-

Advance Data Science

Single-family loan data

Mid Term

Team 5
Ankit Bhayani
Rajat Agrawal
Vishakha Sawant

SUMMARY

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 section:

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 1: 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 re
import os
from bs4 import BeautifulSoup
from urllib.request import urlopen
from zipfile import ZipFile
from io import BytesIO


url='https://freddiemac.embs.com/FLoan/secure/auth.php'
postUrl='https://freddiemac.embs.com/FLoan/Data/download.php'

def assure_path_exists(path):
    if not os.path.exists(path):
            os.makedirs(path)


def extracrtZip(monthlistdata,path):
    for month in monthlistdata:
        r = s.get(month)
        z = ZipFilequest(BytesIO(r.content))
        z.extractall(path)

payload={'username':'rajatddun@gmail.com','password':'Mu9GytRz'}
```

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.

```
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
    for li in ziplist:
        zipatags=li.findAll('a')
        for zipa in zipatags:
            if re.match('sample',zipa.text):
                link = zipa.get('href')
                foldername= 'Sample'
                Samplepath=str(os.getcwd())+"\\"+foldername
                assure_path_exists(Samplepath)
                finallink ='https://freddiemac.embs.com/FLoan/Data/' + link
                sampledata.append(finallink)
            elif re.match('historical',zipa.text):
                link = zipa.get('href')
                foldername= 'Historical'
                Historicalpath=str(os.getcwd())+"\\"+foldername
                assure_path_exists(Historicalpath)
                finallink ='https://freddiemac.embs.com/FLoan/Data/' + link
                historicaldata.append(finallink)
    print(len(historicaldata))
    extracrtZip(historicaldata,Historicalpath)
    print(len(sampledata))
    extracrtZip(sampledata,Samplepath)
```

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

fico	dt_first_pi	flag_fthb	dt_matr	cd_msa	mi_pct	cnt_units	occpy_sts	cltv	dti	...	prop_type	zipcode	id_loan	loan_purpose	orig_loan_term	cnt_bo
799	199903	N	202901	37620	0	1	O	70	25	...	SF	26100	F199Q1000012	N	359	1
635	200212	N	202904	10420	0	1	O	76	21	...	SF	44700	F199Q1000024	P	317	2
787	199905	N	202904	12060	0	1	O	80	18	...	PU	30500	F199Q1000036	P	360	2
726	199904	N	202903	28140	0	1	O	80	37	...	SF	66000	F199Q1000060	P	360	2
748	199905	X	202904	17140	0	1	O	80	28	...	SF	45200	F199Q1000097	N	360	2
720	199905	X	202904	42044	0	1	I	24	11	...	SF	92800	F199Q1000109	C	360	2
695	199905	X	202904	42044	0	1	O	40	6	...	SF	92800	F199Q1000121	C	360	2

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 fillNAN(df):
    df['fico'] = df['fico'].fillna(0)
    df['flag_fthb']=df['flag_fthb'].fillna('X')
    df['cd_msa']=df['cd_msa'].fillna(0)
    df['mi_pct']=df['mi_pct'].fillna(0)
    df['cnt_units']=df['cnt_units'].fillna(0)
    df['occpy_sts']=df['occpy_sts'].fillna('X')
    df['cltv']=df['cltv'].fillna(0)
    df['dti']=df['dti'].fillna(0)
    df['ltv']=df['ltv'].fillna(0)
    df['channel']=df['channel'].fillna('X')
    df['ppmt_pnlty']=df['ppmt_pnlty'].fillna('X')
    df['prop_type']=df['prop_type'].fillna('XX')
    df['zipcode']=df['zipcode'].fillna(0)
    df['loan_purpose']=df['loan_purpose'].fillna('X')
    df['cnt_borr']=df['cnt_borr'].fillna(0)
    df['flag_sc']=df['flag_sc'].fillna('N')
    return df

def changedatatype(df):
    #Change the data types for all column
    df[['fico','cd_msa','mi_pct','cnt_borr','cnt_units','cltv','dti','orig_upb','ltv','zipcode','orig_loan_term']] = df[
    df[['flag_sc','servicer_name']] = df[['flag_sc','servicer_name']].astype('str')
    return 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 23 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 summarizes the input file for all the year during its preprocessing. Some of the major columns are defined below:

id_loan	max_current_upb	min_current_upb	max_delq_sts	min_delq_sts	max_cd_zero_bal	min_cd_zero_bal	max_non_mi_recoveries	min_non_mi_re
F199Q1000012	42058.58	0.00	0	0	1	0	0.0	0.0
F199Q1000024	116426.12	0.00	0	0	1	0	0.0	0.0
F199Q1000036	124000.00	0.00	0	0	1	0	0.0	0.0
F199Q1000060	108000.00	0.00	0	0	1	0	0.0	0.0
F199Q1000097	115000.00	0.00	0	0	1	0	0.0	0.0
F199Q1000109	41000.00	0.00	1	0	1	0	0.0	0.0
F199Q1000121	135000.00	0.00	0	0	1	0	0.0	0.0
F199Q1000133	240000.00	0.00	0	0	1	0	0.0	0.0
F199Q1000157	196000.00	0.00	9	0	1	0	0.0	0.0
F199Q1000193	62000.00	41778.16	20	0	0	0	0.0	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 fillNA(df):
    df['delq_sts'] = df['delq_sts'].fillna(0)
    df['repch_flag']=df['repch_flag'].fillna('X')
    df['flag_mod']=df['flag_mod'].fillna('N')
    df['cd_zero_bal']=df['cd_zero_bal'].fillna(00)
    df['dt_zero_bal']=df['dt_zero_bal'].fillna('189901')
    df['non_int_brng_upb']=df['non_int_brng_upb'].fillna(0)
    df['dt_lst_pi']=df['dt_lst_pi'].fillna('189901')
    df['mi_recoveries']=df['mi_recoveries'].fillna(0)
    df['net_sale_proceeds']=df['net_sale_proceeds'].fillna('U')
    df['non_mi_recoveries']=df['non_mi_recoveries'].fillna(0)
    df['expenses']=df['expenses'].fillna(0)
    df['legal_costs']=df['legal_costs'].fillna(0)
    df['maint_pres_costs']=df['maint_pres_costs'].fillna(0)
    df['taxes_ins_costs']=df['taxes_ins_costs'].fillna(0)
    df['misc_costs']=df['misc_costs'].fillna(0)
    df['actual_loss']=df['actual_loss'].fillna(0)
    df['modcost']=df['modcost'].fillna(0)
    return df

def changedtype(df):
    #Change the data types for all column
    df[['loan_age','mths_remng','cd_zero_bal','non_int_brng_upb','delq_sts','actual_loss']] = df[['loan_age','mths_remng
    df[['svcg_cycle','dt_zero_bal','dt_lst_pi']] = df[['svcg_cycle','dt_zero_bal','dt_lst_pi']].astype('str')
    return 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.

```
def get_current_upb(group):
    return {'min_current_upb': group.min(), 'max_current_upb': group.max()}
def get_delq_sts(group):
    return {'min_delq_sts': group.min(), 'max_delq_sts': group.max()}
def get_cd_zero_bal(group):
    return {'min_cd_zero_bal': group.min(), 'max_cd_zero_bal': group.max()}
def get_mi_recoveries(group):
    return {'min_mi_recoveries': group.min(), 'max_mi_recoveries': group.max()}
def get_net_sale_proceeds(group):
    return { 'min_net_sale_proceeds': group.min(), 'max_net_sale_proceeds': group.max()}
def get_non_mi_recoveries(group):
    return {'min_non_mi_recoveries': group.min(), 'max_non_mi_recoveries': group.max()}
def get_expenses(group):
    return {'min_expenses': group.min(), 'max_expenses': group.max()}
def get_legal_costs(group):
    return {'min_legal_costs': group.min(), 'max_legal_costs': group.max()}
def get_maint_pres_costs(group):
    return {'min_maint_pres_costs': group.min(), 'max_maint_pres_costs': group.max()}
def get_taxes_ins_costs(group):
    return {'min_taxes_ins_costs': group.min(), 'max_taxes_ins_costs': group.max()}
def get_misc_costs(group):
    return {'min_misc_costs': group.min(), 'max_misc_costs': group.max()}
def get_actual_loss(group):
    return {'min_actual_loss': group.min(), 'max_actual_loss': group.max()}
def get_modcost(group):
    return {'min_modcost': group.min(), 'max_modcost': group.max()}
```

1.3 Creating Summarized CSV file (Output)

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

We have also created some derived column like 'Year' and 'Quarter' which will help to create the summary metrics.

```
#Create a data frame for all 18 Origination files
#code originated file
writeHeader1 = True
filename= "OriginationCombinedCode.csv"
with open(filename, 'w',encoding='utf-8',newline="") as file:
    for f in glob.glob(Samplepath +'\\sample_orig_*.txt'):
        sample_df = pd.read_csv(f ,sep="|", names=['fico','dt_first_pi','flag_fthb','dt_matr','cd_msa',"mi_pct",'cnt_uni
        sample_df = fillNAN(sample_df)
        sample_df = changedatatype(sample_df)
        sample_df['Year'] = ['19'+x if x=='99' else '20'+x for x in (sample_df['id_loan'].apply(lambda x: x[2:4]))]
        if writeHeader1 is True:
            sample_df.to_csv(file, mode='a', header=True,index=False)
            writeHeader1 = False
        else:
            sample_df.to_csv(file, mode='a', header=False,index=False)
```

```
writeHeader2 = True
filename= "PerformanceCombinedSummary.csv"
with open(filename, 'w',encoding='utf-8',newline="") as file:
    for f in glob.glob(Samplepath +'\\sample_svcg_*.txt'):
        perf_df = pd.read_csv(f ,sep="|", names=['id_loan','svcg_cycle','current_upb','delq_sts','loan_age','mths_remng'
        perf_df['delq_sts'] = [ 999 if x=='R' else x for x in (perf_df['delq_sts'].apply(lambda x: x))]
        perf_df['delq_sts'] = [ 0 if x=='XX' else x for x in (perf_df['delq_sts'].apply(lambda x: x))]
        perf_df = fillNA(perf_df)
        perf_df = changedtype(perf_df)
        summ_df = pd.DataFrame()
        summ_df['id_loan']= perf_df['id_loan'].drop_duplicates()
        summ_df=summ_df.join((perf_df['current_upb'].groupby(perf_df['id_loan']).apply(get_current_upb).unstack()),on='i
        summ_df=summ_df.join((perf_df['delq_sts'].groupby(perf_df['id_loan']).apply(get_delq_sts).unstack()),on='id_loan
        summ_df=summ_df.join((perf_df['cd_zero_bal'].groupby(perf_df['id_loan']).apply(get_cd_zero_bal).unstack()),on='i
        summ_df=summ_df.join((perf_df['non_mi_recoveries'].groupby(perf_df['id_loan']).apply(get_non_mi_recoveries).unst
        summ_df=summ_df.join((perf_df['expenses'].groupby(perf_df['id_loan']).apply(get_expenses).unstack()),on='id_loan
        summ_df=summ_df.join((perf_df['legal_costs'].groupby(perf_df['id_loan']).apply(get_legal_costs).unstack()),on='i
        summ_df=summ_df.join((perf_df['maint_pres_costs'].groupby(perf_df['id_loan']).apply(get_maint_pres_costs).unstac
        summ_df=summ_df.join((perf_df['taxes_ins_costs'].groupby(perf_df['id_loan']).apply(get_taxes_ins_costs).unstack(
        summ_df=summ_df.join((perf_df['misc_costs'].groupby(perf_df['id_loan']).apply(get_misc_costs).unstack()),on='id_
        summ_df=summ_df.join((perf_df['actual_loss'].groupby(perf_df['id_loan']).apply(get_actual_loss).unstack()),on='i
        summ_df=summ_df.join((perf_df['modcost'].groupby(perf_df['id_loan']).apply(get_modcost).unstack()),on='id_loan')
        if writeHeader2 is True:
            summ_df.to_csv(file, mode='a', header=True,index=False)
            writeHeader2 = False
        else:
            summ_df.to_csv(file, mode='a', header=False,index=False)
```

NOTE: As we are working on two different file, we need to know about the data and the relationship between the two files created. In origination file, we have idloan 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.

