MID TERM REPORT

Under the guidance of

Prof. Srikanth Krishnamurthy

Team7: Vipra Shah, Snigdha Joshi

Part1: Data Downloading and Processing

Docker image:docker pull joshisn/assignment3:pt1final

Docker run -it joshisn/assignment3:pt1final /bin/bash

Python part1_Script.py

- Download Origination and performance files from https://freddiemac.embs.com/FLoan/Data/download.php downloded using mechanicalsoup by passing and saving cookies.

In [1]:
```
import mechanicalsoup as ms
import requests
from zipfile import ZipFile
from tempfile import mktemp
import urllib
import os
from io import BytesIO
from urllib import request
from os.path import basename
from requests import get  # to make GET request
url = "https://freddiemac.embs.com/FLoan/secure/auth.php"
login = "shah.vip@husky.neu.edu"
password = "3MFE5d}`"
url2 = "https://freddiemac.embs.com/FLoan/Data/download.php"
s = requests.Session()
print(s)

browser = ms.Browser(session = s)
print("Logging in....")
login_page = browser.get(url)
login_form = login_page.soup.find("form",{"class":"form"})
login_form.find("input", {"name":"username"})["value"] = login
login_form.find("input", {"name":"password"})["value"] = password
response = browser.submit(login_form, login_page.url)

login_page2 = browser.get(url2)
print("To the continue page...")

next_form = login_page2.soup.find("form",{"class":"fmform"})
a= next_form.find("input",{"name": "accept"}).attrs
a['checked']=True

response2 = browser.submit(next_form, login_page2.url)

print("Start Downloading from..."+ response2.url)

table = response2.soup.find("table",{"class":"table1"})

t = table.find_all('a')

for x in range(76,88):
    c = 'https://freddiemac.embs.com/FLoan/Data/' + t[x]['href']
    print(c)
    #name, hdrs = urllib.request.urlretrieve(c,f)
    r = s.get(c)
    z = ZipFile(BytesIO(r.content))
    z.extractall(os.getcwd()+ '/data')
```

```
<requests.sessions.Session object at 0x000001FC903F4CF8>
Logging in....
```

- Summerizing and cleaning the data based on the user guide provided. For example: Checking the valid fico(Credit Score), checking and replacing blank values.
- Processing big combinded performance files by summarizing it with maximum no of months, maximum and minimum actual upb, getting maximum of other columns, getting minimum of non mi recoveries, expenses , legal costs and taxes and insurance.

```
#function to change data type of columns
def changedatatype(dataframe):
    dataframe['repurchase_flag'] = dataframe['repurchase_flag'].astype('str')
    dataframe['modification_flag'] = dataframe['modification_flag'].astype('str')
    dataframe['zero_bal_date'] = dataframe['zero_bal_date'].astype('str')
    dataframe['ddlpi'] = dataframe['ddlpi'].astype('str')
    dataframe['net_sale_proceeds'] = dataframe['net_sale_proceeds'].astype('str')
    dataframe['delq_status'] = dataframe['delq_status'].astype('int64')
    dataframe['loan_age'] = dataframe['loan_age'].astype('int64')
    dataframe['rem_months'] = dataframe['rem_months'].astype('int64')
    dataframe['zero_balance_code'] = dataframe['zero_balance_code'].astype('int64')
    dataframe['current_def_upb'] = dataframe['current_def_upb'].astype('int64')
    dataframe['actual_loss_calc'] = dataframe['actual_loss_calc'].astype('int64')
    return dataframe

#function to fill nan values
def fillnulls(dataframe):
    dataframe['delq_status']=dataframe['delq_status'].fillna(0)
    dataframe['loan_age']=dataframe['loan_age'].fillna(0)
    dataframe['rem_months']=dataframe['rem_months'].fillna(0)
    dataframe['repurchase_flag']=dataframe['repurchase_flag'].fillna('NA')
    dataframe['modification_flag']=dataframe['modification_flag'].fillna('Not Modified')
    dataframe['zero_balance_code']=dataframe['zero_balance_code'].fillna(00)
    dataframe['zero_bal_date']=dataframe['zero_bal_date'].fillna('NA')
    dataframe['current_def_upb']=dataframe['current_def_upb'].fillna(0)
    dataframe['ddlpi']=dataframe['ddlpi'].fillna('NA')
    dataframe['mi_recoveries']=dataframe['mi_recoveries'].fillna(0)
    dataframe['net_sale_proceeds']=dataframe['net_sale_proceeds'].fillna('U')
    dataframe['non_mi_recoveries']=dataframe['non_mi_recoveries'].fillna(0)
    dataframe['expenses']=dataframe['expenses'].fillna(0)
    dataframe['legal_costs']=dataframe['legal_costs'].fillna(0)
    dataframe['maint_pres_costs']=dataframe['maint_pres_costs'].fillna(0)
    dataframe['taxes_ins']=dataframe['taxes_ins'].fillna(0)
    dataframe['misc_expenses']=dataframe['misc_expenses'].fillna(0)
    dataframe['actual_loss_calc']=dataframe['actual_loss_calc'].fillna(0)
    dataframe['modification_cost']=dataframe['modification_cost'].fillna(0)
    return dataframe
```

```python
#function to summarize performance data
def get_month(group):
    return {'month': group.max()}
def get_current_actual_upb(group):
    return {'max_current_actual_upb': group.max(), 'min_current_actual_upb': group.min()}
def get_delq_status(group):
    return {'delq_status': group.max()}
def get_loan_age(group):
    return {'loan_age': group.max()}
def get_rem_months(group):
    return {'rem_months': group.min()}
def get_repurchase_flag(group):
    return {'repurchase_flag': group.max()}
def get_modification_flag(group):
    return {'modification_flag': group.max()}
def get_zero_bal_code(group):
    return {'zero_balance_code': group.max()}
def get_zero_bal_date(group):
    return {'zero_bal_date': group.max()}
def get_current_int_rate(group):
    return {'current_int_rate': group.max()}
def get_current_def_upb(group):
    return {'current_def_upb': group.max()}
def get_ddlpi(group):
    return {'ddlpi': group.max()}
def get_mi_recoveries(group):
    return {'mi_recoveries': group.max()}
def get_net_sale_proceeds(group):
    return {'net_sale_proceeds': group.max()}
def get_non_mi_recoveries(group):
    return {'non_mi_recoveries': group.min()}
def get_expenses(group):
    return {'expenses': group.min()}
def get_legal_costs(group):
    return {'legal_costs': group.min()}
def get_maint_pres_costs(group):
    return {'maint_pres_costs': group.max()}
def get_taxes_ins(group):
    return {'taxes_ins': group.min()}
def get_misc_expenses(group):
    return {'misc_expenses': group.max()}
def get_actual_loss_calc(group):
```

```
if filepath.is_file():
    print("'Summarized_performance_data.csv' already exits!")
else:
    with open(fileName, 'w',encoding='utf-8', newline="") as file:
        for f in performance_files:
            print("Processing " + f)
            performance_df = pd.read_csv(f ,sep="|", names=['loan_seq_number','month','current_actual_upb','delq_status','loan_age
            performance_df['delq_status'] = [ 999 if x=='R' else x for x in (performance_df['delq_status'].apply(lambda x: x))]
            performance_df['delq_status'] = [ 0 if x=='XX' else x for x in (performance_df['delq_status'].apply(lambda x: x))]
            performance_df = fillnulls(performance_df)
            performance_df = changedatatype(performance_df)
            summary_df = pd.DataFrame()
            summary_df['loan_seq_number']= performance_df['loan_seq_number'].drop_duplicates()
            summary_df=summary_df.join((performance_df['month'].groupby(performance_df['loan_seq_number']).apply(get_month).unstac
            summary_df=summary_df.join((performance_df['current_actual_upb'].groupby(performance_df['loan_seq_number']).apply(get_
            summary_df=summary_df.join((performance_df['delq_status'].groupby(performance_df['loan_seq_number']).apply(get_delq_st
            summary_df=summary_df.join((performance_df['loan_age'].groupby(performance_df['loan_seq_number']).apply(get_loan_age).
            summary_df=summary_df.join((performance_df['rem_months'].groupby(performance_df['loan_seq_number']).apply(get_rem_mont
            summary_df=summary_df.join((performance_df['repurchase_flag'].groupby(performance_df['loan_seq_number']).apply(get_rep
            summary_df=summary_df.join((performance_df['modification_flag'].groupby(performance_df['loan_seq_number']).apply(get_m
            summary_df=summary_df.join((performance_df['zero_balance_code'].groupby(performance_df['loan_seq_number']).apply(get_z
            summary_df=summary_df.join((performance_df['zero_bal_date'].groupby(performance_df['loan_seq_number']).apply(get_zero_
            summary_df=summary_df.join((performance_df['current_int_rate'].groupby(performance_df['loan_seq_number']).apply(get_cu
            summary_df=summary_df.join((performance_df['current_def_upb'].groupby(performance_df['loan_seq_number']).apply(get_cur
            summary_df=summary_df.join((performance_df['ddlpi'].groupby(performance_df['loan_seq_number']).apply(get_ddlpi).unstac
            summary_df=summary_df.join((performance_df['mi_recoveries'].groupby(performance_df['loan_seq_number']).apply(get_mi_re
            summary_df=summary_df.join((performance_df['net_sale_proceeds'].groupby(performance_df['loan_seq_number']).apply(get_n
            summary_df=summary_df.join((performance_df['non_mi_recoveries'].groupby(performance_df['loan_seq_number']).apply(get_n
            summary_df=summary_df.join((performance_df['expenses'].groupby(performance_df['loan_seq_number']).apply(get_expenses).
            summary_df=summary_df.join((performance_df['legal_costs'].groupby(performance_df['loan_seq_number']).apply(get_legal_c
            summary_df=summary_df.join((performance_df['maint_pres_costs'].groupby(performance_df['loan_seq_number']).apply(get_ma
            summary_df=summary_df.join((performance_df['taxes_ins'].groupby(performance_df['loan_seq_number']).apply(get_taxes_ins
            summary_df=summary_df.join((performance_df['misc_expenses'].groupby(performance_df['loan_seq_number']).apply(get_misc_
            summary_df=summary_df.join((performance_df['actual_loss_calc'].groupby(performance_df['loan_seq_number']).apply(get_ac
            summary_df=summary_df.join((performance_df['modification_cost'].groupby(performance_df['loan_seq_number']).apply(get_m
            if flag == 0:
                summary_df.to_csv(file, mode='a', header=True,index=False)
                flag = 1
            else:
                summary_df.to_csv(file, mode='a', header=False,index=False)
```

