Fannie Mae provides lenders with a reliable source of mortgage financing. For many people, owning a home is an important component of their wealth creation plan because values consistently increase. However, confidence in this system was harmed during the subprime mortgage crisis of 2007–2009, when hundreds of thousands of borrowers missed mortgage payments and had their houses foreclosed upon. There were several factors that led to this catastrophe. This catastrophe sent shockwaves through the financial markets, wiping out wealth worth trillions of dollars. Those that didn't default saw their retirement savings destroyed and the value of their homes cut in half. Numerous banks either requested federal rescue money, were bought by larger firms, or filed for bankruptcy to address the ensuing liquidity issue.

Foreclosures cost lenders money even after a sizable percentage of the systematic risk has been reduced. When a borrower fails on a loan, the bank needs to deal with more than simply the initial foreclosure procedure. They also have to maintain the properties and eventually sell them, sometimes at a loss. If actions are done to avoid lending to clients who may default in the future by increasing screening criteria, consumers may avoid a negative mark on their credit history and financial institutions may be spared of undue risk.

In this project, I gathered the publicly accessible loan data from the Fannie Mae website, cleaned it up, and looked into the differences between borrowers who default and those who don't. Additionally, I used machine learning strategies to identify potential defaults.

Exploring the Data:

On the FNMA website, you can view loan data by quarter. In order to prepare it for the prediction algorithm, I obtained the loan data for the two prior years (2016-2017). Reduce_data.py handles combining the datafile and produces a single file for additional analysis if quarterly files are downloaded

(users can downloaded whole dataset in one go). Multiple reporting periods up to the quarter of the cycle will be provided. After a loan is foreclosed, there is no longer a reporting period and no more data entries. After reading loans for each reporting period, Reduce_data.py only accepts the most recent reporting entry. The most recent information on the loan, including whether or not it has been foreclosed is provided in the most recent reporting row.

Some data values are transformed so that they can be used in machine learning algorithms, such as filling in any missing values or removing them, converting columns to numbers, or formatting columns with the proper date format.

```
(good,default) = pd.value_counts(data['foreclosured'])
print('Total Defaults =',default)
print('Total Good loans =',good)
print('Default Rate {:.1%}'.format(100 *default/(good+default)))

grouped_data = data.groupby('origination_date_year')['foreclosured'].value_counts().unstack()
print(grouped_data)
for year,row in grouped_data.iterrows():
    print(f"{year} had {row[1]} defaults")
```

```
Total Defaults = 5209
Total Good loans = 4330877
Default Rate 12.0%
foreclosured                0       1
origination_date_year
2013                      4.0     NaN
2014                    137.0     NaN
2015                 170561.0   272.0
2016                2351021.0  2629.0
2017                1809154.0  2308.0
2013 had nan defaults
2014 had nan defaults
2015 had 272.0 defaults
2016 had 2629.0 defaults
2017 had 2308.0 defaults
```

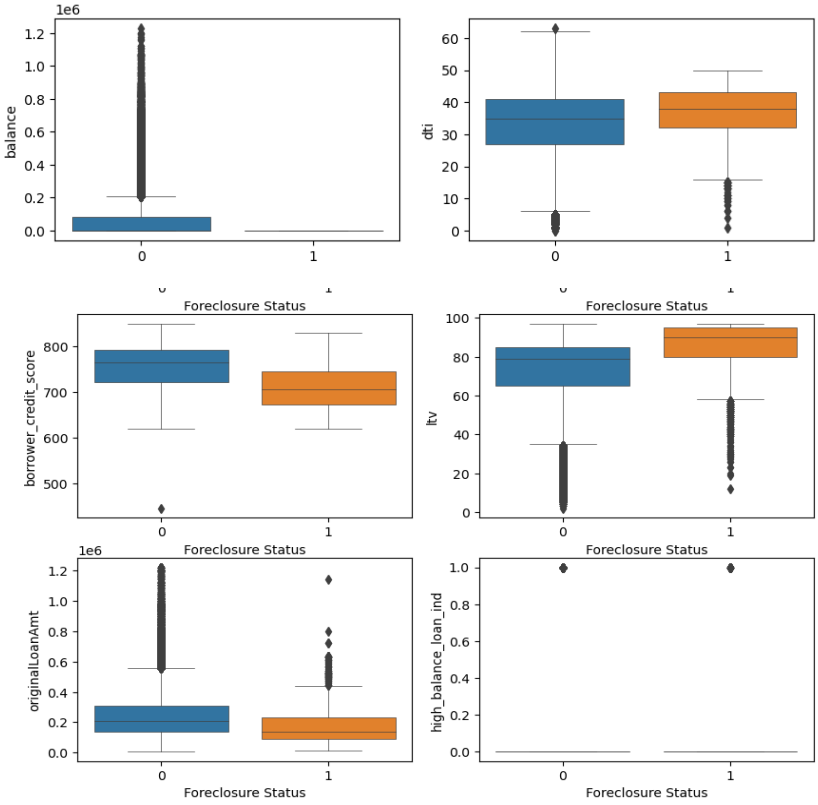
There is a significant imbalance in the loan data for those two years; just 12% of the loans in that dataset were in default. As a result, we must take into account the fact that mortgage defaults are uncommon. Without creating a single line of code, we can make an 88% accurate prediction that no homes will go into foreclosure. While accurate, this model lacks meaning, thus we must take this imbalance into consideration to learn anything.