```
In [5]: summ_df = pd.DataFrame()
        grouped = df.groupby('Year')
        summ_df = summ_df.append(grouped.aggregate(np.mean))
        summ_df['loancount']=df['fico'].groupby(df['Year']).count()
        summ_df['year'] = summ_df.index
        del summ_df['dt_first_pi']
        del summ_df['zipcode']
        del summ_df['cnt_borr']
        del summ_df['orig_loan_term']
        del summ_df['cnt_units']
        del summ_df['cd_msa']
        del summ_df['dt_matr']
        summ_df
```

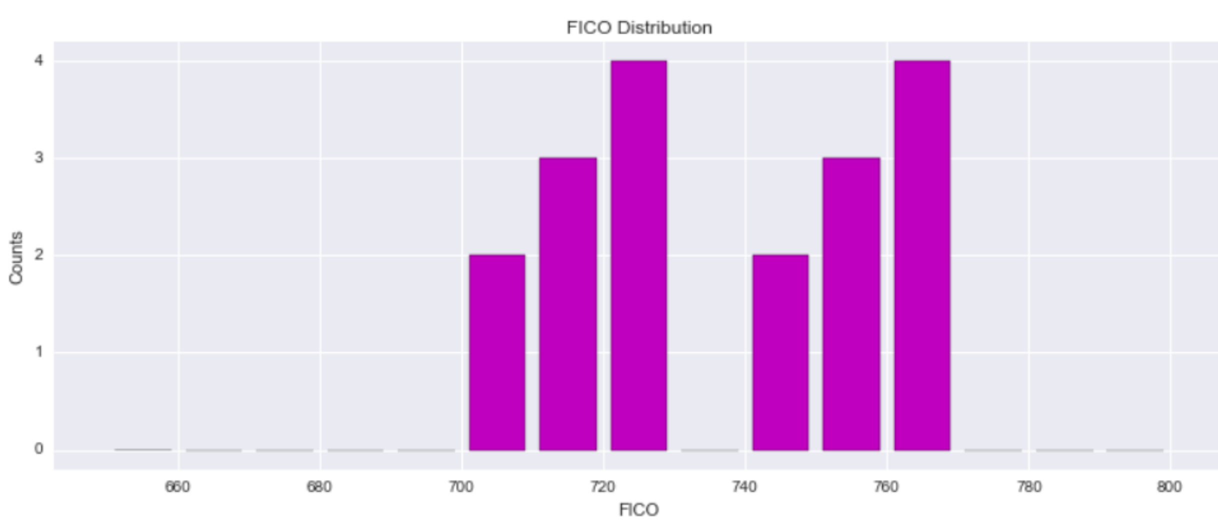
	fico	mi_pct	cltv	dti	orig_upb	ltv	int_rt	loancount	year
Year									
1999	705.985140	9.330280	77.169180	32.025540	125688.940000	77.072040	7.449041	50000	1999
2000	703.075200	8.671820	78.164660	33.993620	130918.920000	77.694460	8.188788	50000	2000
2001	710.203800	6.452120	76.322680	32.580800	148132.340000	75.681380	7.029627	50000	2001
2002	713.010720	5.960520	74.945920	32.924020	154680.200000	74.044140	6.628768	50000	2002
2003	723.291340	4.882500	73.574340	31.936100	160415.000000	72.377340	5.820849	50000	2003
2004	717.246320	4.776880	75.348980	34.395080	166583.300000	73.781940	5.868861	50000	2004
2005	723.672140	3.286360	71.094340	34.214460	170651.580000	69.446860	5.806122	50000	2005
2006	722.155780	3.476260	73.111040	35.806980	179592.580000	70.693960	6.406876	50000	2006
2007	722.662120	5.190540	74.498940	35.891380	183764.160000	72.063640	6.376952	50000	2007
2008	740.569511	4.268185	71.504450	35.466129	203978.499570	70.281966	6.057034	49999	2008
2009	762.136337	1.552389	66.853343	31.759225	213722.905542	65.449951	4.958592	50001	2009
2010	763.085820	1.776420	67.510120	31.639460	208388.820000	66.357120	4.637233	50000	2010
2011	763.800380	2.442760	68.477940	31.753800	217079.460000	67.420520	4.347664	50000	2011
2012	766.538660	3.066260	68.972940	30.830700	222671.620000	67.940540	3.609081	50000	2012
2013	757.935920	4.847660	71.996780	32.309660	217599.680000	71.201500	3.848064	50000	2013
2014	751.329760	7.074460	75.563380	33.728000	219865.720000	75.059080	4.287927	50000	2014
2015	751.733800	6.453200	74.268300	33.827620	229363.160000	73.737140	3.956787	50000	2015
2016	748.459600	6.066400	73.487440	34.333200	228855.920000	73.065600	3.969661	12500	2016

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

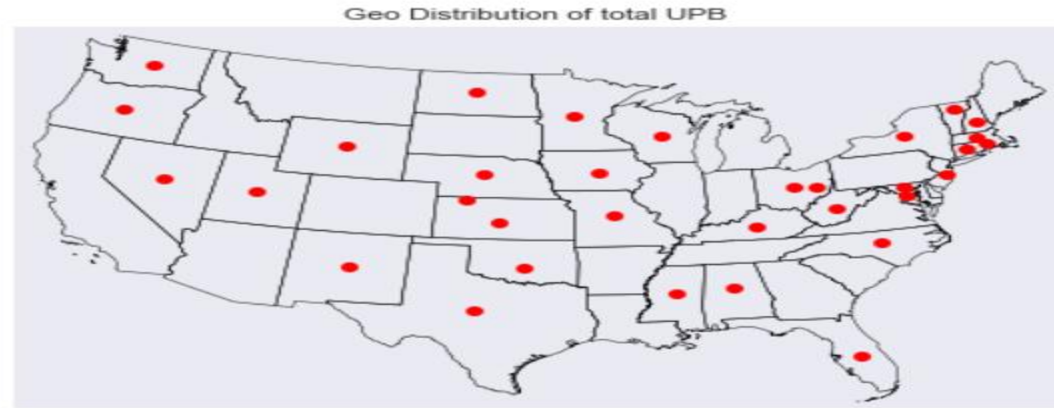
CLTV – LTV & DTI Comparison based on Year

```
def plot_time_trends_1():
    cltv=summ_df['cltv']
    ltv=summ_df['ltv']
    year=df['Year'].drop_duplicates()
    fico=summ_df['fico']
    dti=summ_df['dti']

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

    ax1=plt.subplot(211)
    plt.plot(year,ltv,'r--',year,cltv,'b--',year,dti,'g--')
    plt.xlabel('YEAR')
    plt.ylabel('Combined LTV and Loan-to-value %')
    plt.legend(['LTV','CLTV','DTI'])
    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(fico, bins=bins, histtype='bar', rwidth=0.8 ,color='m')
    plt.xlabel('FICO')
    plt.ylabel('Counts')
    plt.title('FICO 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()
```

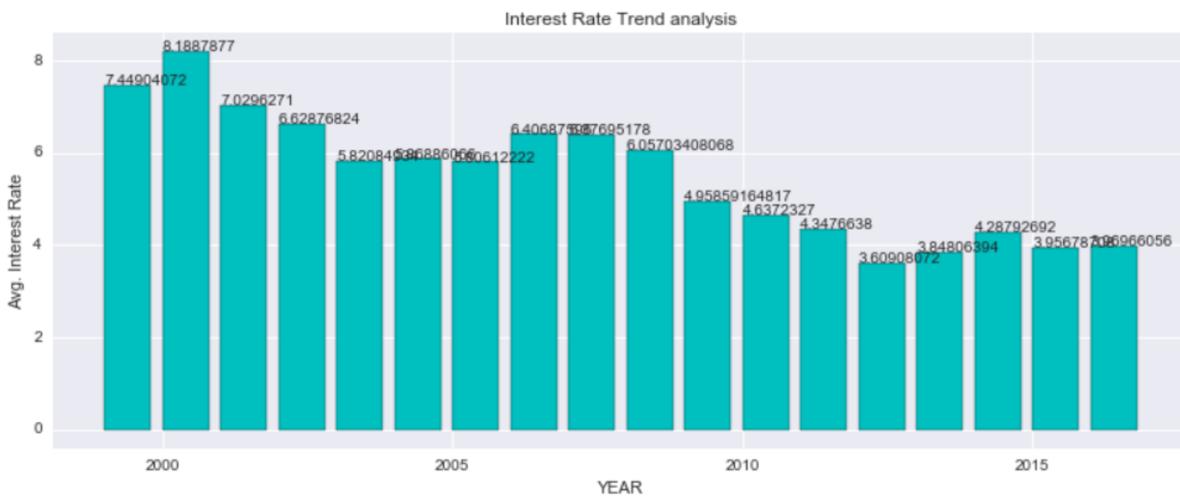
FICO Based on Count



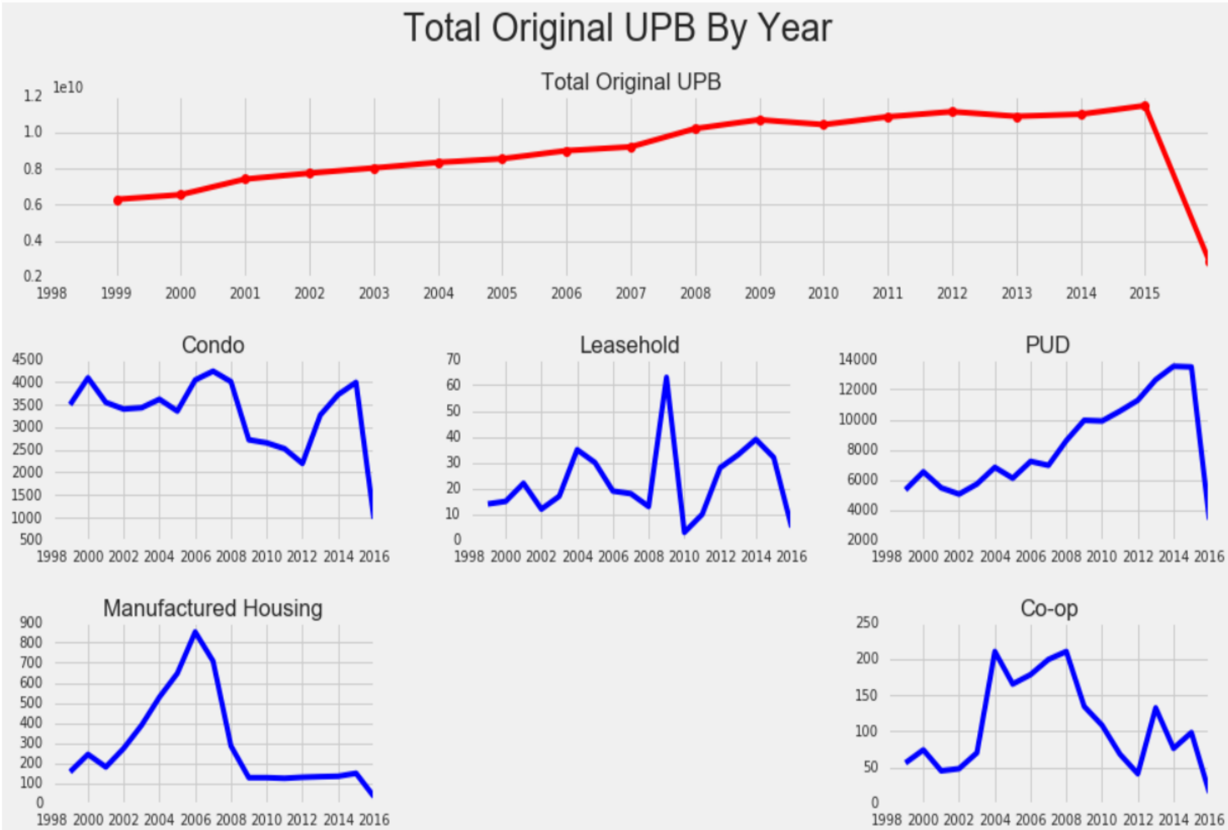
Count of Loan based on Geographical Presence



Average Fico Score based on Year

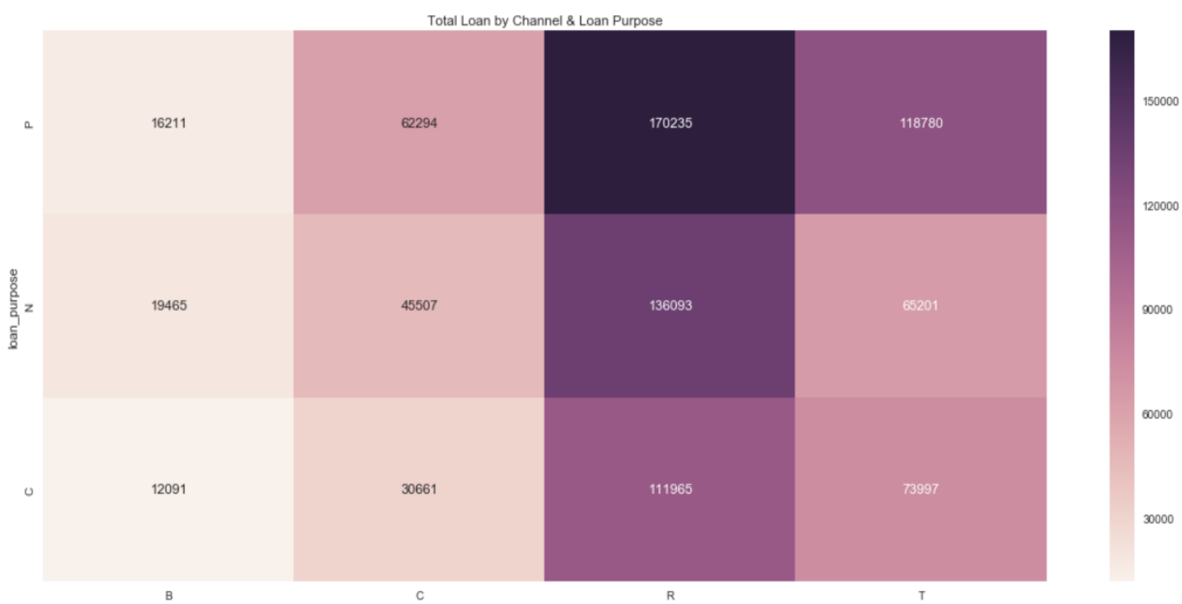


Average UPB based on Year



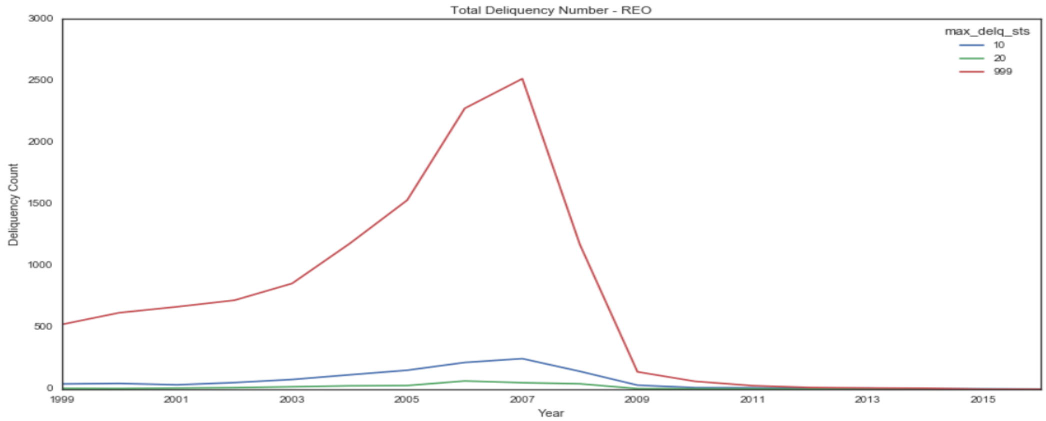
Total Loan Count by Purpose

