ADVANCED DATA SCIENCE

Assignment 3

Midterm Case Studies

Prof. Sri Krishnamurthy

Team Members:

Sayali Borse

Rishi Rajani

Komal Ambekar

- Content:

Sr.No.	Content
1	Data downloading and Pre-processing
2	Exploratory Data Analysis
3	Building and evaluating models
4	Prediction
5	Classification

- <u>Part 1: Data Downloading and Pre-processing</u>

    - Download Origination and performance files from https://freddiemac.embs.com/FLoan/Data/download.php downloaded using mechanicalsoup by passing and saving cookies.
    - Summarizing and cleaning the data based on the user guide provided. For example: Checking the valid Credit Score, checking and replacing blank values.
    - Processing big combined 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.
    - Following are the screen shots of the code snippets:

Programatically downloads the sample_original and sample_svcg files starting from 2005 to 2017 from freddiemac website.

```
from io import BytesIO
from requests import get
from pathlib import Path
from zipfile import ZipFile
from bs4 import BeautifulSoup

import webbrowser


url = "https://freddiemac.embs.com/FLoan/secure/auth.php"
login = "komalambekar26@hotmail.com"
password = "^<4B[3u7"
url2 = "https://freddiemac.embs.com/FLoan/Data/download.php"

#webbrowser.open(url)
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"})
```

```
<requests.sessions.Session object at 0x000001E2CE32CBE0>
Logging in ....
To the continue page....
Start Downloading from...https://freddiemac.embs.com/FLoan/Data/download.php
https://freddiemac.embs.com/FLoan/Data/download.php?f=sample_2005&s=41073026
https://freddiemac.embs.com/FLoan/Data/download.php?f=sample_2006&s=33286483
https://freddiemac.embs.com/FLoan/Data/download.php?f=sample_2007&s=31029360
https://freddiemac.embs.com/FLoan/Data/download.php?f=sample_2008&s=25243310
https://freddiemac.embs.com/FLoan/Data/download.php?f=sample_2009&s=29742643
https://freddiemac.embs.com/FLoan/Data/download.php?f=sample_2010&s=29502562
https://freddiemac.embs.com/FLoan/Data/download.php?f=sample_2011&s=27895583
https://freddiemac.embs.com/FLoan/Data/download.php?f=sample_2012&s=31533051
https://freddiemac.embs.com/FLoan/Data/download.php?f=sample_2013&s=26626765
https://freddiemac.embs.com/FLoan/Data/download.php?f=sample_2014&s=20459253
https://freddiemac.embs.com/FLoan/Data/download.php?f=sample_2015&s=17376778
https://freddiemac.embs.com/FLoan/Data/download.php?f=sample_2016&s=12007433
https://freddiemac.embs.com/FLoan/Data/download.php?f=sample_2017&s=4621125
Downloaded all sample successfully!
```

In [ ]:

The files are then combined to a .csv file and the datatype for the columns are changed.

In [66]:

```
#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('str')
    dataframe['loan_age'] = dataframe['loan_age'].astype('str')
    dataframe['rem_months'] = dataframe['rem_months'].astype('str')
    dataframe['zero_balance_code'] = dataframe['zero_balance_code'].astype('str')
    dataframe['current_def_upb'] = dataframe['current_def_upb'].astype('str')
    dataframe['actual_loss_calc'] = dataframe['actual_loss_calc'].astype('str')
    return dataframe
```

Data cleaning is done on the columns.

In [67]:

```
#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
```

In [ ]:

0]:
```
def clean_merge_generate_origin_csv():
    print("In a directory: " + os.getcwd())
    OrigFiles=str(os.getcwd() + '\All_Samples') + '\sample_orig_*.txt'
    heading = 0
    filename= "sample_orig_combined.csv"
    path= Path(filename)
    files = glob.glob(OrigFiles)
    if len(files) != 0:
        print("Total %d sample original files" %len(files) )
        if path.is_file():
            print("'sample_orig_combined.csv' already exits!")
        else:
            with open(filename, 'w',encoding='utf-8') as file:
                for f in files:
                    df = pd.read_csv(f ,delimiter ="|", names=['credit_score','first_payment_date','fthb_flag','matr_date','m
                    if heading == 0:
                        df = cleanorigin(df)
                        df.to_csv(file, header=True,index=False, mode='a')
                        heading = 1
                    else:
                        df = cleanorigin(df)
                        df.to_csv(file, header=False, index=False, mode='a')
                print("'sample_orig_combined.csv' generated!" )
    else:
        print("Origination file list is empty!!")
```

- ## Exploratory Data Analysis:

It is an approach to analyze data sets to summarize their main characteristics, often with visual methods. EDA is for seeing what the data can tell us beyond the formal modeling. It is typically the first step of analysis.

Step 1: Read the combined CSV file

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
data = pd.read_csv('.\All_Samples\OriginationCombined.csv', low_memory=False)
```

```
data.head()
```

	cred_scr	fst_paymnt_dte	fst_hmebyr_flg	maturty_dte	metro_stat_area	mort_insur_pctg	nbr_units	occu_status	orig_cmbnd_ln_to_value	orig_dbt_to_incm
0	722	200504	N	203503	0	0	1	P	80	48
1	759	200503	N	203502	0	0	1	P	25	25
2	591	200504	N	203503	39100	0	1	P	48	34
3	792	200503	N	203502	39100	0	1	P	90	33
4	725	200503	N	203502	48864	0	1	P	49	41

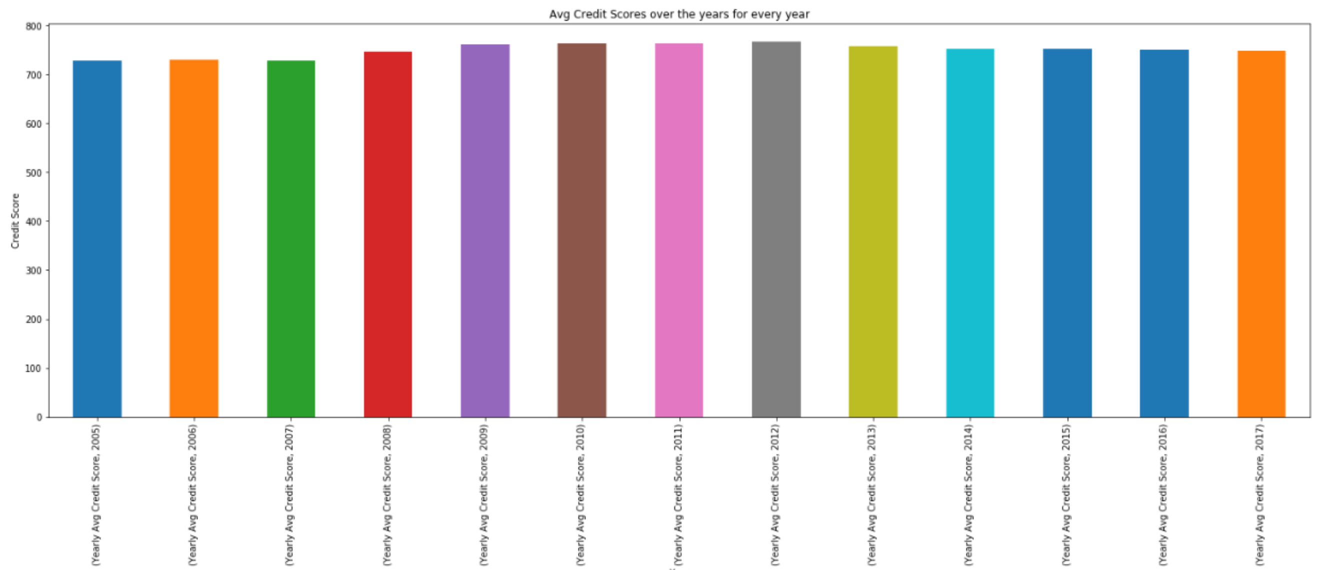
5 rows × 27 columns

Step 2: Plotted graphs to ease data visualizations

- Average Credit score by year

```
#Avg Credit Score by year
year_DF= data.groupby(['Year_Orig'])['cred_scr'].mean().to_frame(name = 'Yearly Avg Credit Score')

year_DF.unstack().plot(title='Avg Credit Scores over the years for every year', kind='bar', stacked=True, figsize=(25,8))

