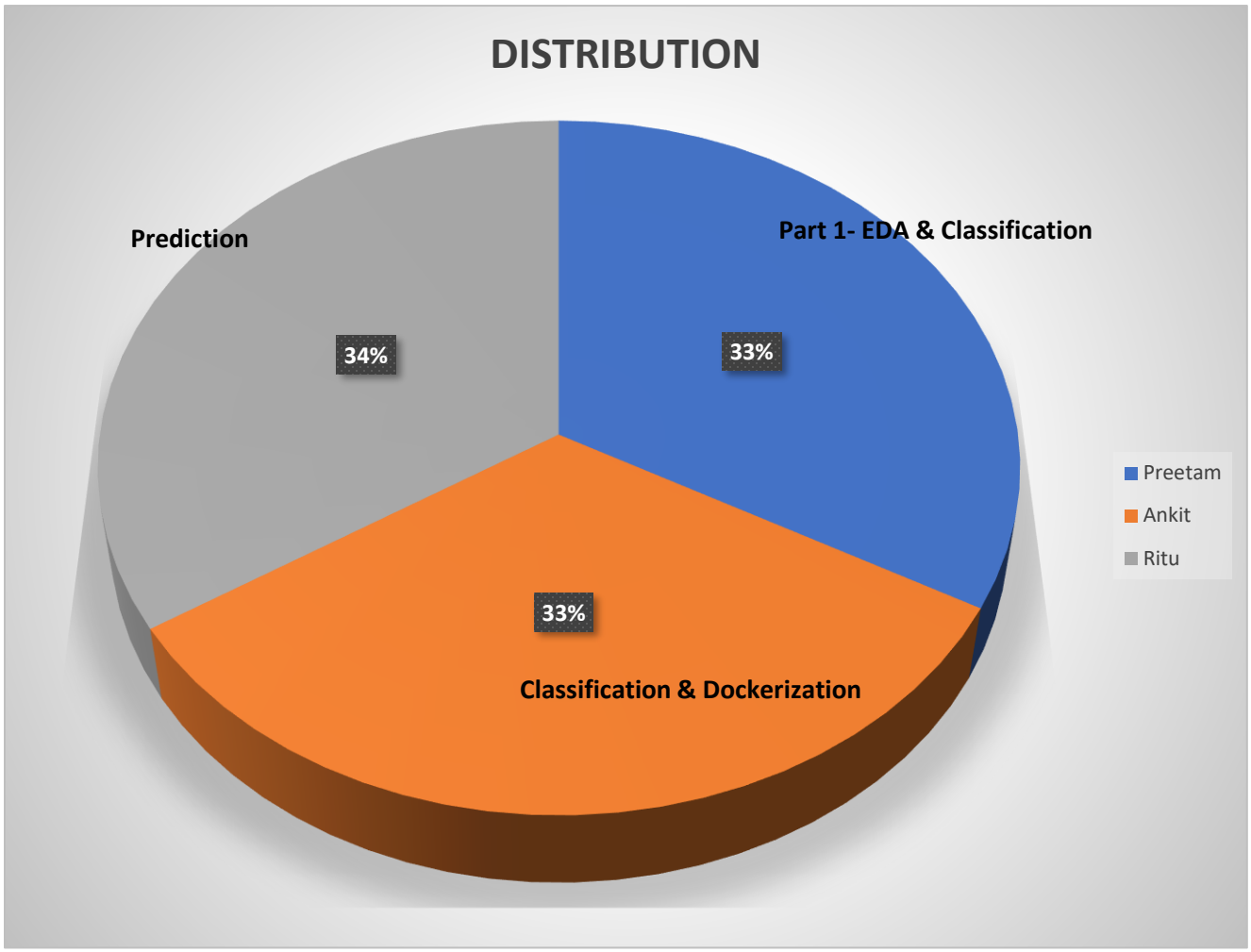


In [6]:
```
pd.DataFrame(100*evalmatrix_df['NumNonDelinquentsImproperlyClassified']/(evalmatrix_df['NumRecordsInDataset']-evalmatrix_df['NumA
plt.show()
```



Work distribution of team:

- Preetam Jain worked on Part 1 completely and helped Ankit in Classification.
- Ritu did complete work on Prediction
- Ankit completed classification and did dockerization of whole assignments including complete pipeline.



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