```
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()
```

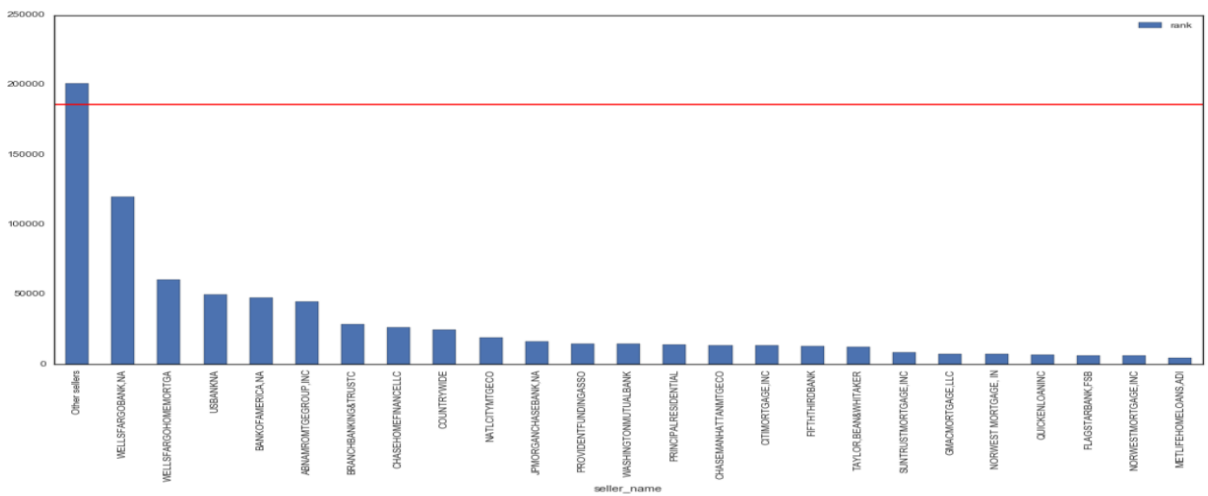


Total Deposition by Year

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



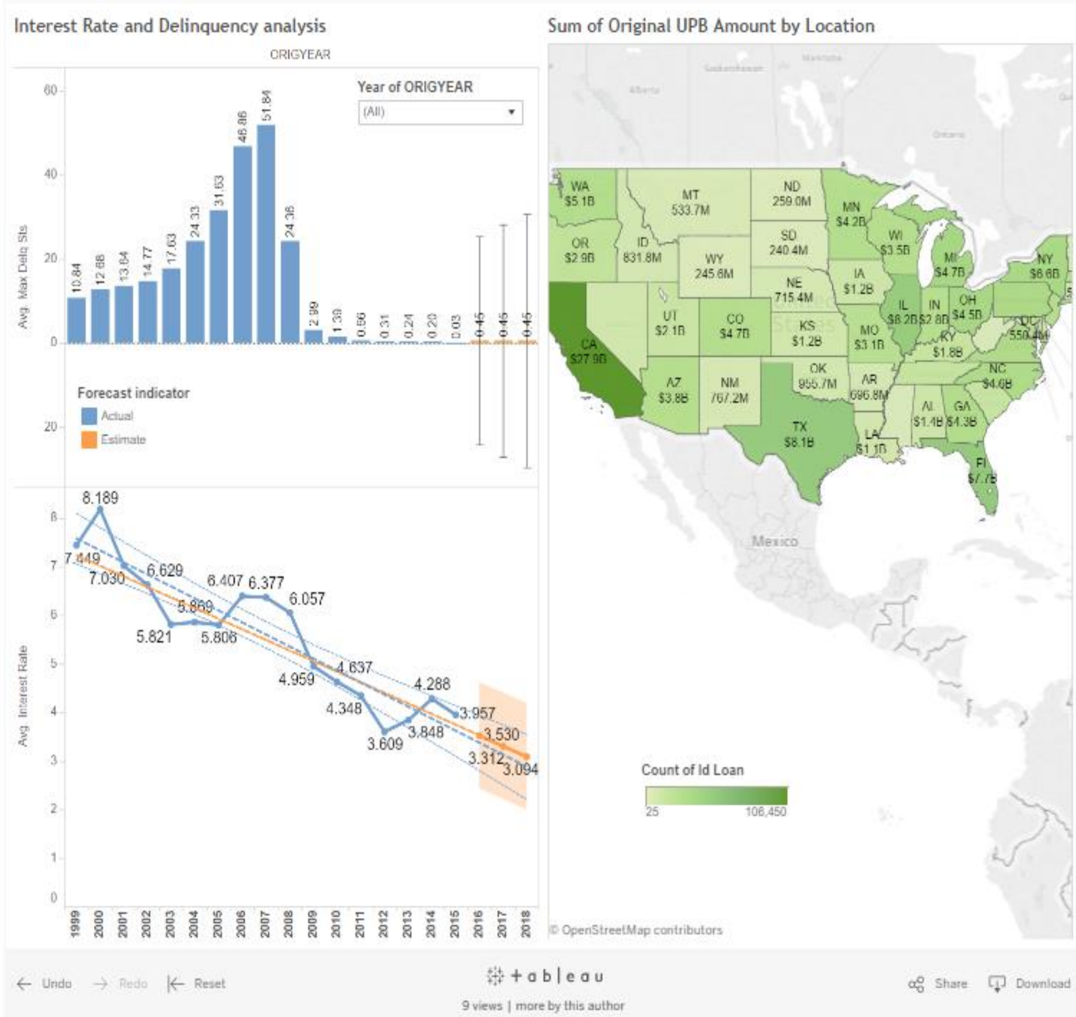
Top 25 Seller



2.1 Analysis - Tableau

Interest Rate and Delinquency analysis

The trends of average of Max Delq Sts (actual & forecast) and average of Int Rt (actual & forecast) for ORIGYEAR Year. Color shows details about Forecast indicator.For pane Average of Max Delq Sts (actual & forecast) : The marks are labeled by average of Max Delq Sts (actual & forecast) . For pane Average of IntRt (actual & forecast) : The marks are labeled by average of Int Rt (actual & forecast). The view is filtered on ORIGYEAR Year, which keeps 18 of 18 members.



FreddieMac-Geographical Analysis of Interest Rate and Delinquency Stats

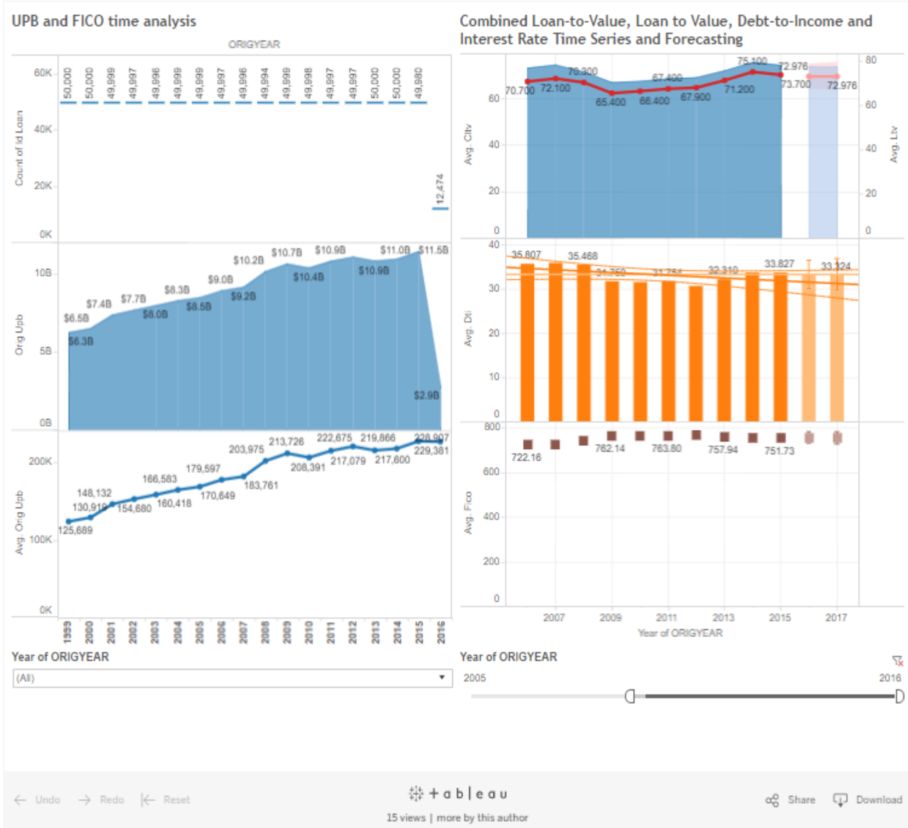
FreddieMac-SingleFamilyLoan-DataAnalysis for Geographical Analysis of Interest Rate and Delinquency

Link:

https://public.tableau.com/profile/ankit.bhayani - !/vizhome/FreddieMac-SingleFamilyLoan-DataAnalysis_1/GeographicalAnalysisofInterestRateandDelinquency

Numerical Measures with Time

The trends of count of Id Loan, sum of Orig Upb and average of Orig Upb for ORIGYEAR Year. For pane Count of Id Loan: The marks are labeled by count of Id Loan. For pane Sum of Orig Upb: The marks are labeled by IF SUM([Orig Upb])>=1000000000 THEN "$"+STR(ROUND((SUM([Orig For pane Average of Orig Upb: The marks are labeled by average of Orig Upb. The view is filtered on ORIGYEAR Year, which keeps 18 of 18 members.



FreddieMac-Numerical Mesaures over time

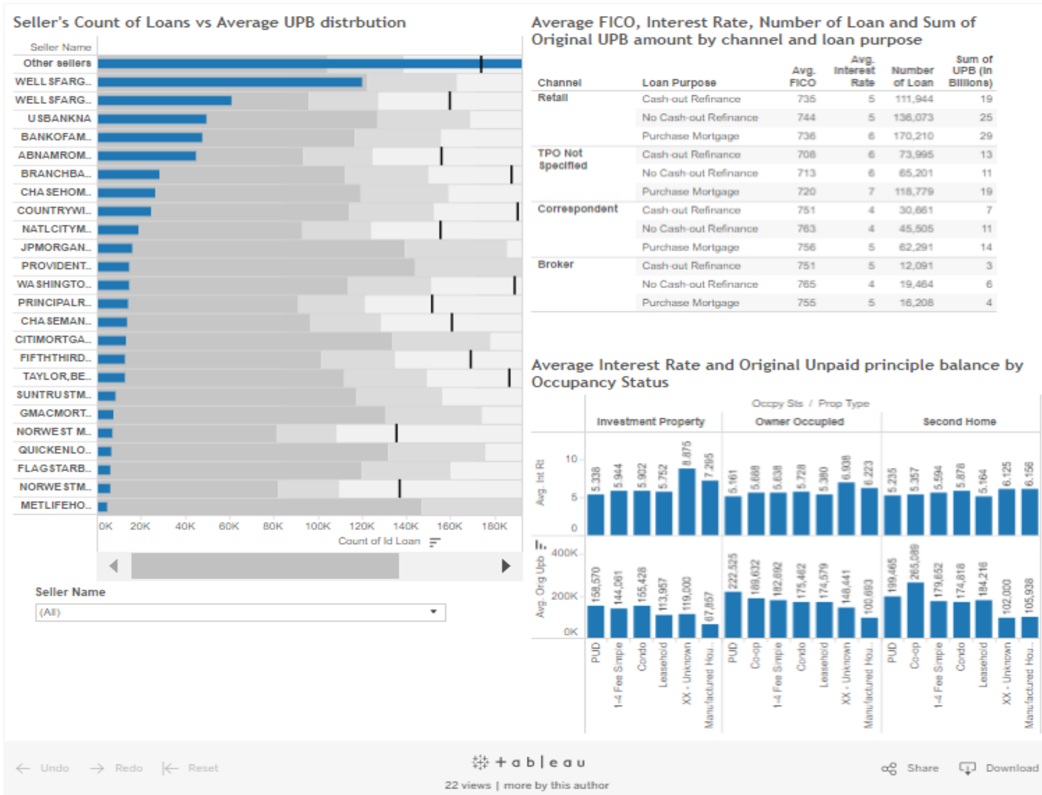
FreddieMac-SingleFamilyLoan-DataAnalysis contains: 1. UPB and FICO time analysis 2. CLTV, LTV, DTI and FICO

Link:

https://public.tableau.com/profile/ankit.bhayani#!/vizhome/FreddieMac-SingleFamilyLoan-DataAnalysis_0/NumericalMesaresovertime

Channel/Seller/Occupancy Insights

Average of Int Rt and average of Orig Upb for each Prop Type broken down by Occpy Sts. For pane Average of Int Rt: The marks are labeled by average of Int Rt. For pane Average of Orig Upb: The marks are labeled by average of Orig Upb. The data is filtered on Action (Seller Name), which keeps 103 members



Channel	Loan Purpose	Avg. FICO	Avg. Interest Rate	Number of Loan	Sum of UPB (In Billions)
Retail	Cash-out Refinance	735	5	111,944	19
	No Cash-out Refinance	744	5	136,073	25
	Purchase Mortgage	736	6	170,210	29
TPO Not Specified	Cash-out Refinance	708	6	73,995	13
	No Cash-out Refinance	713	6	65,201	11
	Purchase Mortgage	720	7	118,779	19
Correspondent	Cash-out Refinance	751	4	30,661	7
	No Cash-out Refinance	763	4	45,505	11
	Purchase Mortgage	756	5	62,291	14
Broker	Cash-out Refinance	751	5	12,091	3
	No Cash-out Refinance	765	4	19,464	6
	Purchase Mortgage	755	5	16,208	4

FreddieMac-Seller, Channel and Occupancy Insights

Link:

https://public.tableau.com/profile/ankit.bhayani#!/vizhome/FreddieMac-SingleFamilyLoan-DataAnalysis/SellerChannelandOccupancyInsights

Summary:

- FICO declined in 2013 and 2014, but remained higher than pre-recession levels.
- LTV was 76.6% in 2014 - a new high since 2000, largely due to the increase of purchase volume over refinances.
- DTI went up in 2013 and 2014, however the concentration in the highest DTI bucket (45-65) remained below the pre-recession average.

As the economy continued to recover, Freddie Mac's loan-level origination data shows a marked increase in the volume of purchase loans, and to a lesser extent investment properties.

2 PART II: Building & Evaluating Models

PREDICTION (Predicting Interest Rates)

Here we are going to create a predictive model based on information from the origination data from the prior quarter using the various regression technique to calculate the following metrics:

MAE (Mean Absolute Error) - In statistics, the mean absolute error (MAE) is a quantity used to measure how close forecasts or predictions are to the eventual outcomes.

RMSE (Root Mean Square Error) - The root-mean-square deviation (RMSD) or root-mean-square error (RMSE) is a frequently used measure of the differences between values (sample and population values) predicted by a model or an estimator and the values observed. The RMSD represents the sample standard deviation of the differences between predicted values and observed values. These individual differences are called residuals when the calculations are performed over the data sample that was used for estimation, and are called prediction errors when computed out-of-sample. The RMSD serves to aggregate the magnitudes of the errors in predictions for various times into a single measure of predictive power. RMSD is a good measure of accuracy, but only to compare forecasting errors of different models for a particular variable and not between variables, as it is scale-dependent

Median Absolute Error- The median_absolute_error is particularly interesting because it is robust to outliers. The loss is calculated by taking the median of all absolute differences between the target and the prediction.