plt.xlabel('Year')
plt.ylabel('Credit Score')
```

```
Text(0,0.5,'Credit Score')
```

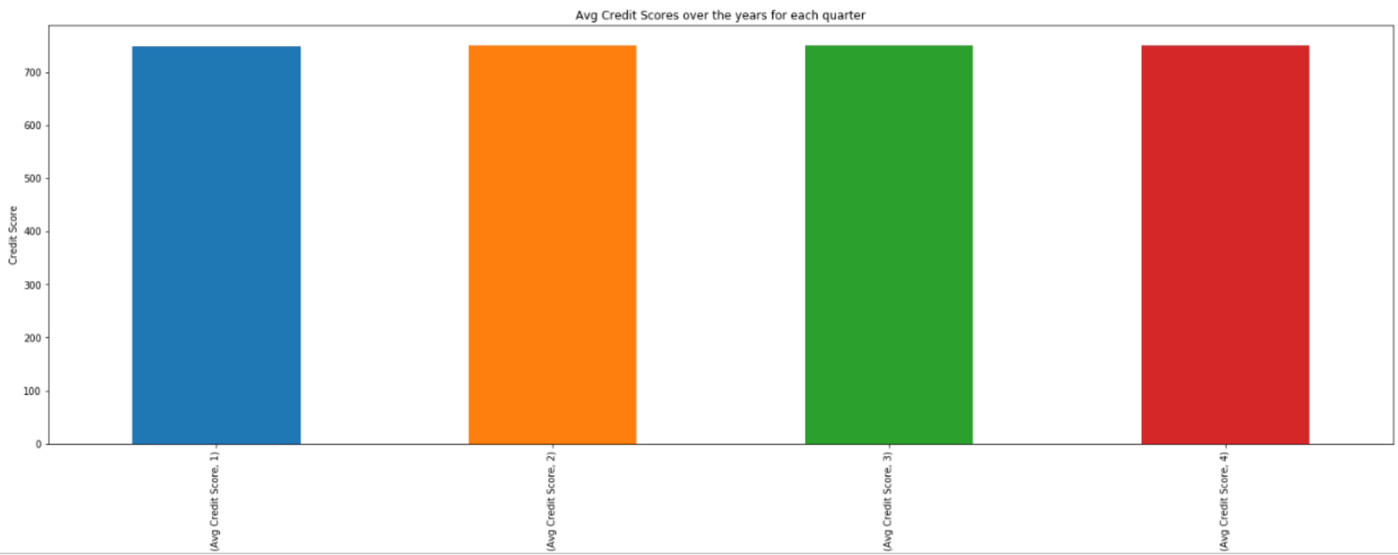


We deduce that credit scores did not have a drastic change in any of the years.

- Adding quarter number for better understanding of EDA

```python
#Adding Quarter number for better understanding of EDA
data['Quarter'] = [x for x in data['ln_sq_nbr'].apply(lambda x: x[5:6])]

Quarter_DF= data.groupby(['Quarter'])['cred_scr'].mean().to_frame(name = 'Avg Credit Score')

Quarter_DF.unstack().plot(title='Avg Credit Scores over the years for each quarter', kind='bar', stacked=True, figsize=(25,8))

plt.xlabel('Quarter')
plt.ylabel('Credit Score')
```

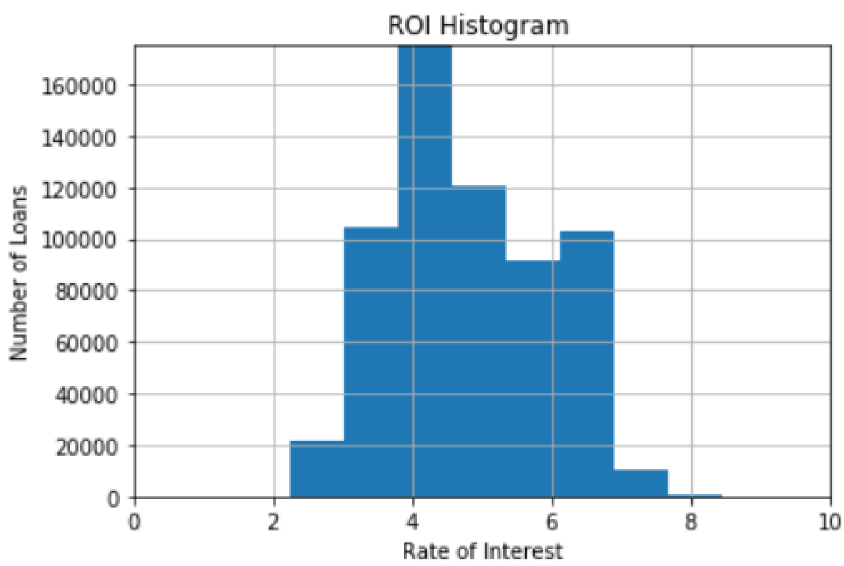
```
Text(0,0.5,'Credit Score')
```



Similarly, credit scores across the quarters also remain same.

- Calculating Rate of Interest against Number of loans using Histogram

```python
n, bins, patches = plt.hist(data.orig_intrst_rate)
plt.xlabel('Rate of Interest')
plt.ylabel('Number of Loans')
plt.title(r'ROI Histogram ')
plt.axis([0, 10, 0, 175000])
plt.grid(True)
plt.show()
```

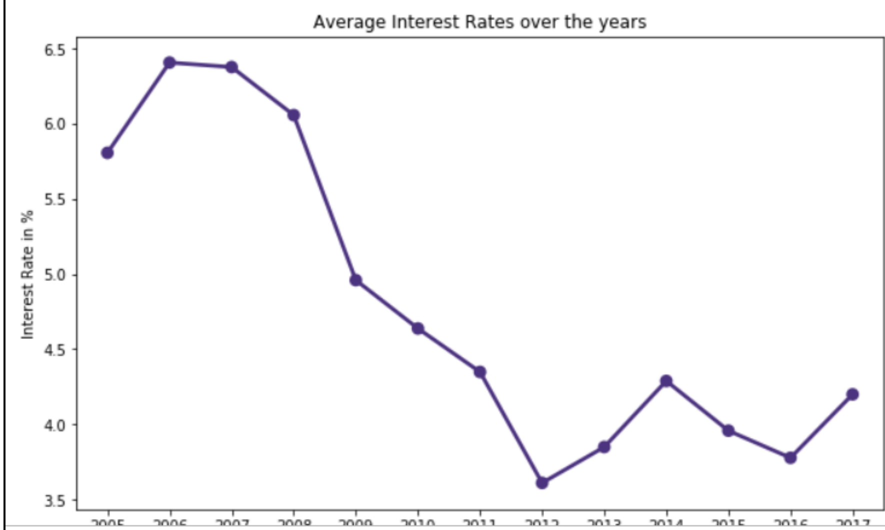


We deduce that most of the interest rates were given out at anywhere around 4-5%

- Plot for Average Interest Rates over the years

```python
interest_df = pd.DataFrame(data.groupby('Year_Orig')['orig_intrst_rate'].mean())
# Creating subplots
fig = plt.figure(figsize=(10,6))

ax1 = fig.add_subplot(1, 1, 1)

# Creating plot for average interest rates
ax1 = sns.pointplot(x=interest_df.index, y=interest_df.orig_intrst_rate, data=interest_df, ax=ax1, color="#4c337f")
ax1.set_title('Average Interest Rates over the years')
ax1.set_xlabel("Year in Account")
ax1.set_ylabel("Interest Rate in %")
```

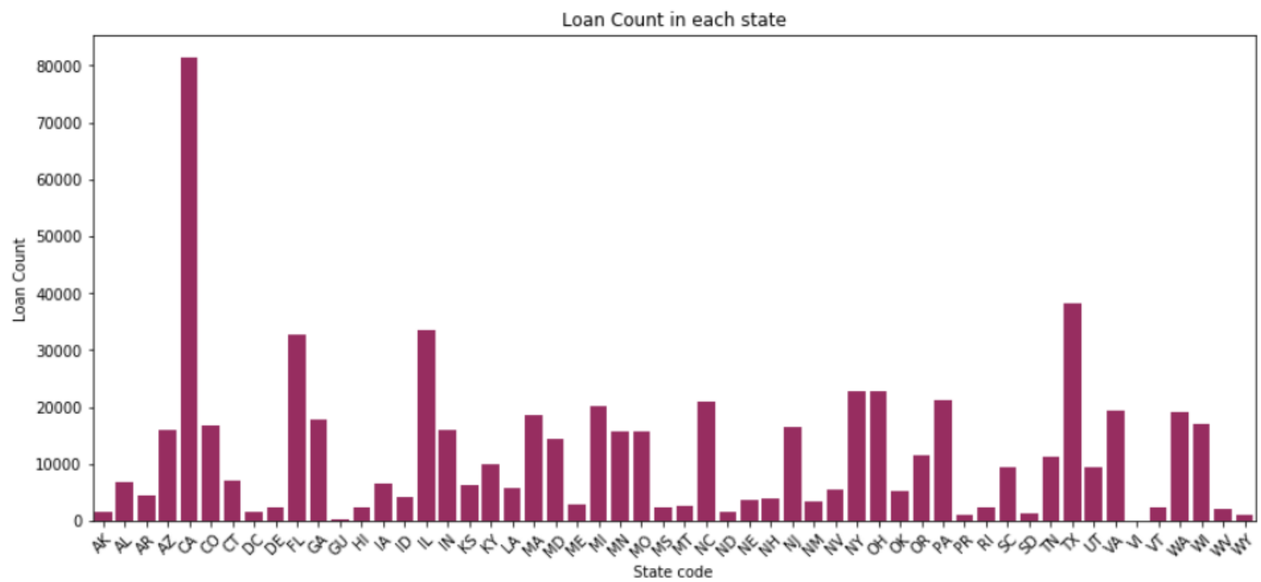
```
Text(0,0.5,'Interest Rate in %')
```



We deduce that rates peaked initially during the first few years but then fell to very low during 2013 and then rose again but not as bad as 2006.

- Plotting graph which states has most loans

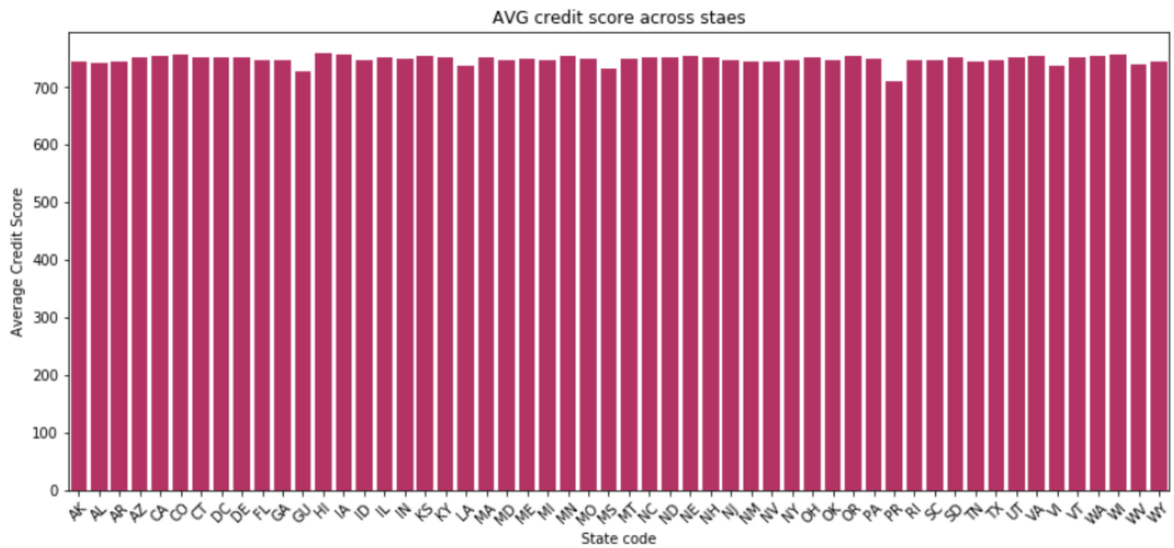
```
In [12]: #Plotting graph to find which states has most loans
         fig = plt.figure(figsize=(14,6))
         ax1 = fig.add_subplot(1, 1, 1)
         ax1 = sns.barplot(x= state_df.propstate, y= state_df.ln_sq_nbr, data=state_df, ax=ax1, color="#a81c5f")
         ax1.set_title('Loan Count in each state')
         ax1.set_xlabel("State code")
         ax1.set_ylabel("Loan Count")
         plt.xticks(rotation=45)