Exploratory Data Analysis:

Read summarized origination data

```
In [10]: import pandas as pd
         import os, matplotlib
         import matplotlib.pyplot as plt
         import numpy as np
         import seaborn as sns
         %matplotlib inline

         sample_clean_file = os.getcwd() + "/sample_orig_combined.csv"
         sample_df = pd.read_csv(sample_clean_file,low_memory=False)
         sample_df.head(2)
```

Out[10]:

	credit_score	first_payment_date	fthb_flag	matr_date	msa	mortage_insurance_pct	no_of_units	occupancy_status	cltv	dti_ratio	...	p
0	591	200504	N	203503	39100	0.0	1	O	48	34	...	N
1	792	200503	N	203502	39100	0.0	1	O	90	33	...	N

2 rows × 26 columns

Read summarized performance data

```
In [11]: performance_file = os.getcwd() + "/Summarized_performance_data.csv"
         perf_df = pd.read_csv(performance_file, low_memory=False)
         perf_df.head(2)
```

Out[11]:

	loan_seq_number	month	max_current_actual_upb	min_current_actual_upb	delq_status	loan_age	rem_months	repurchase_flag	modif
0	F105Q1000064	200912	62000.0	0.0	0	57	303	NaN	Not M
1	F105Q1000076	201011	197000.0	0.0	0	69	291	NaN	Not M

2 rows × 24 columns

Add a new column for Year and Read merged file

```
In [12]: perf_df['Year'] = ['20'+ x for x in (perf_df['loan_seq_number'].apply(lambda x: x[2:4]))]

In [13]: perf_df['Year'].unique()

Out[13]: array(['2005', '2006', '2007', '2008', '2009', '2010', '2011', '2012',
                '2013', '2014', '2015', '2016'], dtype=object)

In [14]: merged_df = pd.merge(sample_df,perf_df,on="loan_seq_number",how="right")

In [15]: merged_df.head(2)
```

Out[15]:

	credit_score	first_payment_date	fthb_flag	matr_date	msa	mortage_insurance_pct	no_of_units	occupancy_status	cltv	dti_ratio	...
0	591.0	200504.0	N	203503.0	39100.0	0.0	1.0	O	48.0	34.0	...
1	792.0	200503.0	N	203502.0	39100.0	0.0	1.0	O	90.0	33.0	...

2 rows × 50 columns

```
In [16]: merged_df.shape

Out[16]: (574957, 50)
```

Group merged file based on year

```
In [17]: yearwise_df = pd.DataFrame()
         grouped = merged_df.groupby('Year')
         yearwise_df = yearwise_df.append(grouped.aggregate(np.mean))
         yearwise_df.drop(['first_payment_date', 'matr_date', 'msa','zipcode', 'ddlpi','month', 'zero_bal_date', 'rem_months'], axis=
         #yearwise_df = yearwise_df.transpose()
         yearwise_df.head(2)
```

Out[17]:

	credit_score	mortage_insurance_pct	no_of_units	cltv	dti_ratio	original_upb	original_ltv	original_int_rt	original_loan_tern
Year									
2005	724.727288	3.027869	1.022480	70.694997	34.262401	177046.460124	68.955302	5.796721	324.040912
2006	723.229643	3.226935	1.024275	72.977019	35.954134	186530.265996	70.394848	6.397613	337.934244

2 rows × 26 columns

UPB trends over the years

In [18]:
```
def upb_trends_over_time():
    original_upb = yearwise_df['original_upb']
    max_current_actual_upb = yearwise_df['max_current_actual_upb']
    min_current_actual_upb = yearwise_df['min_current_actual_upb']
    year = perf_df['Year'].drop_duplicates()
    #year_df = pd.DataFrame(perf_df['Year'].drop_duplicates()).reset_index()
    #year_df.columns = ['Count', 'Year']
    #year_df = year_df.ix[(year_df['Year'] == '2007') | (year_df['Year'] == '2008') | (year_df['Year'] == '2009')]
    plt.figure(num=None, figsize=(14, 12),dpi=50, facecolor='c', edgecolor='b')
    ax1=plt.subplot(211)
    plt.plot(year,original_upb,'y--',year,max_current_actual_upb,'b--',year,min_current_actual_upb,'r--')
    plt.xlabel('YEAR')
    plt.ylabel('UPB')
    plt.legend(['Original UPB','Max UPB','Min UPB'])
    plt.grid(True)
    plt.title('UPB Trends Over Years')

upb_trends_over_time()
```

Zero balance code trends over time

Zero Balance Code Trends over Time

In [325]:
```
total_records = merged_df.shape[0]
print("Total records is {}".format(total_records))
```

```
Total records is 574957
```

In [326]:
```
total_default_records = merged_df.ix[(merged_df['zero_balance_code'] == 3) | (merged_df['zero_balance_code'] == 6) |
                                     (merged_df['zero_balance_code'] == 9)]
count = total_default_records.shape[0]
print("Total number of default record is: {}".format(count))
default_ratio = str(round(count/total_records,2))
print("Total default ratio is {}".format(default_ratio))
```

```
Total number of default record is: 15042
Total default ratio is 0.03
```

In [327]:
```
total_prepaid_records = merged_df.ix[merged_df['zero_balance_code'] == 1]
count_prepaid = total_prepaid_records.shape[0]
print("Total number of prepaid record is {}".format(count_prepaid))
prepaid_ratio = str(round(count_prepaid/total_records,2))
print("Total prepaid ratio is {}".format(prepaid_ratio))
```

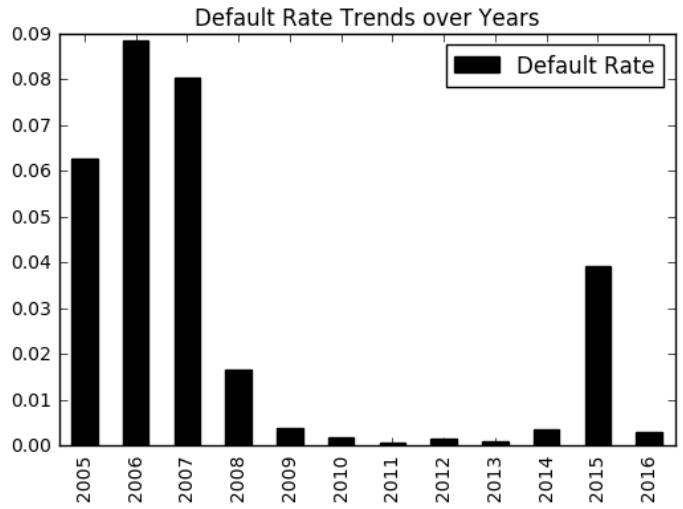
```
Total number of prepaid record is 314001
Total prepaid ratio is 0.55
```

In [328]:
```
total_default_records_by_year = total_default_records.groupby(perf_df['Year'])['loan_seq_number'].count()
total_default_records_by_year
```

Plot graph for Default Rate Trends over Years

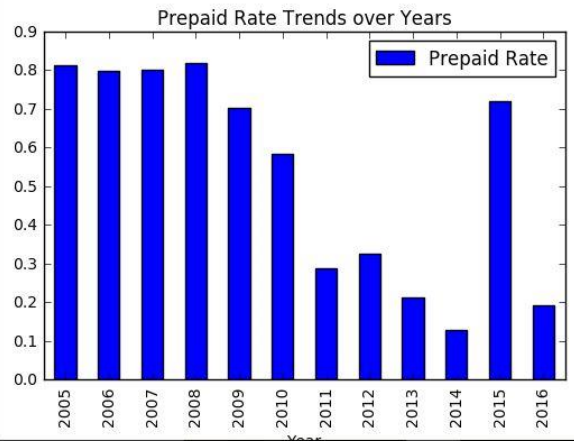
```
In [331]: default_rate_by_year = pd.concat([total_default_records_by_year,total_records_by_year],axis = 1, join='ou
          default_rate_by_year['Default Rate'] = default_rate_by_year['Default Records Number']/default_rate_by_yea
          default_rate_by_year.plot(title ="Default Rate Trends over Years",y='Default Rate', color='k', kind='bar'
```

```
Out[331]: <matplotlib.axes._subplots.AxesSubplot at 0x22553ce82b0>
```