As loan amounts range from 5000 and up to $1 million, loan data is a very diversified set of information. A 30-year mortgage is a feature of the majority of these loans, with interest rates ranging from 1.75% to 6.12%.

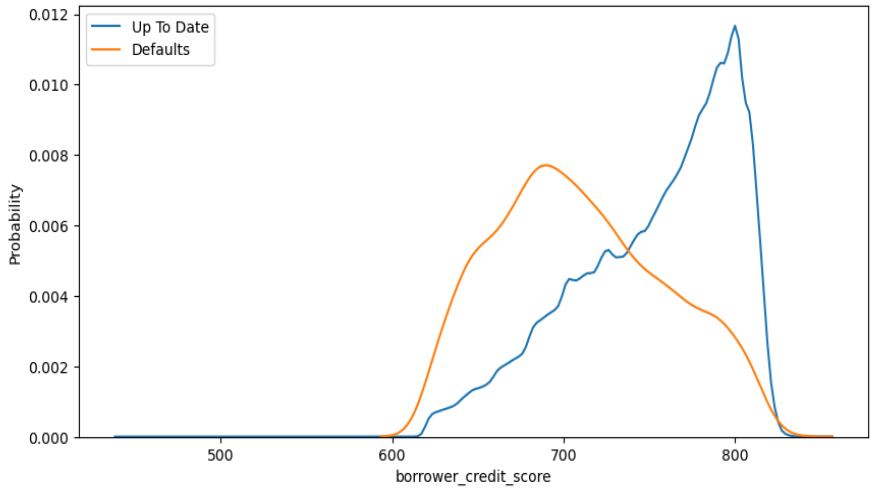
```
description = data[['originalLoanAmt','balance',
                            'interest_rate',
                            'dti',
                            'loan_term',
                            'insurance_percentage'
                            ]].describe()
print(description[1:].round(2))
```

```
      originalLoanAmt     balance  interest_rate    dti  loan_term  \
mean        232440.75    51274.56           3.93  33.79     318.49
std         119673.99    92009.71           0.54   9.10      74.83
min           5000.00        0.00           1.75   0.00      36.00
25%         140000.00        0.00           3.62  27.00     360.00
50%         211000.00        0.00           3.99  35.00     360.00
75%         307000.00    83364.28           4.25  41.00     360.00
max        1223000.00  1227922.05           6.12  63.00     360.00

      insurance_percentage
mean                 24.83
std                   7.00
min                   1.00
25%                  25.00
50%                  25.00
75%                  30.00
max                  42.00
```