Out[12]: (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
                 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
                 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
                 51, 52, 53]), <a list of 54 Text xticklabel objects>)
```



We noticed that the number loans in California were the highest.

- Average credit scores across states

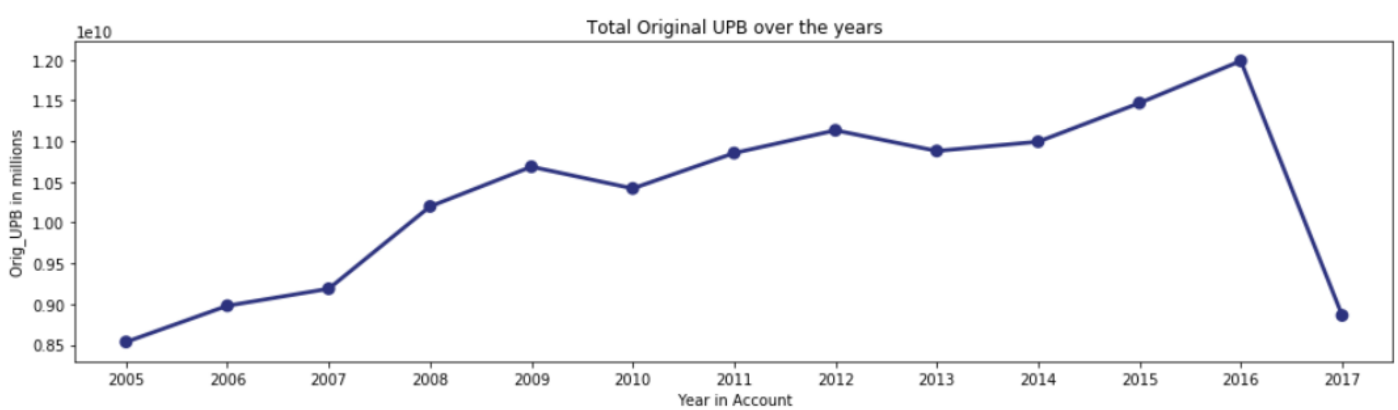
```
In [48]: state_df = data.groupby(['propstate'])['cred_scr'].mean().reset_index()
         fig = plt.figure(figsize=(14,6))
         ax1 = fig.add_subplot(1, 1, 1)
         ax1 = sns.barplot(x= state_df.propstate, y= state_df.cred_scr, data=state_df, ax=ax1, color="#c81f5f")
         ax1.set_title('AVG credit score across staes')
         ax1.set_xlabel("State code")
         ax1.set_ylabel("Average Credit Score")
         plt.xticks(rotation=45)

Out[48]: (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
                 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
                 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
                 51, 52, 53]), <a list of 54 Text xticklabel objects>)
```



We noticed that credit scores in Georgia and Philadelphia are states where the credit scores were slightly less compared to other states.

- Total original UPB over the years

```
prop_df = pd.DataFrame(data.groupby(['Year_Orig'])['orig_upb'].sum())
fig = plt.figure(figsize=(16,4))
ax1 = fig.add_subplot(1, 1, 1)
ax1 = sns.pointplot(x=prop_df.index, y=prop_df.orig_upb, data=prop_df, ax=ax1, color="#2c327e")
ax1.set_title('Total Original UPB over the years')
ax1.set_xlabel("Year in Account")
ax1.set_ylabel("Orig_UPB in millions")

Text(0,0.5,'Orig_UPB in millions')
```



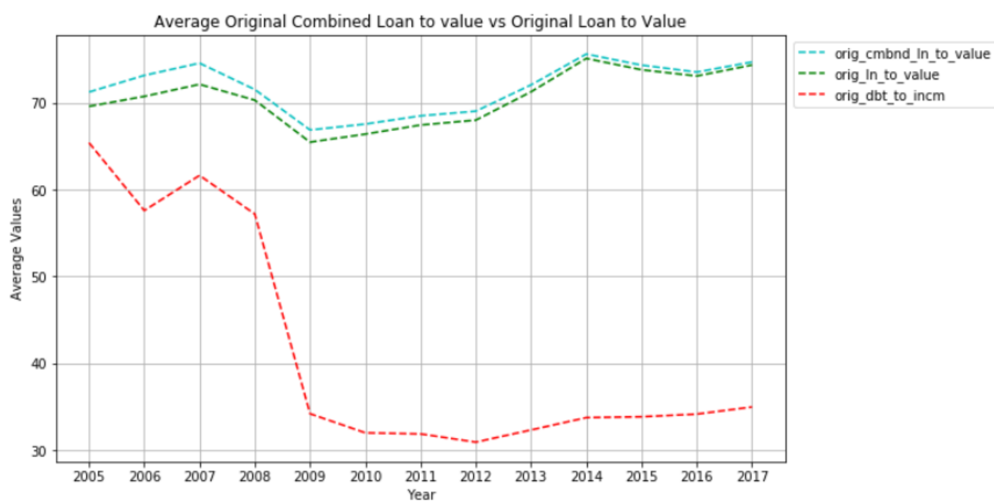
- Average original combined loan vs original loan to value

In [ ]:

In [21]:
```
loantovalue_df = data.groupby('Year_Orig')['orig_cmbnd_ln_to_value', 'orig_ln_to_value', \
                                           'orig_dbt_to_incm'].mean()

fig = plt.figure(figsize=(10,6))
ax1 = fig.add_subplot(111)

ax1.plot(loantovalue_df.index, loantovalue_df.orig_cmbnd_ln_to_value, label='orig_cmbnd_ln_to_value', color='c',  linestyle='
ax1.plot(loantovalue_df.index, loantovalue_df.orig_ln_to_value, label='orig_ln_to_value', color='g', linestyle='--')
ax1.plot(loantovalue_df.index, loantovalue_df.orig_dbt_to_incm, label='orig_dbt_to_incm', color='r', linestyle='--')

ax1.set_title('Average Original Combined Loan to value vs Original Loan to Value')
plt.xticks(loantovalue_df.index)
plt.xlabel('Year')
plt.ylabel('Average Values')

handles, labels = ax1.get_legend_handles_labels()
lgd = ax1.legend(handles, labels, loc='upper center', bbox_to_anchor=(1.15,1))
ax1.grid('on')
plt.show()
```

```
C:\Users\rishi\Anaconda3\lib\site-packages\matplotlib\cbook\deprecation.py:107: MatplotlibDeprecationWarning: Passing one of
'on', 'true', 'off', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
```


C:\Users\rishi\Anaconda3\lib\site-packages\matplotlib\cbook\deprecation.py:107: MatplotlibDeprecationWarning: Passing one of
'on', 'true', 'off', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)



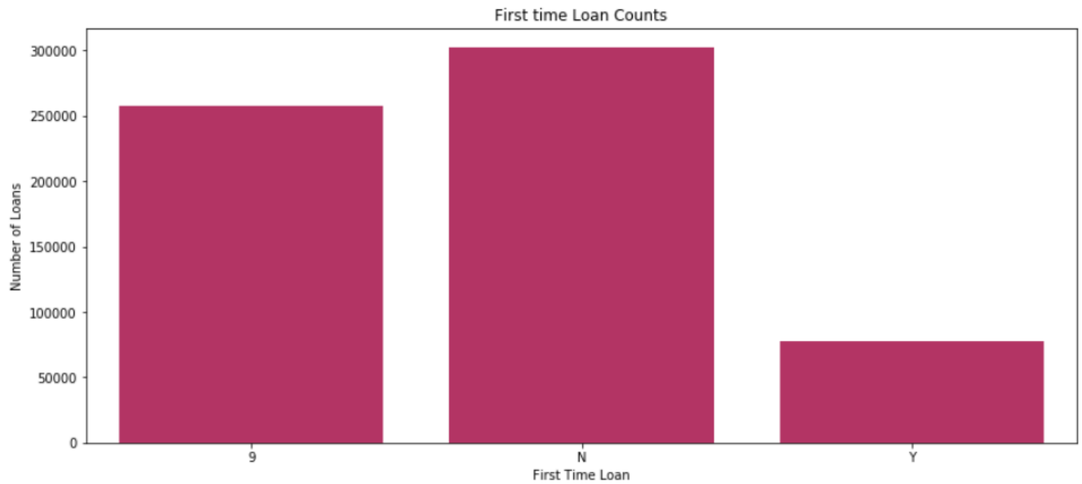
We notice that the debt to income ratio fell drastically in 2009 .

- First time loan counts

Year