```
def computations(org,x,y):
    testlr=org.predict(x)
    #Mean Absolute Error
    mae=mean_absolute_error(y,testlr);
    print("MAE:"+str(mae))
    #RMSE
    rmse=math.sqrt(mean_squared_error(y,testlr))
    print("RMSE:"+str(rmse))
    #Median Absolute error
    Medae=median_absolute_error(y,testlr)
    print("Median Absolute Error:"+str(Medae))
```

2.1 APPROCH - PREDICTION

2.1.1 CREATING THE FUNCTION AND DOWNLOAD THE INPUT

We have created functions that would take hit the Freddie Mac website and download all the Historical file for our purpose. Based on the input Quarter, the file will be imported into python Data Frame. It went through following preprocessing before building model.

```
#Setting Data for model
def trainModel(year):
    foldername= 'historical_data1_'+str(year)
    Historicalpath=str(os.getcwd())+"\\"+foldername
    filename=Historicalpath+"\historical_data1_"+str(year)+".txt"
    Hist_df = pd.read_csv(filename ,sep="|", names=['fico','dt_first_pi','flag_fthb','dt_matr','cd_msa',"mi_pct",'cnt_un
    Hist_df = fillNAN(Hist_df)
    Hist_df = changedatatype(Hist_df)
    Hist_df=createDummies(Hist_df)
    Hist_numaric=Hist_df._get_numeric_data()
    Hist_numaric.drop('cd_msa',axis=1,inplace=True)
    Hist_numaric.drop('dt_first_pi',axis=1,inplace=True)
    Hist_numaric.drop('dt_matr',axis=1,inplace=True)
    Hist_numaric.drop('flag_sc',axis=1,inplace=True)
    Hist_numaric.drop('zipcode',axis=1,inplace=True)
    return Hist_numaric
```

Handling the missing values:

```
def fillNAN(df):
    df['fico'] = df['fico'].fillna(0)
    df['flag_fthb']=df['flag_fthb'].fillna('X')
    df['cd_msa']=df['cd_msa'].fillna(0)
    df['mi_pct']=df['mi_pct'].fillna(0)
    df['cnt_units']=df['cnt_units'].fillna(0)
    df['occpy_sts']=df['occpy_sts'].fillna('X')
    df['cltv']=df['cltv'].fillna(0)
    df['dti']=df['dti'].fillna(0)
    df['ltv']=df['ltv'].fillna(0)
    df['channel']=df['channel'].fillna('X')
    df['ppmt_pnlty']=df['ppmt_pnlty'].fillna('X')
    df['prop_type']=df['prop_type'].fillna('XX')
    df['zipcode']=df['zipcode'].fillna(0)
    df['loan_purpose']=df['loan_purpose'].fillna('X')
    df['cnt_borr']=df['cnt_borr'].fillna(0)
    df['flag_sc']=df['flag_sc'].fillna('N')
    return df
```

Created dummy columns (1…C):

```
def createDummies(df):
    dummies = pd.get_dummies(df['flag_fthb']).rename(columns=lambda x: 'flag_fthb' + str(x))
    train_df=pd.concat([df, dummies], axis=1)
    dummies1 = pd.get_dummies(df['occpy_sts']).rename(columns=lambda x: 'occpy_sts' + str(x))
    train_df=pd.concat([train_df, dummies1], axis=1)
    dummies2 = pd.get_dummies(df['channel']).rename(columns=lambda x: 'channel' + str(x))
    train_df=pd.concat([train_df, dummies2], axis=1)
    dummies3 = pd.get_dummies(df['ppmt_pnlty']).rename(columns=lambda x: 'ppmt_pnlty' + str(x))
    train_df=pd.concat([train_df, dummies3], axis=1)
    dummies4 = pd.get_dummies(df['prop_type']).rename(columns=lambda x: 'prop_type' + str(x))
    train_df=pd.concat([train_df, dummies4], axis=1)
    dummies5 = pd.get_dummies(df['loan_purpose']).rename(columns=lambda x: 'loan_purpose' + str(x))
    train_df=pd.concat([train_df, dummies5], axis=1)
    train_df['flag_sc']=train_df['flag_sc'].map({'Y':1,'N':0})
    return train_df
```

2.1.2 CONVERSION OF DATA TYPE

Data types of the History Origination file is converted as below.

```
def changedatatype(df):
#Change the data types for all column
    df[['fico','cd_msa','mi_pct','cnt_borr','cnt_units','cltv','dti','orig_upb','ltv','zipcode','orig_loan_term']] = df[
    df[['flag_sc','servicer_name']] = df[['flag_sc','servicer_name']].astype('str')
    return df
```

2.1.3 FEATURE SELECTION

Before proceeding with our models, we have done best feature selection using three algorithms. The best features that add to the predictive power of the model and irrelevant features removed from the model. We implemented following feature selection techniques in Python:

**sklearn.feature_selection.SelectKBest : **Select features according to the k highest scores.

```
#SelectKBest
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)
    result = sm.OLS(y,x).fit()
    print(result.summary())
    pred=kb.predict(X_train)
    sc=r2_score(y,pred)
    print("select k best:")
    print(sc)
    return kb
```

sklearn.linear_model.RandomizedLasso: Randomized Lasso works by subsampling the training data and computing a Lasso estimate where the penalty of a random subset of coefficients has been scaled. By performing this double randomization several times, the method assigns high scores to features that are repeatedly selected across randomizations. This is known as stability selection. In short, features selected more often are considered good features.

```
#RandomizedLasso
def randomizedLasso(org,x,y):
    with warnings.catch_warnings():
        warnings.simplefilter('ignore', UserWarning)
        warnings.simplefilter('ignore', ConvergenceWarning)
        lars_cv = LassoLarsCV(cv=6).fit(x, y)
    alphas = np.linspace(lars_cv.alphas_[0], .1 * lars_cv.alphas_[0], 6)
    with warnings.catch_warnings():
        warnings.simplefilter('ignore', DeprecationWarning)
        reg = linear_model.RandomizedLasso(alpha=alphas).fit(x, y)
    with warnings.catch_warnings():
        warnings.simplefilter('ignore', DeprecationWarning)
        X_train = reg.fit_transform(x,y)
    org.fit(X_train,y)
    sc = org.score(X_train, y)
    result = sm.OLS(y,X_train).fit()
    print(result.summary())
    print("Randomized Lasso:")
    print(sc)
    return org
```

sklearn.feature_selection.RFE : Feature ranking with recursive feature elimination.

Given an external estimator that assigns weights to features (e.g., the coefficients of a linear model), the goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and weights are assigned to each one of them. Then, features whose absolute weights are the smallest are pruned from the current set features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.

```
#Recursive feature elimination
def RFElimination(org,x,y):
    selector = RFE(org,28,step=1)
    selector = selector.fit(x, y)
    print(selector.ranking_)
    rankingdf=pd.DataFrame(list(zip(x.columns,selector.ranking_)),columns=["features","ranking"])
    print(rankingdf)
    result = sm.OLS(y,x).fit()
    print(result.summary())
    pred=selector.predict(x)
    sc=r2_score(y,pred)
    print("RFElimination:")
    print(sc)
    return selector
```

We selected variables as per the **RFE ranking** and used those for further analysis while making sure that all the datasets contain same number of columns. Performing all the feature selection methods we shortlisted below features to best predict our model.

```
#Ensures all required features
def checkAllReqColumns(df):
    cols_to_keep=['fico','flag_fthbN','flag_fthbX','flag_fthbY','mi_pct','cnt_units','occpy_stsI','occpy_stsO','occpy_st
    for x in cols_to_keep:
        if not x in df.columns:
            df[x]=0.0
    return df
```

2.1.4 DIFFERENT MACHINE LEARNING ALGORITHMS AND OUTPUT

2.1.4.1 REGRESSION

We have used the following regression techniques and compared the R2 and selected Liner Regression.

Regression using Linear Regression:

sklearn.linear_model.LinearRegression : Ordinary least squares Linear Regression.

```
#Linear regression
def linearRegression(org,x,y):
    org.fit(x,y)
    score=org.score(x,y)
    pd.DataFrame(list(zip(x.columns,org.coef_)),columns=["features","estimatedCoefficients"])
    result = sm.OLS(y,x).fit()
    print(result.summary())
    pred=org.predict(x)
    sc=r2_score(y,pred)
    print("linear regression:")
    print(sc)
    return org
```

Regression using LassoLars

sklearn.linear_model.LassoLars : Lasso model fit with Least Angle Regression a.k.a. Lars
It is a Linear Model trained with an L1 prior as regularize.