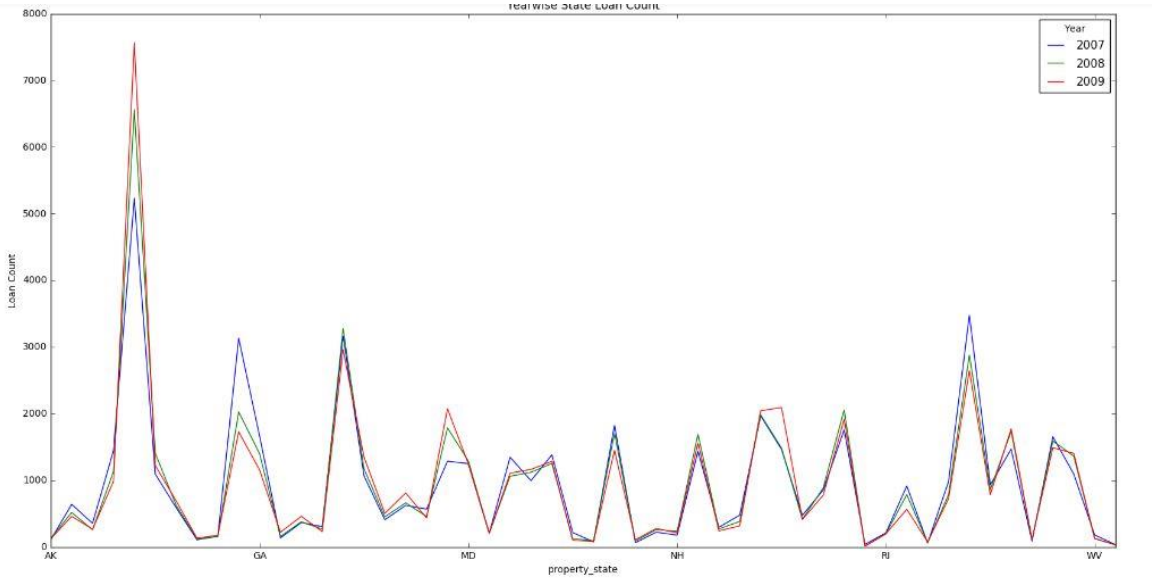



```
In [380]: prepaid_rate_by_year = pd.concat([total_prepaid_records_by_year,total_records_by_year],axis = 1, join='outer')
          prepaid_rate_by_year['Prepaid Rate'] = prepaid_rate_by_year['Prepaid Records Number']/prepaid_rate_by_year['Total Records']
          prepaid_rate_by_year.plot(title ="Prepaid Rate Trends over Years",y='Prepaid Rate', color='b', kind='bar')
```

```
Out[380]: <matplotlib.axes._subplots.AxesSubplot at 0x2255f562f28>
```



State wise loan analysis for year 2007,2008 and 2009



Zero balance code Trends over the years

Zero Balance Code Trends ¶

In [432]:
```
def get_zero_bal_code(df, x):
    if (df.ix[x,'zero_balance_code'] == 0) or (df.ix[x,'zero_balance_code'] == 1):
        return 'Prepaid'
    elif (df.ix[x,'zero_balance_code'] == 3):
        return "Foreclosure Alternative Group"
    elif (df.ix[x,'zero_balance_code'] == 6):
        return "Repurchse"
    elif (df.ix[x,'zero_balance_code'] == 9):
        return "REO Disposition"
    else:
        return 'NA'
```

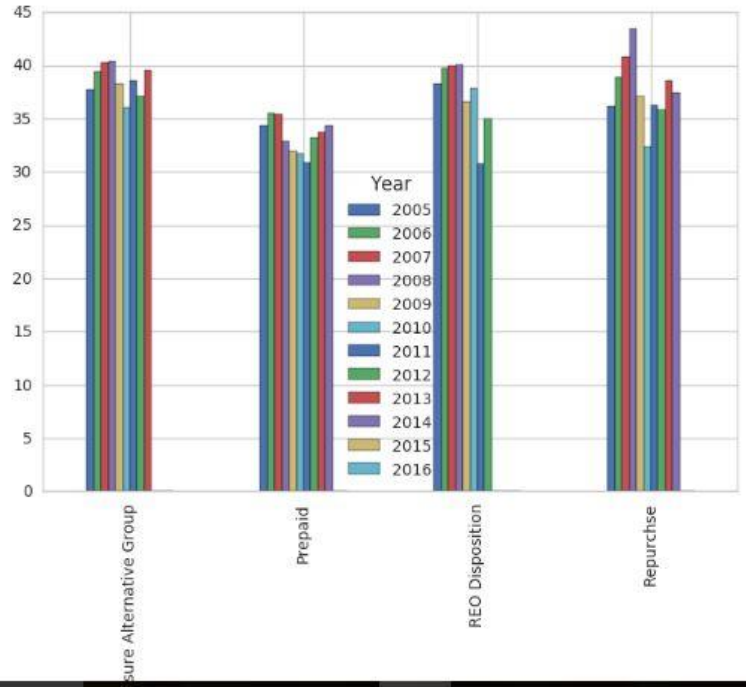
In [433]:
```
loan_purpose_status = merged_df.groupby([lambda x: get_zero_bal_code(merged_df, x), 'loan_purpose'])['loan_seq_number'].coun
loan_purpose_status = loan_purpose_status.unstack(level=0)
plt.figure(figsize=(20, 12))
plt.title('Loan Purpose vs Zero Balance Code')
ax =sns.heatmap(loan_purpose_status.T, mask= loan_purpose_status.T.isnull(),annot=True, fmt='d',cmap='viridis');
```

Debt-to-income trends over the years

Debt-to-income trends over years

```
In [448]: dti_trends = merged_df.groupby([lambda x: get_zero_bal_code(merged_df, x),perf_df['Year']])['dti_ratio'].mean()
          dti_trends.unstack().plot(kind='bar')
```

```
Out[448]: <matplotlib.axes._subplots.AxesSubplot at 0x2255e53d5f8>
```



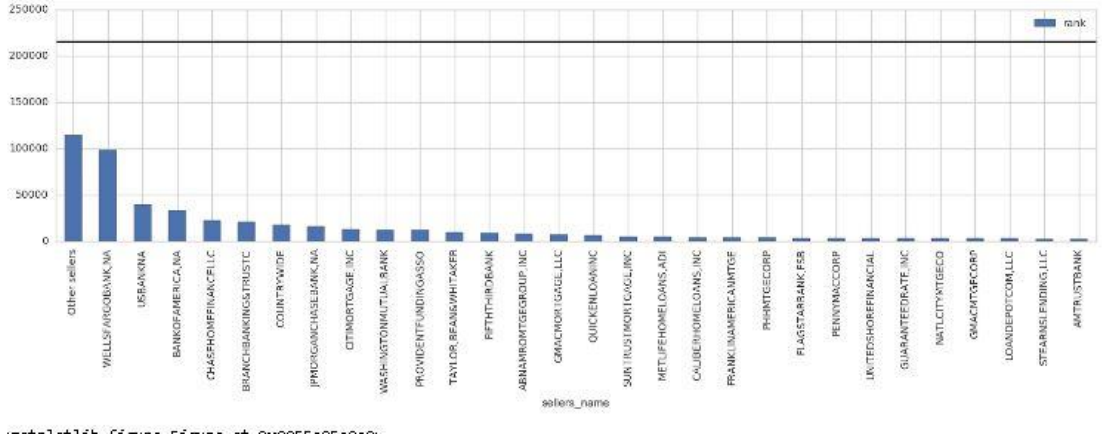
Sellers and Services Ranking

Sellers and Servicer Ranking Analysis

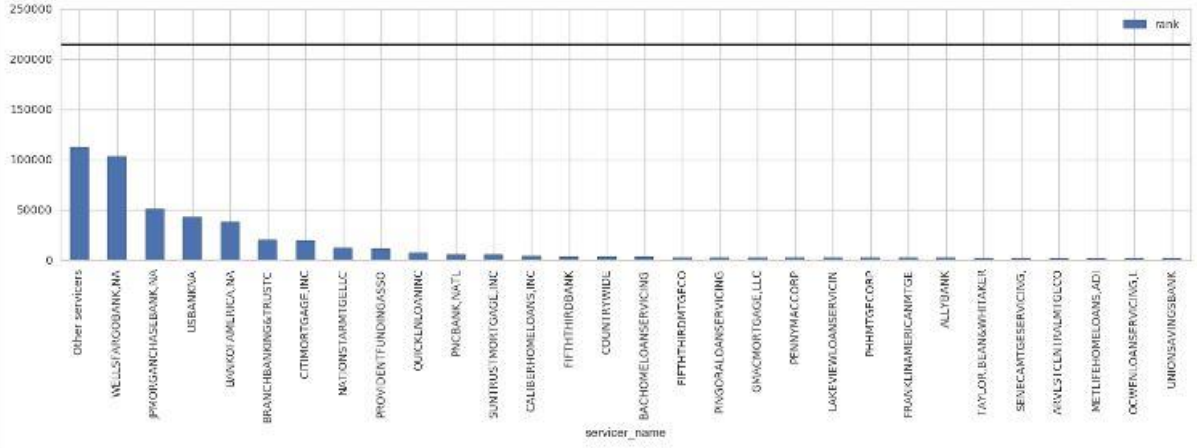
In [465]:
```
def plot_ranking_graph(df):
    plt.figure()
    df.head(n=30).plot(figsize=(18,4), kind='bar')
    plt.axhline(merged_df.original_upb.mean(), color='k')
```

In [468]:
```
#sellers ranking
sellers_rank = pd.DataFrame()
sellers_rank['rank'] = merged_df['loan_seq_number'].groupby(merged_df['sellers_name']).count().sort_values(ascending = False)
plot_ranking_graph(sellers_rank)

#servicer ranking
servicer_rank = pd.DataFrame()
servicer_rank['rank'] = merged_df['loan_seq_number'].groupby(merged_df['servicer_name']).count().sort_values(ascending = Fals
plot_ranking_graph(servicer_rank)
```

```
<matplotlib.figure.Figure at 0x2255e555390>
```



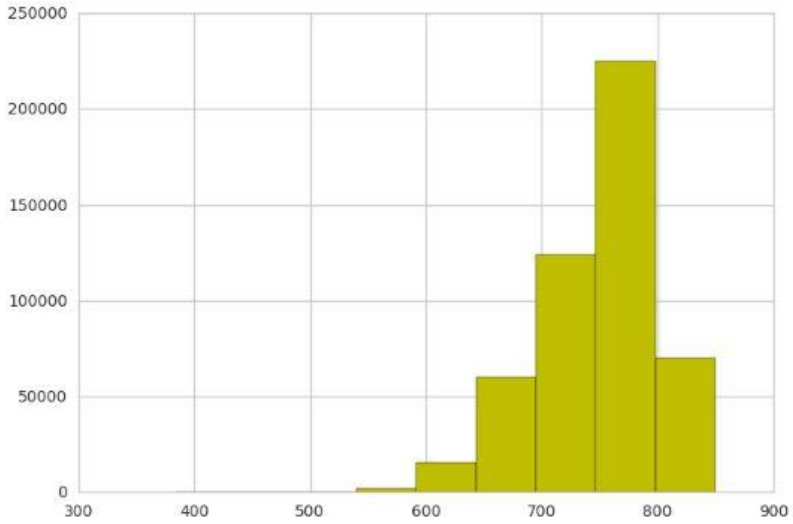

```
<matplotlib.figure.Figure at 0x2255c85e2e8>
```



Credit Score Trends

Credit Score Analysis

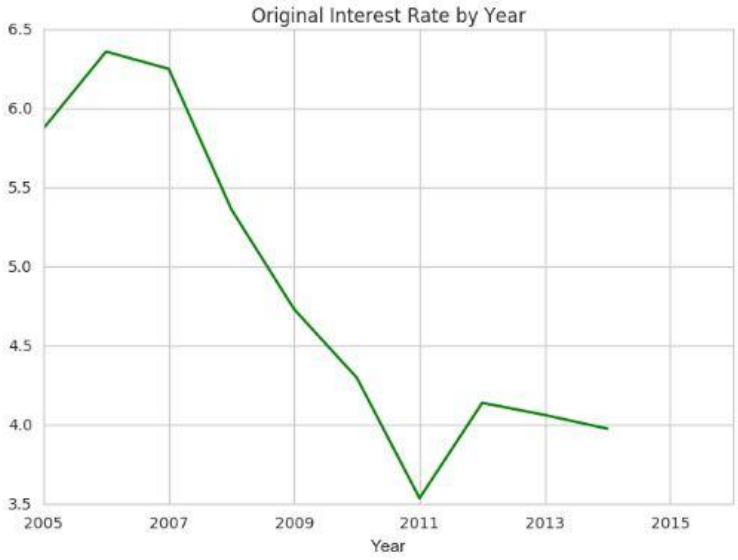
```
In [477]: merged_df['credit_score'].hist(color='y')
Out[477]: <matplotlib.axes._subplots.AxesSubplot at 0x2255c986208>
```



Original Interest Rate Trends

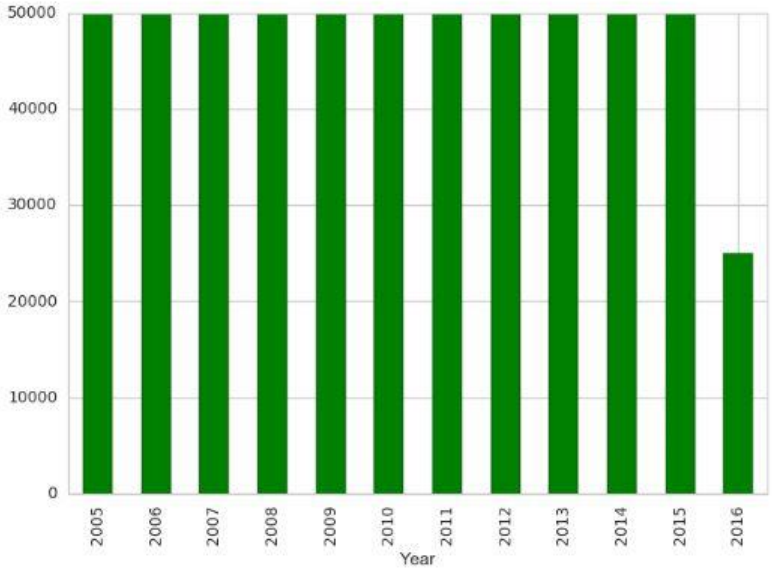
Interest rate trends over time

```
In [491]: rate_by_year = merged_df.groupby(perf_df['Year'])['original_int_rt'].mean()
          rate_by_year.plot(title="Original Interest Rate by Year", color='g')
Out[491]: <matplotlib.axes._subplots.AxesSubplot at 0x225540d5198>
```



Number of loan application over the years:

```
In [507]: total_applications_by_year = merged_df.groupby(perf_df['Year'])['loan_seq_number'].count()
          total_applications_by_year.plot(kind='bar', color='g')

Out[507]: <matplotlib.axes._subplots.AxesSubplot at 0x2255401ac50>
```



Part2: Predication and Classification

Docker image: docker pull joshisn/assignment3:part2

Docker run -it joshisn/assignment3:part2 /bin/bash

Python Script_part2.py

1. Build models
    a. Clean Data

```
def cleanData(data):
    data = data.drop(data['credit_score'].loc[(data['credit_score'] < 301) | (data['credit_score'] > 850)].index)
    data = data.dropna(subset=['credit_score'])
    data['fthb_flag'] = data['fthb_flag'].fillna("NA")
    data = data.dropna(subset=['msa'])
    data['mortage_insurance_pct'] = data['mortage_insurance_pct'].fillna(0)
    data['no_of_units'] = data['no_of_units'].fillna(0)
    data['cltv'] = data['cltv'].fillna(0)
    data['dti_ratio'] = data['dti_ratio'].fillna(0)
    data['original_ltv'] = data['original_ltv'].fillna(0)
    data['ppm_flag'] = data['ppm_flag'].fillna("U")
    data['prop_type']=data['prop_type'].fillna('NA')
    data['loan_purpose']=data['loan_purpose'].fillna('NA')
    data = data.dropna(subset=['zipcode'])
    data['number_of_borrowers'] = data['number_of_borrowers'].fillna(1)
    data['super_conforming_flag'] = data['super_conforming_flag'].fillna("N")
    return data
```

    b. Convert Columns to numeric and change data type

```
def convertNumer(data):
    data['fthb_flag'] = data['fthb_flag'].replace(['Y','N','NA'],[1,2,3])
    data['occupancy_status'] = data['occupancy_status'].replace(['I','O','S'],[1,2,3])
    data['channel'] = data['channel'].replace(['B','C','R','T'],[1,2,3,4])
    data['ppm_flag'] = data['ppm_flag'].replace(['Y','N','U'],[1,2,3])
    data['prop_type'] = data['prop_type'].replace(['CO','LH','PU','MH','SF','CP','NA'],[1,2,3,4,5,6,7])
    data['loan_purpose'] = data['loan_purpose'].replace(['P','C','N','NA'],[1,2,3,4])
    data['super_conforming_flag'] = data['super_conforming_flag'].replace(['Y','N'],[0,1])
    return data

def changedatatype(data):
    data[['credit_score','msa','no_of_units','mortage_insurance_pct','cltv','dti_ratio','original_ltv','zipcode','number_of_borro
    data[['fthb_flag','occupancy_status','channel']] = data[['fthb_flag','occupancy_status','channel']].astype('int64')
    data[['ppm_flag','prop_type','loan_purpose','super_conforming_flag']]= data[['ppm_flag','prop_type','loan_purpose','super_con
    data[['product_type','property_state']] = data[['product_type','property_state']].astype('str')
    data[['loan_seq_number','sellers_name','servicer_name']] = data[['loan_seq_number','sellers_name','servicer_name']].astype('s
    return data
```

c. Perform linear regression

```
def perform_linear_regression(lr, xaxis, yaxis):
    print ("Start of linear regression")
    #Fit the linear model
    lr.fit(xaxis,yaxis)
    print ("Intercept is ",lr.intercept_)
    #Calculate variance score
    var_score =lr.score(xaxis,yaxis)
    #Print coefficients for x-axis only once
    print("Coefficient is ",len(lr.coef_))
    #print(pd.DataFrame(list(zip(xaxis.columns,lr.coef_ )), columns=['Features','Estimated Coefficients']))
    #To calculate difference between estimated and actual y-axis values
    diff =r2_score(yaxis,lr.predict(xaxis))
    print("Linear Regression Score is: ",diff)
    print ("End of linear regression")
    return lr
```

d. Feature Selection using selectKBest

```
def perform_selectKBest(lr, xaxis, yaxis):
    feat=SelectKBest(f_regression,k=10)
    feat = feat.fit(xaxis, yaxis)
    #To find out score, we need to reduce xaxis to selected features
    X = feat.fit_transform(xaxis,yaxis)
    fit = lr.fit(X,yaxis)
    var_score = lr.score(X, yaxis)
    diff=r2_score(yaxis,lr.predict(X))
    #idxs_selected = feat.get_support(indices=True)
    features=pd.DataFrame(list(zip(xaxis,sorted(feat.scores_, reverse = True))),columns=["features","scores"])
    print("K best r2 Score is: ",diff)
    print(features)
```

e. Feature Selection using Lasso

f. Feature selection using RME

```
def perform_lassoLinear(lr,xaxis, yaxis):
    lasso = linear_model.Lasso(alpha=0.1)
    lasso.fit(xaxis,yaxis)
    predict=lasso.predict(xaxis)
    score=r2_score(yaxis,predict)
    features=pd.DataFrame(list(zip(xaxis,sorted(lasso.coef_, reverse = True))),columns=["features","coefficient"])
    print("Lasso Regression r2 score:", score)
    print(features)


def perform_recursiveFE(lr,xaxis,yaxis):
    selector = RFE(lr,10)
    feat = selector.fit(xaxis, yaxis)
    prediction=feat.predict(xaxis)
    score=r2_score(yaxis,prediction)
    print("RFE r2 score: ",score)
    rankfeatures=pd.DataFrame(list(zip(xaxis.columns,sorted(feat.ranking_))),columns=["features","ranking"])
    print(rankfeatures)
```

We have selected recursive feature selection algorithm to select features

```
def calculatestat(lr,xaxis,yaxis):
    y_pred=lr.predict(xaxis)
    MAE=mean_absolute_error(yaxis,y_pred)
    print("Mean Absolute Error: ", MAE)
    RMSE=math.sqrt(mean_squared_error(yaxis,y_pred))
    print("Root of Mean Squared Deviation: ",RMSE)
    MAPE=np.mean(np.abs((yaxis - y_pred) / yaxis)) * 100
    print("Mean Absolute Percentage Error: ",MAPE)

def keepReqColumns(df):
    df=df.drop('zipcode',axis=1)
    df=df.drop('number_of_borrowers',axis=1)
    df=df.drop('original_loan_term',axis=1)
    df=df.drop('original_ltv',axis=1)
    df=df.drop('channel',axis=1)
    df=df.drop('ppm_flag',axis=1)
    df=df.drop('prop_type',axis=1)
    df=df.drop('loan_purpose',axis=1)
    df=df.drop('super_conforming_flag',axis=1)
    return df
```

g. Neural Network algorithm

```
def Neural_Network(xtrain,ytrain,xtest,ytest):
    #Hidden nodes
    hidden_net = 2
    #Epoch is a single pass through the entire training set, followed by testing of the verification set.
    epoch = 2
    ytrain = ytrain.reshape(-1,1)
    input_cnt = xtrain.shape[1]
    target_cnt = ytrain.shape[1]
    dataset = SupervisedDataSet(input_cnt, target_cnt)
    dataset.setField( 'input', xtrain )
    dataset.setField( 'target', ytrain )
    network = buildNetwork( input_cnt, hidden_net, target_cnt, bias = True )
    #Trainer that trains the parameters of a module according to a supervised dataset (potentially sequential) by backpropagating
    trainer = BackpropTrainer( network,dataset )
    print("---------------Neural Network---------------")
    print("Train Data")
    for e in range(epoch):
        mse = trainer.train()
        rmse = math.sqrt(mse)
    print("MSE, epoch {}: {}".format(e + 1, mse))
    print("RMSE, epoch {}: {}".format(e + 1, rmse))

    ytest=ytest.reshape(-1,1)
    input_size = xtest.shape[1]
    target_size = ytest.shape[1]
    dataset = SupervisedDataSet( input_size, target_size )
    dataset.setField( 'input', xtest)
    dataset.setField( 'target', ytest)
    model = network.activateOnDataset(dataset)
```

h. Linear Regression
i. Random forest

```
def LinearRegressionAnalysis(xtrain,ytrain,xtest,ytest):
    trainreg  = lr.fit(xtrain, ytrain)
    print("---------------Linear Regression---------------")
    print("Train Data:")
    print(calculatestat(trainreg, xtrain, ytrain ))
    print("Test Data:")
    print(calculatestat(trainreg, xtest, ytest))

def Random_Forest(xtrain,ytrain,xtest,ytest):
    rand_forest = RandomForestRegressor(n_estimators=15,max_depth=10)
    rand_forest = rand_forest.fit(xtrain,ytrain)
    print("---------------Random Forest---------------")
    print("Train Data:")
    print(calculatestat(rand_forest, xtrain, ytrain ))
    print("Test Data:")
    print(calculatestat(rand_forest,xtest,ytest))
```

```
---------------Linear Regression---------------
Train Data:
Mean Absolute Error:  0.215635923889
Root of Mean Squared Deviation:  0.28904986796221477
Mean Absolute Percentage Error:  3.8205931686665737
None
Test Data:
Mean Absolute Error:  0.248198869421
Root of Mean Squared Deviation:  0.322479782238875
Mean Absolute Percentage Error:  4.241883897814012
None
---------------Random Forest---------------
Train Data:
Mean Absolute Error:  0.202088167802
Root of Mean Squared Deviation:  0.26988543697197087
Mean Absolute Percentage Error:  3.5827581077356547
None
Test Data:
Mean Absolute Error:  0.231182052829
Root of Mean Squared Deviation:  0.30108961650858196
Mean Absolute Percentage Error:  3.9872682618105753
None
```

```
---------------Neural Network---------------
Train Data
MSE, epoch 2: 0.06650558148048123
RMSE, epoch 2: 0.25788676096395724
```

```
Test Data:
MSE:  0.130481074978
RMSE:  0.36122164245478
```

From above 3 algorithms we chose **random forest** because:

- Better results than Linear Regression
- Lot less processing time than Neural networks(Fast and scalable)

Furthermore,

- Processing time does not increase substantially with increase in number of observations.
- Easy to interpret, adjust(tune) parameters to achieve desired results.
- It is Non-parametric; we don't have to worry about outliers.

2. Financial crisis and Economic Boom (https://www.stlouisfed.org/financial-crisis/full-timeline)

```
def financial_crisis_economic(trainQ, testQ):
    from sklearn.ensemble import RandomForestRegressor
    data1 = getTrainTest(trainQ, testQ)
    df2 = cleanData(data1[0])
    df2 = convertNumer(df2)
    trainframe = changedatatype(df2)
    trainf=keepReqColumns(trainframe)
    df3 = cleanData(data1[1])
    df3 = convertNumer(df3)
    testframe = changedatatype(df2)
    testf=keepReqColumns(testframe)
    ytrain = trainf.original_int_rt
    xtrain = trainf.drop('original_int_rt',axis=1)._get_numeric_data()
    #n-jobs:No ristriction on use of processors
    #The out-of-bag (OOB) error is the average error for each z_i calculated using predictior
    rand_forest = RandomForestRegressor(n_jobs=-1,  oob_score = True,max_depth=10)
    rand_forest.fit(xtrain,ytrain)
    print("---------------Financial Crisis---------------")
    print("Train Data: ",trainQ)
    print(calculatestat(rand_forest, xtrain, ytrain ))
    ytest=testf.original_int_rt
    xtest = testf.drop('original_int_rt',axis=1)._get_numeric_data()
    print("Test Data: ",testQ)
    print(calculatestat(rand_forest,xtest,ytest))
```

During financial crisis of 2007 Freddie Mac declared following actions:

- In Q1 & 2: it will no longer buy the riskiest subprime mortgages and mortgage-related securities.
- In Q3: Countrywide Financial Corporation warns of “difficult conditions.
- In Q4: Financial market pressures intensify, reflected in diminished liquidity in interbank funding markets.

Train	Q12007	Q22007	Q32007	Q42007
MAE	0.2294	0.2387	0.236	0.2574
RMS	0.3023	0.3134	0.3133	0.3409
MAPE	3.7052	3.797	3.5654	4.0535

Test	Q22007	Q32007	Q42007	Q12008
MAE	0.2947	0.2889	0.2721	0.3013
RMS	0.3723	0.3689	0.3564	0.3866
MAPE	4.8494	4.6764	4.1559	4.8085

MAE	28.4656	21.03058	15.29661	17.05517
RMS	23.1558	17.709	13.75678	13.40569
MAPE	30.8809	23.16039	16.56196	18.62588

As we can see, difference between Training and Testing MAE and RMS has increasing substantially in Q1 and Q2 of year 2007.

1. In Q12007 train and test data RMS has increased by 23.15%
2. As model is trained, RMS is decreasing

The federal takeover of Freddie Mac in **September 2008** improved situations. It was one of the financial events among many in the ongoing subprime mortgage crisis.

What the Fed did:

The Fed initiated purchases of $500 billion in mortgage-backed securities.

- It announced purchases of up to $100 billion in debt obligations of mortgage giants Fannie Mae, Freddie Mac, Ginnie Mae and Federal Home Loan Banks.
- The Fed cut the key interest rate to near zero, Dec. 16, 2008.
- In March 2009, the Fed expanded the mortgage buying program and said it would purchase $750 billion more in mortgage-backed securities.
- The Fed also announced it would invest another $100 billion in Freddie debt and purchase up to $300 billion of longer-term Treasury securities over a period of six months.
- In the first quarter of 2010, with a total of $1.25 trillion in purchases of mortgage-backed securities and $175 billion of agency debt purchases.

Thus:

Mortgage rates dropped significantly, to as low as 5%, in year 2009.

Train	Q12009	Q22009	Q32009	Q42009
MAE	0.2261	0.2051	0.2339	0.176
RMS	0.3031	0.2753	0.2995	0.2257
MAPE	4.5276	4.1961	4.5929	3.5638

Test	Q22009	Q32009	Q42009	Q12010
MAE	0.2304	0.2121	0.2662	0.187
RMS	0.3172	0.2917	0.3358	0.2431
MAPE	4.5705	4.2903	5.1118	3.7502

MAE	1.901813357	3.412969283	13.80932022	6.25
RMS	4.651930056	5.957137668	12.12020033	7.709348693
MAPE	0.947521866	2.244941732	11.2978728	5.230372075

As we can see, in Q1 & Q2 2007 RMS and MAE is substantially lower than 2007.

As situations began to improve in 2009, observations recorded changes, which in turn increased prediction error.

Economic Boom

The 1990s economic boom in the United States was an extended period of economic prosperity, during which GDP increased continuously for almost ten years (the longest recorded expansion in the history of the United States). It commenced after the end of the early 1990s recession in March 1991, and ended in March 2001 with the start of the early 2000s recession, following the bursting of the dot com bubble**.**

1995-2000 is also remembered for a series of global economic financial crises that threatened the U.S. economy.

Despite occasional stock market downturns and some distortions in the trade deficit, the US economy remained resilient until the dot-com bubble peaked in March 2000.

Train	Q11999	Q21999	Q31999	Q41999
MAE	0.2107	0.2347	0.259	0.2358
RMS	0.292	0.3156	0.3564	0.3305
MAPE	3.0332	3.28	3.3647	2.9768

Test	Q21999	Q31999	Q41999	Q12000
MAE	0.3438	0.2722	0.2773	0.2645
RMS	0.4246	0.3686	0.3818	0.3663
MAPE	5.0457	3.8271	3.5847	3.3405

MAE	63.17038443	15.97784406	7.065637066	12.17133164
RMS	45.4109589	16.79340938	7.126823793	10.83207262
MAPE	66.34907029	16.67987805	6.538472969	12.21781779

Observations:

1. As financial crisis began towards 2000, there were drastic changes in values in datasets.
2. Which caused RMS to be increased drastically in Q12009.
3. Model started to get settled towards Q2 and Q3 of 2009.

The Fed was buying another $40 billion in mortgage-backed investments each month until the economy improved. That's on top of the tens of billions of dollars in mortgages it already had been buying each month, making U.S. banks flush with cash.

Starting in Dec 2013, the Fed began to reduce its $85 billion-per-month asset purchases by $10 billion per month at each Fed meeting, cutting them to $35 billion in June 2014.

Train	Q12013	Q22013	Q32013	Q42013
MAE	0.1599	0.1765	0.2508	0.1744
RMS	0.2094	0.2299	0.3233	0.2269
MAPE	4.7259	5.0445	6.2844	4.0814

Test	Q22013	Q32013	Q42013	Q12014
MAE	0.1685	0.1868	0.2767	0.2001
RMS	0.226	0.251	0.3508	0.2603
MAPE	4.9119	5.2668	6.7444	4.5782

MAE	5.378361476	5.835694051	10.32695375	14.73623853
RMS	7.927411652	9.177903436	8.50603155	14.72014103
MAPE	3.935758268	4.406779661	7.319712303	12.17229382

Observations:

1. At the end of 2013, there was sudden increse in mortgage rates, which lasted till end of Q12014.
2. As a result, we can seen gradual increase in RMS values towards end of year.

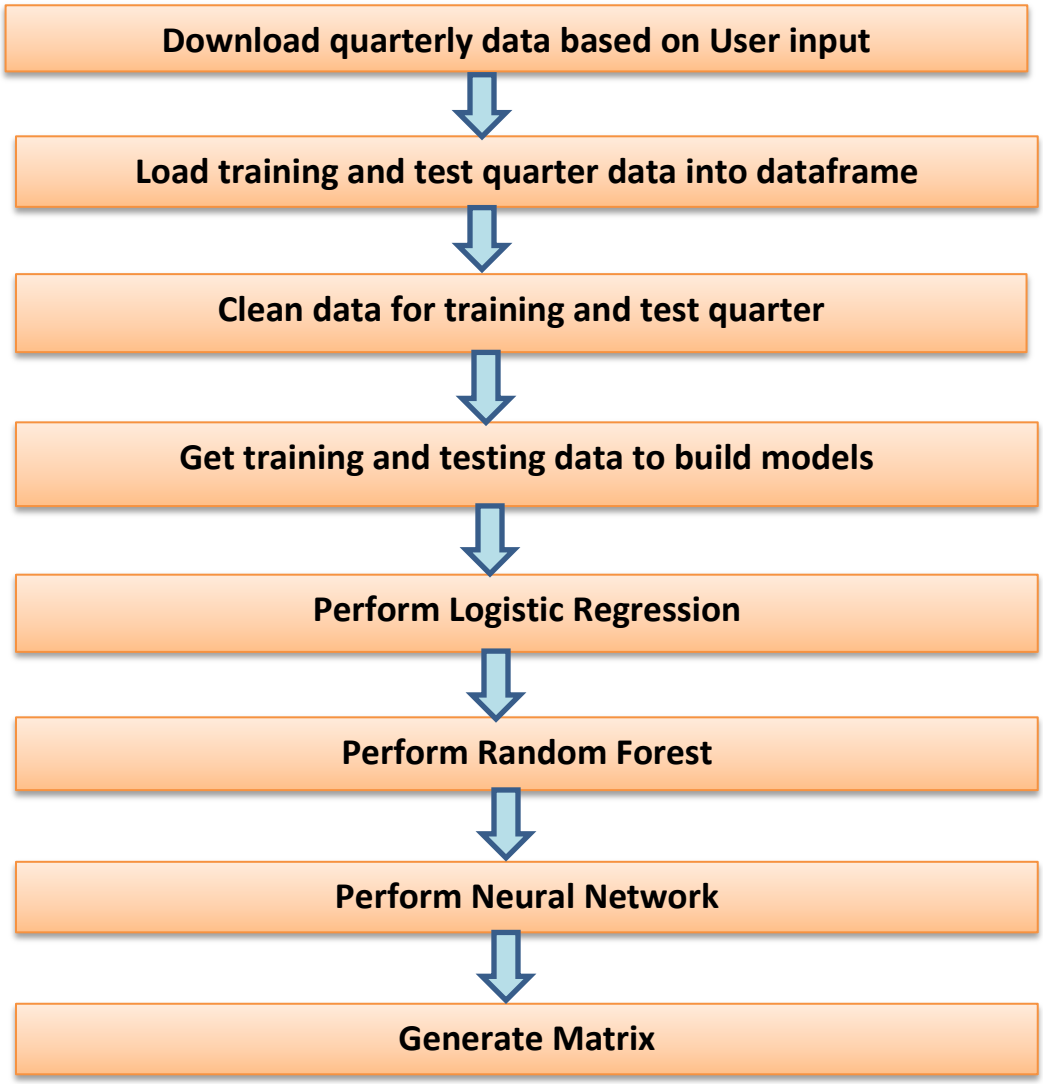
Would you recommend using this model for the next quarter? Justify

As per observations, our model is correctly picking up economic changes.

In stable period RMS percentage differences between RMS values are rarely greater than **13%**

Therefore I would recommend to use this model for next quarter.

3. Classification:



Step1: download historical data based on user input

```
def downloadhistoricaldata(trainQ, testQ, t,s, flag):
    for l in t:
        if(trainQ in l['href'] or testQ in l['href']):
            c = 'https://freddiemac.embs.com/FLoan/Data/' + l['href']
            r = s.get(c)
            z = ZipFile(BytesIO(r.content))
            z.extractall(os.getcwd()+ '/data_part2')
            flag = 1
    return flag

def login(login, password, trainQ, testQ):
    flag = 0
    s = requests.Session()
    url = "https://freddiemac.embs.com/FLoan/secure/auth.php"
    url2 = "https://freddiemac.embs.com/FLoan/Data/download.php"
    browser = ms.Browser(session = s)
    print("Logging in....")
    login_page = browser.get(url)
    login_form = login_page.soup.find("form",{"class":"form"})
    login_form.find("input", {"name":"username"})["value"] = login
    login_form.find("input", {"name":"password"})["value"] = password
    response = browser.submit(login_form, login_page.url)
    login_page2 = browser.get(url2)
    print("To the continue page...")

    next_form = login_page2.soup.find("form",{"class":"fmform"})
    a= next_form.find("input",{"name": "accept"}).attrs
    a['checked']=True
```

Step2: Load dataframe for training and test quarter

```
def load_df(quarter1,quarter2):
    foldername= 'data_part2'
    historical_path= str(os.getcwd())+"\\"+ foldername
    train_path= historical_path+"\historical_data1_time_"+str(trainQ)+".txt"
    test_path= historical_path+"\historical_data1_time_"+str(testQ)+".txt"
    global train_df
    global test_df
    column_names=['loan_seq_number', 'month', 'current_actual_upb', 'delq_status','loan_age', 'rem_months', 'repurchase_flag
                  'zero_bal_date', 'current_int_rate', 'current_def_upb', 'ddlpi', 'mi_recoveries', 'net_sales_proceeds'
                  'non_mi_recoveries', 'expenses', 'legal_costs', 'maint_pres_costs', 'taxes_ins', 'misc_expenses',
                  'actual_loss_calc', 'modification_cost']

    train_df = pd.read_table(os.path.join(train_path),
                      delimiter='|', names=column_names, index_col=None, nrows=1048576, usecols=list(np.arange(23)))
    test_df = pd.read_table(os.path.join(test_path),
                      delimiter='|', names=column_names, index_col=None, nrows=1048576, usecols=list(np.arange(23)))
```

Step3: Clean testing and training data

```
def cleandf(df):
    df.delq_status = df.delq_status.replace('R', '1').astype('float64')
    df.rem_months = df.rem_months.replace(np.nan, 0)
    df.rem_months = df.rem_months.astype('category')
    df.repurchase_flag = df.repurchase_flag.replace(np.nan, 0)
    df.repurchase_flag = df.repurchase_flag.astype('category')
    df.modification_flag = df.modification_flag.replace(np.nan, 0)
    df.modification_flag = df.modification_flag.astype('category')
    df.zero_balance_code = df.zero_balance_code.replace(np.nan, 0)
    df.zero_balance_code = df.zero_balance_code.astype('category')
    df.zero_bal_date = df.zero_bal_date.replace(np.nan, 0)
    df.zero_bal_date = df.zero_bal_date.astype('category')
    df.current_def_upb = df.current_def_upb.replace(np.nan, 0)
    df.current_def_upb = df.current_def_upb.astype('category')
    df.ddlpi = df.ddlpi.replace(np.nan, 0)
    df.ddlpi = df.ddlpi.astype('category')
    df.mi_recoveries = df.mi_recoveries.replace(np.nan, 0)
    df.net_sales_proceeds = df.net_sales_proceeds.replace(np.nan, 0)
    df.net_sales_proceeds = df.net_sales_proceeds.replace('C', 1)
    df.net_sales_proceeds = df.net_sales_proceeds.replace('U', 0)
    df.net_sales_proceeds.astype('float64')
    df.non_mi_recoveries = df.non_mi_recoveries.replace(np.nan, 0)
    df.expenses = df.expenses.replace(np.nan, 0)
    df.legal_costs = df.legal_costs.replace(np.nan, 0)
    df.maint_pres_costs = df.maint_pres_costs.replace(np.nan, 0)
    df.taxes_ins = df.taxes_ins.replace(np.nan, 0)
    df.misc_expenses = df.misc_expenses.replace(np.nan, 0)
    df.actual_loss_calc = df.actual_loss_calc.replace(np.nan, 0)
    df.modification_cost = df.modification_cost.replace(np.nan, 0)
```

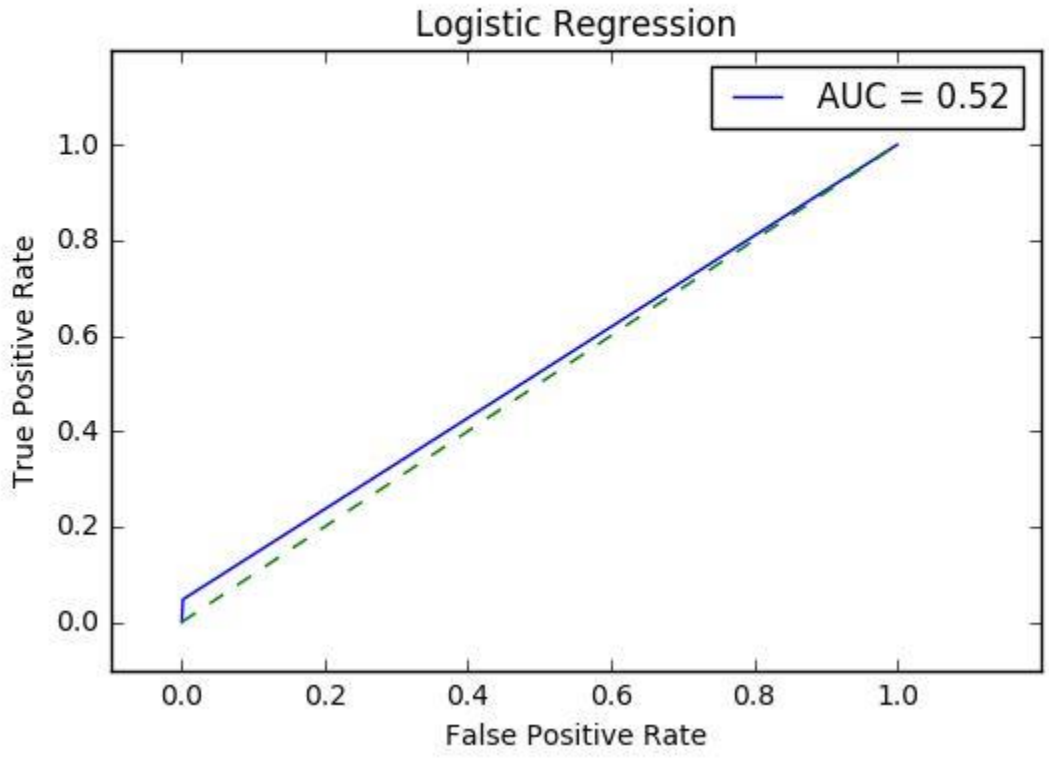
Step4: Get training and testing data based on requirment and drop not required columns

```
def get_train_test_set():

    def f(row):
        if row['delq_status'] > 0:
            val = 1
        else:
            val = 0
        return val

    #Create dummy variables
    train_df_dummies = pd.get_dummies(train_df[['repurchase_flag', 'modification_flag']])
    test_df_dummies = pd.get_dummies(test_df[['repurchase_flag', 'modification_flag']])

    dropped_train_df = train_df.drop(['loan_seq_number','repurchase_flag', 'modification_flag'],axis=1)
    dropped_test_df = test_df.drop(['loan_seq_number','repurchase_flag', 'modification_flag'],axis=1)

    global final_train_df
    global final_test_df
    final_train_df = pd.concat([dropped_train_df, train_df_dummies], axis=1)
    final_test_df = pd.concat([dropped_test_df, test_df_dummies], axis=1)

    #create target variable
    final_train_df['Deliquent'] = final_train_df.apply(f, axis=1)
    final_test_df['Deliquent'] = final_test_df.apply(f, axis=1)

    #create testing and training set
    X_train = final_train_df.drop(['delq_status', 'Deliquent'], axis=1)
    y_train = final_train_df['Deliquent']

    X_test = final_test_df.drop(['delq_status', 'Deliquent'], axis=1)
    y_test = final_test_df['Deliquent']

    #preprocess to scale it between 0 and 1
    X_train = preprocessing.minmax_scale(X_train)
    X_test = preprocessing.minmax_scale(X_test)

    return X_train,y_train,X_test,y_test
```

Step5: Build Logistic Regression Model

```
def logisticRegression(X_train, y_train,X_test,y_test,logire_return_dict):
    logisticReg = LogisticRegression()

    #fit the data
    logisticReg.fit(X_train, y_train)

    #prediction data
    y_train_predicted = logisticReg.predict(X_train)
    y_test_predicted = logisticReg.predict(X_test)
    conf_mat_logire = metrics.confusion_matrix(y_test, y_test_predicted)
    print("Confusion matrix for " + trainQ + " is: " + str(conf_mat_logire))

    #ROC and AUC
    false_positive_rate, true_positive_rate, thresholds = metrics.roc_curve(y_test, y_test_predicted)
    roc_auc_logire = metrics.auc(false_positive_rate, true_positive_rate)
    logire_return_dict['roc_auc_logire']=roc_auc_logire
    logire_return_dict['conf_mat_logire']=conf_mat_logire
    roc_auc_logire = round(roc_auc_logire,2)
    print("ROC curve for " + trainQ + " is: " + str(roc_auc_logire))

    #plot Curve
    plt.title("Logistic Regression")
    plt.plot(false_positive_rate, true_positive_rate, 'b', label='AUC = ' + str(roc_auc_logire))
    plt.legend(loc='upper right')
    plt.plot([0,1],[0,1],'g--')
    plt.xlim([-0.1,1.2])
    plt.ylim([-0.1,1.2])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
```

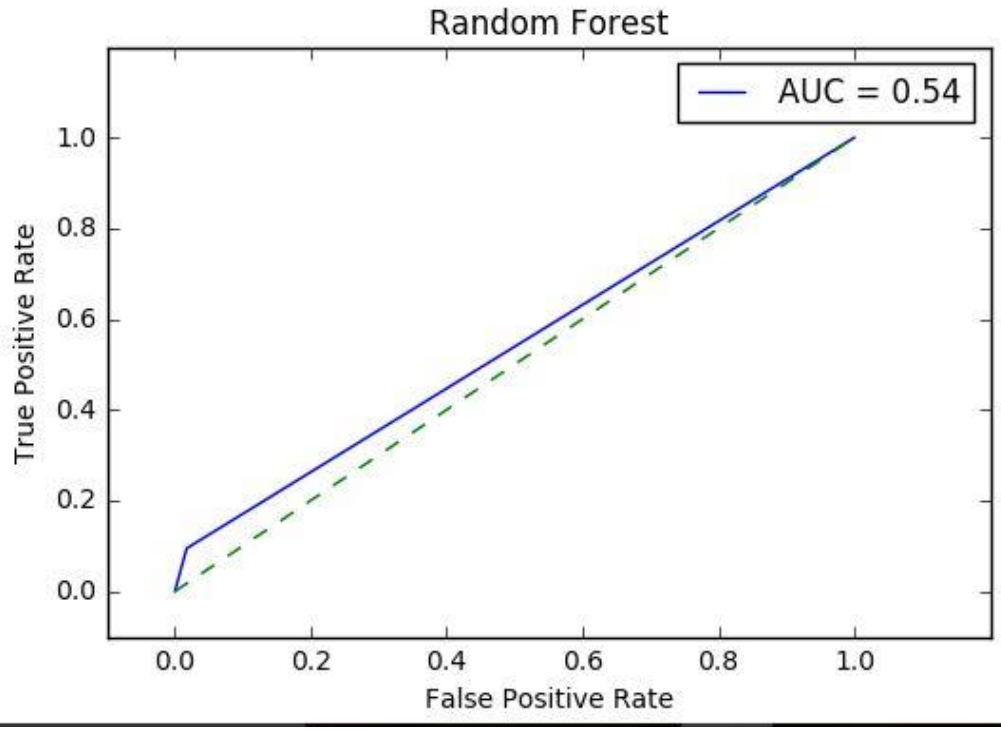
```
Confusion matrix for Q12005 is: [[997305   1496]
 [ 47412   2363]]
ROC curve for Q12005 is: 0.52
```



Step6: Build Random Forest Model

```
def random_forest(X_train, y_train,X_test,y_test,randf_return_dict):
    ranf = RandomForestClassifier(n_estimators=100)

    #fit the data
    ranf.fit(X_train, y_train)

    #prediction data
    y_train_predicted = ranf.predict(X_train)
    y_test_predicted = ranf.predict(X_test)

    #confusion matrix
    ranf_conf_mat = metrics.confusion_matrix(y_test, y_test_predicted)
    print("Confusion matrix for " + trainQ + " is: " + str(ranf_conf_mat))

    #ROC and AUC curve
    false_positive_rate, true_positive_rate, thresholds = metrics.roc_curve(y_test, y_test_predicted)
    ranf_roc_auc = metrics.auc(false_positive_rate, true_positive_rate)
    ranf_roc_auc = round(ranf_roc_auc, 2)
    print("ROC curve for " + trainQ + " is: " + str(ranf_roc_auc))
    randf_return_dict['ranf_roc_auc']=ranf_roc_auc
    randf_return_dict['ranf_conf_mat']=ranf_conf_mat

    #plot curve
    plt.title("Random Forest")
    plt.plot(false_positive_rate, true_positive_rate, 'b', label='AUC = ' + str(ranf_roc_auc))
    plt.legend(loc='upper right')
    plt.plot([0,1],[0,1],'g--')
    plt.xlim([-0.1,1.2])
    plt.ylim([-0.1,1.2])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
```

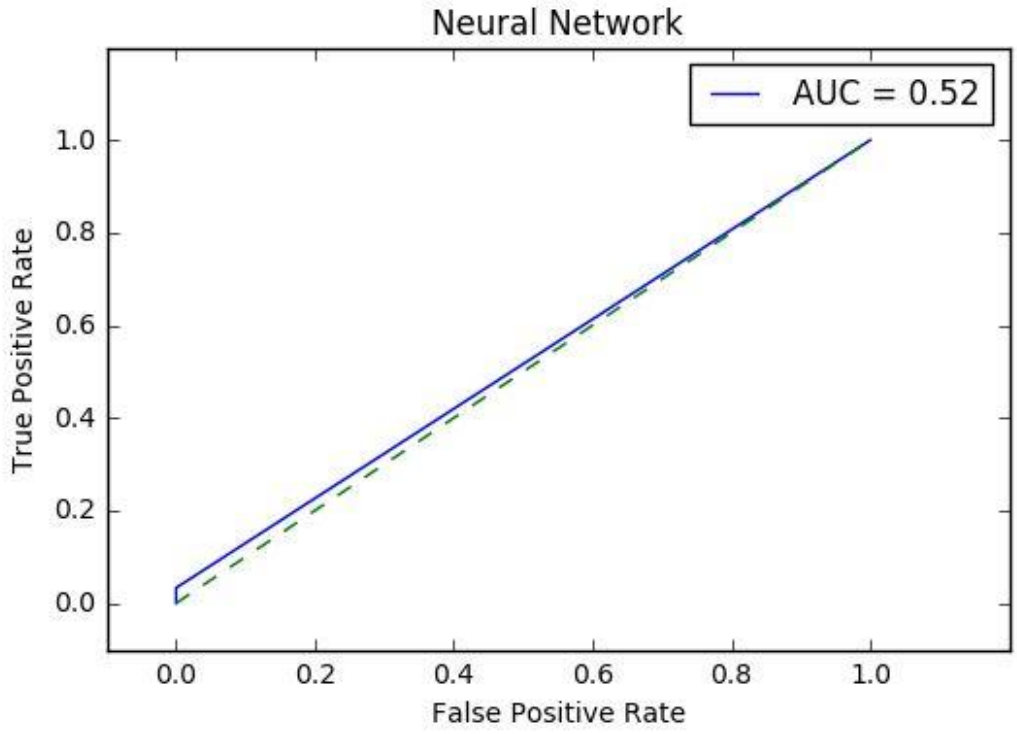
```
Confusion matrix for Q12005 is: [[981453  17348]
 [ 45071   4704]]
ROC curve for Q12005 is: 0.54
```



Step7: Build Neural Network

```
def neural_network(X_train, y_train,X_test,y_test,neun_return_dict):
    neun = MLPClassifier()

    #fit the data
    neun.fit(X_train, y_train)

    #prediction data
    y_train_predicted = neun.predict(X_train)
    y_test_predicted = neun.predict(X_test)

    #confusion matrix
    neun_conf_mat = metrics.confusion_matrix(y_test, y_test_predicted)
    print("Confusion matrix for " + trainQ + " is: " + str(neun_conf_mat))

    #ROC and AUC
    false_positive_rate, true_positive_rate, thresholds = metrics.roc_curve(y_test, y_test_predicted)
    neun_roc_auc = metrics.auc(false_positive_rate, true_positive_rate)
    neun_roc_auc = round(neun_roc_auc,2)
    print("ROC curve for " + trainQ + " is: " + str(neun_roc_auc))
    neun_return_dict['neun_roc_auc']=neun_roc_auc
    neun_return_dict['neun_conf_mat']=neun_conf_mat

    #plot curve
    plt.title("Neural Network")
    plt.plot(false_positive_rate, true_positive_rate, 'b',label='AUC = ' + str(neun_roc_auc ))
    plt.legend(loc='upper right')
    plt.plot([0,1],[0,1],'g--')
    plt.xlim([-0.1,1.2])
    plt.ylim([-0.1,1.2])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
```

```
Confusion matrix for Q12005 is: [[998771     30]
 [ 48126   1649]]
ROC curve for Q12005 is: 0.52
```



Step8: Generate matrix for number of delinquents and number of predicted delinquents

```
def get_matrix(quarter,roc_auc_logire,ranf_roc_auc,neun_roc_auc,conf_mat_logire,ranf_conf_mat,neun_conf_mat,matrix):
    all_auc={'logisticRegression':roc_auc_logire,'random_forest':ranf_roc_auc,'neural_network':neun_roc_auc}
    print(all_auc)
    model=max(all_auc.items(),key=operator.itemgetter(1))[0]
    all_conf={'logisticRegression':conf_mat_logire,'random_forest':ranf_conf_mat,'neural_network':neun_conf_mat}
    all_conf_array = all_conf[model]
    No_of_actual_delq = all_conf_array[1][0] + all_conf_array[1][1]
    No_of_pred_delq = all_conf_array[0][1] + all_conf_array[1][1]
    No_of_records = all_conf_array[0][1] + all_conf_array[1][1] + all_conf_array[1][0] + all_conf_array[0][0]
    No_of_delq_properly_classified = all_conf_array[1][1]
    No_of_nonDelq_improperly_classified_as_delq = all_conf_array[0][1]
    all_conf_df = pd.DataFrame(OrderedDict((('Quarter',[quarter]),
                                    ('Number_of_Actual_Delinquents',[No_of_actual_delq]),
                                    ('Number_of_Predicted_Delinquents',[No_of_pred_delq]),
                                    ('Number_of_records_in_the_dataset',[No_of_records]),
                                    ('Number_of_Delinquents_properly_classified',[No_of_delq_properly_classified]),
                                    ('Number_of_non_delinquents_improperly_classified_as_delinquents',[No_of_nonDelq_improperly_classi

    matrix=pd.concat([all_conf_df, matrix],axis=0)
    return matrix
```

```
if __name__ == '__main__':

    global X_train
    global y_train
    global X_test
    global y_test
    global matrix
    global conf_mat_logire
    global ranf_conf_mat
    global neun_conf_mat
    global roc_auc_logire
    global ranf_roc_auc
    global neun_roc_auc
    manager=Manager()
    logire_return_dict=manager.dict()
    randf_return_dict=manager.dict()
    neun_return_dict=manager.dict()
    matrix = pd.DataFrame(OrderedDict((('Quarter',[]),
                                    ('Number_of_Actual_Delinquents',[]),
                                    ('Number_of_Predicted_Delinquents',[]),
                                    ('Number_of_records_in_the_dataset',[]),
                                    ('Number_of_Delinquents_properly_classified',[]),
                                    ('Number_of_non_delinquents_improperly_classified_as_delinquents',[])))))
    #Get user input to download quarterly historical data
    quarter_input=str(os.getcwd())+"\\"+'quarter_input.csv'
    reader=csv.reader(open(quarter_input),delimiter=',')
    quarter_data=[]
    for row in reader:
        quarter_data.append(row)
        trainQ= quarter_data[0][0]
        testQ = quarter_data[0][1]
        username = quarter_data[0][2]
        password = quarter_data[0][3]

    quarters=[]
    quarters.append(trainQ)
    #login(username, password, trainQ, testQ)
    for q in quarters:
        nextQuarter=get_next_quarter(q)
        load_df(q,nextQuarter)
        cleandf(train_df)
        cleandf(test_df)
        X_train,y_train,X_test,y_test=get_train_test_set()
        print("Traing and Testing set are ready to build models.")

        p1 = Process(target=logisticRegression,args=(X_train, y_train,X_test,y_test,logire_return_dict))
```

```
        username = quarter_data[0][2]
        password = quarter_data[0][3]

    quarters=[]
    quarters.append(trainQ)
    #login(username, password, trainQ, testQ)
    for q in quarters:
        nextQuarter=get_next_quarter(q)
        load_df(q,nextQuarter)
        cleandf(train_df)
        cleandf(test_df)
        X_train,y_train,X_test,y_test=get_train_test_set()
        print("Traing and Testing set are ready to build models.")

        p1 = Process(target=logisticRegression,args=(X_train, y_train,X_test,y_test,logire_return_dict))
        p2 = Process(target=random_forest,args=(X_train, y_train,X_test,y_test,randf_return_dict))
        p3 = Process(target=neural_network,args=(X_train, y_train,X_test,y_test,neun_return_dict))
        p1.start()
        p2.start()
        p3.start()
        p1.join()
        p2.join()
        p3.join()
        logisticRegression(X_train, y_train,X_test,y_test,logire_return_dict)
        random_forest(X_train, y_train,X_test,y_test,randf_return_dict)
        neural_network(X_train, y_train,X_test,y_test,neun_return_dict)
        matrix=get_matrix(q,logire_return_dict['roc_auc_logire'],randf_return_dict['ranf_roc_auc'],neun_return_dict['neun_roc_auc'])

    print(matrix)
    filename = os.getcwd() + '\matrix_classification.csv'
    matrix.to_csv(filename,index= False, encoding='utf-8')
```

	A	B	C	D	E	F
1	Quarter	Number_of_Actual_Delinqu	Number_of_Predicted_Delinq	Number_of_records_in_the_d	Number_of_Delinquents_properly_cla	Number_of_non_delinquents_improperly_classified_as_delinquents
2	Q11999	32068	29551	1048576	3107	26444
3	Q21999	37465	6258	1048576	2211	4047
4	Q12005	49775	30972	1048576	5536	25436
5	Q22005	54668	49924	1048576	9571	40353
6	Q32005	61791	12201	1048576	4165	8036
7	Q42005	76714	342517	1048576	39926	302591
8	Q12006	78771	53982	1048576	15389	38593
9	Q22006	82420	46380	1048576	12092	34288

References:

https://www.thebalance.com/stock-market-returns-by-year-2388543

http://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html

http://bigdata-madesimple.com/how-to-run-linear-regression-in-python-scikit-learn/

https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/

http://www.bankrate.com/finance/federal-reserve/financial-crisis-timeline.aspx

https://www.quora.com/What-are-the-advantages-of-different-classification-algorithms