```
In [37]: firstloan_df = pd.DataFrame(data.groupby('fst_hmebyr_flg')['ln_sq_nbr'].count())
```

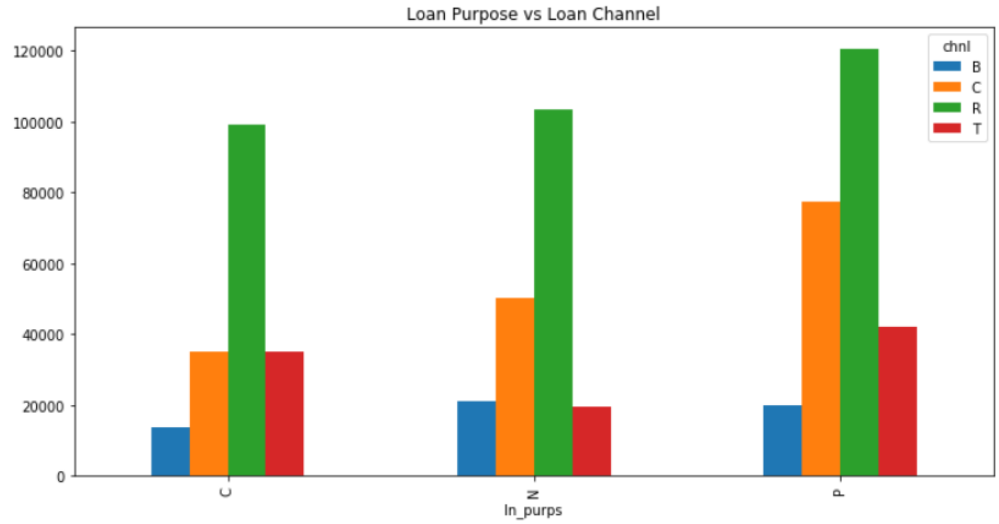
```
In [39]: fig = plt.figure(figsize=(14,6))
ax1 = fig.add_subplot(1, 1, 1)
ax1 = sns.barplot(x= firstloan_df.index, y= firstloan_df.ln_sq_nbr, data=firstloan_df, ax=ax1, color="#c81f5f")
ax1.set_title('First time Loan Counts')
ax1.set_xlabel("First Time Loan")
ax1.set_ylabel("Number of Loans")

Out[39]: Text(0,0.5,'Number of Loans')
```

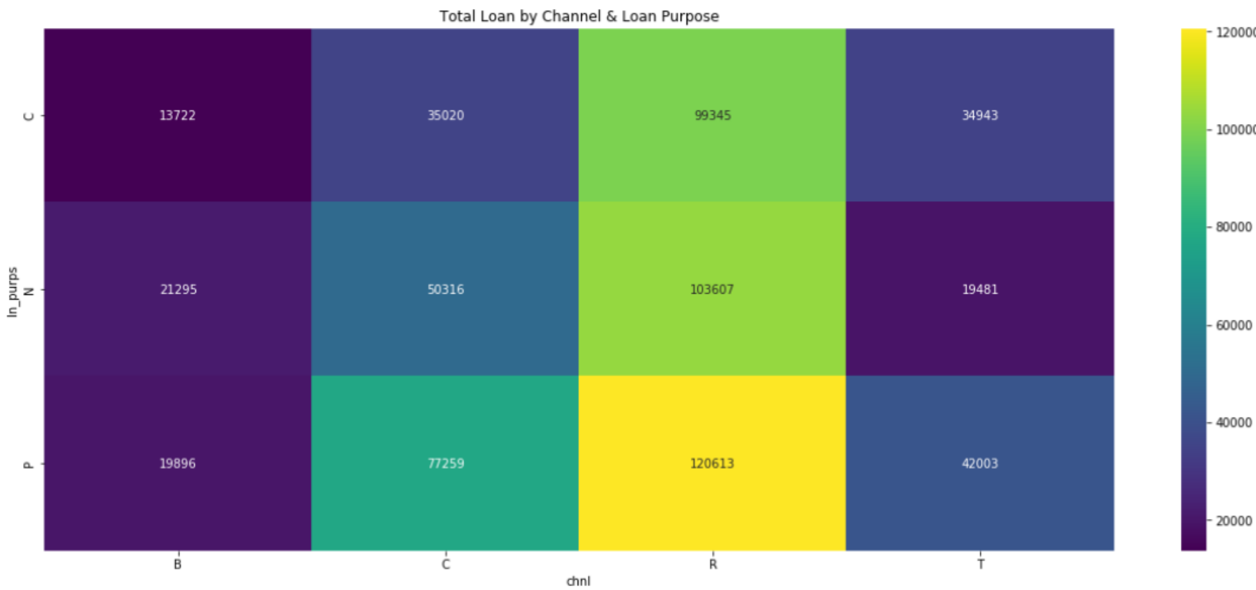


- Loan purpose vs loan channel

First Time Loan

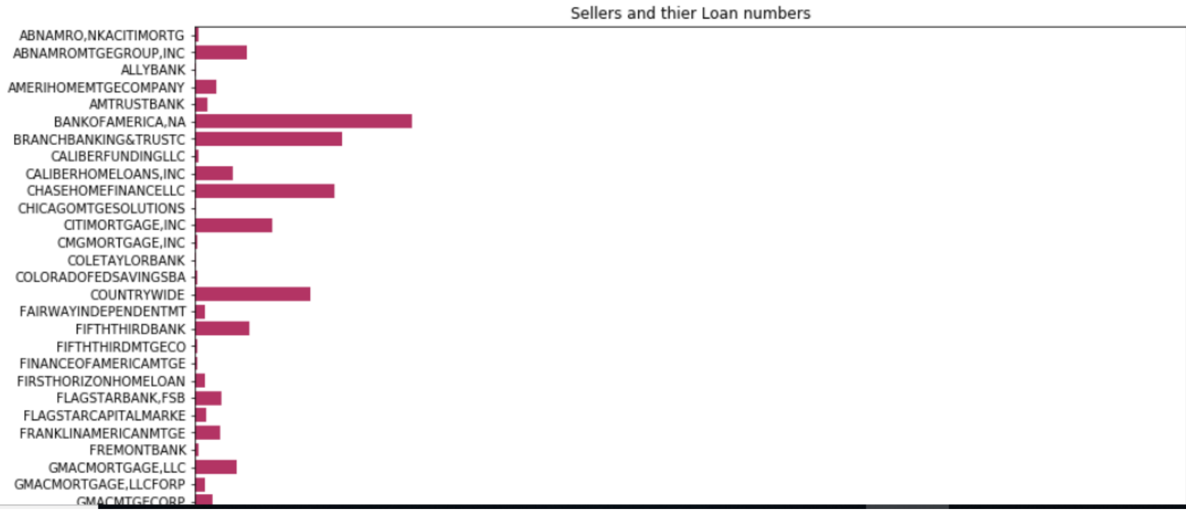
```
In [40]: loan_pur_df = data.groupby(['ln_purps', 'chnl']).size().unstack()
loan_pur_df.plot(title='Loan Purpose vs Loan Channel', kind='bar', stacked=False, figsize=(12,6))
plt.figure(figsize=(20, 8))
plt.title('Total Loan by Channel & Loan Purpose')
sns.heatmap(loan_pur_df, annot=True, fmt="g", cmap='viridis')
plt.show()
```





- Sellers and their loan numbers

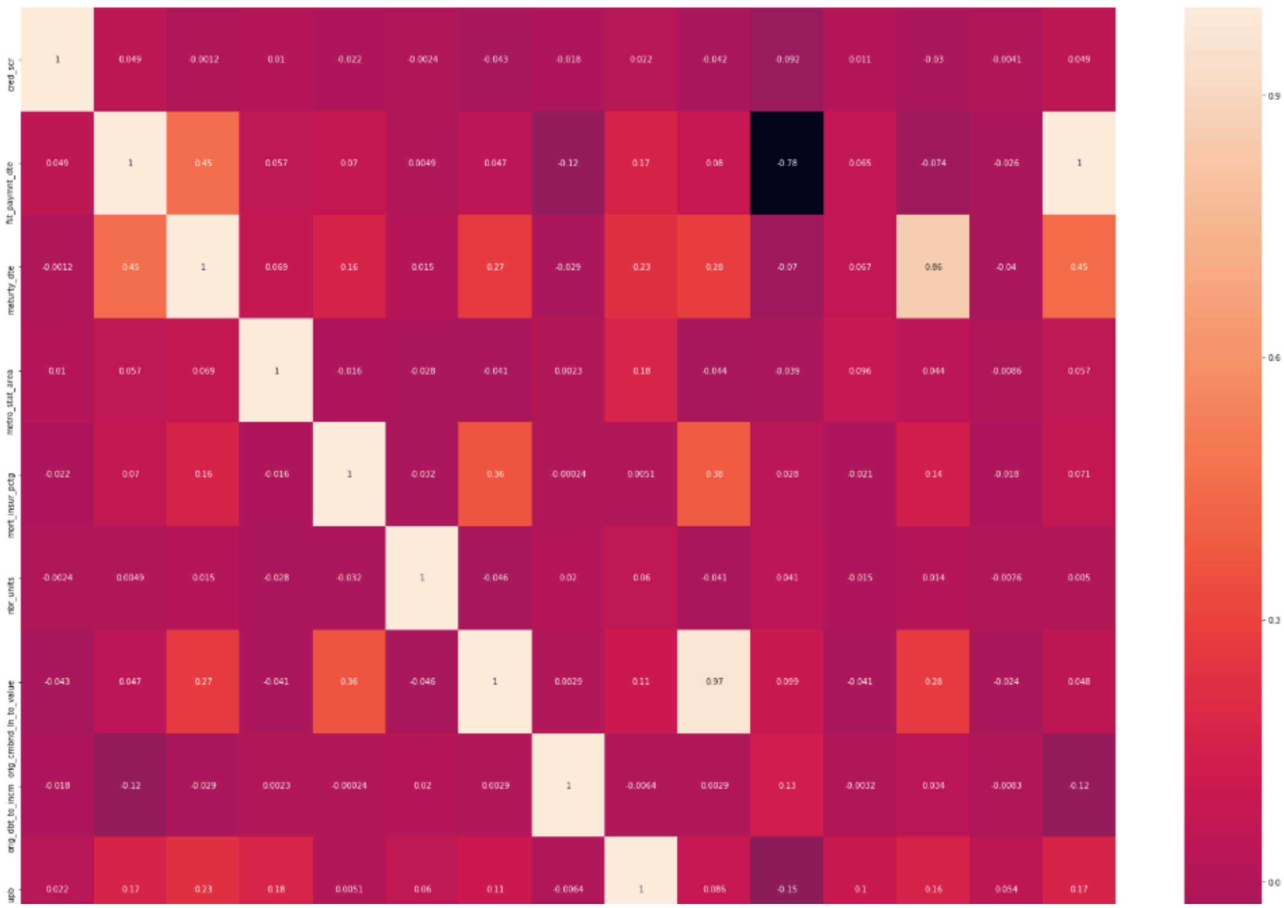
```
In [47]: seller_df = pd.DataFrame(data.groupby('slr_name')['ln_sq_nbr'].count())
         fig = plt.figure(figsize=(14,16))
         ax1 = fig.add_subplot(1, 1, 1)
         ax1 = sns.barplot(y= seller_df.index, x= seller_df.ln_sq_nbr, data=seller_df, ax=ax1, color="#c81f5f")
         ax1.set_title('Sellers and thier Loan numbers')
         ax1.set_xlabel("Number of Loans")
         ax1.set_ylabel("Loan Provider")