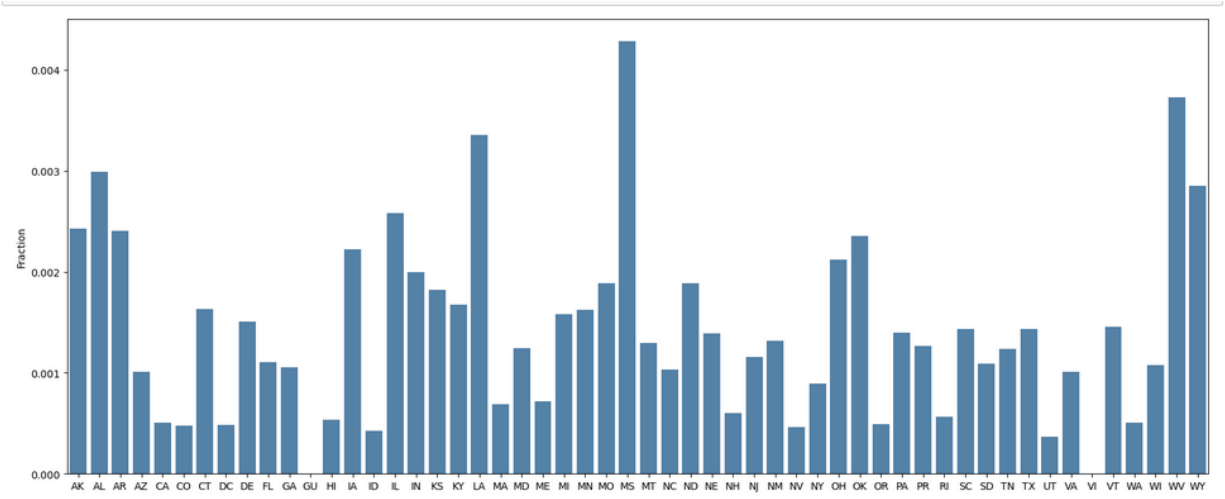
Let's make a few box plots to examine the loan balances, debt-to-income ratios, and credit ratings of the major borrowers. It's interesting to observe that the balance column frequently displays a number of 0 when a loan is foreclosed. The differences between good and default loans are pretty small when we look at the values of other variables.

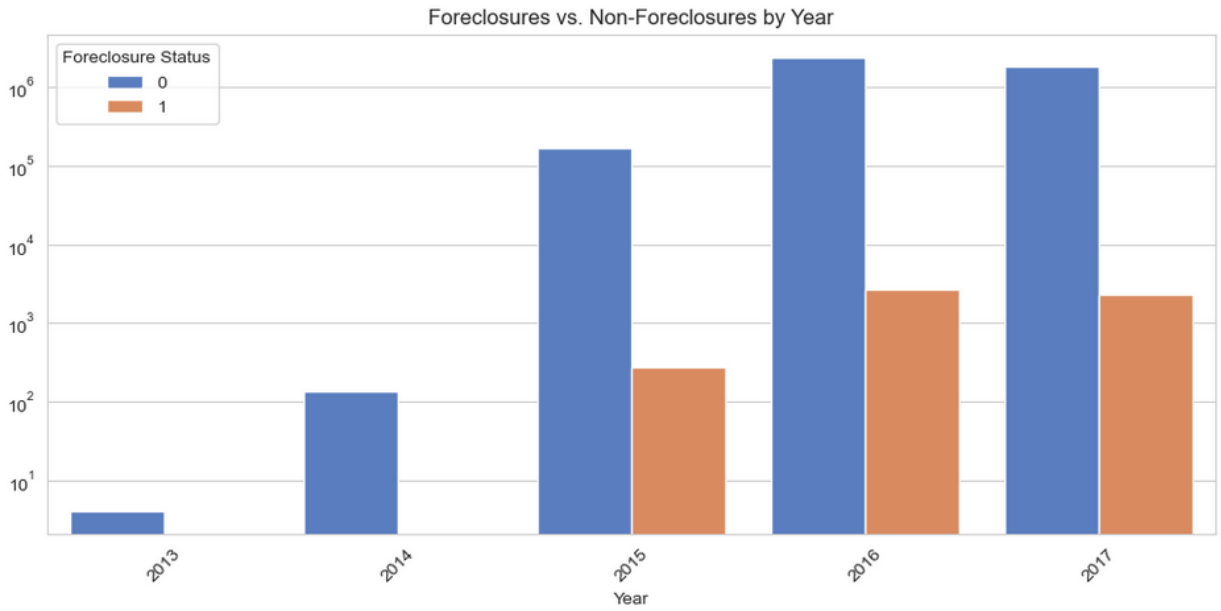


The probability density graph shows clear distinctions between default and good loans when we look at the primary borrower's credit scores. Higher credit scores show a right-skewed distribution in good loans, whereas the apex of the curve for default loans is concentrated around a credit score of about 700.



Data from 2016 and 2017 demonstrates that the rates of foreclosure vary by state. Few states have zero foreclosures, while few states have very high foreclosure rates. Since the data come from two years, 2016 and 2017, we can also determine whether the foreclosure rate varies from year to year. Loans that date back to 2013 are contained in the 2016 and 2017 data files. Let's look at home foreclosures by origination date.