```
#LassoLars regression
def lassoLarsRegression(org,x,y):
    reg = linear_model.LassoLars(alpha=0.000)
    reg.fit(x,y)
    result = sm.OLS(y,x).fit()
    print(result.summary())
    pred=reg.predict(x)
    sc=r2_score(y,pred)
    print("ridge regression:")
    print(sc)
    return reg
```

Implementing Liner Regression

Linear regression is an approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables.

```
#Linear Regression
def LinearRegressionAnalysis(q1,q2):
    org=lm
    Data1=trainModel("Q12005")
    TestData1=trainModel("Q22005")
    Data1=checkAllReqColumns(Data1)
    TestData1=checkAllReqColumns(TestData1)

    y_train1=Data1.int_rt
    Data1.drop('int_rt',axis=1,inplace=True)
    x_train1=Data1
    orglr1=linearRegression(org,x_train1,y_train1)
    print("Linear Regression---------")
    print("Training Data")
    computations(orglr1,x_train1,y_train1)
    print("Testing Data:")
    y_test1=TestData1.int_rt
    TestData1.drop('int_rt',axis=1,inplace=True)
    x_test1=TestData1
    computations(orglr1,x_test1,y_test1)

    plt.scatter(orglr1.predict(x_train1),orglr1.predict(x_train1)-y_train1,c='b',s=40,alpha=0.5)
    plt.scatter(orglr1.predict(x_test1),orglr1.predict(x_test1)-y_test1,c="g",s=40)
    plt.hlines(y=0,xmin=2,xmax=10)
    plt.title('Residual plot using training(blue) and test(green) data')
    plt.ylabel('Residuals')
```

2.1.4.2 KNN

KNN or k-nearest neighbors' algorithm is one of the simplest machine learning algorithms and is an example of instance-based learning, where new data are classified based on stored, labeled instances. More specifically, the distance between the stored data and the new instance is calculated by means of some kind of a similarity measure.

This similarity measure is typically expressed by a distance measure such as the Euclidean distance, cosine similarity or the Manhattan distance.

sklearn.neighbors.**KNeighborsRegressor**

```
#KNN
def KNNAnalysis(q1,q2):
    org=lm
    Data1=trainModel("Q12007")
    TestData1=trainModel("Q22007")
    Data1=checkAllReqColumns(Data1)
    TestData1=checkAllReqColumns(TestData1)
    print("Training Data")
    y_train1=Data1.int_rt
    Data1.drop('int_rt',axis=1,inplace=True)
    x_train1=Data1

    neigh = KNeighborsRegressor(n_neighbors=6)
    neigh.fit(x_train1,y_train1)
    print("KNN---------------")
    print("Training Data:")
    computations(neigh,x_train1,y_train1)
    print("Testing Data:")
    y_test1=TestData1.int_rt
    TestData1.drop('int_rt',axis=1,inplace=True)
    x_test1=TestData1
    computations(neigh,x_test1,y_test1)

    plt.scatter(neigh.predict(x_train1),neigh.predict(x_train1)-y_train1,c='b',s=40,alpha=0.5)
    plt.scatter(neigh.predict(x_test1),neigh.predict(x_test1)-y_test1,c="g",s=40)
    plt.hlines(y=0,xmin=2,xmax=10)
    plt.title('Residual plot using training(blue) and test(green) data')
    plt.ylabel('Residuals')
```

2.1.4.3 RANDOM FOREST

The random forest starts with a standard machine learning technique called a “decision tree”. This is a type of additive model that makes predictions by combining decisions from a sequence of base models.

sklearn.ensemble.RandomForestRegressor

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default).

```
#Random Forest
def RandomForestAnalysis(q1,q2):
    org=lm
    Data1=trainModel("Q12007")
    TestData1=trainModel("Q22007")
    Data1=checkAllReqColumns(Data1)
    TestData1=checkAllReqColumns(TestData1)
    print("Training Data")
    y_train1=Data1.int_rt
    Data1.drop('int_rt',axis=1,inplace=True)
    x_train1=Data1

    regr_rf=RandomForestRegressor(max_depth=8)
    regr_rf.fit(x_train1,y_train1)
    print("Random Forest-------------")
    print("Training Data:")
    computations(regr_rf,x_train1,y_train1)
    print("Testing Data:")
    y_test1=TestData1.int_rt
    TestData1.drop('int_rt',axis=1,inplace=True)
    x_test1=TestData1
    computations(regr_rf,x_test1,y_test1)

    plt.scatter(regr_rf.predict(x_train1),regr_rf.predict(x_train1)-y_train1,c='b',s=40,alpha=0.5)
    plt.scatter(regr_rf.predict(x_test1),regr_rf.predict(x_test1)-y_test1,c="g",s=40)
    plt.hlines(y=0,xmin=2,xmax=10)
    plt.title('Residual plot using training(blue) and test(green) data')
    plt.ylabel('Residuals')
```

2.1.4.4 NEURAL NETWORK

Neural network terminology is inspired by the biological operations of specialized cells called neurons. A neuron is a cell that has several inputs that can be activated by some outside process.

The artificial equivalent of a neuron is a node (also sometimes called neurons, but I will refer to them as nodes to avoid ambiguity) that receives a set of weighted inputs, processes their sum with its activation function, and passes the result of the activation function to nodes further down the graph.

```
#Neural Network
def NeuralNetworkAnalysis(q1,q2):
    print("Neural Network Analysis-------------")
    Data1=trainModel("Q12007")
    TestData1=trainModel("Q22007")
    Data1=checkAllReqColumns(Data1)
    TestData1=checkAllReqColumns(TestData1)

    y_train1=Data1.int_rt
    y_train1=y_train1.reshape(-1,1)
    Data1.drop('int_rt',axis=1,inplace=True)
    x_train1=Data1

    hidden_size = 3
    epochs = 2
    input_size = x_train1.shape[1]
    target_size = y_train1.shape[1]
    ds = SDS( input_size, target_size )
    ds.setField( 'input', x_train1 )
    ds.setField( 'target', y_train1 )

    net = buildNetwork( input_size, hidden_size, target_size, bias = True )
    trainer = BackpropTrainer( net,ds )

    print("Training for {} epochs...".format( epochs ))

    for i in range( epochs ):
        mse = trainer.train()
        rmse = math.sqrt( mse )
    print("Training RMSE, epoch {}: {}".format( i + 1, rmse ))

    y_test1=TestData1.int_rt
    y_test1=y_test1.reshape(-1,1)
    TestData1.drop('int_rt',axis=1,inplace=True)
    x_test1=TestData1

    input_size = x_test1.shape[1]
    target_size = y_test1.shape[1]

    ds = SDS( input_size, target_size )
    ds.setField( 'input', x_test1)
    ds.setField( 'target', y_test1)

    p = net.activateOnDataset( ds )

    mse = mean_squared_error(y_test1, p )
    rmse =math.sqrt(mse)
    print("Testing rmse:"+str(rmse))
```

2.1.4.5 STORING AND RETURNING THE RESULTS

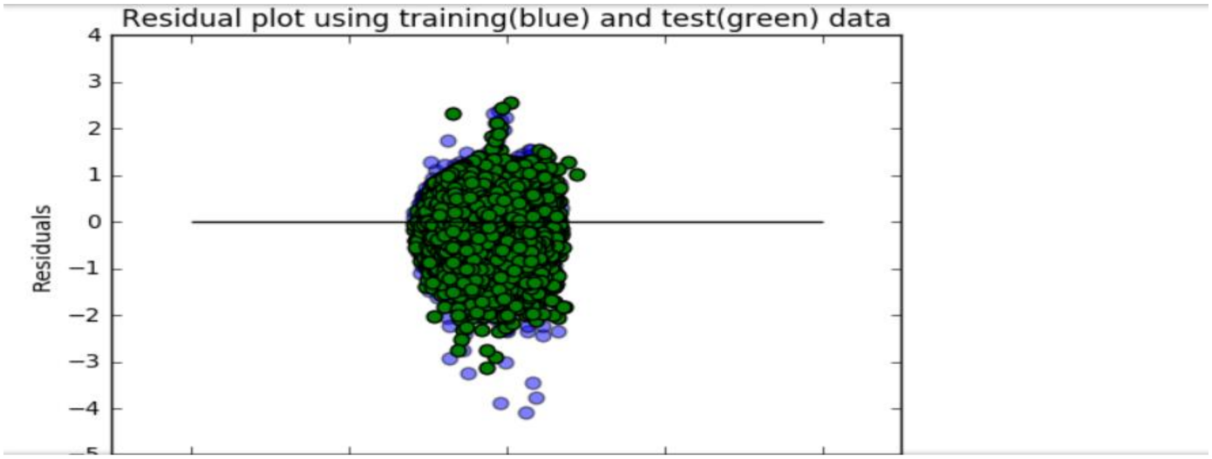
We are calculating the results from the different machine learning algorithms, where we are capturing following details:
ModelName, RMSE.baseline, MAE.baseline and Median Absolute error.

		Training Dataset					Testing Dataset	
Algorithms		MAE	RMSE	Median Absolute Error		MAE	RMSE	Median Absolute Error
Random Forest		0.204556	0.276251	0.156092487	max-depth=8	0.243165509	0.316026152	0.193520286
Neural Network			0.262086				0.360823293	
KNN		0.19929	0.264042	0.166666667	n_neighbours=6	0.267208844	0.346366994	0.208333333
Linear Regression		0.211134	0.284766	0.161359315		0.246939266	0.321061933	0.198301346

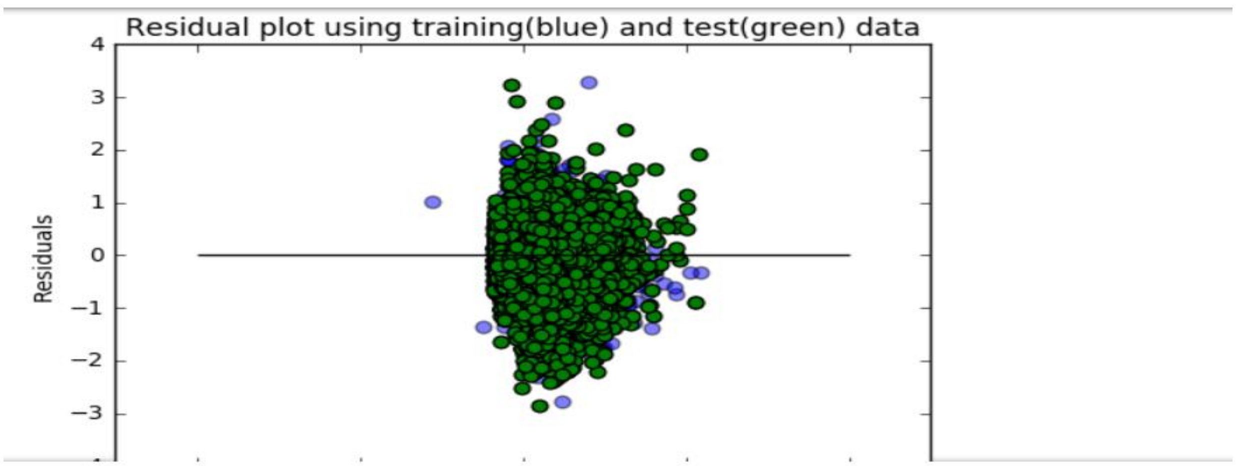
COMPUTE RESIDUALS

We predict the values using our models. We will then calculate the residuals.we can calculate Residuals as shown below, which is nothing but absolute difference of actual verses predicted values.

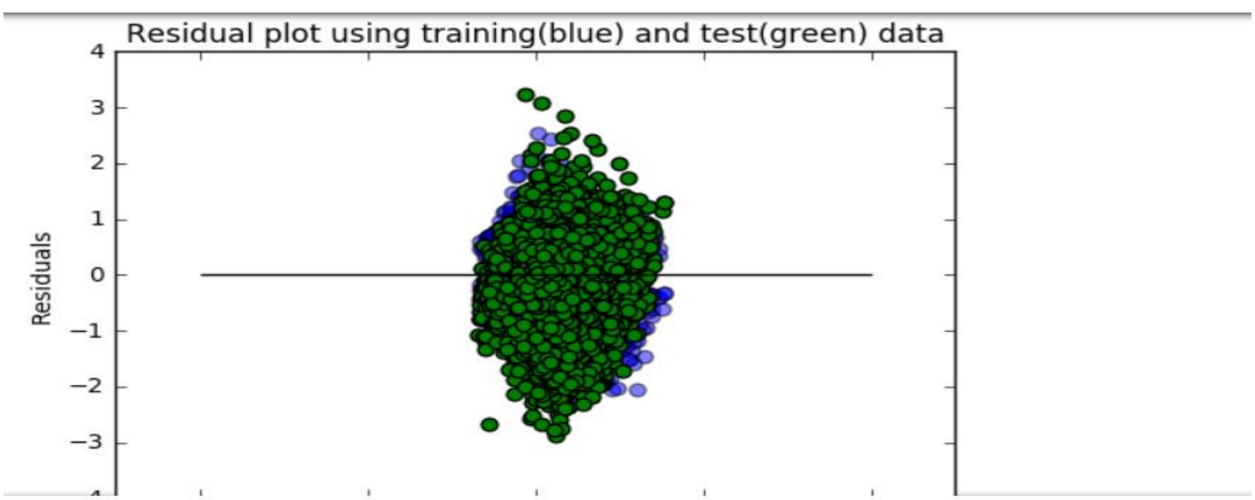
Residual Plot for Linear Regression:



Residual Plot for Random Forest Regression:



Residual Plot for KNN:



2.1.4.6 PREDICTION MODEL EVALUATION: WHICH MODEL TO CHOOSE

We had compared results of all the models decided **Random Forest** gives us much better result.

1. Higher average predictive accuracy
2. Moderate prediction speed
3. Performs well with large number of observations
4. Handles lots of irrelevant features well

WHAT -IF Analysis

Financial Crisis Analysis

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

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

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

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

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

Random Forest	MAE	RMSE	Median Absolute Error		MAE	RMSE	Median Absolute Error
Training				Testing			
Q12007	0.22037	0.29061	0.176166799	Q22007	0.25589	0.33927	0.200377993
Q22007	0.24303	0.31807	0.193599463	Q32007	0.39763	0.47529	0.372476098
Q32007	0.23474	0.30838	0.188043684	Q42007	0.38371	0.46971	0.332195177
Q42007	0.26135	0.34174	0.208010299	Q12008	0.48068	0.5646	0.458545948

Two Years Later (2009):

Financial markets recovered substantially since March 2009 when the financial stress began to ease and market conditions started to improve. (http://www.oecd.org/daf/fin/financial-markets/44563803.pdf)

In 2009, Freddie Mac played a critical role in supporting the nation's housing market by:

- Providing $548.4 billion of liquidity to the mortgage market, helping finance approximately 2.2 million conforming single-family loans and approximately 253,000 units of multifamily rental housing.

- Helping more than 272,000 borrowers stay in their homes or sell their properties through the company's long-standing foreclosure avoidance programs and the Home Affordable Modification program (HAMP), including 129,380 loans that remained in HAMP trial periods as of December 31, 2009 according to information provided by the Making Home Affordable (MHA) program administrator.
- Refinancing approximately $379 billion of single-family loans, creating an estimated $4.5 billion in annual interest savings for borrowers nationwide – this includes approximately 169,000 borrowers whose payments were reduced by an average of $2,000 annually under the Freddie Mac Relief Refinance MortgageSM

(http://www.freddiemac.com/news/archives/investors/2010/2009er-4q09.html)

2009 Data	MAE	RMSE	Median Absolute Error		MAE	RMSE	Median Absolute Error
Training				Testing			
Q12009	0.21955	0.29414	0.162661829	Q22009	0.23189	0.30247	0.179771366
Q22009	0.20055	0.26903	0.150556221	Q32009	0.32368	0.39975	0.291385669
Q32009	0.22342	0.28753	0.180727091	Q42009	0.25334	0.30906	0.233000532
Q42009	0.17149	0.22059	0.142356483	Q12010	0.19357	0.25603	0.153753615

As clearly evident from the analysis, Measures are pretty much stable for 2009.

Economic Boom (1999,2013):

1999:
The easing of credit also coincided with spectacular stock market run-ups from 1999 to 2000 Freddie Mac financed homes for more than 2 million families and achieved record earnings per share of $2.96, an increase of 28 percent over 1998.
(http://www.freddiemac.com/investors/pdffiles/annual99.pdf)
(https://en.wikipedia.org/wiki/1990s_United_States_boom)

1999 Data	MAE	RMSE	Median Absolute Error		MAE	RMSE	Median Absolute Error
Training				Testing			
Q11999	0.2132	0.293303	0.153752	Q21999	0.3092	0.412171	0.239315
Q21999	0.25411	0.333798	0.204376	Q31999	0.63266	0.7228	0.618753
Q31999	0.273483	0.365196	0.210249	Q41999	0.264779	0.365526	0.195572
Q41999	0.231152	0.316921	0.170996	Q12000	0.410728	0.497624	0.385304

2013:
In 2013, Mortgage rates peaked at 4.6% in August and have held steady since September and several accounting events had significant impacts on the Enterprises' reported financial results. Fannie Mae and Freddie Mac reported levels of 2013 net income are greater than at any prior time in their respective histories. Their historically high net income was driven by reversals of previously

accrued losses associated with deferred tax assets (DTA) and their allowance for loan and lease losses (ALLL)—plus revenue from legal settlements of representation and warranties claims and lawsuits regarding private-label securities that the Enterprises purchased as investments. FHFA does not expect benefits of this nature to be repeated in future years and does not expect the 2013 levels of net income to be approached anytime in the foreseeable future.
(https://www.fhfa.gov/AboutUs/Reports/ReportDocuments/FHFA_2013_Report_to_Congress.pdf#page=18)

(http://www.foxbusiness.com/features/2013/12/23/housing-market-in-2013-prices-rise-as-lending-remains-tight.html)

Drastic change in Training and Testing measures for the highlighted rows clearly shows the transition in economic trends during Q2 and Q3 around 48%.

2013 Data	MAE	RMSE	Median Absolute Error		MAE	RMSE	Median Absolute Error
Training				Testing			
Q12013	0.161657	0.2117	0.128463	Q22013	0.186717	0.253939	0.1375
Q22013	0.173338	0.226789	0.13868	Q32013	0.619496	0.70979	0.657801
Q32013	0.284823	0.351997	0.24627	Q42013	0.222674	0.289432	0.176208
Q42013	0.174623	0.224723	0.144539	Q12014	0.16665	0.217698	0.135756

Would you recommend using this model for the next quarter? Justify

The proposed model will perform well for the next quarter with accuracy ranging up to 15% error, if there are not major changes in the data patterns such as financial crisis or economic boom.

CLASSIFICATION (LOAN DELINQUENCY STATUS)

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

3.3 GENERIC APPROACH: CLASSIFCATION

2.1.1 CREATING THE FUNCTION AND DOWNLOAD THE INPUT

We have created functions that would take hit the Freddie Mac website and download all the Historical file for our purpose. Based on the input Quarter, the file will be imported into python Data Frame. It went through following preprocessing before building model.

Handling the missing values:

```
def fillNA(df):
    df['delq_sts'] = df['delq_sts'].fillna(0)
    df['repch_flag']=df['repch_flag'].fillna('X')
    df['flag_mod']=df['flag_mod'].fillna('N')
    df['cd_zero_bal']=df['cd_zero_bal'].fillna(0)
    df['dt_zero_bal']=df['dt_zero_bal'].fillna('189901')
    df['non_int_brng_upb']=df['non_int_brng_upb'].fillna(0)
    df['dt_lst_pi']=df['dt_lst_pi'].fillna('189901')
    df['mi_recoveries']=df['mi_recoveries'].fillna(0)
    df['net_sale_proceeds']=df['net_sale_proceeds'].fillna(0)
    df['non_mi_recoveries']=df['non_mi_recoveries'].fillna(0)
    df['expenses']=df['expenses'].fillna(0)
    df['legal_costs']=df['legal_costs'].fillna(0)
    df['maint_pres_costs']=df['maint_pres_costs'].fillna(0)
    df['taxes_ins_costs']=df['taxes_ins_costs'].fillna(0)
    df['misc_costs']=df['misc_costs'].fillna(0)
    df['actual_loss']=df['actual_loss'].fillna(0)
    df['modcost']=df['modcost'].fillna(0)
    return df
```

Created dummy columns (1…C):

```
def createDummies(df):
    dummies = pd.get_dummies(df['repch_flag']).rename(columns=lambda x: 'repch_flag' + str(x))
    df = pd.concat([df, dummies], axis=1)
    dummies1 = pd.get_dummies(df['cd_zero_bal']).rename(columns=lambda x: 'cd_zero_bal' + str(x))
    df = pd.concat([df, dummies1], axis=1)
    return df
```

2.1.2 CONVERSION OF DATA TYPE

Data types of the History Performance file is converted as below.

```
def changedtype(df):
    #Change the data types for all column
    df[['non_int_brng_upb','actual_loss','modcost','misc_costs','taxes_ins_costs','maint_pres_costs','legal_costs','expe
    df[['loan_age','mths_remng','cd_zero_bal','delq_sts','flag_mod_n']] = df[['loan_age','mths_remng','cd_zero_bal','del
    df[['svcg_cycle','dt_zero_bal','dt_lst_pi']] = df[['svcg_cycle','dt_zero_bal','dt_lst_pi']].astype('str')
    return df


def createDataFrame(str):
    perf_df = pd.read_csv(str ,sep="|", names=['id_loan','svcg_cycle','current_upb','delq_sts','loan_age','mths_remng',
    perf_df['delq_sts'] = [ 999 if x=='R' else x for x in (perf_df['delq_sts'].apply(lambda x: x))]
    perf_df['delq_sts'] = [ 0 if x=='XX' else x for x in (perf_df['delq_sts'].apply(lambda x: x))]
    perf_df['flag_mod_n'] = [ 0 if x=='N' else 1 for x in (perf_df['flag_mod'].apply(lambda x: x))]
    perf_df[['net_sale_proceeds']] = [ 0 if x=='U' else x for x in (perf_df['net_sale_proceeds'].apply(lambda x: x))]
    perf_df[['net_sale_proceeds']] = [ perf_df['current_upb'] if x=='C' else x for x in (perf_df['net_sale_proceeds'].ap
    perf_df['Year'] = ['19'+x if x=='99' else '20'+x for x in (perf_df['id_loan'].apply(lambda x: x[2:4]))]
    perf_df = fillNA(perf_df)
    perf_df = changedtype(perf_df)
    return perf_df

#Ensures all required features
def checkAllReqColumns(df):
    for x in cols_to_keep:
        if not x in df.columns:
            df[x]=0.0
    return df
```

2.1.3 FEATURE SELECTION

Before proceeding with our models, we have done best feature selection using three algorithms. The best features that add to the predictive power of the model and irrelevant features removed from the model. We implemented following feature selection techniques in Python:

```
def featureSelectionRFE():
    from sklearn.feature_selection import RFE
    from sklearn.linear_model import LogisticRegression
    model = LogisticRegression()
    # create the RFE model and select 10 attributes
    rfe = RFE(model, 10)
    rfe = rfe.fit(train_data[0:,1:], train_data[0:,0])
    # summarize the selection of the attributes
    print(rfe.support_)
    print(rfe.ranking_)
    print(rfe.n_features_)
    #Check the accuracy of the model
    rfe.score(train_data[0:,1:], train_data[0:,0])
```

We selected variables as per the **RFE ranking** and used those for further analysis while making sure that all the datasets contain same number of columns. Performing all the feature selection methods we shortlisted below features to best predict our model.

We will discuss the algorithm used in the below section for all the Machine Learning algorithm used for classifications

3.3.1 DIFFERENT MACHINE LEARNING ALGORITHMS AND OUTPUT

3.3.1.1 LOGISTIC REGRESSION

Binary Logistic Regression is a special type of regression where binary response variable is related to a set of explanatory variables, which can be discrete and/or continuous. We are using the logistic regression model for training the model for the quarter supplied and predicting the delinquency status based on the trained model.

```
In [12]: #Select response y and predictors X
         from sklearn.linear_model import LogisticRegression
         model = LogisticRegression()

         cols_to_keep = ['cd_zero_bal6', 'cd_zero_bal1', 'repch_flagX','cd_zero_bal0','repch_flagN','repch_flagY','current_int_rt
         train_num_df = train_num_df[cols_to_keep]

         model = model.fit(train_num_df, delinquent_train)
         model.score(train_num_df, delinquent_train)

Out[12]: 0.95340575025311358
```

We will calculate the model Coefficient:

```
In [13]: model.coef_

Out[13]: array([[ -6.17702993, -12.65217804,  -4.54919556,  -4.53871363,
                   3.5160986 ,   1.32974397,   0.58789696,   0.36538899,
                   0.296647  ,   0.01463028]])
```

Import the following libraries to calculate the logit summary for the logistic regression:

from sklearn.linear_model import LogisticRegression
model = LogisticRegression()

import statsmodels.api as sm
from statsmodels.formula.api import logit, probit, poisson, ols

Apply Logit Function:

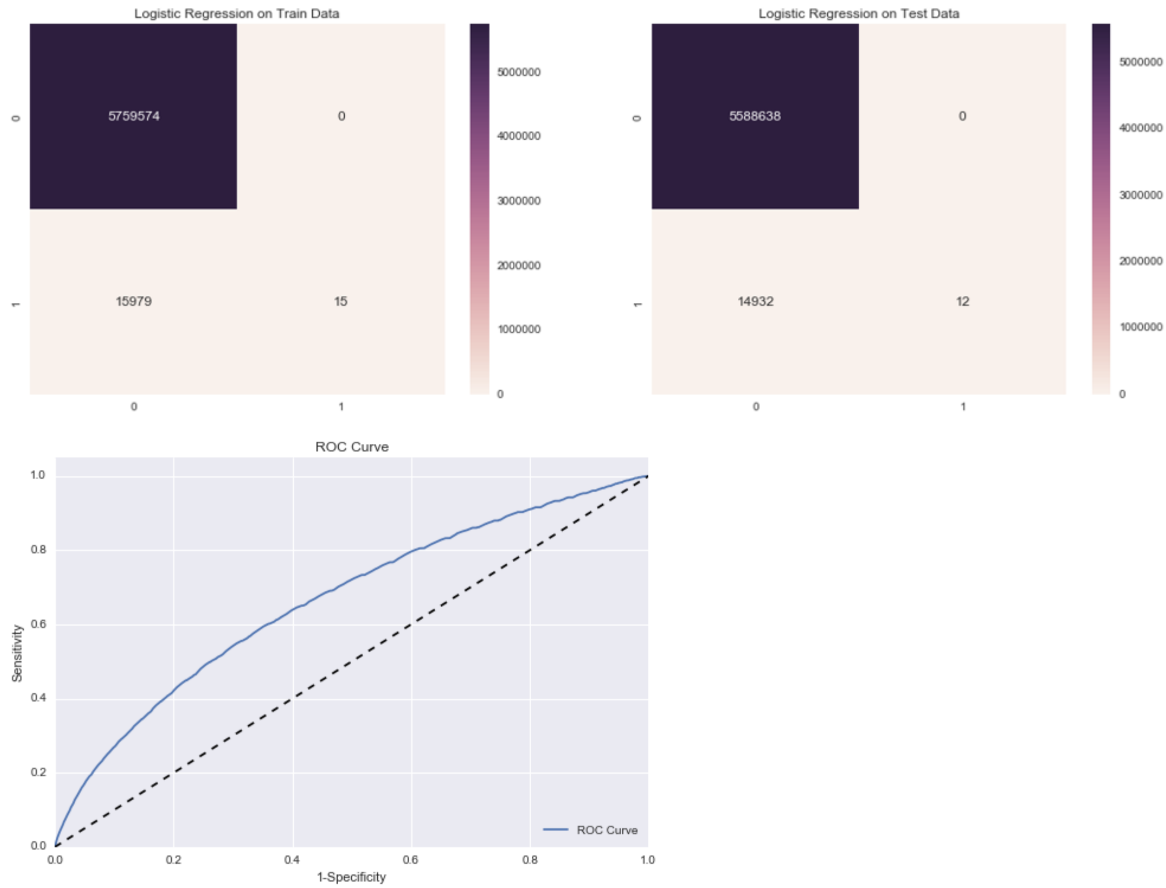
```
In [14]: import statsmodels.api as sm
         from statsmodels.formula.api import logit, probit, poisson, ols
         logit = sm.Logit(delinquent_train,train_num_df[cols_to_keep])
         dli_mod = logit.fit()
         print(dli_mod.summary())
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.174059
         Iterations: 35

C:\Users\Rajat\Anaconda3\lib\site-packages\statsmodels\base\model.py:466: ConvergenceWarning: Maximum Likelihood opti
mization failed to converge. Check mle_retvals
  "Check mle_retvals", ConvergenceWarning)

                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:              3934992
Model:                          Logit   Df Residuals:                  3934983
Method:                           MLE   Df Model:                            8
Date:                Mon, 13 Mar 2017   Pseudo R-squ.:                 0.08223
Time:                        22:37:09   Log-Likelihood:            -6.8492e+05
converged:                      False   LL-Null:                   -7.4629e+05
                                        LLR p-value:                     0.000
==================================================================================
                     coef    std err          z      P>|z|      [95.0% Conf. Int.]
----------------------------------------------------------------------------------
cd_zero_bal6     -30.9884    262.568     -0.118      0.906      -545.612   483.635
cd_zero_bal1     -25.1910    108.820     -0.231      0.817      -238.475   188.093
repch_flagX       -8.6468   6.06e+04     -0.000      1.000     -1.19e+05  1.19e+05
cd_zero_bal0     -11.2351    336.635     -0.033      0.973      -671.028   648.557
repch_flagN        5.2585   6.06e+04   8.68e-05      1.000     -1.19e+05  1.19e+05
repch_flagY       14.6739   6.06e+04      0.000      1.000     -1.19e+05  1.19e+05
current_int_rt     0.6000      0.005    125.252      0.000         0.591     0.609
cd_zero_bal3     -17.8542    108.821     -0.164      0.870      -231.140   195.432
flag_mod_n        11.2856   6.06e+04      0.000      1.000     -1.19e+05  1.19e+05
loan_age           0.0146   4.39e-05    333.618      0.000         0.015     0.015
==================================================================================
```

Now predict the delinquency status based on the Test data and generate the accuracy and Classification report.

```
def build_logistic_Regression(train_num_df_X, train_y, test_num_df_X, test_y ):
    model = LogisticRegression()
    #Train the data on the one quater
    model = model.fit(train_num_df_X,train_y)
    model.score(train_num_df_X,train_y)

    #Train the data on the one quater
    scaler = preprocessing.StandardScaler()
    X = scaler.fit_transform(train_num_df_X)
    logistic_reg_acc_matrix=metrics.accuracy_score(train_y, stratified_cv(X, train_y, linear_model.LogisticRegression))
    logistic_reg_class_matrix=metrics.classification_report(train_y, stratified_cv(X, train_y, linear_model.LogisticRegr
    logistic_reg_conf_matrix = metrics.confusion_matrix(train_y, stratified_cv(X, train_y, linear_model.LogisticRegressi
    print('Logistic Regression accuracy on train data:           {:.2f}'.format(logistic_reg_acc_matrix))
    print('Logistic Regression classification reprot on train data:\n {}\n'.format(logistic_reg_class_matrix))
    dli_pred_test=model.predict(test_num_df_X)
    logistic_reg_conf_matrix_test = confusion_matrix(test_y,dli_pred_test)
    print('Creating confusion Matrix on Train and Test data')
    conf_matrix = {
                    1: {
                        'matrix': logistic_reg_conf_matrix,
                        'title': 'Logistic Regression on Train Data',
                       },
                    2: {
                        'matrix': logistic_reg_conf_matrix_test,
                        'title': 'Logistic Regression on Test Data',
                       },
                 }
    confusion_matrix_data(conf_matrix)
    expected = test_y
    predicted = model.predict(test_num_df_X)
    acc = np.sum(predicted == expected)/len(expected)
    print("")
    print("Model Coeffiecient")
    print(model.coef_)
    print("")
    print('accuracy on Test data={}'.format(acc))
```

In this case, our function will result the Confusion matrix as shown below and the ROC curve:



Accuracy Result & Classification Report

```
Logistic Regression accuracy on train data:           1.00
Logistic Regression classification reprot on train data:
              precision    recall  f1-score   support

          0       1.00      1.00      1.00     49849
          1       0.00      0.00      0.00       151

avg / total       0.99      1.00      1.00     50000


Creating confusion Matrix on Train and Test data

Model Coeffiecient
[[-0.13453252 -1.07913452 -1.42683869 -1.42683869 -1.07913452 -0.13453252
   0.35490597  0.          -2.64050573  0.0915565 ]]

accuracy on Test data=0.99776

Classification report for Test data LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False):
             precision    recall  f1-score   support

          0       1.00      1.00      1.00     49888
          1       0.00      0.00      0.00       112

avg / total       1.00      1.00      1.00     50000
```

3.3.1.3 RANDOM FOREST

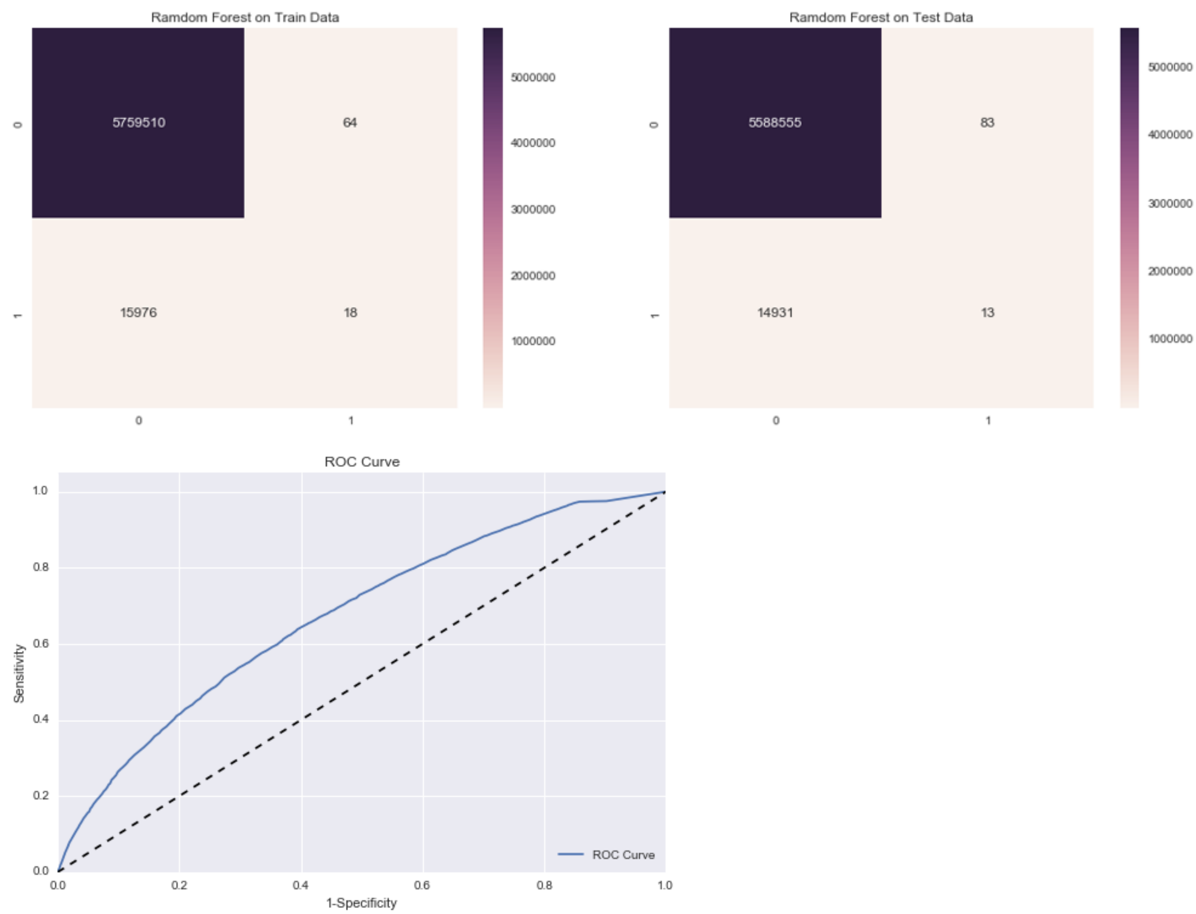
Random Forests grows many classification trees. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

```
def build_Random_Forest(train_num_df_X, train_y, test_num_df_X, test_y ):
    model = RandomForestClassifier(n_estimators = 10)
    scaler = preprocessing.StandardScaler()
    X = scaler.fit_transform(train_num_df_X)
    #Train the data on the one quater
    random_acc_matrix=metrics.accuracy_score(train_y, stratified_cv(X, train_y, ensemble.RandomForestClassifier))
    random_class_matrix=metrics.classification_report(train_y, stratified_cv(X, train_y, ensemble.RandomForestClassifie
    random_conf_matrix = metrics.confusion_matrix(train_y, stratified_cv(X, train_y, ensemble.RandomForestClassifier))
    print('Random Forest accuracy on train data:           {:.2f}'.format(random_acc_matrix))
    print('Random Forest classification reprot on train data:\n {}\n'.format(random_class_matrix))
    model = model.fit(train_num_df_X,train_y)
    model.score(train_num_df_X,train_y)
    dli_pred_test=model.predict(test_num_df_X)
    random_conf_matrix_test = confusion_matrix(test_y,dli_pred_test)
    print('Creating confusion Matrix on Train and Test data')
    conf_matrix = {
                    1: {
                        'matrix': random_conf_matrix,
                        'title': 'Ramdom Forest on Train Data',
                       },
                    2: {
                        'matrix': random_conf_matrix_test,
                        'title': 'Ramdom Forest on Test Data',
                       },
                 }
    confusion_matrix_data(conf_matrix)
    preds=model.predict_proba(test_num_df_X)[:,1]
    fpr, tpr, _ = roc_curve(test_y,preds)
    #Plot ROC Curve
    print('Creating ROC curve on Test data')
    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()
```

<u>Accuracy Result & Classification Report</u>

```
Calling Random Forest Classification with train and test dataframes
Random Forest accuracy on train data:           1.00
Random Forest classification reprot on train data:
              precision    recall  f1-score   support

          0       1.00      1.00      1.00     49849
          1       0.00      0.00      0.00       151

avg / total       0.99      1.00      1.00     50000


Creating confusion Matrix on Train and Test data

accuracy on Test data=0.99776

Classification report for Test data RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False):
             precision    recall  f1-score   support

          0       1.00      1.00      1.00     49888
          1       0.00      0.00      0.00       112

avg / total       1.00      1.00      1.00     50000


Creating ROC curve on Test data
```

In this case, our function will result the Confusion matrix as shown below and the ROC curve:



3.3.1.4 NEURAL NETWORK

Artificial neural networks are relatively crude electronic networks of neurons based on the neural structure of the brain. They process records one at a time, and learn by comparing their classification of the record (i.e., largely arbitrary) with the known actual classification of the record.