Out[47]: Text(0,0.5,'Loan Provider')
```



➢ Correlation matrix

```
In [19]: corr = data.corr()
         f, ax = plt.subplots(figsize=(30,35))
         cmap = sns.diverging_palette(220, 10, as_cmap=True)
         sns.heatmap(corr,annot = True)

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x2d444450160>
```

```
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x2d444456106>
```



➢ Prediction:

- Here we will use the historical origination data to predict the interest rate for quarters.
- Here we will use Q1 2005 as a training data and we will predict the values for Q22005 quarter.
- We will calculate and evaluate different algorithms based on below parameters:

- **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

- The **mean absolute percentage error** (**MAPE**), also known as mean absolute percentage deviation (MAPD), is a measure of prediction accuracy of a forecasting method in statistics, for example in trend estimation.

1. Building Model

   a. Clean Data

Function to clean Data

```
In [6]: def cleaningdata(data):
    #Dropping Credit scores above 850 and less than 301
    data=data.drop(data['credit_score'].loc[(data['credit_score'] < 301) | (data['credit_score'] > 850)].i
ndex)
    data=data.dropna(subset=['credit_score'])
    data=data.dropna(subset=['first_payment_date'])
    data['fthb_flag'] = data['fthb_flag'].fillna("NA")
    #Dropping not applicable MSA data
    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 Data Into Numbers

Function to convert Data into Numbers

```
In [7]: def convertnumbersdata(data):
    data['fthb_flag'] = data['fthb_flag'].replace(['Y','N','NA'],[1,2,3])
    data['occupancy_status'] = data['occupancy_status'].replace(['I','O','S','P'],[1,2,3,4])
    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','99'],[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
```

c. Change Data Type to Integer

Function to Change data type to integer for linear regression

```
In [8]: def changedatatype(data):
    data[['credit_score','msa','no_of_units','mortage_insurance_pct','cltv','dti_ratio','original_ltv','zi
pcode','number_of_borrowers']]=data[['credit_score','msa','no_of_units','cltv','mortage_insurance_pct','dt
i_ratio','original_ltv','zipcode','number_of_borrowers']].astype('int64')
    data[['fthb_flag','occupancy_status','channel']] = data[['fthb_flag','occupancy_status','channel']].as
type('int64')
    data[['ppm_flag','prop_type','loan_purpose','super_conforming_flag']]= data[['ppm_flag','prop_type','l
oan_purpose','super_conforming_flag']].astype('int64')
    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','ser
vicer_name']].astype('str')
    return data
```

d. Perform Linear Regression

```
In [118]: def linearRegression(x1,y1,x2,y2):
    regressor = LinearRegression()
    regressor.fit(x1,y1)
    y_pred_train = regressor.predict(x1)
    y_pred_test = regressor.predict(x2)

    print('\nTraining Data')
    print('Score:',regressor.score(x1,y1))
    MAE = mean_absolute_error(y1,y_pred_train)
    print('MAE of Training Data =', MAE)
    ## Mean squared error
    MSE = mean_squared_error(y1,y_pred_train)
    RMSE = math.sqrt(MSE)
    print('RMSE of Training Data =',RMSE)
    ## R-square score of this model
    R2 = r2_score(y1,y_pred_train)
    print('R2 of Training Data =',R2)
    ## MAPE of this model
    MAPE=mean_absolute_percentage_error(y1,y_pred_train)
    print('MAPE of Training Data =',MAPE,'\n')

    print('\nTesting Data')
    print('Score:',regressor.score(x2,y2))
    MAE = mean_absolute_error(y2,y_pred_test)
    print('MAE of Training Data =', MAE)
    ## Mean squared error
    MSE = mean_squared_error(y2,y_pred_test)
    RMSE = math.sqrt(MSE)
    print('RMSE of Training Data =',RMSE)
    ## R-square score of this model
    R2 = r2_score(y2,y_pred_test)
    print('R2 of Training Data =',R2)
    ## MAPE of this model
    MAPE=mean_absolute_percentage_error(y2,y_pred_test)
    print('MAPE of Training Data =',MAPE)
```

e. Perform Random Forest Regressor

In [163]:

```
from sklearn.ensemble import RandomForestRegressor
def RandomForestRegression(x1,y1,x2,y2):
    rfc = RandomForestRegressor(n_estimators=50,random_state=np.random)
    rfc.fit(x1,y1)
    y_pred_train = rfc.predict(x1)
    y_pred_test = rfc.predict(x2)

    print('\nTraining Data')
    print('\nScore',rfc.score(x1,y1))
    MAE = mean_absolute_error(y1,y_pred_train)
    print('MAE of Training Data =', MAE)
    ## Mean squared error
    MSE = mean_squared_error(y1,y_pred_train)
    RMSE = math.sqrt(MSE)
    print('RMSE of Training Data =',RMSE)
    ## R-square score of this model
    R2 = r2_score(y1,y_pred_train)
    print('R2 of Training Data =',R2)
    ## MAPE of this model
    MAPE=mean_absolute_percentage_error(y1,y_pred_train)
    print('MAPE of Training Data =',MAPE)

    print('\n Testing Data')
    print('Score',rfc.score(x2,y2))
    MAE = mean_absolute_error(y2,y_pred_test)
    print('MAE of Training Data =', MAE)
    ## Mean squared error
    MSE = mean_squared_error(y2,y_pred_test)
    RMSE = math.sqrt(MSE)
    print('RMSE of Training Data =',RMSE)
    ## R-square score of this model
    R2 = r2_score(y2,y_pred_test)
    print('R2 of Training Data =',R2)
    ## MAPE of this model
    MAPE=mean_absolute_percentage_error(y2,y_pred_test)
    print('MAPE of Training Data =',MAPE)
```

f. Perform Neural Network

In [122]:

```
def neuralnetworks(x1,y1,x2,y2):
    neuralNetwork = MLPRegressor(hidden_layer_sizes=(15,15,15))
    neuralNetwork.fit(x1,y1)
    y_pred_train = neuralNetwork.predict(x1)
    y_pred_test = neuralNetwork.predict(x2)

    print('\nTraining Data')
    print('Score',neuralNetwork.score(x1,y1))
    MAE = mean_absolute_error(y1,y_pred_train)
    print('MAE of Training Data =', MAE)
    ## Mean squared error
    MSE = mean_squared_error(y1,y_pred_train)
    RMSE = math.sqrt(MSE)
    print('RMSE of Training Data =',RMSE)
    ## R-square score of this model
    R2 = r2_score(y1,y_pred_train)
    print('R2 of Training Data =',R2)
    ## MAPE of this model
    MAPE=mean_absolute_percentage_error(y1,y_pred_train)
    print('MAPE of Training Data =',MAPE)

    print('\nTesting Data')
    print('Score: ',neuralNetwork.score(x2,y2))
    MAE = mean_absolute_error(y2,y_pred_test)
    print('MAE of Training Data =', MAE)
    ## Mean squared error
    MSE = mean_squared_error(y2,y_pred_test)
    RMSE = math.sqrt(MSE)
    print('RMSE of Training Data =',RMSE)
    ## R-square score of this model
    R2 = r2_score(y2,y_pred_test)
    print('R2 of Training Data =',R2)
    ## MAPE of this model
    MAPE=mean_absolute_percentage_error(y2,y_pred_test)
    print('MAPE of Training Data =',MAPE)
```

```
Testing Data
Score: 0.15267794422249192
MAE of Training Data = 0.24428970743232073
RMSE of Training Data = 0.318664522425998
R2 of Training Data = 0.15267794422249192
MAPE of Training Data = 99.18266484244911
```