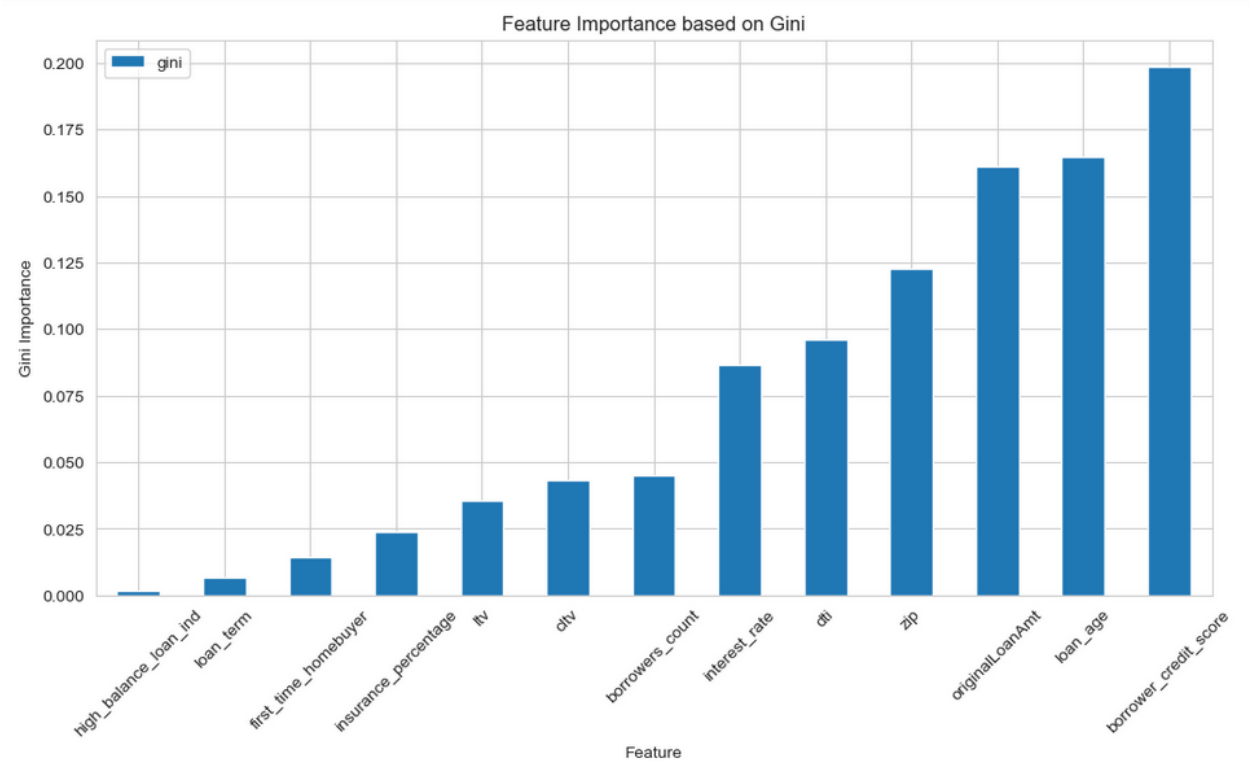


"Building the Machine Learning Classifier:"

"In order to make predictions in a binary classification environment, we must choose a suitable algorithm. There are just two possible values for our target variable, foreclosure: True or False. We need to pay great attention to class weighting when dividing the data into training and testing sets because this dataset is very unbalanced.

To streamline our model, we have removed uninformative features, such as recovery cost columns and insurance percentages. Conducting a Feature Importance Analysis is the following step. To do this, we train each feature in the filtered dataset using a Random Forest Classifier. We then keep track of the feature importance scores for each feature, allowing us to determine which characteristics have the greatest predictive power.

```
[0 1]

In [32]: bestFeatures, bestParams = best_random_forest_parameters(data)

****filtered_data*****
Best parameters for random forest classification are  {'n_estimators': 30, 'min_samples_split': 5, 'min_samples_leaf':
2, 'max_depth': None, 'criterion': 'gini'}
**********Features with importance ******************************
{'originalLoanAmt': 0.16102288121837935, 'interest_rate': 0.08661349163973363, 'borrowers_count': 0.045113756438833016,
'dti': 0.09599071502115944, 'loan_term': 0.006577517095304195, 'cltv': 0.04308895291895008, 'ltv': 0.035803163584265, '
loan_age': 0.1645429360891354, 'borrower_credit_score': 0.1986631638199225, 'high_balance_loan_ind': 0.0015682772400273
822, 'first_time_homebuyer': 0.014175278333886052, 'insurance_percentage': 0.024025166975759096, 'zip': 0.1228146996246
4489}
```



As next step, we'll divide the data into subsets for training and testing the model. Keep in mind that our dataset is unbalanced. We must first address this issue in order to fit the model.

For balancing the data here, I attempted two distinct approaches.

- SMOTE (Synthetic Minority Over-sampling Technique) from imbalanced-learn library
- Manually dividing the dataset to balance each set

SMOTE:

Data is divided into training and test sets (70/30 split) while 'stratify' is used to ensure that the class distribution is maintained. SMOTE (Synthetic Minority Over-sampling Technique) was then used to address the dataset's class imbalance. The training and test sets are each given a separate application of SMOTE. Later, the optimal hyperparameters are used to initialize the Random Forest Classifier, and predictions on resampled test data are computed along with prediction probabilities.

```
****filtered_data*****
selected columns for forclosure prediction
['dti', 'borrower_credit_score', 'interest_rate']
Training Data: foreclosured and good loans count before SMOTE :
 0    3029698
1       3640
Name: foreclosured, dtype: int64
Test Data: foreclosured and good loans count before SMOTE :
 0    1298442
1       1560
Name: foreclosured, dtype: int64
Training Data: foreclosured and good loans count After SMOTE :
 0    1298442
1    1298442
Name: foreclosured, dtype: int64
Test Data: foreclosured and good loans count After SMOTE :
 0    1298442
1    1298442
```

Manually Segmenting the Dataset:

We determine the oversampling fraction during the manual dataset division to achieve balance. In order to do this, a subset of the good loans must be chosen, oversampling the minority class, which reflects defaulted loans.

```
defaults = filtered_data[filtered_data['foreclosured']==1.0]
good = filtered_data[filtered_data['foreclosured']==0.0]
frac = (1.0*defaults.shape[0])/(1.0*good.shape[0])
filtered_data = pd.concat([defaults, good.sample(frac=frac*2.)]).sample(frac=1.0)
```

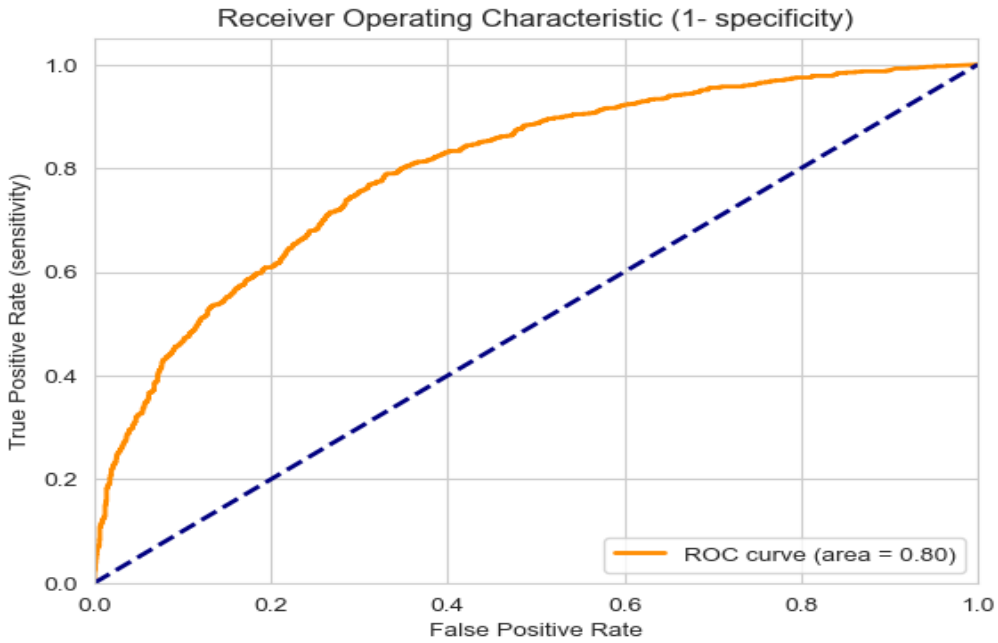
```
****filtered_data*****
foreclosured loans in filtered_data : 3057
Good loans in filtered_data : 1223241
After Sampling foreclosured loans in filtered_data : 3057
After Sampling  Good loans in filtered_data : 6114
selected columns for forclosure prediction
```

Let's look at the performance of this model:

```
**************************************************************************************************************
RandomForest Classifer - Test Data: Confusion matrix
[[1522  313]
 [ 385  532]]
****** RandomForest Classifer, with randomized Search Classification report ***Test Data ****
              precision    recall  f1-score   support

           0       0.80      0.83      0.81      1835
           1       0.63      0.58      0.60       917

    accuracy                           0.75      2752
   macro avg       0.71      0.70      0.71      2752
weighted avg       0.74      0.75      0.74      2752


RandomForest Classifer tree: Accuracy score - Test Data : 0.7463662790697675
**************************************************************************************************************
```

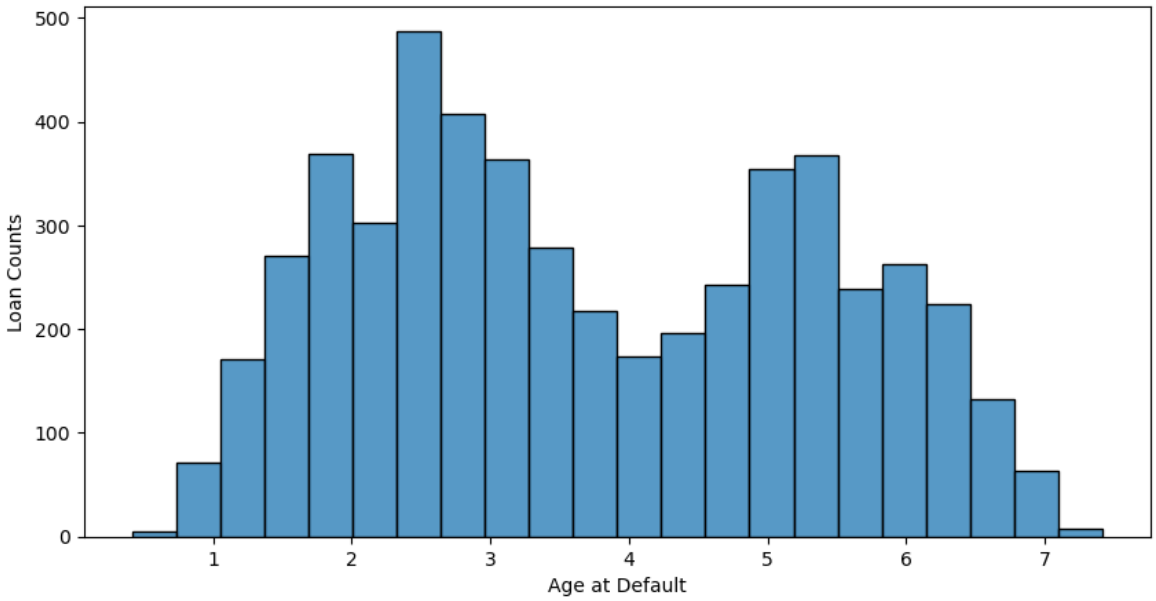
The recall and precision of the Random Forest Classifier model are both 64% for fore closured loans. In real terms, this indicates that 64% of the foreclosure predictions were accurate. On the test data, the classifier has an accuracy of roughly 75%. With an F1-score of roughly 0.60, it predicts foreclosures with a decent combination of precision and recall.

We look at the ROC (Receiver Operating Characteristic) curve to evaluate the random forest model's quality. The true positive and true negative rates (false positive) are quantified in relation to how the discrimination threshold is changed. As the criteria are tightened, an efficient model shows little variation in the false positive rate. The ROC curve should, in theory, be located in the upper-left corner of the TP-FP region, which denotes a higher probability of true detections as opposed to false alarms.



Conclusion:

Defaulted loans in the dataset shows the around 2.5 years most of the loans gets to the foreclosure state. With this analogy when predicted fore closed results are analyzed in the test set, as of 2017 following loan counts have ~75% chances of getting into default state in ~2.5years.




```
false_neg1 = false_neg[['origination_date_year','foreclosured','prediction']]
grouped_result = false_neg1.groupby(['origination_date_year']).count()
grouped_result = grouped_result.rename(columns={'foreclosured': 'foreclosured = 0 '})
print(grouped_result)
```

```
                        foreclosured = 0   prediction
origination_date_year
2015                                   8           8
2016                                 141         141
2017                                 164         164
```

References:

Data set is downloaded by FNMA website

Fannie Mae Single-Family Loan Performance Data | Fannie Mae

Drew Hogg ‘s project – handling imbalanced data