```
# build a neural network
def build_neural_network(train_num_df_X, train_y, test_num_df_X, test_y ):

    #Calculating rows and columns for input dfs
    trn_rows,trn_cols=train_num_df_X.shape
    tst_rows,tst_cols=test_num_df_X.shape

    # build train dataset
    print("Inside build_neural_network : ")
    print("Building train dataset")
    train_data = ClassificationDataSet(trn_cols, 1 , nb_classes=2)
    for k in range(len(train_num_df_X)):
        train_data.addSample(train_num_df_X.iloc[k],train_y.iloc[k])

    # build test dataset
    print("Building test dataset")
    test_data = ClassificationDataSet(tst_cols, 1 , nb_classes=2)
    for k in range(len(test_num_df_X)):
        test_data.addSample(test_num_df_X.iloc[k],test_y.iloc[k])


    print("Train Dataset input length: {}".format(len(train_data['input'])))
    print("Train Dataset output length: {}".format(len(train_data['target'])))
    print("Train Dataset input|output dimensions are {}|{}".format(train_data.indim, train_data.outdim))

    print("Train Data length: {}".format(len(train_data)))
    print("Test Data length: {}".format(len(test_data)))

    # encode with one output neuron per class
    train_data._convertToOneOfMany()
    test_data._convertToOneOfMany()

    print("Train Data input|output dimensions are {}|{}".format(train_data.indim, train_data.outdim))
    print("Test Data input|output dimensions are {}|{}".format(test_data.indim, test_data.outdim))

    # build network (INPUT=10,HIDDEN=5,CLASSES=2,outclass=SoftmaxLayer)
    print("Building Neural network with 5 hidden layer")
    network = buildNetwork(train_data.indim,5,train_data.outdim,outclass=SoftmaxLayer)

    # train network
    print("Training the network, it may take a while...")
    trainer = BackpropTrainer(network,dataset=train_data,momentum=0.1,verbose=True,weightdecay=0.01)
    trainer.trainOnDataset(train_data, 1) #training model on One epoch

    print("Total epochs: {}".format(trainer.totalepochs))
```

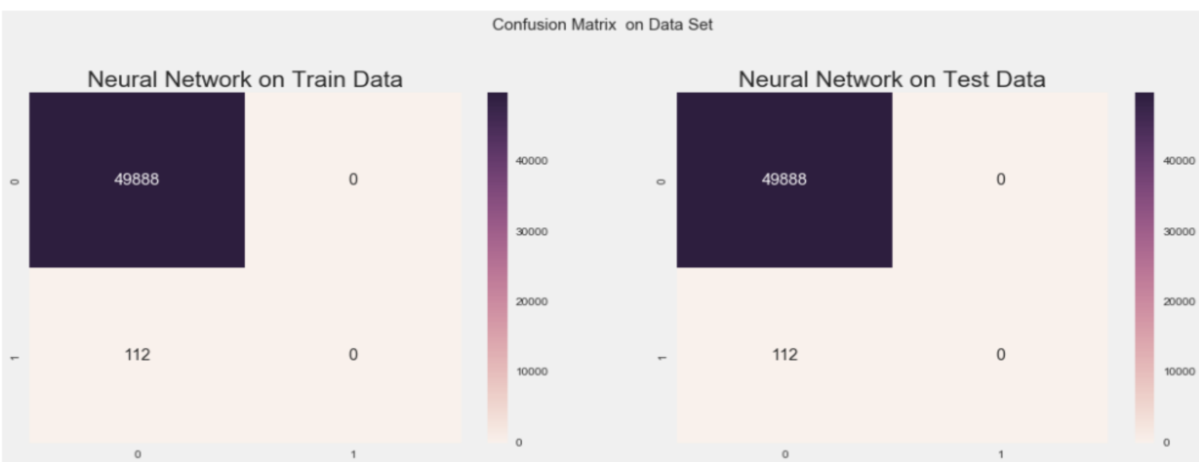
```
    # test network
    print("Predicting the output array with the trained model")
    output = network.activateOnDataset(test_data).argmax(axis=1)

    #Neural network Percent error and accuracy
    print("Percent error: {}".format(percentError(output, test_data['class'])))
    accuracy=Validator.classificationPerformance(output, test_y)
    print("Model Accuracy: {}".format(accuracy))

    #Compute confusion metrics
    cm = confusion_matrix(test_y,output)
    print(cm)
```

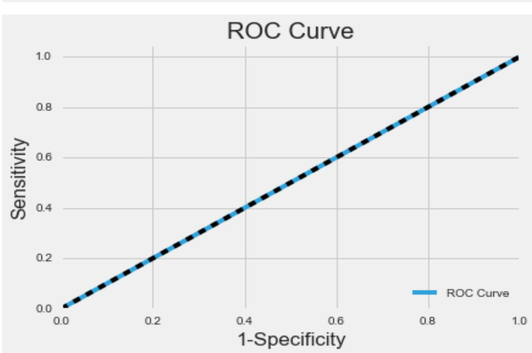
```
#Calling Neural Network
print("Calling neural network with train and test dataframes")
build_neural_network(train_num_df_X, delinquent_train_y, test_num_df_X, delinquent_test_y)
```

Accuracy Result & Classification Report

```
Calling neural network with train and test dataframes
Inside build_neural_network :
Building train dataset
Building test dataset
Train Dataset input length: 50000
Train Dataset output length: 50000
Train Dataset input|output dimensions are 10|1
Train Data length: 50000
Test Data length: 50000
Train Data input|output dimensions are 10|2
Test Data input|output dimensions are 10|2
Building Neural network with 5 hidden layer
Training the network, it may take a while...
Total error:  0.00176370039322
Total epochs: 1
Predicting the output array with the trained model
Percent error: 0.224
Model Accuracy: 0.99776
Classification report for Test data FeedForwardNetwork-44
   Modules:
    [<BiasUnit 'bias'>, <LinearLayer 'in'>, <SigmoidLayer 'hidden0'>, <SoftmaxLayer 'out'>]
   Connections:
    [<FullConnection 'FullConnection-40': 'hidden0' -> 'out'>, <FullConnection 'FullConnection-41': 'bias' -> 'out'>,
<FullConnection 'FullConnection-42': 'bias' -> 'hidden0'>, <FullConnection 'FullConnection-43': 'in' -> 'hidden0'>]
:
             precision    recall  f1-score   support

          0       1.00      1.00      1.00     49888
          1       0.00      0.00      0.00       112

avg / total       1.00      1.00      1.00     50000
```

In this case, our function will result the Confusion matrix as shown below and the ROC curve:





3.3.1.4 SUPPORT VECTOR MACHINE

SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

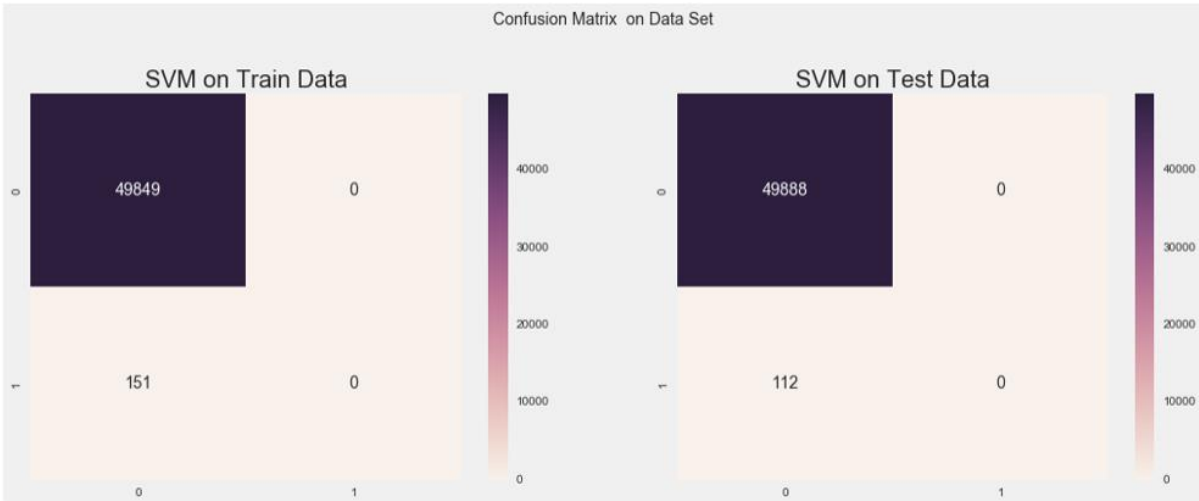
```
def build_SVM(train_num_df_X, train_y, test_num_df_X, test_y ):
    classifier = svm.LinearSVC(C=1)
    scaler = preprocessing.StandardScaler()
    X = scaler.fit_transform(train_num_df_X)
    #Train the data on the one quater
    svc_acc_matrix=metrics.accuracy_score(train_y, stratified_cv(X, train_y, svm.LinearSVC))
    svc_class_matrix=metrics.classification_report(train_y, stratified_cv(X, train_y,svm.LinearSVC))
    svc_conf_matrix = metrics.confusion_matrix(train_y, stratified_cv(X, train_y, svm.LinearSVC))
    print('Support Vector accuracy on train data:           {:.2f}'.format(svc_acc_matrix))
    print('Support Vector classification reprot on train data:\n {}\n'.format(svc_class_matrix))
    model = classifier.fit(train_num_df_X,train_y)
    classifier.score(train_num_df_X,train_y)
    dli_pred_test=classifier.predict(test_num_df_X)
    svm_conf_matrix_test = confusion_matrix(test_y,dli_pred_test)
    print('Creating confusion Matrix on Train and Test data')
    conf_matrix = {
                    1: {
                        'matrix': svc_conf_matrix,
                        'title': 'SVM on Train Data',
                       },
                    2: {
                        'matrix': svm_conf_matrix_test,
                        'title': 'SVM on Test Data',
                       },
                  }
    confusion_matrix_data(conf_matrix)
    preds=classifier.predict_proba(test_num_df_X)
    fpr, tpr, _ = roc_curve(test_y,preds)
    #Plot ROC Curve
    print('Creating ROC curve on Test data')
    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()
```

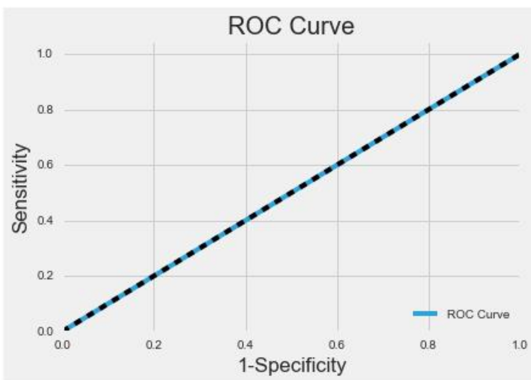
Accuracy Result & Classification Report

```
Support Vector accuracy on train data:           1.00
Support Vector classification reprot on train data:
              precision    recall  f1-score   support

          0       1.00      1.00      1.00     49849
          1       0.00      0.00      0.00       151

avg / total       0.99      1.00      1.00     50000


Creating confusion Matrix on Train and Test data

accuracy on Test data=0.99776

Classification report for Test data LinearSVC(C=1, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0):
             precision    recall  f1-score   support

          0       1.00      1.00      1.00     49888
          1       0.00      0.00      0.00       112

avg / total       1.00      1.00      1.00     50000
```

In this case, our function will result the Confusion matrix as shown below and the ROC curve:

```
Creating ROC curve on Test data
```





NOTE: These images does not contain the actual data set but we have used a subset of data set to show the working of our model.

Performance Metrics:

```
In [93]: def CopyToCSV(delinquent_test,cm,q1):
             q1="2005"
             columns=["Quarter","Total Number of actual Deliquent","Total Number of Predicted Deliquent","Total Number of Records
             df=pd.DataFrame(columns);
             df.append({"Quarter":q1,"Total Number of actual Deliquent":np.count_nonzero(delinquent_test==1),"Total Number of Pre
             writeHeader = True
             filename= "DelinquentStatus.csv"
             if not os.path.exists(filename):
                 writeHeader = False
             with open(filename, 'w',encoding='utf-8',newline="") as f:
                 if writeHeader is False:
                     df.to_csv(f, mode='a', header=True,index=False)
                 else:
                     df.to_csv(f, mode='a', header=False,index=False)
```

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

Random forest (or random forests) is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the class's output by individual trees

Why – Random Forest?

- Rand Forest is fast to build. Even faster to predict!
- Practically speaking, not requiring cross-validation alone for model selection significantly speeds training by 10x-100x or more.
- Automatic predictor selection from large number of candidates
- Resistance to over training
- Ability to handle data without preprocessing
- data does not need to be rescaled, transformed, or modified resistant to outliers
- automatic handling of missing values
- Cluster identification can be used to generate tree-based clusters through sample proximity