In [126]:
```
RandomForestRegression(x1,y1,x2,y2)
```

```
Training Data

Score 0.9218957303180438
MAE of Training Data = 0.07514743491903093
RMSE of Training Data = 0.10141110859760777
R2 of Training Data = 0.9218957303180438
MAPE of Training Data = 100.04384400037725

 Testing Data
Score 0.2081089446503399
MAE of Training Data = 0.23777447041809666
RMSE of Training Data = 0.30806487126844917
R2 of Training Data = 0.2081089446503399
MAPE of Training Data = 100.11708185075616
```

In [123]:
```
neuralnetworks(x1,y1,x2,y2)
```

```
Training Data
Score -2838.7491458511613
MAE of Training Data = 18.75910829067956
RMSE of Training Data = 19.33695698718669
R2 of Training Data = -2838.7491458511613
MAPE of Training Data = 166.50619658433507

Testing Data
Score:  -3184.5549780756583
MAE of Training Data = 18.987486471990255
RMSE of Training Data = 19.53898556491105
R2 of Training Data = -3184.5549780756583
MAPE of Training Data = 164.76354734501652
```

g. Feature Selection using Stepwise

```
In [81]: import statsmodels.api as sm
def stepwise_selection(X1, y1,
                       initial_list=[],
                       threshold_in=0.01,
                       threshold_out = 0.05,
                       verbose=True):

    included = list(initial_list)
    while True:
        changed=False
        # forward step
        excluded = list(set(X1.columns)-set(included))
        new_pval = pd.Series(index=excluded)
        for new_column in excluded:
            model = sm.OLS(y1, sm.add_constant(pd.DataFrame(X1[included+[new_column]]))).fit()
            new_pval[new_column] = model.pvalues[new_column]
        best_pval = new_pval.min()
        if best_pval < threshold_in:
            best_feature = new_pval.argmin()
            included.append(best_feature)
            changed=True
            if verbose:
                print('Add  {:30} with p-value {:.6}'.format(best_feature, best_pval))

        # backward step
        model = sm.OLS(y1, sm.add_constant(pd.DataFrame(X1[included]))).fit()
        # use all coefs except intercept
        pvalues = model.pvalues.iloc[1:]
        worst_pval = pvalues.max() # null if pvalues is empty
        if worst_pval > threshold_out:
            changed=True
            worst_feature = pvalues.argmax()
            included.remove(worst_feature)
            if verbose:
                print('Drop {:30} with p-value {:.6}'.format(worst_feature, worst_pval))
        if not changed:
            break
    return included
```

h. Feature Selection using RFE

```
In [100]: def perform_recursiveFE(xaxis,yaxis):
    lr = LinearRegression()
    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)
    lis = rankfeatures.loc[rankfeatures['ranking'] == 1]
    return lis['features'].values
```

In [151]:
```python
x1 = data_train.copy(deep = True)
y1 = x1.original_int_rt
x1 . drop(['loan_seq_number','product_type','property_state','sellers_name','servicer_name', 'original_int
_rt',], axis =1 , inplace = True)
```

In [152]:
```python
result = stepwise_selection(x1,y1)
print (result)
```

```
C:\Users\rishi\Anaconda3\lib\site-packages\ipykernel_launcher.py:30: FutureWarning: 'argmin' is deprecate
d, use 'idxmin' instead. The behavior of 'argmin'
will be corrected to return the positional minimum in the future.
Use 'series.values.argmin' to get the position of the minimum now.
```

```
Add  credit_score                   with p-value 0.0
Add  original_ltv                   with p-value 0.0
Add  original_upb                   with p-value 0.0
Add  original_loan_term             with p-value 0.0
Add  cltv                           with p-value 0.0
Add  matr_date                      with p-value 0.0
Add  super_conforming_flag          with p-value 0.0
Add  loan_purpose                   with p-value 0.0
Add  occupancy_status               with p-value 0.0
Add  first_payment_date             with p-value 0.0
Add  zipcode                        with p-value 2.99006e-114
Add  prop_type                      with p-value 5.72251e-58
Add  ppm_flag                       with p-value 2.52127e-56
Add  channel                        with p-value 2.73204e-56
Add  fthb_flag                      with p-value 2.96895e-65
Add  mortage_insurance_pct          with p-value 3.2769e-40
Add  number_of_borrowers            with p-value 1.58178e-14
Add  no_of_units                    with p-value 1.13671e-06
Add  msa                            with p-value 0.00701658
['credit_score', 'original_ltv', 'original_upb', 'original_loan_term', 'cltv', 'matr_date', 'super_conform
ing_flag', 'loan_purpose', 'occupancy_status', 'first_payment_date', 'zipcode', 'prop_type', 'ppm_flag',
'channel', 'fthb_flag', 'mortage_insurance_pct', 'number_of_borrowers', 'no_of_units', 'msa']
```

In [104]:
```python
lis = perform_recursiveFE(x1,y1)
print(lis)
```

```
RFE r2 score:  0.33164643323586684
                 features  ranking
0            credit_score        1
```

In [104]:
```python
lis = perform_recursiveFE(x1,y1)
print(lis)
```

```
RFE r2 score:  0.33164643323586684
                 features  ranking
0            credit_score        1
1      first_payment_date        1
2               fthb_flag        1
3               matr_date        1
4                     msa        1
5   mortage_insurance_pct        1
6             no_of_units        1
7        occupancy_status        1
8                    cltv        1
9               dti_ratio        1
10           original_upb        2
11           original_ltv        3
12                channel        4
13               ppm_flag        5
14              prop_type        6
15                zipcode        7
16           loan_purpose        8
17     original_loan_term        9
18    number_of_borrowers       10
19  super_conforming_flag       11
['credit_score' 'first_payment_date' 'fthb_flag' 'matr_date' 'msa'
 'mortage_insurance_pct' 'no_of_units' 'occupancy_status' 'cltv'
 'dti_ratio']
```

```
ati_ratio ]
```

```
In [153]: stepwise_data_train= x1[result]
          rfe_data_train = x1[lis]
```

```
In [154]: x2 = data_test.copy(deep = True)
          y2 = x2.original_int_rt
          stepwise_data_est = x2[result]
          rfe_data = x2[lis]
```

```
In [134]: linearRegression(stepwise_data_train, y1, stepwise_data_est,y2)
```

```
Training Data
Score: 0.38608112429318964
MAE of Training Data = 0.21067809832173592
RMSE of Training Data = 0.28431752875494376
R2 of Training Data = 0.38608112429318964
MAPE of Training Data = 100.12489128928084


Testing Data
Score: 0.1539055593141737
MAE of Training Data = 0.24399943675945607
RMSE of Training Data = 0.31843359537152577
R2 of Training Data = 0.1539055593141737
MAPE of Training Data = 99.18236412891345
```

```
In [135]: linearRegression(rfe_data_train, y1, rfe_data,y2)
```

```
Training Data
Score: 0.3381300989541415
MAE of Training Data = 0.22065949208642552
RMSE of Training Data = 0.2952123067163569
R2 of Training Data = 0.3381300989541415
MAPE of Training Data = 100.13487432019272


Testing Data
```

From above three 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.

- H2O.AI

jupyter H20 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run | Code | Show running Spark jobs

H2O API Extensions:	Algos, AutoML, Core V3, Core V4
Python version:	3.7.0 final

In [41]:

```
df = h2o.import_file('./train_Q12005.csv')
```

```
Parse progress: |██████████████████████████████████████████████████████████| 100%
```

In [42]:

```
df.describe()
```

```
Rows:351634
Cols:26
```

	credit_score	first_payment_date	fthb_flag	matr_date	msa	mortage_insurance_pct	no_of_units	occupa
type	int	int	enum	int	int	int	int	
mins	300.0	200001.0		201002.0	10180.0	0.0	1.0	
mean	728.15774356291	200504.72467110673		203110.87020026497	30555.53146320928	3.552696838189709	1.030398084371818	
maxs	9999.0	201509.0		205109.0	49740.0	999.0	99.0	
sigma	234.93710185479145	10.618324353022441		651.0857095038862	11364.642833370082	16.490476286362334	0.39454088326869285	
zeros	0	0		0	0	302104	0	
missing	0	0	0	0	54684	0	0	
0	699.0	200505.0	N	203504.0	39300.0	0.0	1.0	
1	691.0	200504.0	N	203503.0	36420.0	25.0	1.0	
2	713.0	200503.0	N	203502.0	28740.0	0.0	1.0	
3	719.0	200505.0	N	203504.0	nan	0.0	1.0	
4	656.0	200503.0	N	203502.0	40340.0	0.0	1.0	
5	641.0	200504.0	N	203503.0	19500.0	30.0	1.0	

jupyter H20 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Run Code Show running Spark jobs

7	586.0	200503.0	N	203502.0	28740.0	0.0	1.0
8	582.0	200503.0	N	203502.0	nan	0.0	1.0
9	720.0	200503.0	N	203502.0	36500.0	30.0	1.0

In [47]:
```
y= 'original_int_rt'
```

In [44]:
```
aml = H2OAutoML(max_runtime_secs = 6000, seed = 1)
```

In [45]:
```
splits = df.split_frame(ratios = [0.8], seed = 1)
train = splits[0]
test = splits[1]
```

In [48]:
```
aml.train(y = y, training_frame = train, leaderboard_frame =test )
```

```
AutoML progress: |████████████████████████████████████████████████████████| 100%| 100%
```

In [49]:
```
aml.leaderboard
```

model_id	mean_residual_deviance	rmse	mse	mae	rmsle
StackedEnsemble_AllModels_AutoML_20181128_184249	0.0589145	0.242723	0.0589145	0.179068	0.0362195
StackedEnsemble_BestOfFamily_AutoML_20181128_184249	0.0594723	0.243869	0.0594723	0.179735	0.0363882
GBM_4_AutoML_20181128_184249	0.0597037	0.244343	0.0597037	0.180205	0.0364556
GBM_5_AutoML_20181128_184249	0.0598676	0.244679	0.0598676	0.180461	0.0365096
GBM_3_AutoML_20181128_184249	0.0600509	0.245053	0.0600509	0.180747	0.0365571
GBM_2_AutoML_20181128_184249	0.0602627	0.245485	0.0602627	0.181252	0.0366201
GBM_grid_1_AutoML_20181128_184249_model_6	0.0602709	0.245501	0.0602709	0.181097	0.0366342
GBM_grid_1_AutoML_20181128_184249_model_5	0.0602904	0.245541	0.0602904	0.181585	0.0366414
GBM_1_AutoML_20181128_184249	0.060748	0.246471	0.060748	0.182013	0.0367763
GBM_grid_1_AutoML_20181128_184249_model_1	0.0619842	0.248966	0.0619842	0.184051	0.0371497

Out[49]:

In [50]:
```
perf = aml.leader.model_performance(test)
perf
```

```
ModelMetricsRegressionGLM: stackedensemble
** Reported on test data. **

MSE: 0.05891454988282391
RMSE: 0.2427231960131209
MAE: 0.17906808363506316
RMSLE: 0.036219535047174754
R^2: 0.5603185758078277
Mean Residual Deviance: 0.05891454988282391
Null degrees of freedom: 70133
Residual degrees of freedom: 70123
Null deviance: 9398.106447521634
Residual deviance: 4131.913041481972
AIC: 459.5233347959999
```

Out[50]:

In [ ]:

- TPOT:

jupyter TPOT Last Checkpoint: 19 hours ago (unsaved changes)

```
In [10]: y2 = data_test.original_int_rt
```

```
In [11]: x2 = data_test
```

```
In [12]: x2.drop(['loan_seq_number','property_state','sellers_name','servicer_name','super_conforming_flag' ,'product_type', 'original
```

```
In [ ]: tpot = TPOTRegressor(generations=5, population_size=50, verbosity=2)
        tpot.fit(x1, y1)
        print(tpot.score(x2, y2))
```

```
C:\Users\Komal\Anaconda3\lib\importlib\_bootstrap.py:219: ImportWarning: can't resolve package from __spec__ or __package__,
falling back on __name__ and __path__
  return f(*args, **kwds)
C:\Users\Komal\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.py:29: DeprecationWarning: numpy.core.umath_test
s is an internal NumPy module and should not be imported. It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d

Warning: xgboost.XGBRegressor is not available and will not be used by TPOT.
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
Generation 1 - Current best internal CV score: -0.08029201091216487
Generation 2 - Current best internal CV score: -0.07384083598147335
```

```
In [ ]:
```

- ## Classification:

The main goal of classification is to predict the target class (Yes/ No). If the trained model is for predicting any of two target classes. It is known as binary classification. Here we are predicted the derived column Delinquent which is the target class.

We Programmatically downloaded files from the freddiemac website. The input is parameterized. The user provides two inputs one for test data and the other for train data. We have built four models namely: Random Forest, Neural Network, SVN and Logistic Regression.

- ## Programmatically downloading the historical data based on user input:

```
    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')
    flag = 0
    flag = downloadhistoricaldata(trainQ, testQ, t,s, flag)

    if flag == 1:
        print("Data downloaded successfully!!")
    else:
        print("Error in downloading data")
```

In [7]:

```
login('rishi.r.rajani@gmail.com','jQcQFxI=','Q12005','Q22005')
```

```
Logging in....
To the continue page...
Start Downloading from...https://freddiemac.embs.com/FLoan/Data/download.php
Data downloaded successfully!!
```

- <u>Cleaning the dataframe:</u>

```
In [13]: 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)
```

```
In [10]: train_data_copy =train_data.copy(deep = True)
```

```
In [11]: test_data_copy =test_data.copy(deep = True)
```

Adding delinquent column:

‘Deliquent’ column is added based on delq_status. If delq_status is greater than 0 then Deliquent = 1 else the value of Deliquent is 0.

```
In [50]: def statusDeliquent(row):
                 if row['delq_status'] > 0:
                     val = 1
                 else:
                     val = 0
                 return val
```

```
In [54]: train_data_copy['Deliquent'] = train_data_copy.apply(statusDeliquent, axis=1)
```

```
In [55]: test_data_copy['Deliquent'] = test_data_copy.apply(statusDeliquent, axis=1)
```

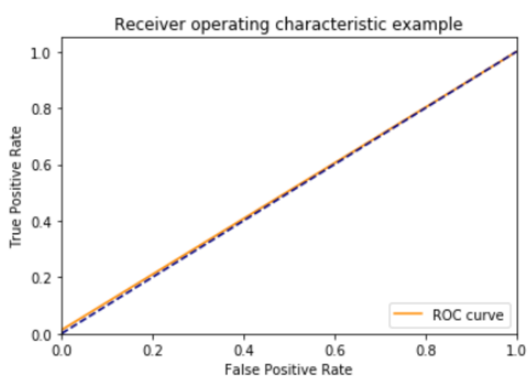
```
In [59]: train_data_copy.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26508106 entries, 0 to 26508105
Data columns (total 27 columns):
loan_number            object
year-month             int64
current_actual_upb     float64
delq_status            float64
loan_age               int64
rem_months             category
repurchase_flag        category
modification_flag      category
zero_balance_code      category
zero_bal_date          category
current_int_rate       float64
current_def_upb        category
ddlpi                  category
mi_recoveries          float64
net_sales_proceeds     object
non_mi_recoveries      float64
```

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 [109]:  logistic_regressor(train_features, y, test_features,y2)

Training results

----------------------Confusion Matrix--------------------
                    Actual Result
                   0                  1
Expected  0         25364890          32

result    1         1129456          13728

Accuracy: 95.73908448985378
```

```
Testing results

----------------------Confusion Matrix--------------------
                    Actual Result
                   0                  1
Expected  0         28448824          55

result    1         1404355          18062

Accuracy: 95.29846311321745
```



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).

```
# Ramdom forest
def Random_forest_classification(training_feature, training_label, testing_feature, testing_label):

    # Creating the model
    rf = RandomForestClassifier(n_estimators=100,class_weight={0:1,1:0.001},n_jobs=-1)

    # Traning the model with traning data
    rf.fit(training_feature, training_label)

    print('Training Data')
    # Testing the model with the testing data
    r=rf.predict(training_feature)

    #Computing the confusion matrix
    cm=confusion_matrix(testing_label,r)
    confusionMatrixPrint(cm)

    result = np.sum(training_label.values.flatten() == r)/r.size
    print("Accuracy:",result*100)

    fpr, tpr, _ = roc_curve(training_label,r)

    plt.plot(fpr, tpr, color='darkorange',label='ROC curve')
    plt.plot([0, 1], [0, 1], color='navy',  linestyle='--')
    plt.plot([0, 0], [0, 1], color='darkorange')
    plt.xlim([-0.1, 1.0])
    plt.ylim([-0.1, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
    Act_del = np.count_nonzero(training_label)
    Pred_del =  np.count_nonzero(z)
    length = r.size
    tp_del = cm[0][0]
    fp_del = cm[1][0]
    print('Number of records in dataset', length,'Actual delequents', Act_del, 'Predicted Delenquents',  Pred_del, 'Proper DE
```

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.

In [ ]:

```
def Neural_net(training_feature, training_label, testing_feature, testing_label):
    nn = MLPClassifier(solver='adam', alpha=1e-6,hidden_layer_sizes=(10, 2), random_state=3, max_iter=300,warm_start=True)

    # we create an instance of Neighbours Classifier and fit the data.
    nn.fit(training_feature, traning_label)

     r=rf.predict(training_feature)

    #Computing the confusion matrix
    cm=confusion_matrix(testing_label,r)
    confusionMatrixPrint(cm)

    result = np.sum(training_label.values.flatten() == r)/r.size
    print("training Data")
    print("Accuracy:",result*100)

    fpr, tpr, _ = roc_curve(training_label,r)

    plt.plot(fpr, tpr, color='darkorange',label='ROC curve')
    plt.plot([0, 1], [0, 1], color='navy',  linestyle='--')
    plt.plot([0, 0], [0, 1], color='darkorange')
    plt.xlim([-0.1, 1.0])
    plt.ylim([-0.1, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()

    print('\nTesting Data')
    z=nn.predict(testing_feature)
    cm=confusion_matrix(testing_label,z)
    confusionMatrixPrint(cm)
    fpr, tpr, _ = roc_curve(testing_label,z)
    result = np.sum(testing_label.values.flatten() == z)/z.size
    print("Accuracy:",result*100)
    plt.plot(fpr, tpr, color='darkorange'
```

AutoSklearn:

AutoSklearn for classification can be executed on Ubuntu. This execution is done on Amazon EC2 instance on Ubuntu 2xlarge.

All the scikit learn libraries for classification are installed and imported. The train and test data is then read in a dataframe.

```
ubuntu@ip-172-31-40-88: /tmp
le/weight_boosting.py:29: DeprecationWarning: numpy.core.umath_tests is an in
ternal NumPy module and should not be imported. It will be removed in a futur
e NumPy release.
  from numpy.core.umath_tests import inner1d
>>> from sklearn import metrics
>>> import autosklearn
>>> import pandas as pd
>>> import numpy as np
>>> from autosklearn.pipeline.components import classification as classificat
ion_components
>>> train_data = pd.read_csv("/home/ubuntu/historical_data_time_Q12005.txt" ,
 skipinitialspace=True, sep = "|" ,low_memory = False , header = None)
>>> test_data = pd.read_csv("/home/ubuntu/historical_data_time_Q22005.txt" ,
skipinitialspace=True, sep = "|" ,low_memory = False , header = None)
>>> test_data.columns=['loan_number','year-month', 'current_actual_upb', 'delq_status','loan_age', 'rem_months', 'repurchase_flag', 'modification_flag', 'zer
o_balance_code',
...                          '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', 'stepmod_ind' , 'dpm_ind' , 'eltv' ]
>>> train_data.columns=['loan_number','year-month', 'current_actual_upb', 'delq_status','loan_age', 'rem_months', 'repurchase_flag', 'modification_flag', 'ze
ro_balance_code',
...                          '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', 'stepmod_ind' , 'dpm_ind' , 'eltv' ]
```

The train_data and test_data dataframe is cleaned with necessary datatypes.

```
>>> 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)
...
>>> cleandf(train_data)
>>> cleandf(test_data)
```

Adding the ‘Deliquent’ column to the dataframe based on delq_status/

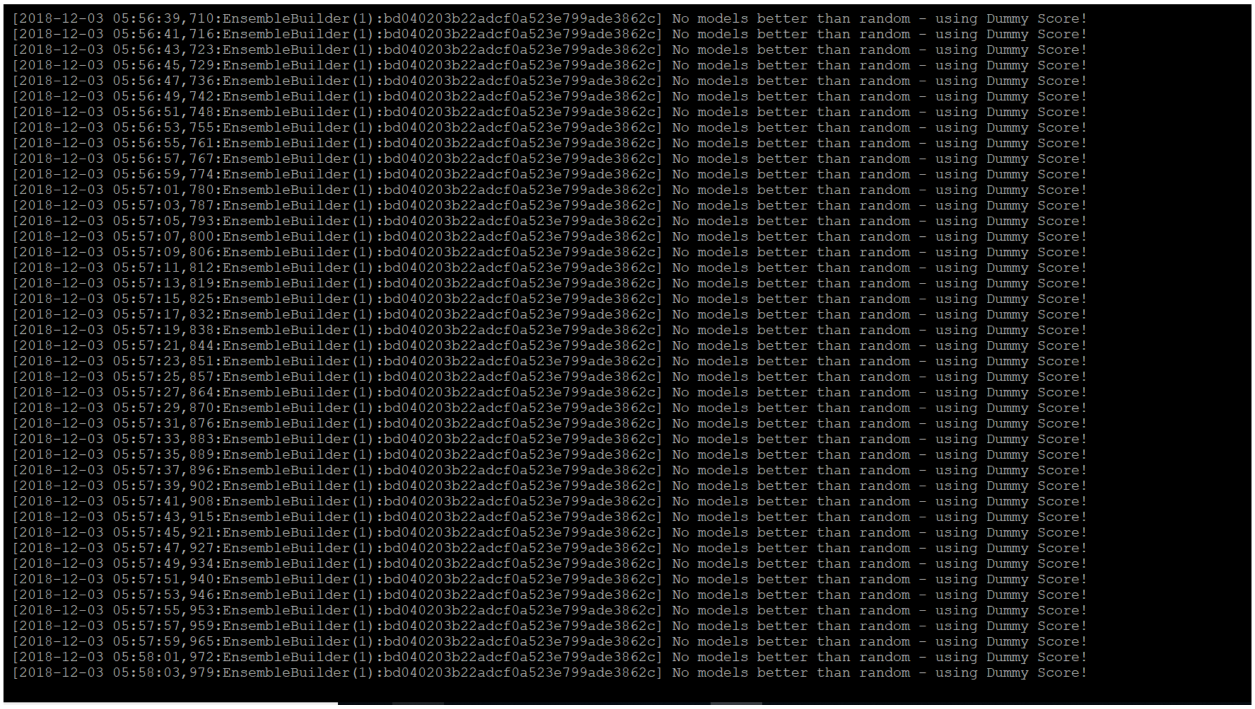
ubuntu@ip-172-31-40-88: /tmp

```
>>> def statusDeliquent(row):
...         if row['delq_status'] > 0:
...             val = 1
...         else:
...             val = 0
...         return val
...
>>> train_data['Deliquent'] = train_data.apply(statusDeliquent, axis=1)
>>> test_data['Deliquent'] = test_data.apply(statusDeliquent, axis=1)
>>> train_data.head()
    loan_number  year-month    ...      eltv  Deliquent
0  F105Q1000001      200504    ...       NaN          0
1  F105Q1000001      200505    ...       NaN          0
2  F105Q1000001      200506    ...       NaN          0
3  F105Q1000001      200507    ...       NaN          0
4  F105Q1000001      200508    ...       NaN          0

[5 rows x 27 columns]
>>> test_data.head()
    loan_number  year-month    ...      eltv  Deliquent
0  F105Q2000001      200507    ...       NaN          0
1  F105Q2000001      200508    ...       NaN          0
2  F105Q2000001      200509    ...       NaN          0
3  F105Q2000001      200510    ...       NaN          0
4  F105Q2000001      200511    ...       NaN          0

[5 rows x 27 columns]
>>> features = [f for f in list(train_data) if 'feature' in f]
>>> x = train_data[features]
>>> y = train_data['target']
```

```
>>> y = train_data['Deliquent']
>>> x_features = test_data[features]
>>> ids = test_data['Deliquent']
>>> train_data.drop['Deliquent']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'method' object is not subscriptable
>>> train_data.drop('Deliquent')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/ubuntu/anaconda3/envs/my_env/lib/python3.7/site-packages/pandas
/core/frame.py", line 3697, in drop
    errors=errors)
  File "/home/ubuntu/anaconda3/envs/my_env/lib/python3.7/site-packages/pandas
/core/generic.py", line 3111, in drop
    obj = obj._drop_axis(labels, axis, level=level, errors=errors)
  File "/home/ubuntu/anaconda3/envs/my_env/lib/python3.7/site-packages/pandas
/core/generic.py", line 3143, in _drop_axis
    new_axis = axis.drop(labels, errors=errors)
  File "/home/ubuntu/anaconda3/envs/my_env/lib/python3.7/site-packages/pandas
/core/indexes/base.py", line 4404, in drop
    '{} not found in axis'.format(labels[mask]))
KeyError: "['Deliquent'] not found in axis"
>>> train_data.head()
    loan_number  year-month    ...      eltv  Deliquent
0  F105Q1000001      200504    ...       NaN          0
1  F105Q1000001      200505    ...       NaN          0
2  F105Q1000001      200506    ...       NaN          0
3  F105Q1000001      200507    ...       NaN          0
4  F105Q1000001      200508    ...       NaN          0

[5 rows x 27 columns]
>>> features.head()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'list' object has no attribute 'head'
```

Fitting the variables X and Y to the model autosklearn.

```
>>> model = autosklearn.classification.AutoSklearnClassifier()
>>> model.fit(x,y)
```



```
[2018-12-03 05:56:39,710:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:56:41,716:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:56:43,723:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:56:45,729:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:56:47,736:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:56:49,742:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:56:51,748:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:56:53,755:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:56:55,761:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:56:57,767:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:56:59,774:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:01,780:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:03,787:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:05,793:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:07,800:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:09,806:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:11,812:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:13,819:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:15,825:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:17,832:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:19,838:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:21,844:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:23,851:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:25,857:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:27,864:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:29,870:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:31,876:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:33,883:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:35,889:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:37,896:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:39,902:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:41,908:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:43,915:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:45,921:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:47,927:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:49,934:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:51,940:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:53,946:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:55,953:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:57,959:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:57:59,965:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:58:01,972:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
[2018-12-03 05:58:03,979:EnsembleBuilder(1):bd040203b22adcf0a523e799ade3862c] No models better than random - using Dummy Score!
```

Conclusion: We get the best model as Random Forest.

TPOT: