Capstone Project – Mortgage Bank Analytics

Table of Contents

Capstone Project – Mortgage Bank Analytics 1
Mortgage Loans Analytics 3
1. Predicting mortgage demand using machine learning techniques 3
Results: 4
2. Decision Tree Algorithm 5
3.1.1 Business Understanding 5
3.1.2 Data Understanding 5
3.1.3 Data Preparation 6
3.1.4 Modeling 6
4.2.2 Mortgage interest rates 6
Random Walk 11
Are Interest Rates or Monthly Loan Returns Prices a Random Walk? 11
Are Interest Rates Auto correlated? 12
4.2.3 Changes in regulations 13
4.2.4 Other predictors 14
4.3 PREDICTIVE ANALYTICS 14
5 DATA UNDERSTANDING 16
5.1 DATA COLLECTION 16
5.2 DATA EXPLORATION 17
6 DATA PREPARATION 37
Encoding Categorical Data 37
Label Encoding 38
One-Hot encoding 38
6.1 DATA PRE-PROCESSING 38
Handling Categorical Data in Python 39
Preprocessing: scaling 39
6.2 FEATURE ENGINEERING 40
6.3 FEATURE SELECTION 41

Feature Selection Methods: 41
I will share 3 Feature selection techniques that are easy to use and also gives good results. 41
2. Feature Importance 42
6.4 PCA (Principal Component Analysis) 44
Variance of the PCA features 45
PCA as dimensionality reduction 49
7 MODELING 50
7.1 SELECTION OF MODELING TECHNIQUES 50
7.2 MODEL BUILDING 51
ARMA Model 51
Linear regression 56
Fitting Linear Regression using statsmodels 58
Fitting Linear Regression using sklearn 60
Lasso Regression 61
Logistic Regression 66
Centering, Scaling and Logistic Regression 66
Scaling Synthesized Data 67
Adding Gaussian noise to the signal: 69
Random Forests (RF) 72
Finding Important Features in Scikit-learn 73
Support Vector Regression (SVR) 75
Support Vector Machine (SVM) 75
Tunning Parameter for SVM 76
k-Nearest Neighbors (KNN) 76
k-Nearest Neighbors: FIT 76
k-Nearest Neighbors: Predict 77
Preprocessing: scaling 78
Decision Tree Classifier 79
7.3 MODEL VALIDATION 82
8 EVALUATION 82
8.1 MODEL EVALUATION 82
8.2 DISCUSSION OF RESULTS 82

10 CONCLUSIONS, LIMITATIONS AND FURTHER RESEARCH 83
10.1 CONCLUSIONS 83
10.2 LIMITATIONS 84
10.3 RECOMMENDATIONS FOR FURTHER RESEARCH 85

Mortgage Loans Analytics

Banks can now use mortgage loan analytics using Data Science techniques. The system can provide detail information of the mortgage loans and the mortgage loan markets. It is a powerful tool for mortgage brokers to seek counterparties and generate trading interests and is useful for the CFOs to conduct what-ifs scenarios on the balance sheets.

Fill in the follow fields in Loan file template:

- Loan ID: to identify the special loan
- Loan Type: to indicate the loan if fixed rate, or balloon loan , or ARM, or AMP (alternative mortgage product).
- Balance:
- Loan program type: to indicate conforming loan, FHA/VA loan, Jumbo loan or sub-prime loan
- Current coupon rate:
- Amortization type: the original amortization term
- Maturity: the maturity loan (the remaining term of the loan)
- FICO Score: the updated fico score
- LTV: the current loan to value ratio
- Loan Size: the loan amount of the loan
- Loan origination location (City & Zip)
- Unit Types (Types of property)

1. Predicting mortgage demand using machine learning techniques

It is difficult for the financial institutions to determine the amount of personnel needed to handle the mortgage applications coming in. There are multiple factors influencing the amount of mortgage applications, such as the mortgage interest rates, which cause the amount of mortgage applications to differ day by day. In this research we aim to provide more insight in the amount of personnel needed by developing a machine learning model that predicts the amount of mortgage applications coming in per day for the next week, using the CRISP-DM framework. After conducting a literature study and interviews, multiple features are generated using historical data from a Dutch financial institution and external data. A number of machine learning models are developed and validated using cross-validation. The predictions of our best model differ on average mortgage applications per day compared to the actual amount of mortgage applications. A dynamic dashboard solution is proposed to visualize the

predictions, in which mortgage interest rate changes can be manually entered in the dashboard, and recommendations have been given for the deployment of the model at the financial institutions.
Methodology

Historical data and publicly available data were used as input for our predictive model, and five machine learning techniques (Decision Tree, Random Forest, Support Vector Machines, Support Vector Regression and KNN) were applied to create the predictions. The models are validated using repeated cross-validation, and evaluated using several evaluation criteria. We have also used ARIMA model, Linear Regression and Logistic Regression for predictive modeling.

Results:

The Random Forest model gave the best result on each of the four evaluation criteria used to evaluate the models. The Random Forest model is mortgage applications per day perform best, then Decision Tree model. The SVR model scored is the worse, SVM on a second place. The percent error of the Random Forest model is around of the actual amount of mortgage applications per day.

Linear Regression,

Logistic Regression

Random Forests (RF)

Support Vector Regression (SVR)

Support Vector Machine (SVM)

k-Nearest Neighbors

Decision Tree Classifier

Using Scikit-learn, optimization of decision tree classifier performed by only pre-pruning. Maximum depth of the tree can be used as a control variable for pre-pruning. In the following the example, we can plot a decision tree on the same data with max_depth=4. Other than pre-pruning parameters, We have also tried other attribute selection measure such as entropy This pruned model is less complex, explainable, and easy to understand than the previous decision tree model plot.

Pros

Decision trees are easy to interpret and visualize.

It can easily capture Non-linear patterns.

It requires fewer data preprocessing from the user, for example, there is no need to normalize columns.

It can be used for feature engineering such as predicting missing values, suitable for variable selection.

The decision tree has no assumptions about distribution because of the non-parametric nature of the algorithm.

Cons

Sensitive to noisy data. It can overfit noisy data.

The small variation(or variance) in data can result in the different decision tree.

Decision trees are biased with imbalance dataset, so we can balance out the dataset before creating the decision tree.

2. Decision Tree Algorithm

A decision tree is a flowchart-like tree structure where an internal node represents feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning. This flowchart-like structure helps you in decision making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.

A predictive model was created using the SVM, KNN, Random Forest, and Deaccession Tree technique, which predicts the amount of mortgage applications per day(LoanInMonth) with a mean absolute error of mortgage applications per day. This can directly be converted to the amount of personnel needed at the mortgage application department of the financial institutions, by dividing it by the amount of mortgage applications handled per person per day. Based on mortgage per months, company can manage core human resources, such as Loan Processors, Mortgage Loan Originators, Underwriters, secondary market analyst, lock desk personal, compliance personals, & others.

The mortgage interest rates have the biggest impact on our model, but are difficult to predict.
Several features (**10 Years US Treasury Rate, Home Supply Index**) can be added to the model in order to improve its predictive power. Furthermore, there is still improvement in the feature regarding mortgage interest rate changes, as a significant part of the error of our model is caused by under-prediction of the outliers.

3.1.1 Business Understanding

The background information is collected on the domain. A theoretical framework is developed by conducting a literature review which contains an overview of the related research in this research area, and an overview of the concepts in predictive analytics and the different models that are feasible for our Capstone project. For the domain analysis, background information on the mortgage application process and the mortgage domain is collected in order to get a better understanding of the different topics.

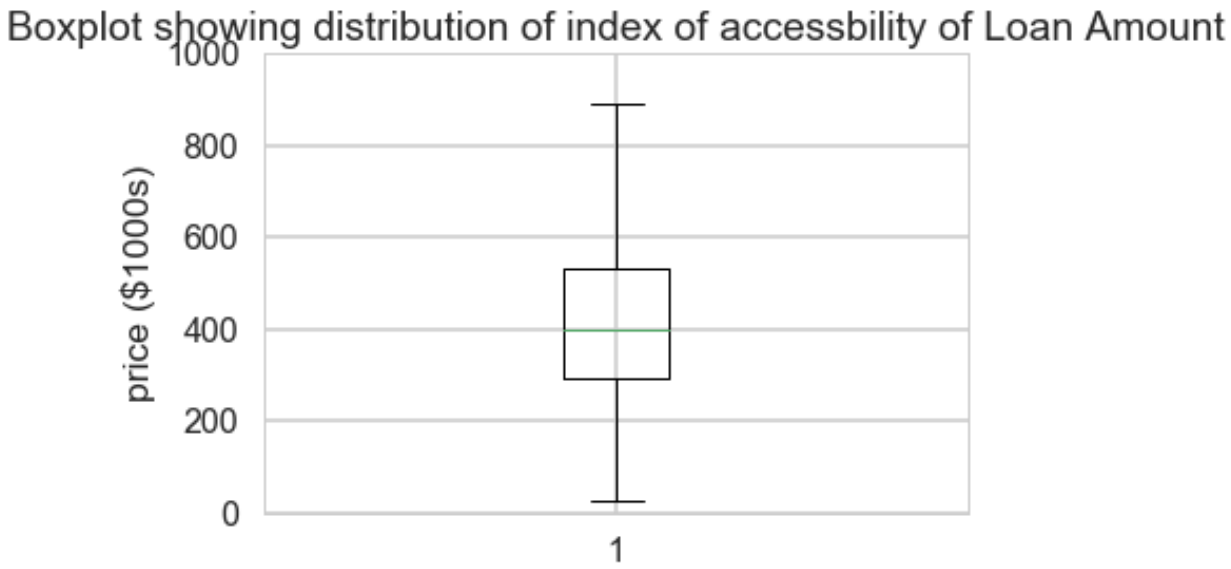
3.1.2 Data Understanding

In the Data Understanding stage the raw data is collected from the database using an SQL query and its characteristics and distributions are explored. Event logs are kept in the database and can be used for predictive modeling. Once the data is collected and saved as **csv** format, the data is explored and visualizations are made of the different variables to get an understanding of the data. With these visualizations we can already see some of the relationships in the data and identify possible features. The data exploration activity is important for becoming familiar with the data and identifying data quality problems.

3.1.3 Data Preparation

The goal of the Data Preparation stage is to transform and enrich the dataset so that it can be fed into the models. After the data is collected and explored, it can be pre-processed so that it can be used directly in our predictive model. With the pre-processed data one can perform feature engineering. Using historical data and external data, different features can be generated. After the feature engineering activity, a subset of features will be selected that provide predictive value for our models.

Outlier detections using Boxplot:
#creating boxplot to see the distribution median and outliers
plt.boxplot(model_data['Loan Amount'])
plt.title("Boxplot showing distribution of index of accessbility of Loan Amount")
plt.ylim(0.1,1000)
plt.ylabel('price ($1000s)')
plt.show()



3.1.4 Modeling

In this stage, several models are developed based on the dataset. First, a selection of predictive models is made (e.g. Linear Regression, Random Forests). These models are trained on the dataset and used tomake predictions. The models are validated using a test set and repeated 5-fold cross-validation.

For each of the models, hyperparameters are optimized and data pre-processing is done if needed (e.g. centering, scaling, multi-collinearity checks). Some of the models have specific requirements on the form of the data, which require specific pre-processing activities

4.2.2 Mortgage interest rates

Mortgage interest rates have a significant impact on the amount of mortgage applications. If the interest rates are low, the mortgages are relatively cheaper for the borrower as they have to pay less interest, which leads to an increased amount of mortgage applications. A high mortgage interest rate means the mortgage borrower pays a high amount of interest to the lender, which makes the mortgage less attractive for the borrower. Interest rate changes have a significant impact on mortgage applications, as was seen in November of last year, where a sudden increase in interest rates led to a large peak in mortgage applications. The main difference between the mortgages offered by these types of companies lies in the mortgage interest rates. Even a small difference in mortgage interest rates can often save or cost the borrower a vast amount of money, due to the large sum of a mortgage.

In general, there are two types of mortgage interest rate: variable rates (**ARM**) and fixed rates. Variable interest rates are generally lower than fixed interest rates, but can change every month. Fixed interest rates are slightly higher, but are fixed for a certain period of time. A fixed interest rate is generally preferred when the mortgage interest rates are expected to rise, or when the borrower wants to know its monthly expenses upfront. A variable interest rate (**ARM**) is preferred when interest rates are expected to decrease. If a financial institution has a significantly higher interest rate than its competitors, it will generally receive fewer mortgage applications as the independent mortgage advisors will forward its customers to a different mortgage lender.

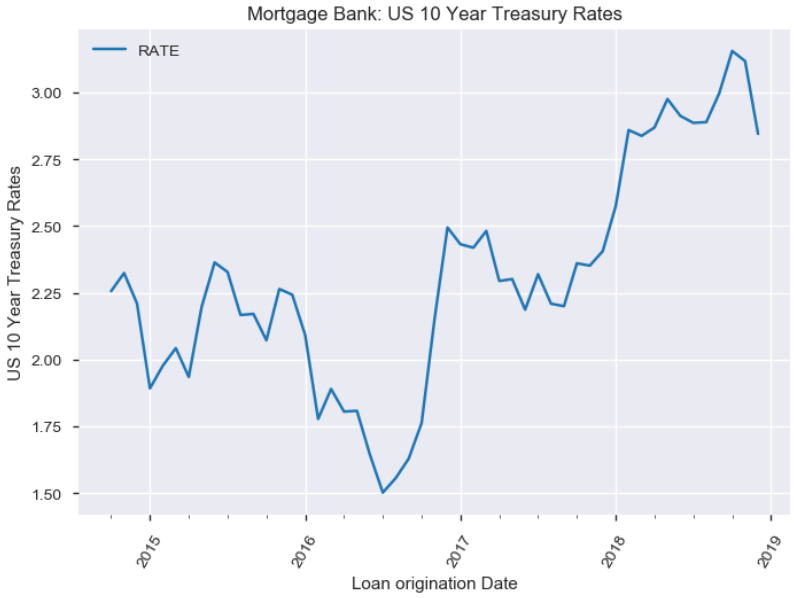
For the financial institutions, there can be a number of reasons to change its mortgage interest rate. First of all, the mortgage interest rate is based on the cost of lending for the financial institutions itself. If the cost of debt is higher, the financial institutions will compensate this by charging higher interest rates for its mortgages, in order to keep a profitable margin on their products. This cost of lending is mainly based on the capital market interest rate, for the long-term loans, and the short-term loans. If either of these changes significantly, one can expect the financial institutions to respond by changing their own mortgage interest rates. This usually happens after a few days.

Second, financial institutions generally work with a budget for their mortgages. Based on the amount of funding they can get, and on the interest rates and the duration of the funding, they determine a budget for their mortgages for the upcoming period. Ideally, financial institutions want to match the duration of the fixed interest period of a mortgage with the duration of the lending of debt for that mortgage. Once a financial institution is almost out of budget for a specific fixed interest period, it may choose to increase the interest rate for mortgages with that fixed interest period. This way, borrowers will apply for mortgages with a different fixed interest period, or may choose to go to another mortgage lender.

Finally, financial institutions sometimes increase their interest rates during the summer months, and at the end of the year, as there is less personnel available to handle the requests due to vacations and holidays. With less personnel available they can handle less mortgage requests, so in order to keep the processing time the same they choose to reduce the input, by increasing the interest rates. Financial institutions may also specifically keep interest rates low for mortgages with a certain fixed interest period. Interest rate changes are not always directly influenced by changes in the cost of lending, but can have numerous reasons.

```
US10Y.index = pd.to_datetime(US10Y.index)
US10Y.resample('M',how=mean).plot(title="Interest Rate for 10 Years Treasury - ")
plt.xlabel('Loan origination Date')
plt.xticks(rotation=60)
plt.ylabel('US 10 Year Treasury Rates')
plt.title('Mortgage Bank: US 10 Year Treasury Rates')
monthly_rate=US10Y.resample('M',how=mean)
type(monthly_rate)
monthly_rate_data=monthly_rate['RATE']
type(monthly_rate_data)
type(monthly_loan_rev_data)
```

Generally, when US 10 Years Treasury Rate fluctuates, that leads lenders to adjust their internal bank rates accordingly. Also interest rates for consumers varies on several risk factors, such as DTI (Debt to Income Ratio), FICO Scores, Recent derogatory events on their credit history, stable job history, W2 or 1099, Stated Income, Profit or Loss Statements, Student Loans, Auto Payments, Credit utilization, Property types, number of households, rental history, etc.

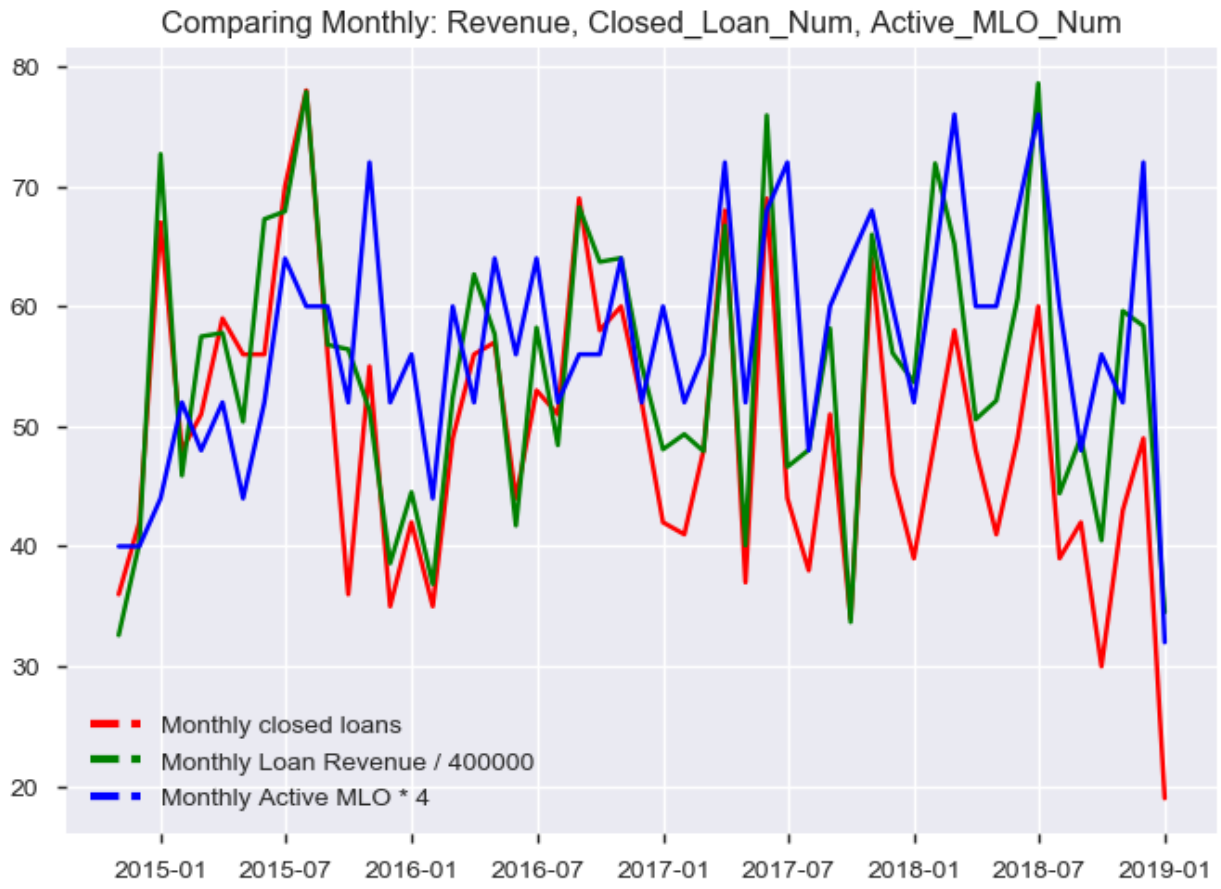


Interest Rate is currently historical low. In the short run rate may go ups and down but in the long run rate will go up. As housing price goes up, interest rate will go up to control the housing price.

Now we will compare Monthly Revenue, Monthly Closed Loan Number and Active Mortgage Loan Originators. We will count number of MLO actively closing loans on any given month.

```
mlo_num=data[['Loan Officer Name']]
mlo_num['date'] = pd.DatetimeIndex(data['Created Date'])
mlo_num = mlo_num.set_index('date')
monthly_mlo_num=mlo_num.resample('M').nunique()
from matplotlib.lines import Line2D
```

```
colors = ['red', 'green', 'blue']
lines = [Line2D([0], [0], color=c, linewidth=3, linestyle='--') for c in colors]
labels = ['Monthly closed loans', 'Monthly Loan Revenue / 400000', 'Monthly Active MLO * 4']
plt.plot(monthly_loan_num_data, color='red')
plt.plot(monthly_loan_rev_data/400000, color='green')
plt.plot(monthly_mlo_num*4, color='blue')
plt.legend(lines, labels)
plt.title('Comparing Monthly: Revenue, Closed_Loan_Num, Active_MLO_Num')
plt.show()
```

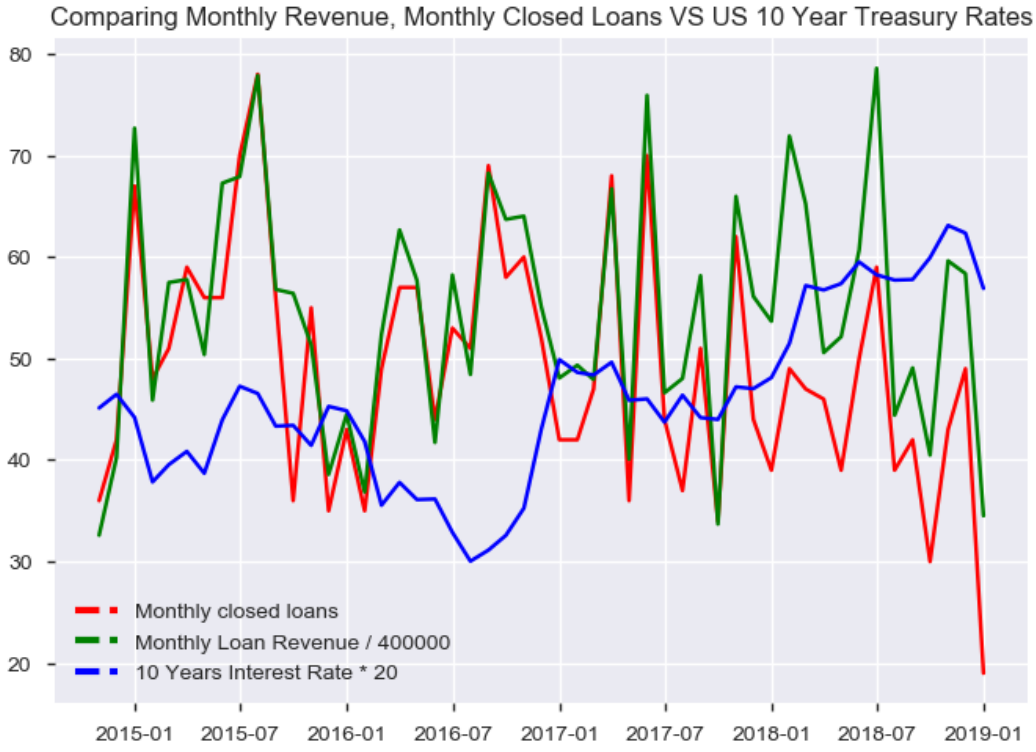


As we know that number of producer is essential component in any given business. MLO (Mortgage loan Originator) is core component in Mortgage business. Many MLO works indedendently and interect directly to clients, involve in marketting and grow their business. There could be many MLO in Mortgage Bank, but active MLO generate more reverue for the bank. As number of active MLO goes up, which will directly and positively impact numbers of loan closed per month, eventually mortgage revenue will go up. On the other hand, once number of active MLO goes down, mortgage revenue and number of loan per month goes down as well. By visualizing the graphs, we can see that monthly data of closed loan numbers , monthly revenue and active MLO numbers ber months, all moving at the same direction.

Let’s find out interest rate effect on Monthly Closed Loans and Monthly Revenue.

```
from matplotlib.lines import Line2D
colors = ['red', 'green', 'blue']
lines = [Line2D([0], [0], color=c, linewidth=3, linestyle='--') for c in colors]
labels = ['Monthly closed loans', 'Monthly Loan Revenue / 400000', '10 Years Interest Rate * 20']
plt.plot(monthly_loan_num_data, color='red')
plt.plot(monthly_loan_rev_data/400000, color='green')
plt.plot(monthly_rate_data*20, color='blue')
plt.legend(lines, labels)
plt.title('Comparing Monthly Revenue, Monthly Closed Loans VS US 10 Year Treasury Rates')
plt.show()
r_monthly_loan_num_data_monthly_loan_rev = pearson_r(monthly_loan_num_data,
monthly_loan_rev_data)
print('Pearson correlation coefficient between Monthly Closed Loans & Monthly loan Rev: ',
r_monthly_loan_num_data_monthly_loan_rev)
print('Pearson correlation coefficient between Monthly Interest & Monthly loans Closed Data: ',
r_monthly_Rate_monthly_loan_data )
```

Pearson correlation coefficient between Monthly Closed Loans & Monthly loan Rev: 0.8295841987344892
Pearson correlation coefficient between Monthly Interest & Monthly loans Closed Data: -0.3343475255276081



We can see the strong positive correlation between Monthly Closed Loans and Monthly Revenue. This graph also suggest that, as interest rates goes down, banks monthly revenue and numbers of loans increases, and when the Rates goes up, both Monthly Closed Loans and Monthly Revenue for the Mortgage bank decline. Pearson correlation coefficient between Monthly Interest & Monthly loans

Closed Data is **-0.334**, which clearly proves that Monthly Interest Rates & Monthly loans Closed Data is **negatively correlated**.

```
# Acquire 1000 bootstrap replicates of Pearson r
bs_replicates_cltv_fico = draw_bs_pairs(
    cltv_data, fico, pearson_r, size=1000)
bs_replicates_loan_num_loan_rev = draw_bs_pairs(
    monthly_loan_num_data, monthly_loan_rev_data, pearson_r, size=1000)
# Compute 95% confidence intervals
conf_int_cltv_fico = np.percentile(bs_replicates_cltv_fico, [2.5, 97.5])
conf_int_loan_num_loan_rev = np.percentile(bs_replicates_loan_num_loan_rev, [2.5, 97.5])
print('CLTV data VS Qualification FICO Data      :', r_cltv_data_fico, conf_int_cltv_fico)
print('Monthly Loan_num_data VS. Monthly_loan_rev :',
r_monthly_loan_num_data_monthly_loan_rev, conf_int_loan_num_loan_rev)
```

Acquire 1000 **pairs bootstrap replicates of the Pearson correlation coefficient using the draw_bs_pairs()** function you wrote in the previous exercise for **CLTV data VS Qualification FICO** Data and **Monthly Loan_num_data VS. Monthly_loan_rev**. Compute the **95% confidence** interval for both using your bootstrap replicates. We have created a NumPy array of percentiles to compute. These are the 2.5th, and 97.5th. By creating a list and convert the list to a NumPy array using np.array(). For example, np.array([2.5, 97.5]) would create an array consisting of the **2.5th and 97.5th percentiles**.

CLTV data VS Qualification FICO Data   : -0.037060429592168175 [-0.08 0.]
Monthly Loan_num_data VS. Monthly_loan_rev : 0.8295841987344892 [0.73 0.9]

''' It shows that there is **statistically significant relationship** between number of loans closed and loan revenue'''

Random Walk

Are Interest Rates or Monthly Loan Returns Prices a Random Walk?

Most returns prices follow a random walk (perhaps with a drift). We will look at a time series of Monthly Sales Revenue, and run the 'Augmented Dickey-Fuller Test' from the statsmodels library to show that it does indeed follow a random walk. With the ADF test, the "null hypothesis" (the hypothesis that we either reject or fail to reject) is that the series follows a random walk. Therefore, a low p-value (say less than 5%) means we can reject the null hypothesis that the series is a random walk.

Print out just the p-value of the test (adfuller_loan_rev_data[0] is the test statistic, and adfuller_loan_rev_data[1] is the p-value). Print out the entire output, which includes the test statistic, the p-values, and the critical values for tests with 1%, 10%, and 5% levels.

```
# Import the adfuller module from statsmodels
from statsmodels.tsa.stattools import adfuller
monthly_loan_rev_data
```

```
# Run the ADF test on the monthly_loan_rev_data series and print out the results
adfuller_loan_rev_data = adfuller(monthly_loan_rev_data)
print(adfuller_loan_rev_data)
print('The p-value of the test on loan_rev is: ' + str(adfuller_loan_rev_data[1]))
```

(-4.49967715626648, 0.00019690419763896495, 10, 40, {'1%': -3.6055648906249997, '5%': -2.937069375, '10%': -2.606985625}, 1307.285137861741)
The p-value of the test on loan_rev is: 0.00019690419763896495

'''According to this test, p-value is very low (lower than 0.05). We reject the hypothesis that monthly_loan_rev_data follow a random walk. '''

Let's try same for Monthly Loan Data:

```
# Import the adfuller module from statsmodels
from statsmodels.tsa.stattools import adfuller
monthly_loan_num_data
# Run the ADF test on the monthly_loan_num_data series and print out the results
adfuller_loan_num_data = adfuller(monthly_loan_num_data)
print(adfuller_loan_num_data)
print('The p-value of the test on loan_num is: ' + str(adfuller_loan_num_data[1]))
```

(-5.7659481612690495, 5.532460937310067e-07, 0, 50, {'1%': -3.568485864, '5%': -2.92135992, '10%': -2.5986616}, 300.2408550906095)
The p-value of the test on loan_num is: 5.532460937310067e-07

According to this test, p-value is very low (lower than 0.05). We reject the hypothesis that monthly_loan_num_data follow a random walk.
Let's try same for Interest Rate Data:

```
monthly_rate_data=monthly_rate['RATE']
type(monthly_rate_data)
# Run the ADF test on the monthly_rate_data series and print out the results
adfuller_monthly_rate_data = adfuller(monthly_rate_data)
print(adfuller_monthly_rate_data)
print('The p-value of the test on monthly_rate_data is: ' + str(adfuller_monthly_rate_data[1]))
```

(-1.396544767435691, 0.5839314748568241, 1, 49, {'1%': -3.5714715250448363, '5%': -2.922629480573571, '10%': -2.5993358475635153}, -34.97121939430423)
The p-value of the test on monthly_rate_data is: 0.5839314748568241

According to this test, **p-value is very is higher than 0.05**. We **cannot reject** the hypothesis that Monthly Interest Rate prices follow a random walk.

Are Interest Rates Auto correlated?

When we look at daily changes in interest rates, the autocorrelation is close to zero. However, if we resample the data and look at monthly or annual changes, the autocorrelation is negative. This implies that while short term changes in interest rates may be uncorrelated, long term changes in interest rates are negatively auto correlated. A daily move up or down in interest rates is unlikely to tell us anything about interest rates tomorrow, but a move in interest rates over a year can tell us something about where interest rates are going over the next year. And this makes some economic sense: over long horizons, when interest rates go up, the economy tends to slow down, which consequently causes interest rates to fall, and vice versa. If we want to look at the data by monthly and annually. We can easily resample and sum it up. I'm using 'M' as the period for resampling which means the data should be resampled on a month boundary and 'A' for annual data'. Finally find the autocorrelation of annual interest rate changes.

```
US10Y['change_rates'] = US10Y.diff()
US10Y['change_rates'] = US10Y['change_rates'].dropna()
# Compute and print the autocorrelation of daily changes
autocorrelation_daily = US10Y['change_rates'].autocorr()
print("The autocorrelation of daily interest rate changes is %4.2f" %(autocorrelation_daily))
```

The autocorrelation of daily interest rate changes is -0.06

```
US10Y.index = pd.to_datetime(US10Y.index)
annual_rate_data = US10Y['RATE'].resample(rule='A').last()
# Repeat above for annual data
annual_rate_data['diff_rates'] = annual_rate_data.diff()
annual_rate_data['diff_rates'] = annual_rate_data['diff_rates'].dropna()
print(annual_rate_data['diff_rates'])
autocorrelation_annual = annual_rate_data['diff_rates'].autocorr()
print("The autocorrelation of annual interest rate changes is %4.2f" %(autocorrelation_annual))
```

DATE
2015-12-31 0.10
2016-12-31 0.18
2017-12-31 -0.05
2018-12-31 0.37
Freq: A-DEC, Name: RATE, dtype: float64
The autocorrelation of annual interest rate changes is -0.97

'''Notice how the daily autocorrelation is small but the annual autocorrelation is large and negative'''

4.2.3 Changes in regulations

Another factor that impacts the amount of mortgage applications is changes in regulations. Depending on the type of regulations change and the impact of the change, there is generally an increase or decrease in mortgage applications before and after the regulations change. Changes in the mortgage loan regulations include amongst others changes in the maximum mortgage, also called the Loan-To-Value (LTV) ratio, and changes in the mortgage interest deduction. The LTV ratio is a financial term that indicates the ratio of the mortgage loan to the value of the property.

4.2.4 Other predictors

Besides the mortgage interest rates and changes in regulations, several other predictors were mentioned in the literature and interviews. These will be discussed briefly below. Furthermore, besides looking at the predictors of the amount of mortgage applications, we will also look at factors that affect the housing market. Since there is a strong relationship between the housing market and the mortgage market (i.e. the amount of houses sold and the amount of mortgage applications), we can assume that the factors that influence the housing market may affect the amount of mortgage applications. The amount of mortgage applications are low in Winter and there are peaks in May, June, July. We also expect a drop in the amount of mortgage applications during the weekends, as most of the mortgage advisors are not working during these days. Furthermore, holidays are expected to have a negative impact on the amount of mortgage applications. Finally, even though there seems to be no autocorrelation between the amount of mortgage applications over time, we still want to include the historical amount of mortgage applications in our model. Even though the amount of mortgage applications are not dependent on the amount of mortgage applications for the previous day, week or month, there is still a time component present and there may be a correlation between these factors.

First of all, the average house prices and expected changes in house prices have an effect on the amount of mortgage applications. If house prices are expected to increase, people might think it is a good moment to buy a house, and thus more mortgage applications may be coming in at the financial institutions. House prices do not only influence the amount of mortgage applications, but mortgage applications in return also influence house prices.

Second, rental prices are important. If the cost of renting is high, buying becomes more attractive compared to renting. Third, the amount of houses available has an effect on the amount of mortgages sold. If the supply of houses is large, the average house prices are going to drop and it will become more attractive to buy a house. All of these factors influence the amount of houses sold and thus the amount of mortgage applications. In the interviews each of these factors was identified as a possible predictor.

The economic features may also impact the amount of mortgage applications, but this is generally caused by the media. If there are a relatively large number of news stories about economic growth or an economic crisis within a short period of time, then this will have a certain impact on the consumers.

4.3 PREDICTIVE ANALYTICS

Predictive analytics is a field in data mining that encompasses different statistical and machine learning techniques that are aimed at making empirical predictions. These predictions are based on empirical data, rather than predictions that are based on theory only. In predictive analytics, several statistical and machine learning techniques can be used to create predictive models. These models are used to exploit patterns in historical data, and make use of these patterns in order to predict future events. These

models can be validated using different methods to determine the quality of such a model, in order to see which model performs best.

There are generally two types of problems predictive analytics is used for: classification problems and regression problems. The main difference between these two problems is the dependent variable, the target variable that is being predicted. In classification problems, the dependent variable is categorical (e.g. credit status). In regression problems, the dependent variable is continuous (e.g. pricing).
The techniques that are used in predictive analytics to create a model depend heavily on the type of problem. For classification problems, classification techniques are used such **Random Forest and decision trees**. These techniques often consist out of one or multiple algorithms that can be used to construct a model. For decision trees, some of the algorithms are Classification and Regression Trees.

For regression problems, **regression techniques** such as multiple **linear regression, support vector machines or time series** are used. These techniques focus on providing a mathematical equation in order to represent the interdependencies between the independent variables and the dependent variable, and use these to make predictions. One of the most popular regression techniques is linear regression. When applied correctly, regression is a powerful technique to show the relationships between the independent and the dependent variables. However, linear regression requires some assumptions in the dataset. One of these assumptions is that there has to be a linear interdependency between the independent variables and the dependent variable. A pitfall of linear regression is that the regression line contains no information about the distribution of the data. It needs to be combined with a visualization of the regression line in order to draw conclusions.

It can be seen that different datasets that have the same **means, variances, correlation** and linear fit, still have a completely different distribution, even though their regression lines are the same. Hence, a regression line always needs to be combined with a visualization in order to draw conclusions about the distribution of the data.

In order to compensate for the disadvantages of the individual models, ensemble models can be used. An ensemble model is a set of individually trained models, which predictors are combined to increase the predictive performance. Ensemble models are generally more accurate than any of the individual models that make up the ensemble model.

Examples of techniques used for creating an **ensemble model** are bagging and boosting. With bagging, multiple versions of a predictor are used to create an aggregated predictor, in order to increase the accuracy of the model. An example of a bagging algorithm is **random forests**, which combines a set of **decision trees** to increase the model performance. A combination of the machine learning techniques mentioned above can be used to create predictive models. These models can then be validated and compared based on predictive power, which can be calculated using a set of statistical measures, such as **Mean Absolute Error (MAE), Root Mean Square Error (RMSE) and R^2**

5 DATA UNDERSTANDING

The Data Understanding stage has been split up in two parts: data collection and data exploration. In the data collection we will discuss characteristics of the event log data, and how the data has been extracted from the database. In the data exploration, the data and its characteristics are explored to extract useful information for our models.

5.1 DATA COLLECTION

Since we are only interested in the event log data we will only be using one of the tables. This table contains data about every mortgage application. Every action performed by the system or by a user on a mortgage application is logged, and the status before and after that specific action is logged. For our analysis we are mainly interested in the date and time at which each of the mortgage applications have entered the system.

Besides Mortgage Application DataSet, we have joined two separate (**10 Years US Treasury Rate, Home Supply Index)** with our existing Mortgage Application DataSet to enhance predictive power of our model.

```
US10Y.replace('.', -99999, inplace=True)
US10Y= US10Y.replace(to_replace=[-99999], value=[np.nan])
US10Y= US10Y.fillna(method='ffill')
US10Y= US10Y[['RATE']].astype('float64')
total_missing_rate=US10Y.isnull().sum()
model_data1['Loan Amount'].pct_change()
US10Y['RATE'].pct_change()
US10Y['RATE'].autocorr()
US10Y['RATE'].diff()
```

#Join two DataFrames model_data1 & US10Y save the results in model_data

```
model_data2 = model_data1.join(US10Y)
total_missing=model_data2.isnull().sum()
model_data2= model_data2.fillna(method='ffill')
total_missing=model_data2.isnull().sum()
```

#Integrate Monthly housing supply index data and merging with current dataset

```
total_missing_home_supply=home_supply.isnull().sum()
model_data = model_data2.join(home_supply)
total_missing=model_data.isnull().sum()
model_data= model_data.fillna(method='ffill')
total_missing=model_data.isnull().sum()
```

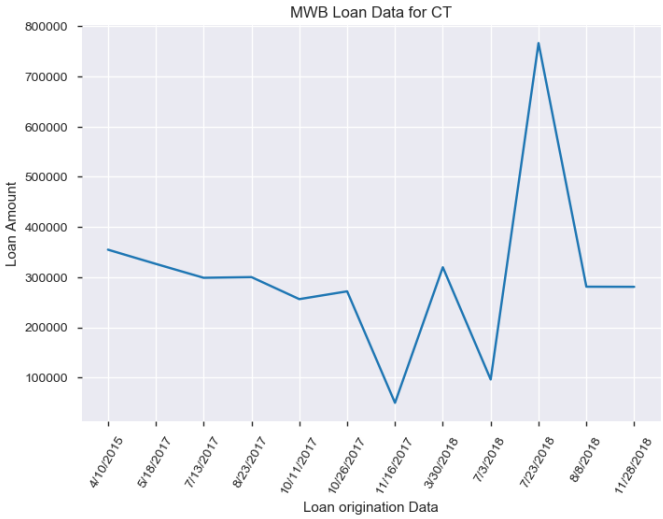
5.2 DATA EXPLORATION

Since our dataset can be grouped per day to create meaningful visualizations. The dataset contains data from October 2014 until December 2018. In order to get a feel of the amount of mortgage applications per day and the distribution of the mortgage applications, different visualizations can be made using Python. Two graphs have been created, which can be found in Figure 1 and Figure 2. Both of these graphs only contain the amount of mortgage applications on the weekdays. As there are almost no applications coming in on the weekends they have been excluded from the graphs.
As can be seen from the graphs, there seems to be a seasonal pattern on a monthly level, but from these graphs it is not very clear. It also seems like there are some outliers, so these data points will have to be investigated to see if they will have to be included in our model, as there can be multiple underlying reasons for outliers in our dataset. It also seems there is an increase in mortgage applications during the last few months of each year. The amount of applications per day during these months is higher compared to the other months. This can have multiple explanations so this will have to be accounted for in the model.

The density plot shows the distribution of the amount of mortgage applications. It seems the distribution of the amount of mortgage applications is normally distributed, slightly skewed to the right with a long tail. This is due to the outliers mentioned before. The median seems to be at around mortgage applications per day.

Let's plot number of Loan Application for Connecticut:

```
#Plot loan amount per Statae
plt.xlabel('Loan origination Data')
plt.xticks(rotation=60)
plt.ylabel('Loan Amount')
plt.title('MWB Loan Data for CT')
plt.plot(ct_data['Created Date'], ct_data['Loan Amount'])
```



We can see a spike of new loan application on summer 2018 for state of Connecticut.

We can analyze mortgage loan origination for the bank per state; we also can find the total counts of different types of loan for the bank. Based on this information bank can allocate resources:

```
print('=================================================')
print('=========    Total Sales by State    ==============')
print(' ')
print('Total Sales in Cunnecticut   : $', ct_loan_amount)
print('Total Sales in Florida       : $', fl_loan_amount)
print('Total Sales in New York      : $', ny_loan_amount)
print('Total Sales in New Jersey    : $', nj_loan_amount)
print('Total Sales in Pennsylvania  : $', pa_loan_amount)
print(' ')
print('=================================================')
print(' ')
########################################
loan_types=data['Loan Type'].unique()
group_loan_types=data.groupby(data['Loan Type']).size()
print('Unique Loan Types       : ', loan_types)
print(' ')
print('=================================================')
print('Number of loan per Types : ', group_loan_types)
```

========= Total Sales by State ==============

Total Sales in Connecticut : $ 3604641.0
Total Sales in Florida : $ 3371495.0
Total Sales in New York : $ 868836552.28
Total Sales in New Jersey : $ 236456261.06
Total Sales in Pennsylvania : $ 600920.0

===

Unique Loan Types : ['Residential' 'FHA' 'Commercial' 'Conventional' 'Other' 'VA']

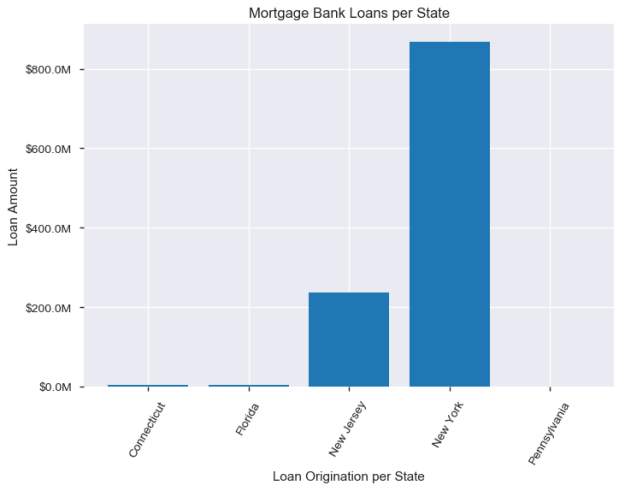
===

Number of loan per Types : Loan Type	
Commercial	***100***
Conventional	***1736***
FHA	***418***
Other	***31***
Residential	***215***
VA	***1***

Let's visualize the loan data per state;

```
from matplotlib.ticker import FuncFormatter
x = np.arange(5)
```

```
money = [1.5e5, 2.5e6, 5.5e6, 1.0e7, 2.0e7, 3.0e7, 4.0e7, 5.0e7, 6.0e7]
def millions(x, pos):
    'The two args are the value and tick position'
    return '$%1.1fM' % (x * 1e-6)
formatter = FuncFormatter(millions)
fig, ax = plt.subplots()
ax.yaxis.set_major_formatter(formatter)
plt.bar(St, loan_amount_per_state)
plt.xticks(x, ('Connecticut', 'Florida', 'New Jersey', 'New York', 'Pennsylvania'))
plt.ylabel('Loan Amount')
plt.xlabel('Loan Origination per State')
plt.xticks(rotation=60)
plt.title('Mortgage Bank Loans per State')
plt.show()
```



We can see that New York and New Jersey is the major Loan Origination market for the Bank. Let's explore Loan Origination Data with Unit Types:

```
total_unit_type = data.groupby(data['Unit Type']).size()
print('Loan originated in all States per unit types : \n', total_unit_type)
```

==========

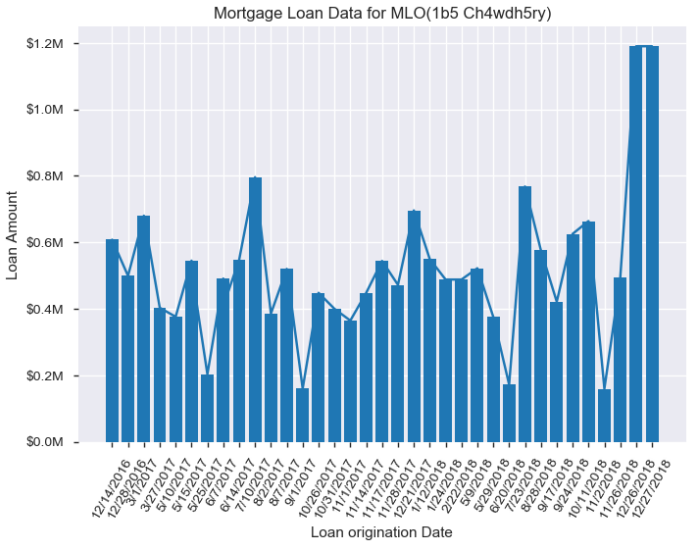
Loan originated in all States per unit types :

Unit Type	
Condo	**216**
Coop	**76**
FourFamily	**36**
Industrial	**1**
Land	**1**
MixedUse	**25**
MultiFamily	**20**
OneFamily	**1276**
PUD	**6**
ThreeFamily	**129**
TwoFamily	**707**
Warehouse	**8**

We can see that One Family and Two Family property dominates the Loan Origination shares for the mortgage bank. As a result company should focus on suitable loan products particularly for 1-4 family loan products. This Mortgage Bank also originates good number of Condo and Coop.

We can analyze data for individual MLO (Mortgage Loan Originator)

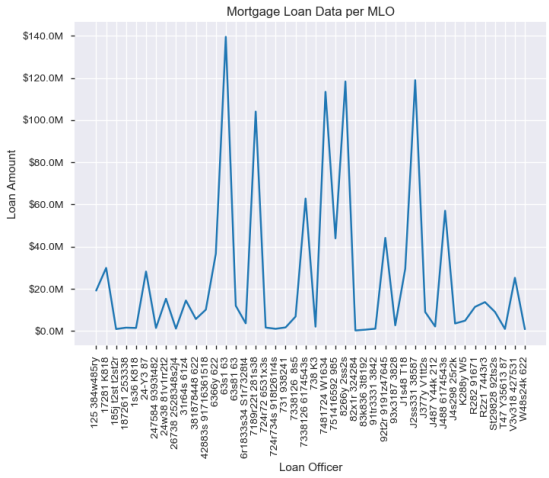
```
formatter = FuncFormatter(millions)
fig, ax = plt.subplots()
ax.yaxis.set_major_formatter(formatter)
plt.xlabel('Loan origination Date')
plt.xticks(rotation=60)
plt.ylabel('Loan Amount')
plt.title('Mortgage Loan Data for MLO(1b5 Ch4wdh5ry)')
plt.plot(abu_loan['Created Date'], abu_loan['Loan Amount'])
plt.xlabel('Loan origination Date')
plt.xticks(rotation=60)
plt.ylabel('Loan Amount')
plt.title('Mortgage Loan Data for MLO(1b5 Ch4wdh5ry)')
plt.bar(abu_loan['Created Date'], abu_loan['Loan Amount'])
```



We can easily explore the Loan Origination history for particular MLO

Here is total Sales volume per MLO.

```
formatter = FuncFormatter(millions)
fig, ax = plt.subplots()
ax.yaxis.set_major_formatter(formatter)
plt.xlabel('Loan Officer')
plt.ylabel('Loan Amount')
plt.title('Mortgage Loan Data per MLO')
plt.xticks(rotation=90)
plt.plot(sales_totals_lo)
```



Some MLO generates major portions of the sales revenue for the Bank, many of the MLO is performing way below company standards. Additional research required for the root causes behind the performance. Let's explore the Loan Statistics data for the Mortgage Bank:

*********** *Loan Statistics for Mortgage Bank* **********

Average Loan Amount is	***: $ 445099.91***
Median Loan Amount is	***: $ 400000.00***
Standard deviation of is	***: $ 288803.96***
Minimum Loan Amount is	***: $ 23600.00***
Maximum Loan Amount is	***: $ 6125000.00***
Total of Loan Amount is	***: $ 1113194869.34***
10% of Loan Amount is below	***: $ 200000.00***
25% of Loan Amount is below	***: $ 292500.00***
50% of Loan Amount is below	***: $ 417000.00***
75% of Loan Amount is below	***: $ 533000.00***
90% of Loan Amount is below	***: $ 696500.00***

FICO Score is one of the critical information for processing mortgage loan application.

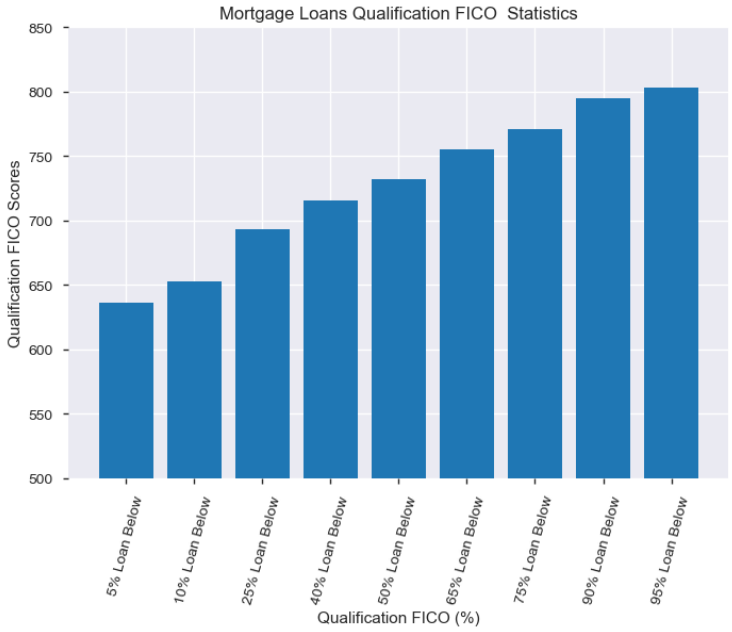
********** *Qualification FICO Statistics for Mortgage Bank* **********

Average FICO is	***: $ 727.81***
Median FICO is	***: $ 732.00***
Standard deviation is	***: $ 56.61***
Minimum FICO is	***: $ 0.00***
Maximum FICO is	***: $ 825.00***
5% of FICO is below	***: $ 636.00***
10% of FICO is below	***: $ 653.00***
25% of FICO is below	***: $ 693.00***
40% of FICO is below	***: $ 716.00***
50% of FICO is below	***: $ 732.00***
65% of FICO is below	***: $ 755.00***
75% of FICO is below	***: $ 771.00***
90% of FICO is below	***: $ 795.00***
95% of FICO is below	***: $ 803.00***

==========

We can see that 50% of the consumers FICO score is between 693 and 771. Let's visualize FICO statistics
fico_score = [fico_q7, fico_q0, fico_q1, fico_q5, fico_q2, fico_q6, fico_q3, fico_q4, fico_q8]
fico_pct=['5% Loan Below', '10% Loan Below', '25% Loan Below', '40% Loan Below', '50% Loan Below', '65% Loan Below', '75% Loan Below','90% Loan Below', '95% Loan Below']

```
plt.xticks(rotation=75)
plt.ylim((500,850))
plt.bar(fico_pct, fico_score)
plt.ylabel('Qualification FICO Scores')
plt.xlabel('Qualification FICO (%)')
plt.title('Mortgage Loans Qualification FICO  Statistics')
plt.show()
```

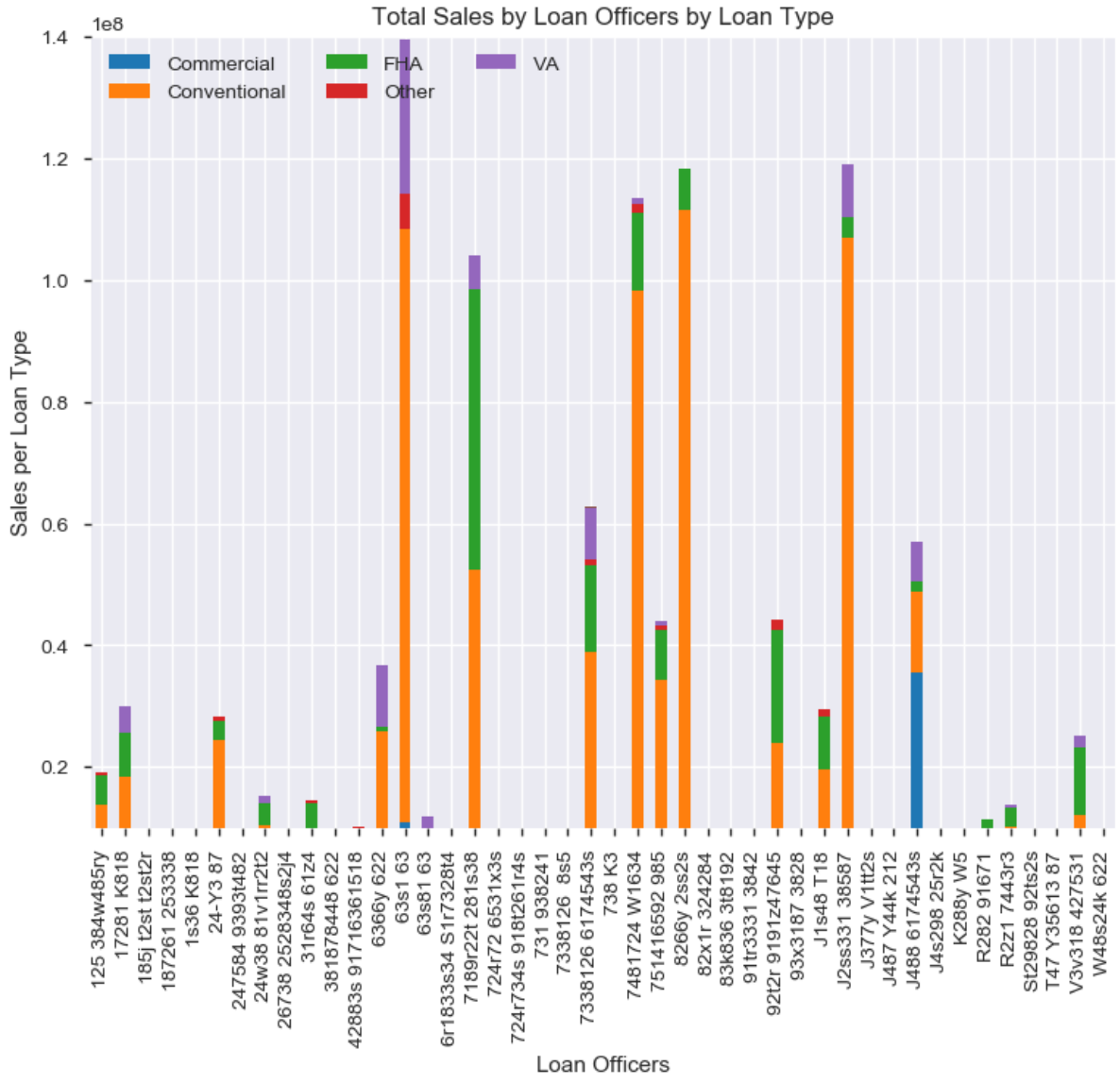


We can see that 90% of the consumers FICO score is over 650. Most of cases applicant with lower FICO score will not qualify for Conventional mortgages. In many cases, FHA Loan type could be only option remaining for the applicants with FICO score; many bank uses cut-off points for FICO Score (640-680) for conventional mortgages. FHA accepts FICO score below 600.

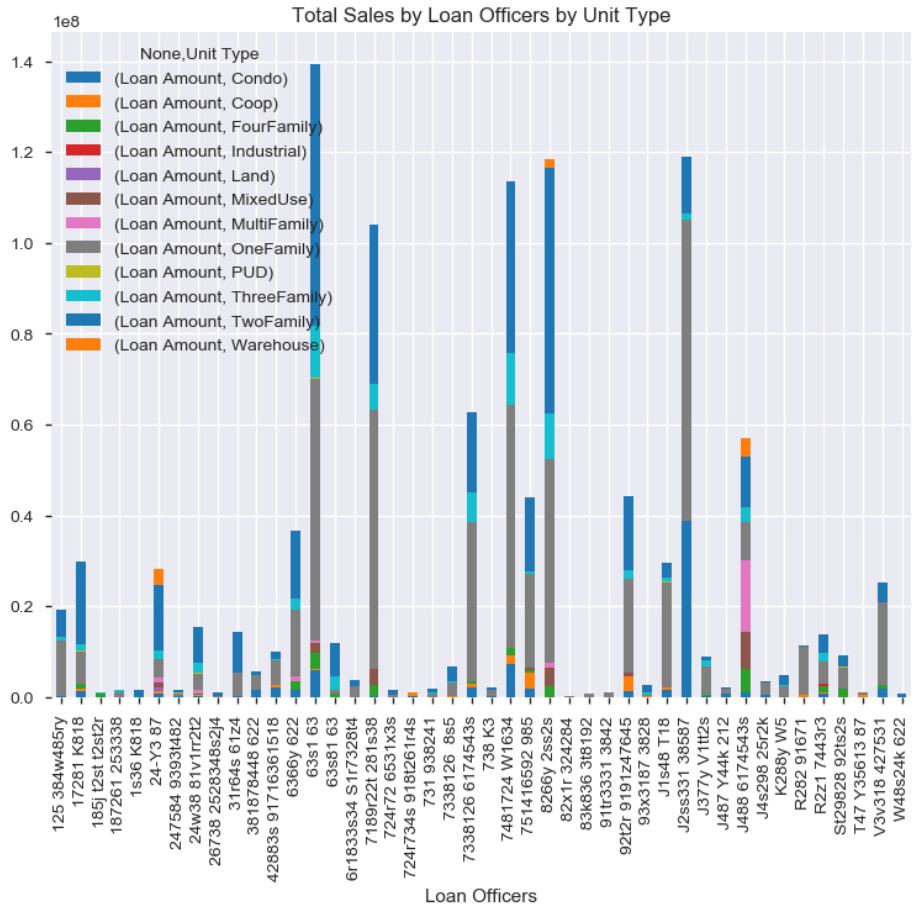
Let's Analyze Mortgage Loan Officer's Sales Volume per Loan Type. Processing different loan types need different expertise. Some loan officer do not have experience for VA or commercial loan at all. Few Loan officers are expert in commercial side of loan origination process. In order to Loan Origination for 1-4 units of residential properties, MLO must be licensed by the State. On the other hand other properties such as multi-family, mixed use or commercial loans do not require MLO to be licensed by the State.

```
print('**************** Mortgage Loan Ofiicer'' Sales Volume per Loan Type
***************')
lo_loan = data[['Loan Officer Name', 'Loan Type', 'Created Date', 'Loan Amount']]
#We can use groupby to organize the data by category and name.
lo_loan_group = lo_loan.groupby(['Loan Officer Name', 'Loan Type']).sum()
lo_loan_group_count = lo_loan.groupby(['Loan Officer Name', 'Loan Type']).count()
```

```
print(lo_loan_group)
print(lo_loan_group_count)
lo_loan['Loan Type'].describe()
'''The category representation looks good but we need to break it apart
to graph it as a stacked bar graph. unstack can do this for us.'''
lo_loan_group.unstack().head()
lo_loan_group_plot = lo_loan_group.unstack().plot(kind='bar',stacked=True,title="Total Sales by
Loan Officers by Loan Type",figsize=(9, 7))
lo_loan_group_plot.set_xlabel("Loan Officers")
lo_loan_group_plot.set_ylabel("Sales per Loan Type")
lo_loan_group_plot.legend(["Commercial","Conventional","FHA","Other","VA"], loc=2,ncol=3)
plt.ylim(10000000, 140000000)
```



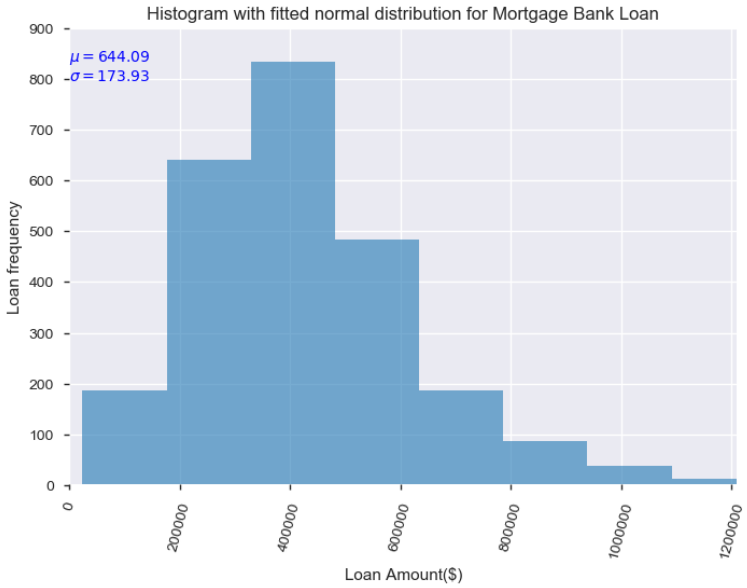
Similarly, we can explore Loan Officer's Sales Volume per Unit Type.



Now that we know who the biggest customers are and how they purchase products,
we might want to look at purchase patterns in more detail. Let's take another look at the data and try to see how large the individual purchases are. A histogram allows us to group purchases together so we can see how big the customer transactions are.

```
from IPython.display import display   # A notebook function to display more complex data (like tables)
import scipy.stats as stats

import numpy as np
from scipy.stats import kurtosis
from scipy.stats import skew
loan_patterns = data[['Loan Amount', 'Created Date']]
loan_patterns_plot = loan_patterns['Loan Amount'].hist(alpha=0.6, bins=40, grid=True)
loan_patterns_plot = loan_patterns['Loan Amount'].apply(np.sqrt)
param = stats.norm.fit(loan_patterns_plot)
x = np.linspace(0, 100000, 1250000)      # Linear spacing of 100 elements between 0 and 20.
pdf_fitted = stats.norm.pdf(x, *param)    # Use the fitted paramters to
loan_patterns_plot.plot.hist(alpha=0.6, bins=40, grid=True, density=True, legend=None)
plt.text(x=np.min(loan_patterns_plot), y=800, s=r"$\mu=%0.2f$" % param[0] + "\n" +
r"$\sigma=%0.2f$" % param[1], color='b')
```

```
# Plot a line of the fitted distribution over the top
# Standard plot stuff
plt.xticks(rotation=75)
plt.ylim((1,900))
plt.xlim((1,1210000))
plt.xlabel("Loan Amount($)")
plt.ylabel("Loan frequency")
plt.title("Histogram with fitted normal distribution for Mortgage Bank Loan")
print( 'excess kurtosis of normal distribution (should be 0): {}'.format(
kurtosis(loan_patterns_plot) ))
print( 'skewness of normal distribution (should be 0): {}'.format( skew(loan_patterns_plot) ))
print("mean : ", np.mean(loan_patterns_plot))
print("var  : ", np.var(loan_patterns_plot))
print("skew : ",skew(loan_patterns_plot))
print("kurt : ",kurtosis(loan_patterns_plot))
```

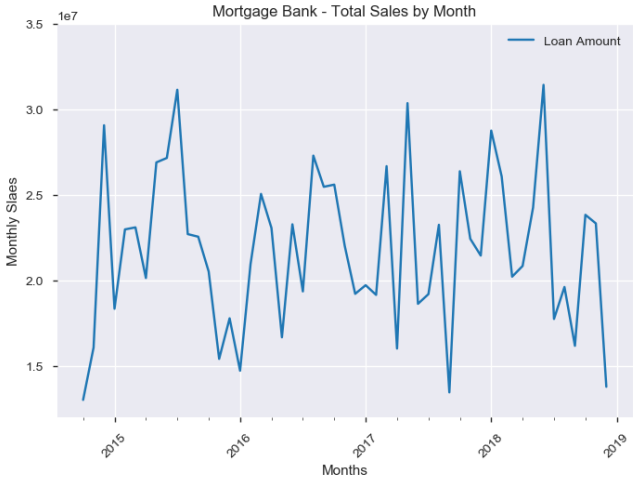


excess kurtosis of normal distribution (should be 0): 10.20423896299732
skewness of normal distribution (should be 0): 1.6231675369974472
mean : 644.086535155362
var : 30252.443004451427
skew : 1.6231675369974472
kurt : 10.20423896299732

The fancy text to show us what the parameters of the distribution are (mean and standard deviation). We can create a histogram with 20 bins to show the distribution of purchasing patterns. Fit a normal distribution to the data then plot the histogram.

One of the really cool things that **Pandas** allows us to do is **resample** the data. We want to look at the data by month, we can easily resample and sum it all up. I'm using 'M' as the period for resampling which means the data should be resampled on a month boundary.

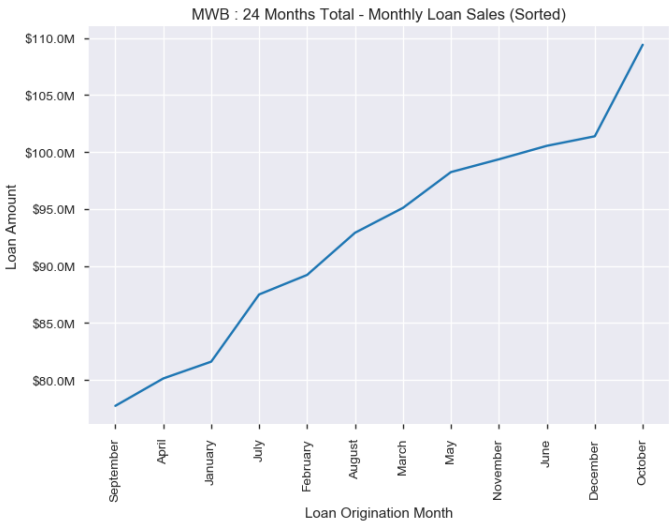
```
loan_patterns.resample('M',how=sum)
monthly_loan_rev=loan_patterns.resample('M',how=sum)
loan_patterns.info()
loan_patterns_month_plot = loan_patterns.resample('M',how=sum).plot(title="Mortgage Bank -
Total Sales by Month",legend=True)
loan_patterns_month_plot.set_xlabel("Months")
loan_patterns_month_plot.set_ylabel("Monthly Slaes")
plt.xticks(rotation=45)
plt.ylim((12000000, 35000000))
fig = loan_patterns_month_plot.get_figure()
```



We can see that monthly mortgage loan sales volume varies between $15M and $32M. Another interesting find is Loan sales are at the peak during summer seasons. Winter sales are normally slow but 2018 was an exception.
data = pd.read_csv('c:/scripts/mwb2014.csv', index_col='Created Date', parse_dates=True, encoding='cp1252')

Best month for Sales

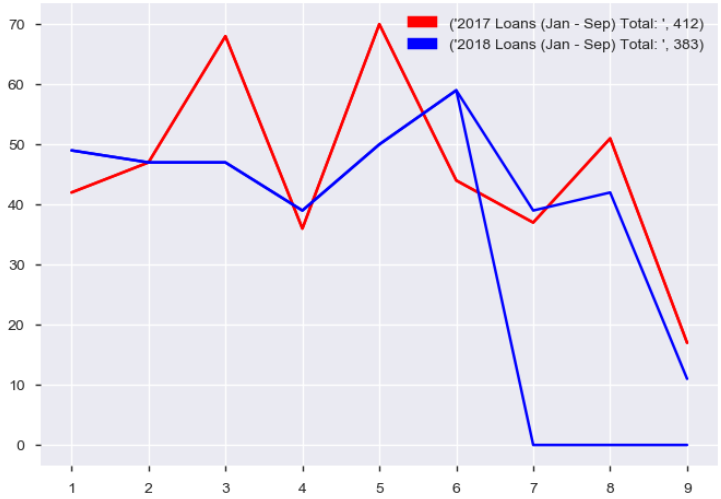
```
monthly_loan_amount_sum = by_month['Loan Amount'].sum()
daily_loan_amount_sum = by_day['Loan Amount'].sum()
yearly_loan_amount_sum = by_year['Loan Amount'].sum()
monthly_loan_amount_sum = monthly_loan_amount_sum
monthly_loan_amount_sum_sorted= monthly_loan_amount_sum.sort_values()
fig, ax = plt.subplots()
ax.yaxis.set_major_formatter(formatter)
plt.xlabel('Loan Origination Month')
plt.ylabel('Loan Amount')
plt.title('MWB : 24 Months Total - Monthly Loan Sales (Sorted)')
plt.xticks(rotation=90)
plt.plot(monthly_loan_amount_sum_sorted)
```



Monthly Worst Sales: January, April and September and Best is June, December and October.

Comparing year-over-year performance:

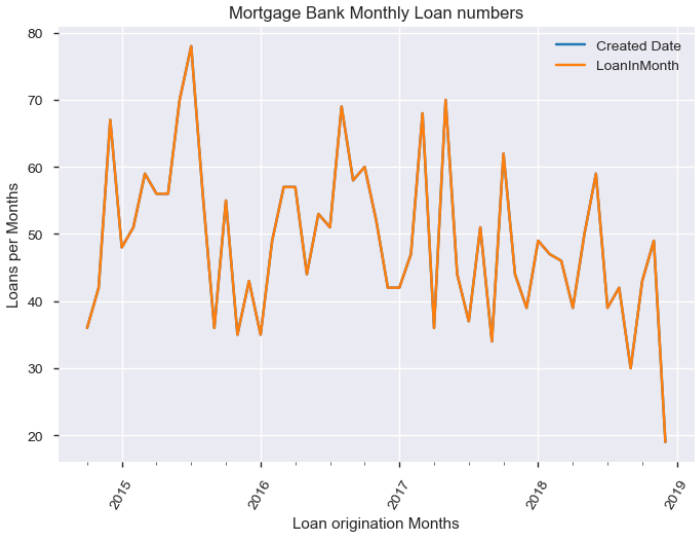
```
plt.plot(M17, color='red')
plt.plot(M18, color='blue')
red_patch = mpatches.Patch(color='red', label=('2017 Loans (Jan - Sep) Total: ', M17['Created Date'].sum()) )
blue_patch = mpatches.Patch(color='blue', label=('2018 Loans (Jan - Sep) Total: ',  M18['Created Date'].sum()) )
plt.legend(handles=[red_patch, blue_patch])
plt.xticks(range(len(M17)), [calendar.month_name[month] for month in M17.index], rotation=60)
plt.xlabel('Loan origination Months')
#plt.xticks(rotation=60)
plt.ylabel('Loans per Months')
plt.title(Mortgage Bank Monthly Loan numbers 2017 & 2018: ')
plt.show()
```



In 2017, Mortgage Bank has closed more loan compare to 2018

In 2017, Mortgage Bank sales revenue was (M_amount_17.sum()) = $180M
For 2018, (M_amount_18.sum())= $197M

monthly_loan_num.resample('M', how='count')
monthly_loan_num_data= monthly_loan_num.resample('M', how='count')
monthly_loan_num_plot = monthly_loan_num.resample('M',how='count').plot(title="Mortgage Bank - Total Sales by Month",legend=True)
plt.xlabel('Loan origination Months')
plt.xticks(rotation=60)
plt.ylabel('Loans per Months')
plt.title('Mortgage Bank Monthly Loan numbers')



r_monthly_loan_num_data_monthly_loan_rev = pearson_r(monthly_loan_num_data, monthly_loan_rev_data)
print('Pearson correlation coefficient between Monthly Closed Loans & Monthly loan Rev: ', r_monthly_loan_num_data_monthly_loan_rev)

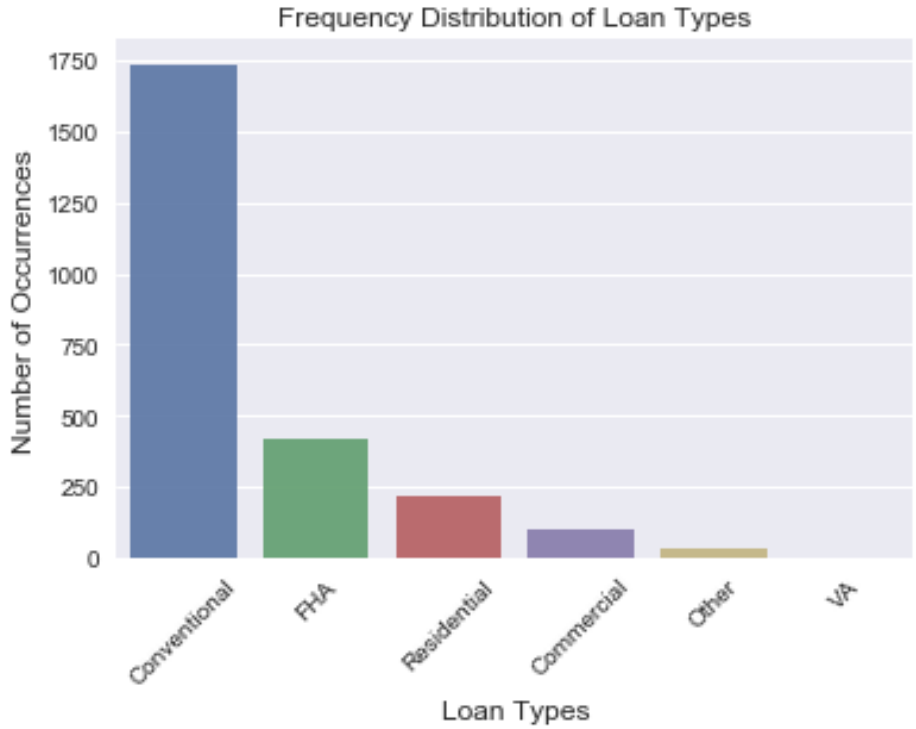
Pearson correlation coefficient between Monthly Closed Loans & Monthly loan Rev: 0.8295841987344892

Monthly Loan numbers and Monthly Sales volumes are strongly positively correlated. Once we analyze number of the loans closed per month, we find the similarity between sales volume and loan numbers. As average loan numbers goes up, total monthly sales volume goes up. Any given months, if the number of loans closed are higher; we find that average loan amounts are low for that particular months. In other words, this may suggest that loan processing requirements and guidelines loans with higher loan amount take longer time to close the loans with lower loan amounts.

'''Visual exploration is the most effective way to extract information between variables.

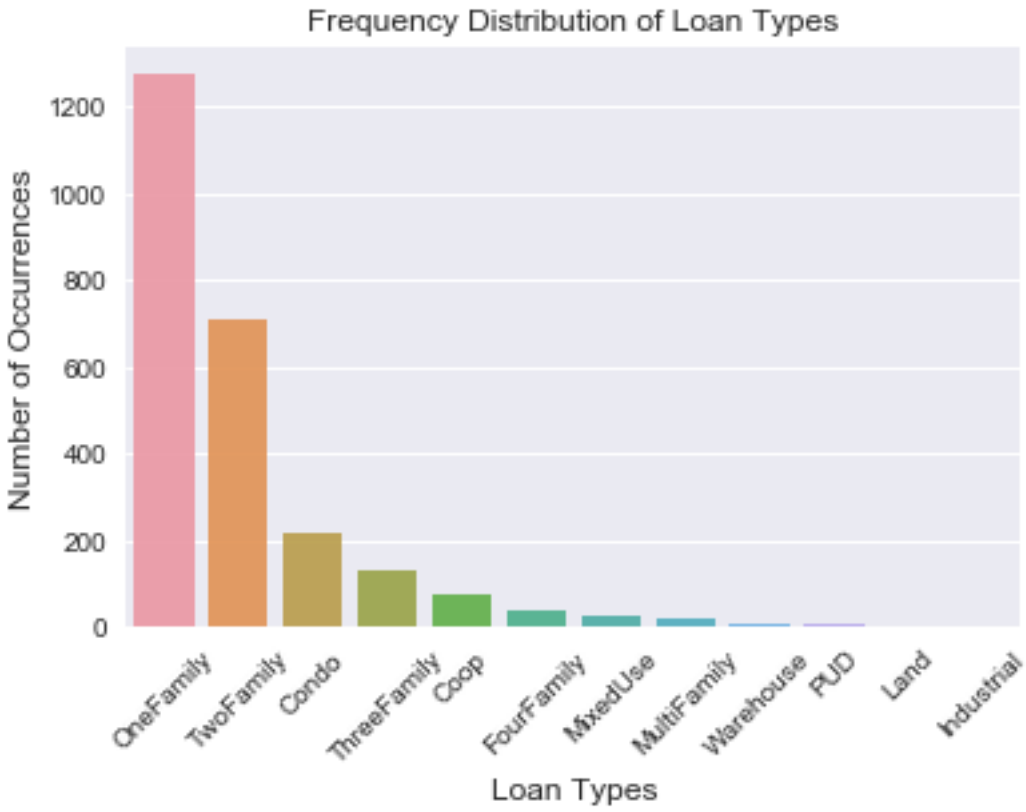
We can plot a barplot of the frequency distribution of a categorical feature using the seaborn package, which shows the frequency distribution of the mortgage dataset column.

```
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt
loan_type_count = data['Loan Type'].value_counts()
sns.set(style="darkgrid")
sns.barplot(loan_type_count.index, loan_type_count.values, alpha=0.9)
plt.title('Frequency Distribution of Loan Types')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Loan Types', fontsize=12)
plt.xticks(rotation=45)
plt.show()
```



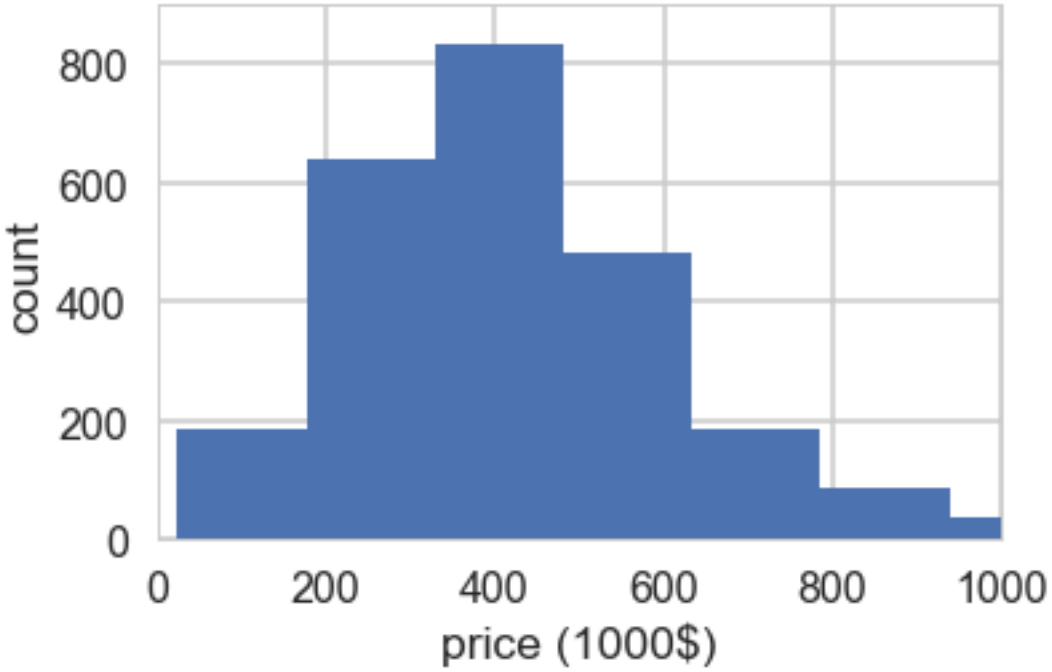

```
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt
unit_type_count = data['Unit Type'].value_counts()
```

```
sns.set(style="darkgrid")
sns.barplot(unit_type_count.index, unit_type_count.values, alpha=0.9)
plt.title('Frequency Distribution of Loan Types')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Loan Types', fontsize=12)
plt.xticks(rotation=45)
plt.show()
```



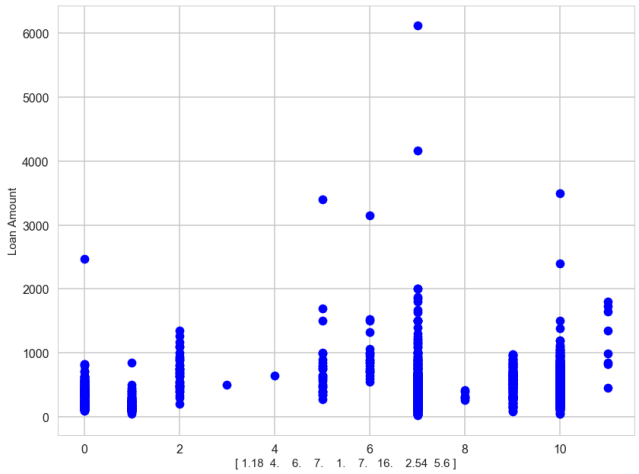

```
%matplotlib inline
import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
import sklearn
import seaborn as sns
# special matplotlib argument for improved plots
from matplotlib import rcParams
sns.set_style("whitegrid")
sns.set_context("poster")

# Target variable
Y=model_data['Loan Amount'].values
#plotting histogram of the target variable Loan Amount
import  matplotlib.pyplot  as plt
plt.figure(figsize=(6, 4))
plt.xlim(0.1,1000)
```

```
plt.ylim(0.8,900)
plt.hist(Y, bins=40)
plt.xlabel('price (1000$)')
plt.ylabel('count')
plt.tight_layout()
```

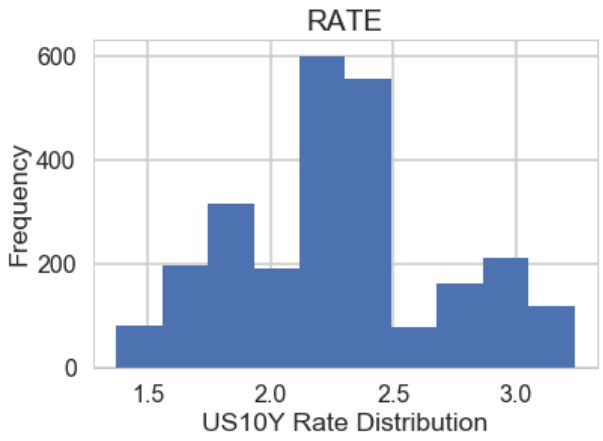


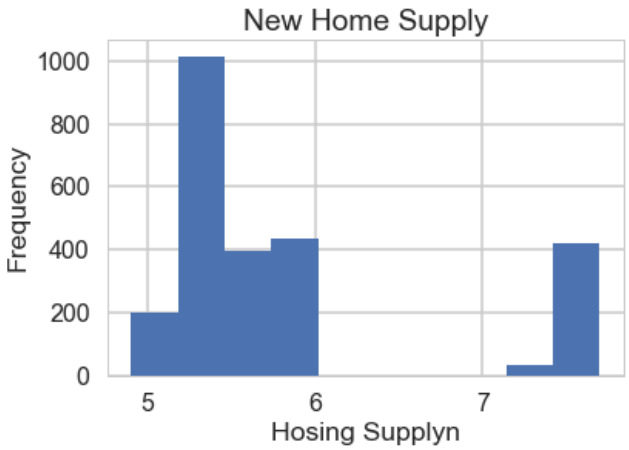
Visualization

```
# Print the scatter plot for each (Column 1-4) feature with respect to price
for index, columns in enumerate(X[1:5]):
    plt.figure(figsize=(12, 9))
    plt.scatter(X[:, index], y, color='g')
    plt.ylabel('Loan Amount', size=15)
    plt.xlabel(columns, size=15)
    plt.tight_layout()
#Print the scatter plot for each (Column 6-9) feature with respect to price
for index, columns in enumerate(X[6:10]):
    plt.figure(figsize=(12, 9))
    plt.scatter(X[:, index], y, color='b')
    plt.ylabel('Loan Amount', size=15)
    plt.xlabel(columns, size=15)
    plt.tight_layout()
```

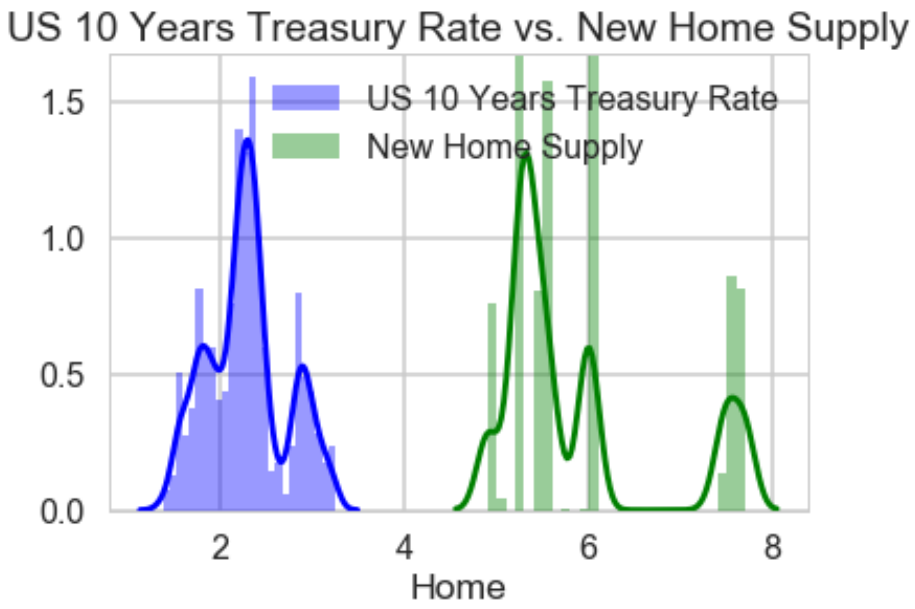



```
plt.hist(model_data['RATE'])  #This distribution is somewhat normal distribution
plt.title("RATE")
plt.xlabel("US10Y Rate Distribution")
plt.ylabel("Frequency")
plt.show()
plt.hist(model_data['Home'])
plt.title("New Home Supply")
plt.xlabel("Hosing Supplyn")
plt.ylabel("Frequency")
plt.show() #The distribution is somewhat left skewed with a mean to the left
```



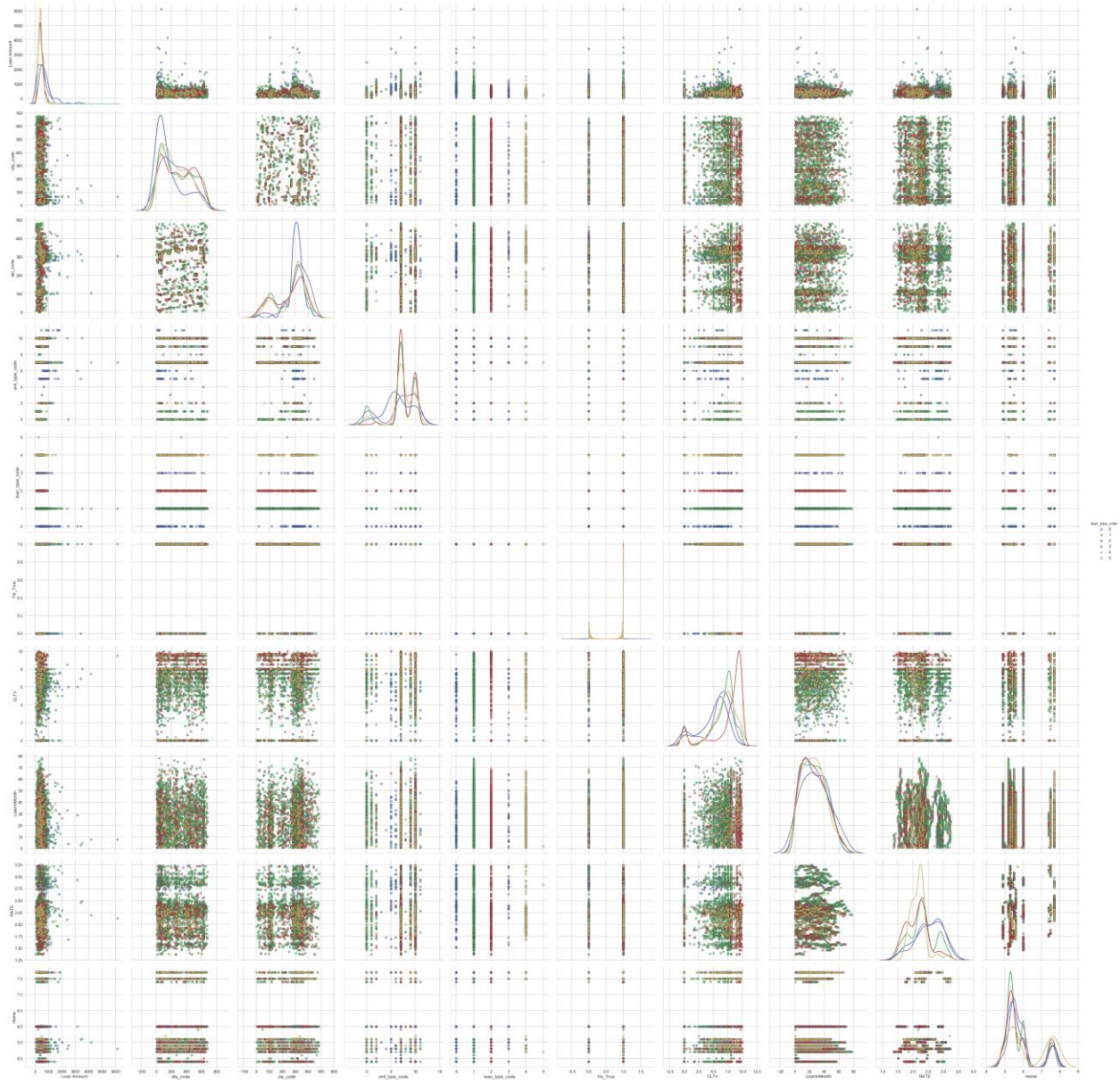


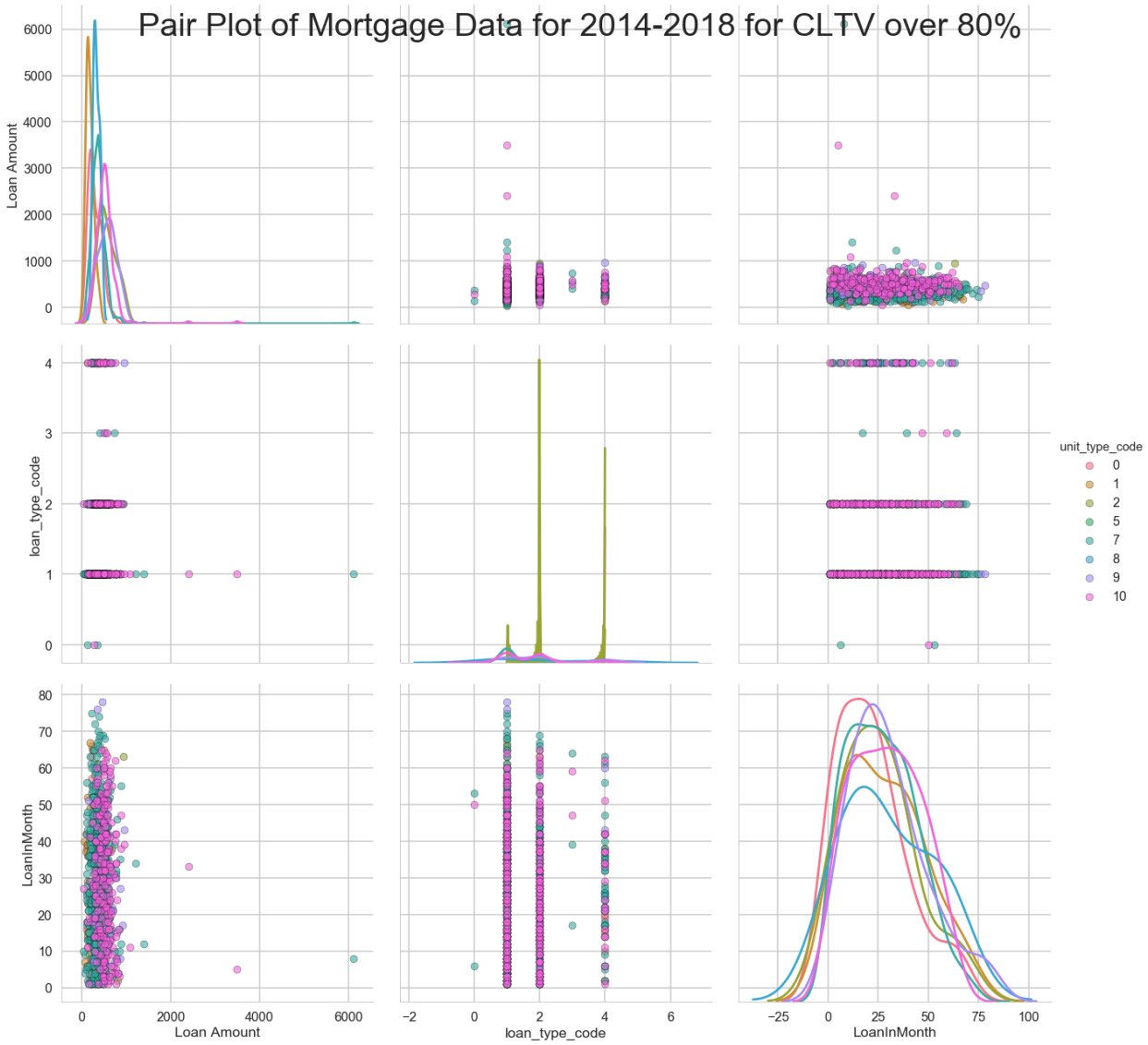

```
#Plot 'RM' and 'PTRATIO' on the same Axis
sns.distplot( model_data['RATE'] , color="blue", label="US 10 Years Treasury Rate")
sns.distplot( model_data['Home'] , color="green", label="New Home Supply")
plt.legend()
plt.title("US 10 Years Treasury Rate vs. New Home Supply")
plt.show()
```



''' Home Supply goes up, RATE goes up ''' '''
The distribution plot comparing US 10 Years Treasury Rate & New Home Supply shows that US 10Y RATE is normally distributed and New Home Supply is skewed to the right.

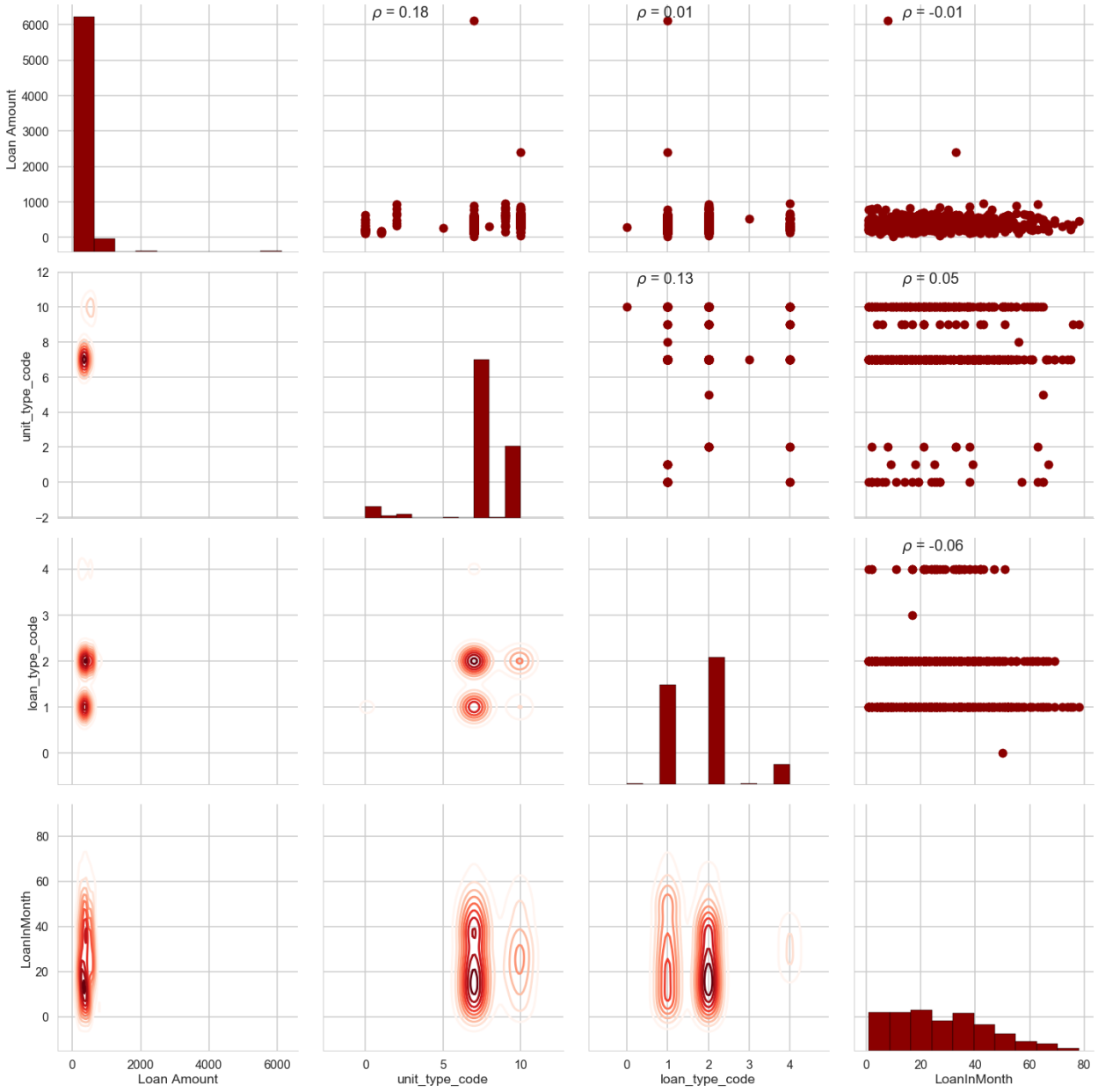
```
# Create a pair plot colored by Loan Types with a density plot of the
# diagonal and format the scatter plots.
sns.pairplot(fit_data, hue = 'loan_type_code', diag_kind = 'kde',  plot_kws = {'alpha': 0.6, 's': 80,
'edgecolor': 'k'},   size = 6)
# Plot colored by continent for years 2000-2007
sns.pairplot(fit_data[fit_data['CLTV'] >= 8], vars = ['Loan Amount', 'loan_type_code',
'LoanInMonth'],
        hue = 'unit_type_code', diag_kind = 'kde',
        plot_kws = {'alpha': 0.6, 's': 80, 'edgecolor': 'k'},    size = 6);
plt.suptitle('Pair Plot of Mortgage Data for 2014-2018 for CLTV over 80%',  size = 40);
```






```
# Function to calculate correlation coefficient between two arrays
def corr(x, y, **kwargs):

    # Calculate the value
    coef = np.corrcoef(x, y)[0][1]
    # Make the label
    label = r'$\rho$ = ' + str(round(coef, 2))
      # Add the label to the plot
    ax = plt.gca()
    ax.annotate(label, xy = (0.2, 0.95), size = 20, xycoords = ax.transAxes)

# Create a pair grid instance
grid = sns.PairGrid(data= fit_data[fit_data['CLTV'] > 8],
                    vars = ['Loan Amount', 'unit_type_code',
       'loan_type_code', 'LoanInMonth'], size = 5)
```

```
# Map the plots to the locations
grid = grid.map_upper(plt.scatter, color = 'darkred')
grid = grid.map_upper(corr)
grid = grid.map_lower(sns.kdeplot, cmap = 'Reds')
grid = grid.map_diag(plt.hist, bins = 10, edgecolor =  'k', color = 'darkred');
```



6 DATA PREPARATION

The Data Preparation stage contains three elements: data pre-processing, feature engineering and feature selection. In the data pre-processing we pre-process the data so that it contains the right format for our models. In the feature engineering we create a list of features, using the predictors. In the feature selection we select a subset of features that are useful for our model.

Encoding Categorical Data

There are different techniques to encode the categorical features to numeric quantities.
The techniques are as following:

- Replacing values
- Encoding labels
- One-Hot encoding
- Binary encoding
- Backward difference encoding
- Miscellaneous features
- Replace Values

Let's start with the most basic method, which is just replacing the categories with the desired numbers. This can be achieved with the help of the replace() function in pandas. The idea is that you have the liberty to choose whatever numbers we want to assign to the categories according to the business use case.

#Categorical value 'Unit Type' has been update based on frequency of the value_counts
it's a good practice to typecast categorical features to a category dtype because they make the operations on such columns much faster than the object dtype. You can do the typecasting by using .astype() method on your columns like shown below:'''

```
data_lc = data.copy()
data_lc['City'] = data_lc['City'].astype('category')
data_lc['Zip'] = data_lc['Zip'].astype('category')
data_lc['Loan Type'] = data_lc['Loan Type'].astype('category')
data_lc['Unit Type'] = data_lc['Unit Type'].astype('category')
data_lc['loan_purpose_code'] = data_lc['loan_purpose_code'].astype('category')
print(data_lc.dtypes)
```

validate the faster operation of the category dtype by timing the execution time of the same operation done on a DataFrame with columns as category dtype and object dtype by using the time library.

```
import time
%timeit data.groupby(['Loan Type','Zip']).count()
%timeit data_lc.groupby(['Loan Type','Zip']).count()
```

```
data['Loan Type']= data['Loan Type'].astype('int')'''
'''We can achieve the label encoding using scikit-learn's LabelEncoder:'''
data_le = cat_df_flights.copy()

from sklearn.preprocessing import LabelEncoder
lb_make = LabelEncoder()
data_lc['loan_purpose_code'] = lb_make.fit_transform(data_lc['loan_purpose_code'])
data_lc['loan_type_code'] = lb_make.fit_transform(data_lc['Loan Type'])
data_lc['unit_type_code'] = lb_make.fit_transform(data_lc['Unit Type'])
data_lc.head(20) #Results in appending a new column to df
data_lc.info()
```

Label Encoding

Another approach is to encode categorical values with a technique called "label encoding", which allows you to convert each value in a column to a number. Numerical labels are always between 0 and n_categories-1.

We can do label encoding via attributes .cat.codes on your DataFrame's column.'''

```
data_lc['city_code'] = data_lc['City'].cat.codes
data_lc['zip_code'] = data_lc['Zip'].cat.codes
```

One-Hot encoding

The basic strategy is to convert each category value into a new column and assign a 1 or 0 (True/False) value to the column. This has the benefit of not weighting a value improperly.

```
data_lc = pd.get_dummies(data_lc, columns=['Fix'], prefix = ['Fix'])
data_lc['Fix_True']
data_lc['Fix_False']
data_lc['Fix_True'].head()
data_lc['Fix_True'].value_counts()
```

6.1 DATA PRE-PROCESSING

In the data pre-processing phase we pre-process our data to a suitable format for our predictive model. This phase consists out of the following steps. First, the data is grouped by day. Since we want to get insight in the amount of personnel needed on a daily level, we want to group the amount of mortgage applications per working day.

Second, as there are minimal applications coming in during the weekends, we choose to transfer these applications to Monday. Third, mortgage applications that entered the system before the 14th of October 2014 have been removed from our dataset. As this data would not be useful for our model, it is excluded. Finally, missing dates have been added to the model, as these need to be predicted as well.

Handling Categorical Data in Python

Learn the common tricks to handle categorical data and preprocess it to build machine learning models! The categorical features in many datasets generally include different categories or levels associated with the observation, which are non-numerical and thus need to be converted so the computer can process them. The difference between categorical and continuous data in your dataset and identifying the type of data to do basic exploration of such data to extract information from it.

One of the most common data pre-processing steps is to check for null values in the dataset. You can get the total number of missing values in the DataFrame by the following one liner code. We can do a mode imputation for those null values. The function fillna() is handy for such operations.

```
print(data['Zip'].isnull().values.sum())
#Let's also check the column-wise distribution of null values:
print(data['Zip'].isnull().sum())
```

Find Unique Values

The chaining of method .value_counts() in the code below. This returns the frequency distribution of each category in the feature, and then selecting the top category, which is the mode, with the .index attribute.

Another Exploratory Data Analysis (EDA) step to do on categorical features is the frequency distribution of categories within the feature, which can be done with the .value_counts() method.'''

```
data['City'].value_counts()
```

To know the count of distinct categories within the feature you can chain the previous code with the .count() method:

```
data['City'].value_counts().count()
```

Preprocessing: scaling

Here below I (i) scale the data, (ii) use k-Nearest Neighbors and (iii) check the model performance. I'll use scikit-learn's scale function, which standardizes all features (columns) in the array passed to it.'''

```
from sklearn.preprocessing import scale
Xs = scale(X)
from sklearn.cross_validation import train_test_split
Xs_train, Xs_test, y_train, y_test = train_test_split(Xs, y, test_size=0.2)
knn_model_2 = knn.fit(Xs_train, y_train)
print('k-NN score for test set: %f' % knn_model_2.score(Xs_test, y_test))
print('k-NN score for training set: %f' % knn_model_2.score(Xs_train, y_train))
y_true, y_pred = y_test, knn_model_2.predict(Xs_test)
print(classification_report(y_true, y_pred))
```

k-NN score for test set: 0.678643
k-NN score for training set: 0.770000 or 77%

	precision	*recall*	*f1-score*	*support*
0	*0.46*	*0.38*	*0.41*	*16*
1	*0.73*	*0.88*	*0.80*	*347*
2	*0.41*	*0.21*	*0.27*	*87*
3	*0.00*	*0.00*	*0.00*	*2*
4	*0.48*	*0.21*	*0.29*	*48*
5	*0.00*	*0.00*	*0.00*	*1*
avg / total	*0.64*	*0.68*	*0.64*	*501*

All these measures improved by 0.1325, which is a **13.25% improvement and significant**! As hinted at above, before scaling there were a number of predictor variables with ranges of different order of magnitudes, meaning that one or two of them could dominate in the context of an algorithm such as k-NN. The two main reasons for scaling our data are

Our predictor variables may have significantly different ranges and, in certain situations, such as when implementing k-NN, this needs to be mitigated so that certain features do not dominate the algorithm; We want our features to be unit-independent, that is, not reliant on the scale of the measurement involved. If we both scale our respective data, this feature will be the same for each of us.

6.2 FEATURE ENGINEERING

Feature engineering is the process of encoding the predictors in a way that they can be useful for prediction. We generate features for our model that may have predictive power based on our domain knowledge of the mortgage application domain. There are a number of predictors that influence the amount of mortgage applications. In order to use these predictors in our model, we can encode them into features that can directly be fed into our model. For each of the predictors mentioned, one or multiple features are created. For example, for the predictors related to seasonality, we generate features related to the time of the year, vacations and holidays. We can incorporate external holiday data, to see if the holidays have an effect on the amount of mortgage applications. For the other predictors, we will need to include external data in our model. For example we have included US 10 Years Treasury Rate Data and Housing Supply data into out model. There are a number of institutions that offer publicly available data related to the economy and housing market. We have collected the data from **FRED (https://fred.stlouisfed.org/)**

A single predictor sometimes translates into multiple features. Another factor that we have to take into account is the announcements of interest rate changes and auto-correlation. As soon as an increase in interest rate is announced, we may already expect an increase in mortgage applications. This increase in mortgage applications happens before the actual interest rate change itself.

6.3 FEATURE SELECTION

Feature Selection is one of the core concepts in machine learning which hugely impacts the performance of your model. The data features that you use to train your machine learning models have a huge influence on the performance you can achieve.

How to select features and what are Benefits of performing feature selection before modeling your data?

Reduces Overfitting: Less redundant data means less opportunity to make decisions based on noise.

Improves Accuracy: Less misleading data means modeling accuracy improves.Reduces

Training Time: fewer data points reduce algorithm complexity and algorithms train faster.

Feature Selection Methods:

I will share 3 Feature selection techniques that are easy to use and also gives good results.

- **Univariate Selection**
- **Feature Importance**
- **Correlation Matrix with Heatmap**

1. Univariate Selection

Statistical tests can be used to select those features that have the strongest relationship with the output variable. The scikit-learn library provides the SelectKBest class that can be used with a suite of different statistical tests to select a specific number of features.

The example below uses the chi-squared (chi²) statistical test for non-negative features to select 10 of the best features from the Mobile Price Range Prediction Dataset.

```
import pandas as pd
import numpy as np
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
X = np.array(model_data.drop(['Qualification FICO'],1))
y = np.array(model_data['Qualification FICO'])

#apply SelectKBest class to extract top 10 best features
bestfeatures = SelectKBest(score_func=chi2, k='all')
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(model_data.columns)
featureScores = pd.concat([dfcolumns,dfscores],axis=1)  #concat two dataframes for better visualization
featureScores.columns = ['Specs','Score']  #naming the dataframe columns
print(featureScores.nlargest(11,'Score'))  #print 10 best features
```

```
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="darkgrid")
sns.barplot(featureScores['Specs'], featureScores['Score'], alpha=0.9)
plt.title('Frequency Distribution of Features')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Specs', fontsize=12)
plt.xticks(rotation=45)
plt.show()
```

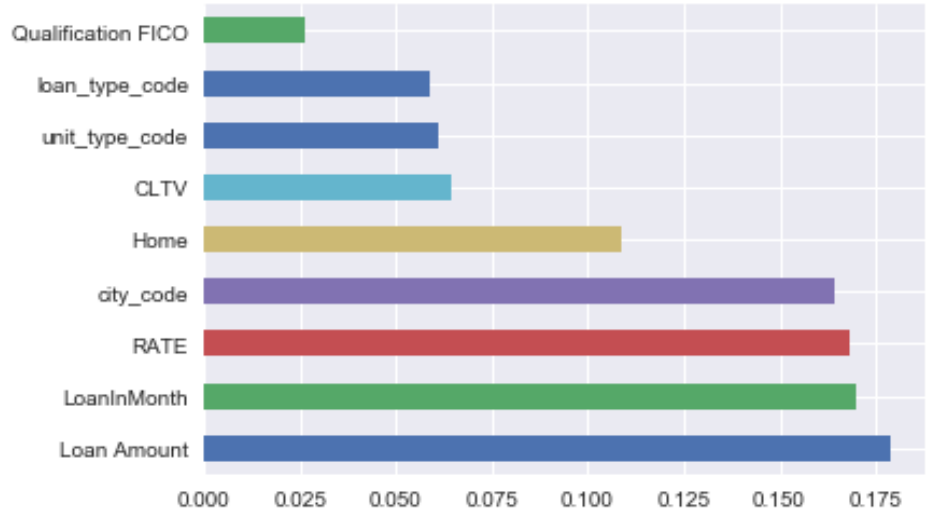
	Specs	Score
6	**CLTV**	**80.536598**
0	**Loan Amount**	**16.250345**
4	**unit_type_code**	**14.523317**
3	**Qualification FICO**	**14.370298**
2	**loan_purpose_code**	**12.874750**
1	city_code	6.169496
5	loan_type_code	1.500709
8	RATE	1.213212
7	LoanInMonth	1.186288

2. Feature Importance

The feature importance of each feature of our dataset by using the feature importance property of the model. Feature importance gives us a score for each feature of our data, the higher the score more important or relevant is the feature towards your output variable. Feature importance is an inbuilt class that comes with Tree Based Classifiers, we will be using Extra Tree Classifier for extracting the top 10 features for the dataset.'''

```
X = np.array(model_data.drop(['CLTV'],1))
y = np.array(model_data['CLTV'])    #target column
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt
model = ExtraTreesClassifier()
model.fit(X,y)
print(model.feature_importances_) #use inbuilt class feature_importances of tree based
classifiers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=Z.columns)
feat_importances.nlargest(11).plot(kind='barh')
plt.show()
```

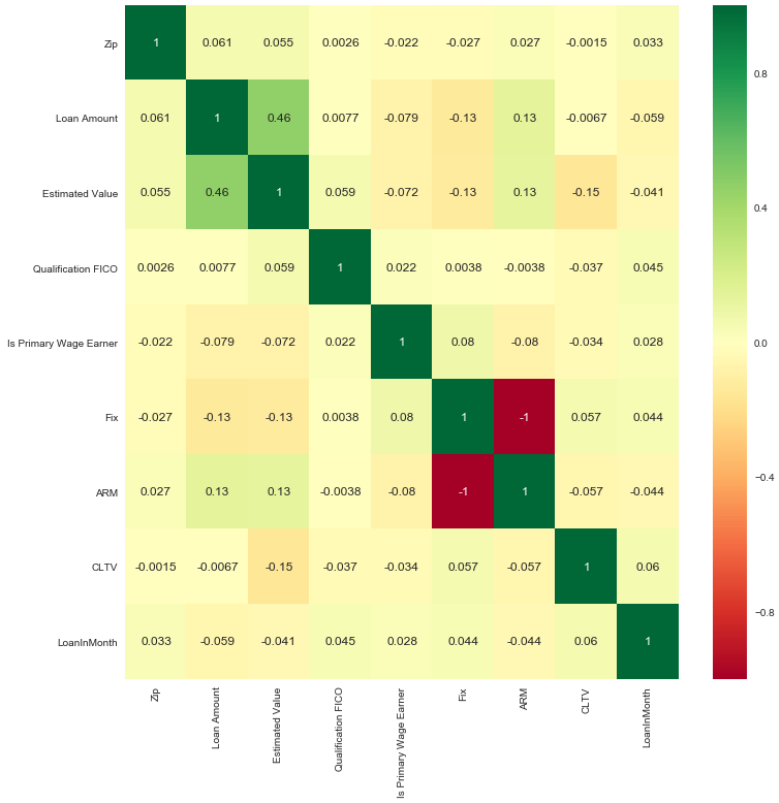
[0.18 0.16 0.03 0.06 0.06 0.06 0.17 0.17 0.11]



3. Correlation Matrix with Heatmap

Correlation states how the features are related to each other or the target variable.
Correlation can be positive (increase in one value of feature increases the value of the target variable) or negative (increase in one value of feature decreases the value of the target variable)
Heatmap makes it easy to identify which features are most related to the target variable, we will plot heatmap of correlated features using the seaborn library.

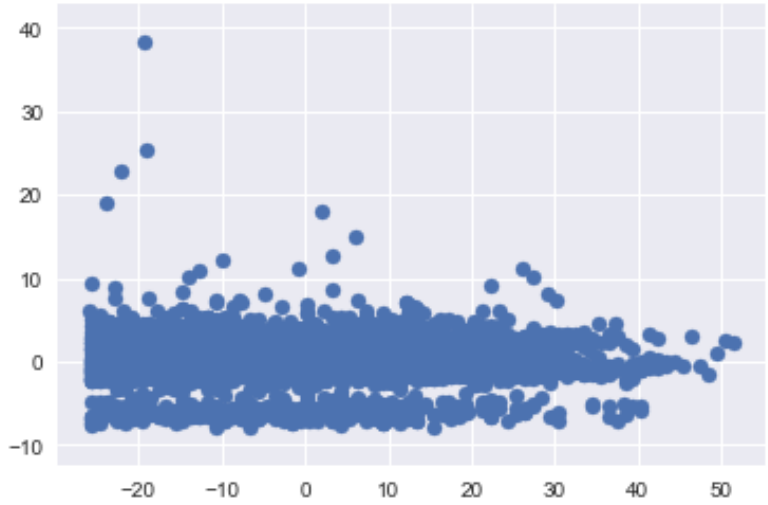
```
import pandas as pd
import numpy as np
import seaborn as sns
X = np.array(model_data.drop(['CLTV'],1))
y = np.array(model_data['CLTV'])    #target column

#get correlations of each features in dataset
corrmat = data.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(12,12))
#plot heat map
g=sns.heatmap(data[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



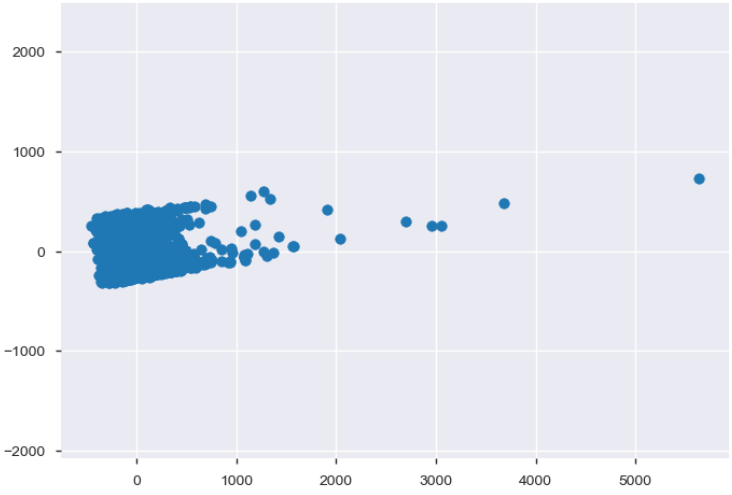
6.4 PCA (Principal Component Analysis)

PCA use PCA to de-correlate these measurements, then plot the de-correlated points and measure their Pearson correlation. Compute Pearson correlation coefficient between two arrays.

```
def pearson_r(x, y):
    # Compute correlation matrix: corr_mat
    corr_mat = np.corrcoef(x, y)
    # Return entry [0,1]
    return corr_mat[0, 1]

# Import PCA
from sklearn.decomposition import PCA
# Create PCA instance: model
model = PCA()
# Apply the fit_transform method of model to grains: pca_features
pca_features = model.fit_transform(model_data)
# Assign 0th column of pca_features: xs
xs = pca_features[:,0]
# Assign 1st column of pca_features: ys
ys = pca_features[:,1]
plt.scatter(xs, ys) # Scatter plot xs vs ys
plt.axis('equal')
plt.show()
```




```
pca_features = model.fit_transform(fit_data)
# Assign 0th column of pca_features: xs
xs = pca_features[:,0]
# Assign 1st column of pca_features: ys
ys = pca_features[:,1]
# Scatter plot xs vs ys
plt.scatter(xs, ys)
plt.axis('equal')
plt.show()
```



Variance of the PCA features

The dataset is 10-dimensional. But what is its intrinsic dimension? Make a plot of the variances of the PCA features to find out. As before, samples is a 2D array, where each row represents a fish. You'll need to standardize the features first. Tthe use of principal component analysis for dimensionality reduction, for visualization of high-dimensional data, for noise filtering, and for feature selection within high-dimensional data. Because of the versatility and interpretability of PCA, it has been shown to be

effective in a wide variety of contexts and disciplines. Given any high-dimensional dataset, I tend to start with PCA in order to visualize the relationship between points), to understand the main variance in the data and to understand the intrinsic dimensionality (by plotting the explained variance ratio).
Perform the necessary imports

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
import matplotlib.pyplot as plt
import numpy as np

X = np.array(fit_data.drop(['Fix_True'],1))
y = np.array(fit_data['Fix_True'])
# Create scaler: scaler
scaler = StandardScaler()
# Create a PCA instance: pca
pca = PCA()
# Create pipeline: pipeline
pipeline = make_pipeline(scaler, pca)

# Fit the pipeline to 'samples'
pipeline.fit(fit_data)
pca.n_components_
#df_pca = pd.read_csv(url, names=['Loan Amount', 'zip_code', 'loan_purpose_code',
'Qualification FICO', 'unit_type_code', 'loan_type_code', 'Fix_True', 'CLTV', 'RATE', 'Home'])
# Plot the explained variances
columns = ['Loan Amount', 'zip_code', 'loan_purpose_code', 'Qualification FICO',
'unit_type_code',
     'loan_type_code', 'Fix_True', 'CLTV', 'RATE', 'Home']
for x in ax.get_xticklabels(minor=True):
   columns.set_rotation(45)
   print(x)
ax.set_xticks(np.arange(len(fit_data.index)))
ax.set_xticklabels([case for case in fit_data.columns], rotation=30)
```

[Text(0,0,'Loan Amount'),
Text(1,0,'city_code'),
Text(2,0,'zip_code'),
Text(3,0,'unit_type_code'),
Text(4,0,'loan_type_code'),
Text(5,0,'Fix_True'),
Text(6,0,'CLTV'),
Text(7,0,'LoanInMonth'),
Text(8,0,'RATE'),
Text(9,0,'Home')]

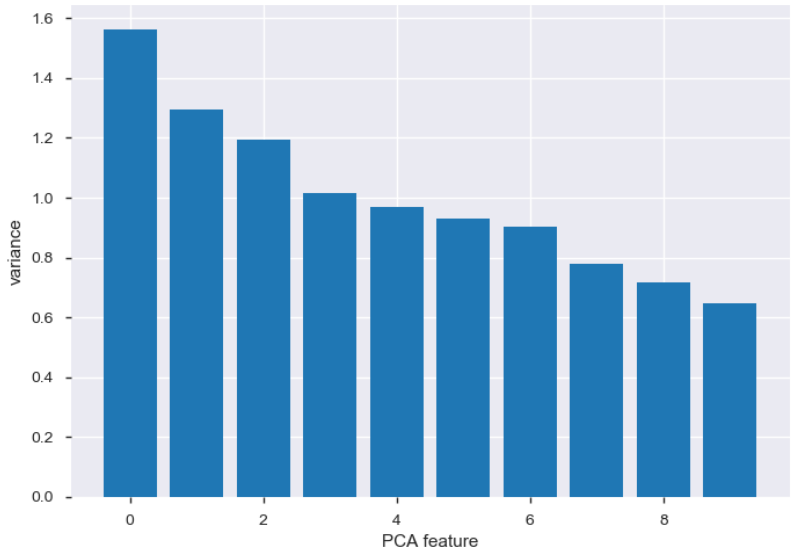
Now we want to know how many principal components we can choose for our new feature subspace. A useful measure is the “explained variance ratio“. The explained variance ratio tells us how much

information (variance) can be attributed to each of the principal components. We can plot bar graph between no. of features on X axis and variance ratio on Y axis.

```
features = range(pca.n_components_)
feature_names = features = range(pca.n_components_)
plt.bar(features, pca.explained_variance_)
plt.xlabel('PCA feature')
plt.ylabel('variance')
plt.xticks(columns.values)
plt.show()

plt.plot([1, 9])
ax = plt.gca()
labels = ax.get_xticklabels()
for label in labels:
    print(label)
pca.fit_transform(X)
print(pca.mean_)
print(pca.components_)
print(pca.explained_variance_)
print(pca.explained_variance_ratio_)
print(pca.singular_values_)
print(pca.n_components_)
print(pca.noise_variance_)
```

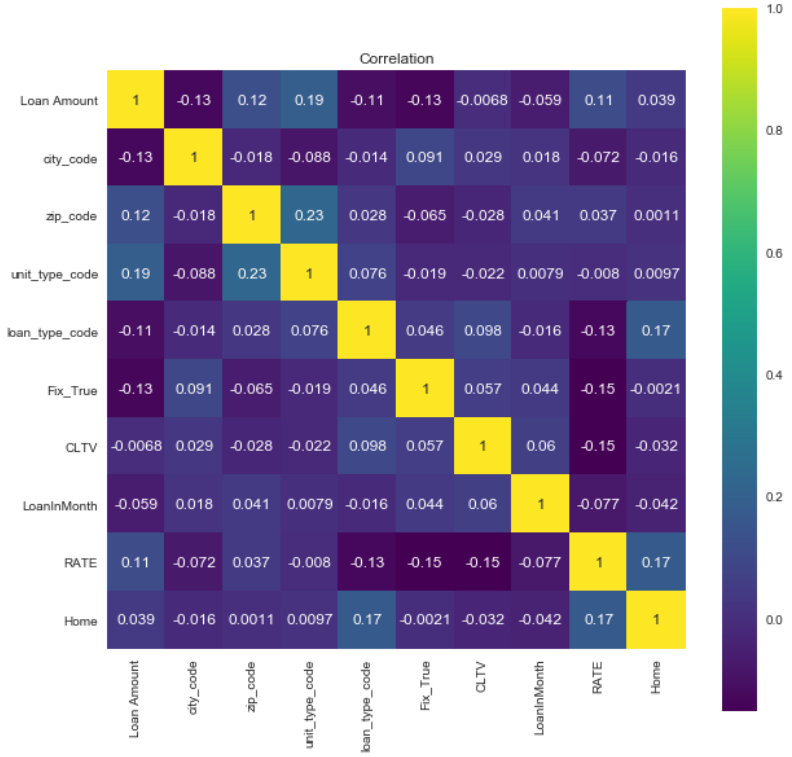
These are the principal components:

Text(0,0,'Loan Amount'),
Text(1,0,'city_code'),
Text(2,0,'zip_code'),
Text(3,0,'unit_type_code'),
Text(4,0,'loan_type_code'),



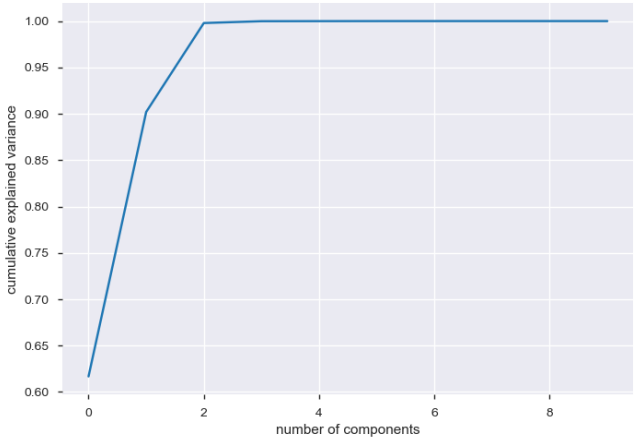
'''Finding Correlation between Features and Target Variable in mortgage Dataset using Heatmap'''

```
correlation = fit_data.corr()
plt.figure(figsize=(10,10))
sns.heatmap(correlation, vmax=1, square=True,annot=True,cmap='viridis')
plt.title('Correlation')
```



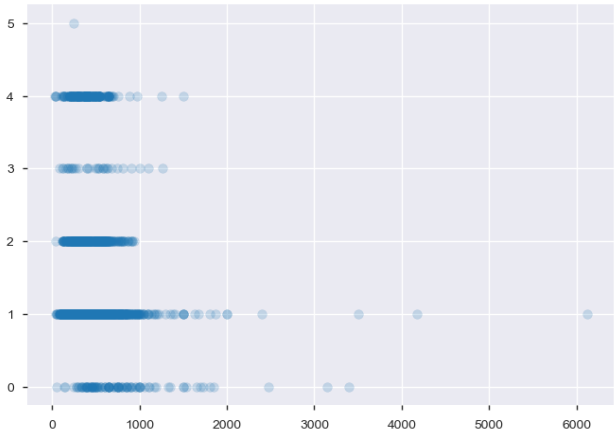
#Let us load the basic packages needed for the PCA analysis

```
pca = PCA().fit(fit_data)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```



To see what these numbers mean, let's visualize them as vectors over the input data, using the "components" to define the direction of the vector, and the "explained variance" to define the squared-length of the vector:

```
def draw_vector(v0, v1, ax=None):
    ax = ax or plt.gca()
    arrowprops=dict(arrowstyle='->',
                    linewidth=2,
                    shrinkA=0, shrinkB=0)
    ax.annotate('', v1, v0, arrowprops=arrowprops)

# plot data
plt.scatter(X[:, 0], X[:, 4], alpha=0.2)
for length, vector in zip(pca.explained_variance_, pca.components_):
    v = vector * 3 * np.sqrt(length)
    draw_vector(pca.mean_, pca.mean_ + v)
plt.axis('equal');
```



PCA as dimensionality reduction

Using PCA for dimensionality reduction involves zeroing out one or more of the smallest principal components, resulting in a lower-dimensional projection of the data that preserves the maximal data variance.

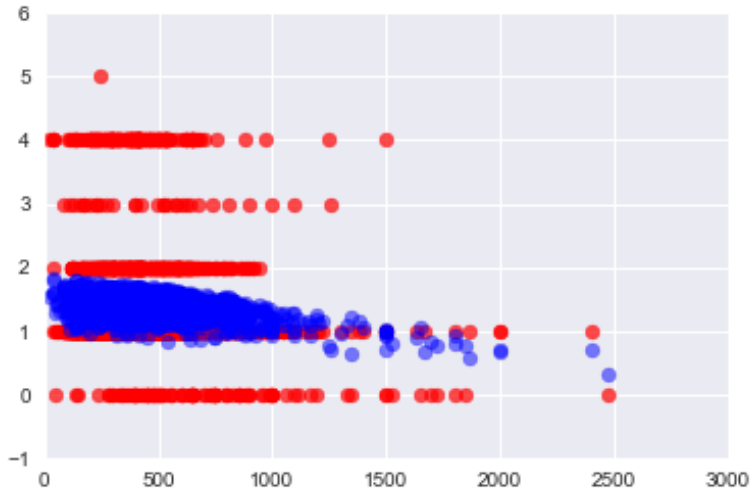
Here is an example of using PCA as a dimensionality reduction transform:

```
pca = PCA(n_components=6)
pca.fit(X)
X_pca = pca.transform(X)
print("original shape:   ", X.shape)
print("transformed shape:", X_pca.shape)
```

original shape: (2501, 9)
transformed shape: (2501, 6)

The transformed data has been reduced to a 6 dimension. To understand the effect of this dimensionality reduction, we can perform the inverse transform of this reduced data and plot it along with the original data:'''

```
X_new = pca.inverse_transform(X_pca)
plt.scatter(X[:, 0], X[:, 4], alpha=0.7, c='red')
plt.scatter(X_new[:, 0], X_new[:, 4], alpha=0.5, c='blue')
#plt.axis('equal');
ylim(-1,6)
xlim(0,3000)
plt.figure(figsize=(20,20))
plt.show()
```



The light points are the original data, while the dark points are the projected version. This makes clear what a PCA dimensionality reduction means: the information along the least important principal axis or axes is removed, leaving only the component(s) of the data with the highest variance. The fraction of variance that is cut out (proportional to the spread of points about the line formed in this figure) is roughly a measure of how much "information" is discarded in this reduction of dimensionality.

This reduced-dimension dataset is in some senses "good enough" to encode the most important relationships between the points: despite reducing the dimension of the data by 50%, the overall relationship between the data points are mostly preserved.

7 MODELING

In the Modeling stage we discuss the activities related to the model building part of our project. A selection of five modeling techniques is made that are applicable to our capstone project. From each of these five modeling techniques, a model is built with the feature set provided earlier, and the models are validated using repeated cross-validation.

7.1 SELECTION OF MODELING TECHNIQUES

For our modeling we use a combination of predictive techniques. Multiple techniques are selected and applied on the data. For the non-linear regression techniques, we use Support Vector Regression (SVR) and Neural Networks (NN). SVR has shown to obtain excellent performances in regression and time series applications. Neural Networks are a widely used method for time series data that generally gives mixed results.

Another technique we use is Classification and Regression Trees(CART), which is a simple technique that is easy to visualize. Also two ensemble techniques are included, in order to improve the performance of the Classification and Regression Trees. These ensemble techniques are Gradient Boosting Machines (GBM) and Random Forests (RF). These techniques create a multitude of regression trees and select a combination of them in order to maximize the performance.

7.2 MODEL BUILDING

Using these five techniques (Linear Regression, Logistic Regression, SVM, SVR, Decision Tree, RF, and KNN) we can create five models. We use the list of features mentioned in section 6.3 as input for our models. A total of 12 features are included, the remaining features were excluded after performing feature selection. For each of the five models hyperparameters were tuned, using grid search. Hyperparameters are the model-specific parameters that are used for optimizing the model. They generally have to be tuned in order to optimize the model's performance, and reduce the variance and bias of the model. By training the model with different values of the size and the decay, and evaluating its performance, we can select the hyperparameters that result in the best performing model in terms of predictive power.

ARMA Model

Estimating an AR Model

We will estimate the AR(1) parameter, φ, of one of the Rate, Revenue, Loan_num, series that generated in the earlier . Since the parameters are known for a series, it is a good way to understand the stimation routines before applying it to real data. For monthly_rate_data with a true φ of 0.9, we will print out the estimate of φ. In addition, we will also print out the entire output that is produced when you fit a time series, so we can get an idea of what other tests and summary statistics are available in statsmodels.

```
from statsmodels.tsa.arima_model import ARMA
# Fit an AR(1) model to the first simulated data
mod_rate = ARMA(np.asarray(monthly_rate_data), order=(1,0))
res_rate = mod_rate.fit()
print(res_rate.summary())
# Print out the estimate for the constant and for phi
print("When the true phi=0.9, the estimate of phi (and the constant) are:")
print(res_rate.params)
```

ARMA Model Results

Dep. Variable:	**y**	**No. Observations:**	**51**
Model:	**ARMA(1, 0)**	**Log Likelihood**	**23.792**
Method:	**css-mle**	**S.D. of innovations**	**0.149**
Date:	**Thu, 31 Jan 2019**	**AIC**	**-41.584**
Time:	**11:49:00**	**BIC**	**-35.788**
Sample:	**0**	**HQIC**	**-39.369**

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	**2.3886**	**0.247**	**9.653**	**0.000**	**1.904**	**2.874**
ar.L1.y	**0.9310**	**0.045**	**20.707**	**0.000**	**0.843**	**1.019**

Roots

	Real	**Imaginary**	**Modulus**	**Frequency**
AR.1	**1.0741**	**+0.0000j**	**1.0741**	**0.0000**

When the true phi=0.9, the estimate of phi (and the constant) are:[2.39 0.93]

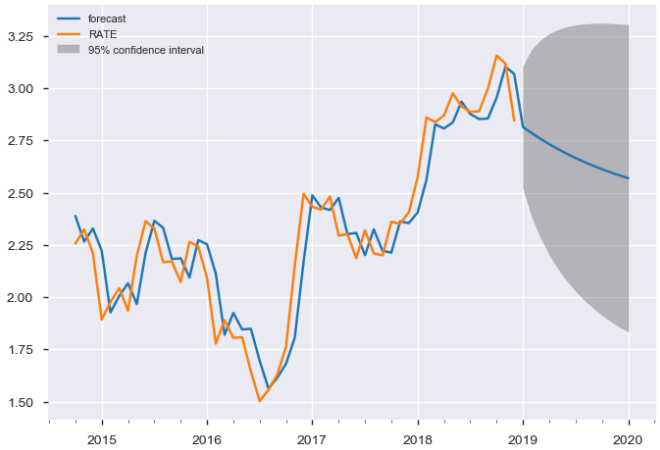
Forecasting with an AR Model

In addition to estimating the parameters of a model, we can also do forecasting using statsmodels. The in-sample is a forecast of the next data point using the data up to that point, and the out-of-sample forecasts any number of data points in the future. These forecasts can be made using either the predict() method if we want the forecasts in the form of a series of data, or using the plot_predict() method we you want a plot of the forecasted data. We will supply the starting point for forecasting and the ending point, which can be any number of data points after the data set ends.

For the simulated series Monthly Interest Rate with **φ=0.9**, we will plot in-sample and out-of-sample forecasts. Being able to forecast interest rates is of enormous importance, not only for bond investors but also for individuals like new **homeowners** who must decide **between fixed and floating rate** mortgages.

There is some **mean reversion** in interest rates over long horizons. In other words, when interest rates are high, they tend to drop and when they are low, they tend to rise over time. Currently they are below long-term rates, so they are expected to rise, but an **AR model** attempts to quantify how much they are expected to rise.

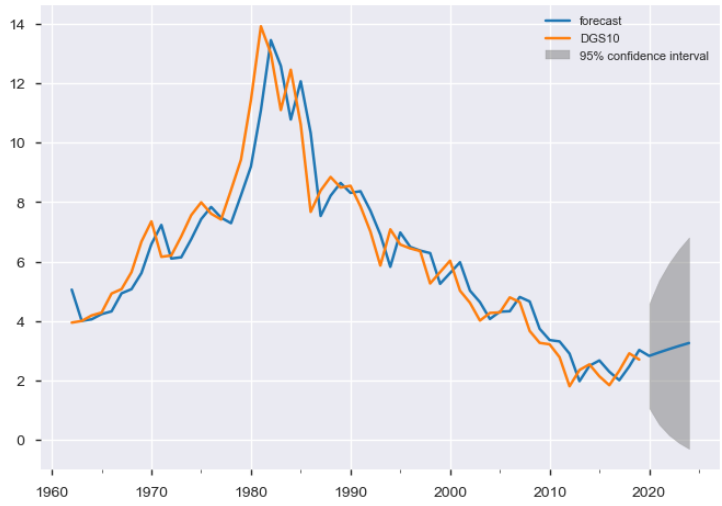
```
from statsmodels.tsa.arima_model import ARMA
# Forecast interest rates using an AR(1) model
mod_monthly_rate_data = ARMA(monthly_rate_data, order=(1,0))
res = mod_monthly_rate_data.fit()
# Plot the original series and the forecasted series
res.plot_predict(start=0, end='2021')
plt.legend(fontsize=8)
plt.show()
```



Since we have only used only 4 years of Monthly Interest Rate Data, we can see the short term downward momentum on the rate. A daily move up or down in interest rates is unlikely to tell us anything about interest rates tomorrow, but a move in interest rates over a year can tell us something about where interest rates are going over the next year. The DataFrame daily_data contains daily data of **10-year interest rates from 1962 to 2017**.

The autocorrelation of daily interest rate changes is 0.07
The autocorrelation of annual interest rate changes is -0.22

```
annual_rate=daily_data.resample('A',how=mean)
type(annual_rate)
annual_rate_data=annual_rate['DGS10']
# Import the ARMA module from statsmodels
from statsmodels.tsa.arima_model import ARMA
mod_annual_rate = ARMA(annual_rate_data, order=(1,0))
res_annual_rate = mod_annual_rate.fit()
res_annual_rate.plot_predict(start=0, end='2024')
plt.legend(fontsize=8)
plt.show()
```

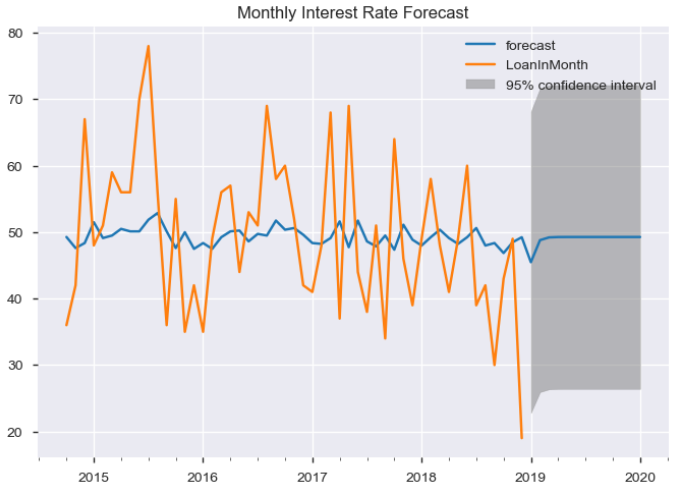


Over long horizons, when interest rates go up, the economy tends to slow down, which consequently causes interest rates to fall, and vice versa. According to an AR(1) model, 10-year interest rates are forecasted to rise from 2.16%, towards the end of 2017 to 3.35% in five years.

Forecast expected monthly closed loans

Our Mortgage DataSet contains data from Oct 2014 to December 2018. Let's forecast Monthly Closed Loans and Monthly Revenue. We will **forecast monthly_loan_num_data** using an AR(1) model

```
from statsmodels.tsa.arima_model import ARMA
monthly_loan_num.resample('M', how='last')
monthly_loan_num_data= monthly_loan_num.resample('M', how='last')
monthly_loan_num_data=monthly_loan_num_data['LoanInMonth']
mod_monthly_loan_num_data = ARMA(monthly_loan_num_data, order=(1,0))
res_monthly_loan_num_data = mod_monthly_loan_num_data.fit()
res_monthly_loan_num_data.plot_predict(start=0, end='2020')
plt.legend(fontsize=10)
plt.title('Monthly Interest Rate Forecast')
plt.show()
```

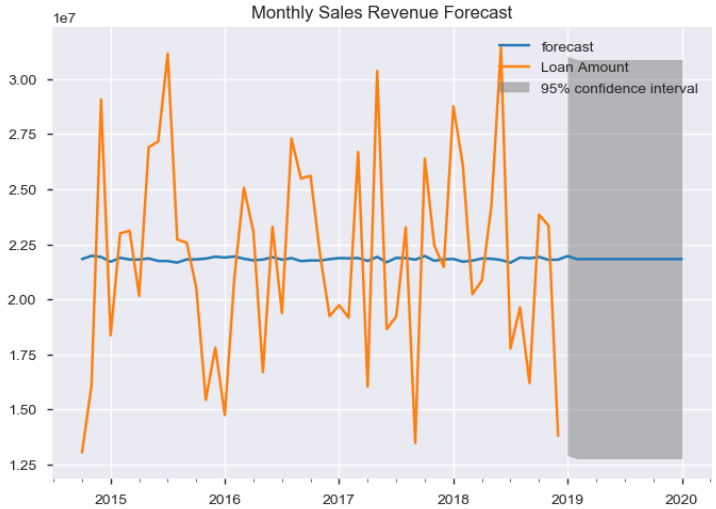


'''Above we have plotted the original series and the forecasted series. With 95% confidence interval, Expected Loans per month will be around 50. Low end is 28 & High End is 70'''

Forecast expected monthly revenue

Similarly, we may forecast expected monthly revenue and plot original series and the forecasted series

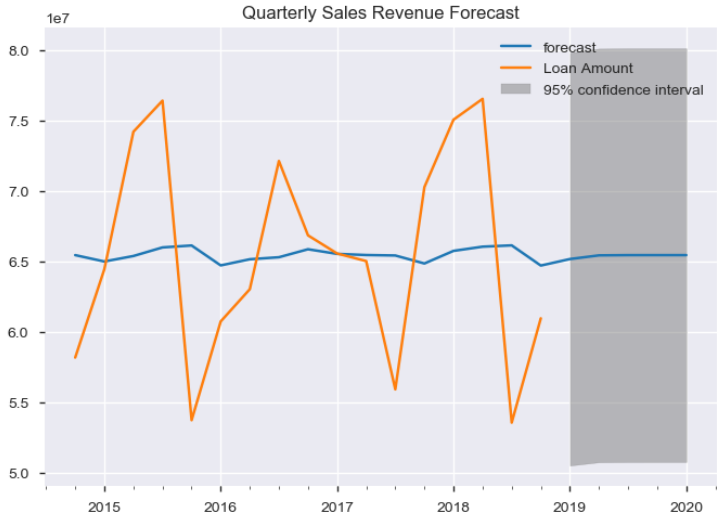
```
mod_monthly_loan_rev_data = ARMA(monthly_loan_rev_data, order=(1,0))
res_monthly_loan_rev_data = mod_monthly_loan_rev_data.fit()
print("The AIC for an AR(1) is: ", res_monthly_loan_rev_data.aic)
res_monthly_loan_rev_data.plot_predict(start=0, end='2020')
plt.legend(fontsize=10)
plt.title('Monthly Sales Revenue Forecast')
plt.show()
```



With 95% confidence interval, Expected Revenue per month will be around 22M. Low end is 13M & High End is 32M

Forecasting Quarterly Sales Revenue

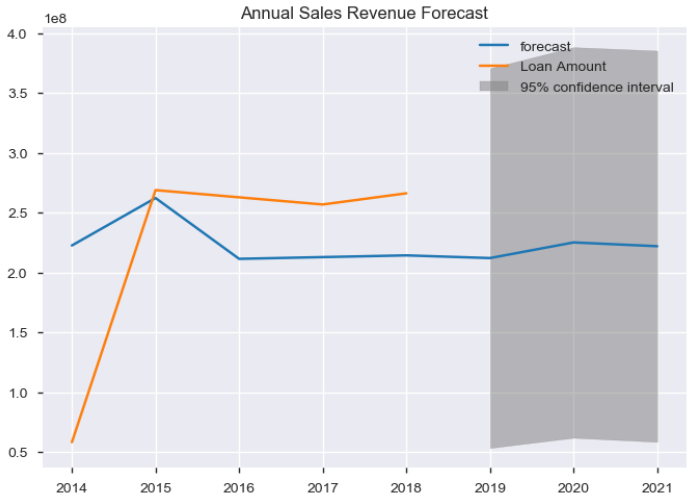
```
loan_patterns['date']= pd.DatetimeIndex(data['Created Date'])
loan_patterns = loan_patterns.set_index('date')
loan_patterns.index
loan_patterns.info()
loan_patterns.head()
quarterly_revenue_data = loan_patterns['Loan Amount'].resample(rule='Q').sum()
mod_quarterly_loan_rev_data = ARMA(quarterly_revenue_data, order=(1,0))
res_quarterly_loan_rev_data = mod_quarterly_loan_rev_data.fit()
res_quarterly_loan_rev_data.plot_predict(start=0, end='2020')
plt.legend(fontsize=10)
plt.title('Quarterly Sales Revenue Forecast')
plt.show()
```



We have plotted the original series and the forecasted series. With **95% confidence** interval, Expected **Revenue per Quarters will be around 67M**. Low end is 52M & High End is 80M

Forecasting Annual Sales Revenue

```
annual_revenue_data = loan_patterns['Loan Amount'].resample(rule='A').sum()
mod_annual_loan_rev_data = ARMA(annual_revenue_data, order=(1,0))
res_annual_loan_rev_data = mod_annual_loan_rev_data.fit()
res_annual_loan_rev_data.plot_predict(start=0, end='2021')
plt.legend(fontsize=10)
plt.title('Annual Sales Revenue Forecast')
plt.show()
```



In the above graph, we have plotted the original series and the forecasted series. We are expecting little drop in annual mortgage revene. Our current annual revenue is $270M. By end of 2021, with 95% confidence interval, Expected Revenue per Years will be around 220M. Low end is 70M & High End is 360M. With more annual data, we should be able to do better annual revenue prediction.

Linear regression

Purpose of linear regression
Given a dataset containing predictor variables X and outcome/response variable Y, linear regression can be used to:

Build a predictive model to predict future values, using new data X where Y is unknown.
Model the strength of the relationship between each independent variable X_i and Y
Many times, only a subset of independent variables X_i will have a linear relationship with Y
Need to figure out which X_i contributes most information to predict Y
It is in many cases, the first pass prediction algorithm for continuous outcomes.

Linear Regression is a method to model the relationship between a set of independent variables X (also knowns as explanatory variables, features, predictors) and a dependent variable Y. This method assumes the relationship between each predictor X is linearly related to the dependent variable Y.

Independence means that the residuals are not correlated -- the residual from one prediction has no effect on the residual from another prediction. Correlated errors are common in time series analysis and spatial analyses.

```
from sklearn.model_selection import train_test_split # for train and test set split
from sklearn.model_selection import cross_val_score
#Sklearn.model_seletion is used instead of sklearn.cross_validation to avoid

X = np.array(model_data.drop(['Loan Amount'],1))
y = np.array(model_data['Loan Amount'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("size of the training feature set is",X_train.shape)
print("size of the test feature set is",X_test.shape)
print("size of the training Target set is",y_train.shape)
print("size of the test Target set is",y_test.shape)

#Linear regression
from sklearn.linear_model import LinearRegression #import from sklearn
linear_reg= LinearRegression() # instantiated linreg
linear_reg.fit(X_train,y_train) #fit the model
#predict using X_test
predicted_train= linear_reg.predict(X_train)
predicted_test= linear_reg.predict(X_test)

from sklearn.metrics import mean_squared_error # import mse from sklearn
#calculate root mean squarred error
rmse_train=np.sqrt(mean_squared_error(y_train, predicted_train))
rmse_test=np.sqrt(mean_squared_error(y_test, predicted_test))

print('The train root mean squarred error is :', rmse_train)
print('The test root mean squarred error is :', rmse_test)
```

size of the training feature set is (2000, 9)
size of the test feature set is (501, 9)
size of the training Target set is (2000,)
size of the test Target set is (501,)
The train root mean squarred error is : 236.63863445055156
The test root mean squarred error is : 399.2661053605523

RMSE of the test data is closer to the training RMSE (and lower) if you have a well trained model. It will be higher if we have an over fitted model.

```
from sklearn import metrics # import metrics from sklearn
Rsquared=linear_reg.score(X_train,y_train) # to determine r square Goodness of fit
# how good the model fits the training data can be determined by R squared metric which 0.12
print('The R squared metric is :', Rsquared)
```

The R squared metric is : 0.10545

'''The R^2 in scikit learn is the coefficient of determination. It is 1 - residual sum of square / total sum of squares. Since R^2 = 1 - RSS/TSS, the only case where RSS/TSS > 1 happens when our model is even worse than the worst model assumed (which is the absolute mean model).

here RSS = sum of squares of difference between actual values(yi) and predicted values(yi^) and TSS = sum of squares of difference between actual values (yi) and mean value (Before applying Regression). So you can imagine TSS representing the best(actual) model, and RSS being in between our best model and the worst absolute mean model in which case we'll get RSS/TSS < 1. If our model is even worse than the worst mean model then in that case RSS > TSS(Since difference between actual observation and mean value < difference predicted value and actual observation).

cv_score is : [-106960.71 -29878.56 -67051.6 -70738.7 -48806.29 -39731.71 -118698.09 -72105.09 -34297.42 -182435.55]
cv_score is : -77070.37197952245
The **cross validation** root mean squarred error is : **277.6155110571498**

Fitting Linear Regression using statsmodels

Statsmodels is a great Python library for a lot of basic and **inferential statistics**. It also provides basic regression functions using an R-like syntax, so it's commonly used by statisticians. The version of least-squares we will use in statsmodels is called ordinary least-squares (OLS). There are many other versions of least-squares such as partial least squares (PLS) and weighted least squares (WLS).

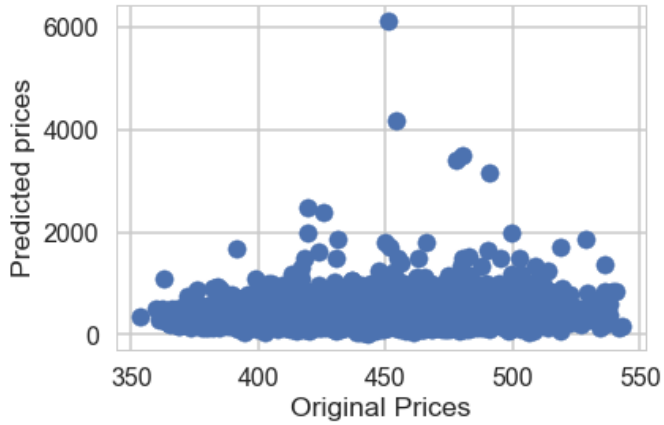
```
# Import regression modules
import statsmodels.api as sm
from statsmodels.formula.api import ols

# statsmodels works nicely with pandas dataframes.  The thing inside the "quotes" is called a formula
m_rate = ols('y ~ RATE',model_data).fit()
print(m_rate.summary())

m_home = ols('y ~ Home',model_data).fit()
print(m_home.summary())
m_cltv = ols('y ~ CLTV',model_data).fit()
print(m_cltv.summary())
m_zip_code = ols('y ~ zip_code',model_data).fit()
print(m_zip_code.summary())
m_Fix_True = ols('y ~ Fix_True',model_data).fit()
print(m_Fix_True.summary())
m_loaninmonths = ols('y ~ LoanInMonth',model_data).fit()
print(m_loaninmonths.summary())

m_rcpi = ols('y ~ LoanInMonth + RATE + Home + Fix_True + CLTV + zip_code',model_data).fit()
print(m_rcpi.summary())
```

OLS Regression Results

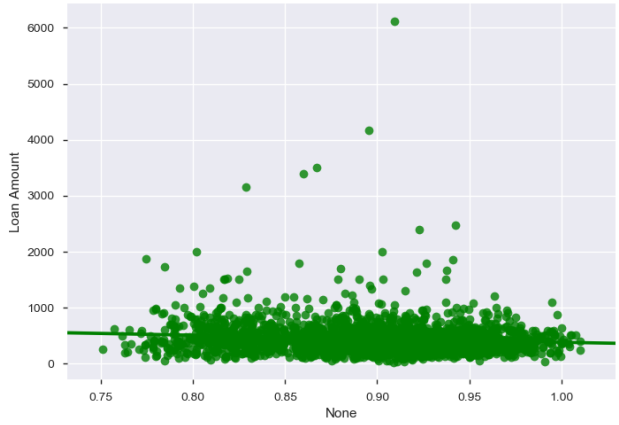
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.026
Model:                            OLS   Adj. R-squared:                  0.025
Method:                 Least Squares   F-statistic:                     16.97
Date:                Fri, 01 Feb 2019   Prob (F-statistic):           9.82e-14
Time:                        03:57:19   Log-Likelihood:                -542.85
No. Observations:                2501   AIC:                             1096.
Df Residuals:                    2496   BIC:                             1125.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      1.0463      0.054     19.536      0.000       0.941       1.151
LoanInMonth    0.0006      0.000      1.602      0.109      -0.000       0.001
RATE          -0.1074      0.015     -7.335      0.000      -0.136      -0.079
Home           0.0088      0.007      1.260      0.208      -0.005       0.023
CLTV           0.0045      0.002      1.913      0.056      -0.000       0.009
==============================================================================
Omnibus:                     1214.830   Durbin-Watson:                   1.884
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             4810.993
Skew:                          -2.509   Prob(JB):                         0.00
Kurtosis:                       7.581   Cond. No.                         291.
==============================================================================
```

fdval = m_rcpi.fittedvalues

plt.scatter(fdval, y)
plt.ylabel('Predicted prices')
plt.xlabel('Original Prices')
plt.show()



sns.regplot(x=fdval, y="Loan Amount", data=model_data, fit_reg = True, color='g')
set_ylim(2,900)
set_xlim(2,900)
plt.show()



Fitting Linear Regression using sklearn

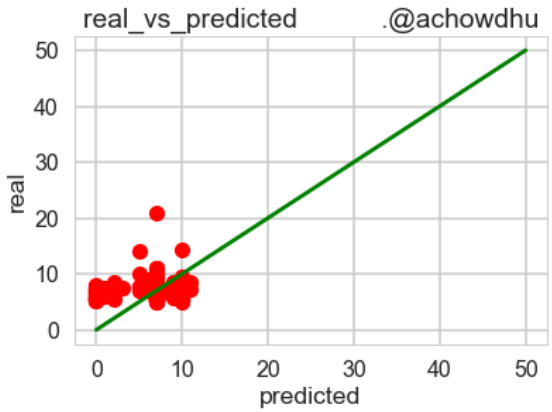
```
from sklearn.linear_model import LinearRegression
X = np.array(fit_data.drop(['Fix_True'],1))
y = np.array(fit_data['Fix_True'])
# This creates a LinearRegression object
lm = LinearRegression()

Fit model: The lm.fit() function estimates the coefficients the linear regression using least squares.
lm.fit(X, y)
lm.coef_
lm.intercept_
# The mean squared error
print("Mean squared error (Fix_True Rate): %.2f" % np.mean((lm.predict(X) - y) ** 2))
X = np.array(fit_data.drop(['loan_type_code'],1))
y = np.array(fit_data['loan_type_code'])
lm.fit(X, y)
lm.coef_
lm.intercept_
print("Mean squared error (loan_type_code): %.2f" % np.mean((lm.predict(X) - y) ** 2))

X = np.array(fit_data.drop(['Loan Amount'],1))
y = np.array(fit_data['Loan Amount'])
lm.fit(X, y)
lm.coef_
lm.intercept_
print("Mean squared error (Loan Amount): %.2f" % np.mean((lm.predict(X) - y) ** 2))
X = np.array(fit_data.drop(['unit_type_code'],1))
y = np.array(fit_data['unit_type_code'])
lm.fit(X, y)
lm.coef_
lm.intercept_
print("Mean squared error (unit_type_code): %.2f" % np.mean((lm.predict(X) - y) ** 2))
```

np.mean((lm.predict(X) - y) ** 2))
Mean squared error (Fix_True Rate): 0.09
Mean squared error (loan_type_code): 0.80
Mean squared error (Loan Amount): 75543.58
Mean squared error (unit_type_code): 7.76

''' Let's try Linear Regression: with Scale'''

```
X = scale_data(X)
X_train, X_test, y_train, y_test = split_data(X,y)
# Create linear regression object
linreg = LinearRegression()

# Train the model using the training sets
linreg.fit(X_train,y_train)
print("Linear model: ", pretty_print_linear(linreg.coef_, sort = True))
# Predict the values using the model
y_lin_predict = linreg.predict(X_test)
# Print the root mean square error
print("Linear Regression - Root Mean Square Error: ",
root_mean_square_error(y_lin_predict,y_test))
plot_real_vs_predicted(y_test,y_lin_predict)
```

Linear Regression - Root Mean Square Error: 2.8077294021457067



Lasso Regression

```
lasso = Lasso(alpha=.3)
# Train the model using the training sets
lasso.fit(X_train, y_train)
print("Lasso model: ", pretty_print_linear(lasso.coef_, sort = True))
# Predict the values using the model
y_lasso_predict = lasso.predict(X_test)
# Print the root mean square error
print("Lasso model - Root Mean Square Error: ",
root_mean_square_error(y_lasso_predict,y_test))
```

```
plot_real_vs_predicted(y_test,y_lasso_predict)
```

Lasso model - Root Mean Square Error: 2.739546

''' Let's try **Ridge Regression**: '''

```
ridge = Ridge(fit_intercept=True, alpha=.3)
# Train the model using the training sets
ridge.fit(X_train, y_train)
print("Ridge model: ", pretty_print_linear(ridge.coef_, sort = True))
# Predict the values using the model
y_ridge_predict = ridge.predict(X_test)
# Print the root mean square error
print("Ridge Regression - Root Mean Square Error: ",
root_mean_square_error(y_ridge_predict,y_test))
plot_real_vs_predicted(y_test,y_ridge_predict)
```

''' Now let's try to do **regression via Elastic Net**. '''

```
elnet = ElasticNet(fit_intercept=True, alpha=.3)
# Train the model using the training sets
elnet.fit(X_train, y_train)
print("Elastic Net model: ", pretty_print_linear(elnet.coef_, sort = True))
# Predict the values using the model
y_elnet_predict = elnet.predict(X_test)
# Print the root mean square error
print("Elastic Net - Root Mean Square Error: ", root_mean_square_error(y_elnet_predict,y_test))
plot_real_vs_predicted(y_test,y_elnet_predict)
```

''' Now let's try to do regression via **Stochastic Gradient Descent**. '''

```
sgdreg = SGDRegressor(penalty='l2', alpha=0.15, n_iter=200)
# Train the model using the training sets
sgdreg.fit(X_train, y_train)
print("Stochastic Gradient Descent model: ", pretty_print_linear(sgdreg.coef_, sort = True))
# Predict the values using the model
y_sgdreg_predict = sgdreg.predict(X_test)

# Print the root mean square error
print("Stochastic Gradient Descent - Root Mean Square Error: ",
root_mean_square_error(y_sgdreg_predict,y_test))
plot_real_vs_predicted(y_test,y_sgdreg_predict)
```

Ridge Regression - Root Mean Square Error: 2.80
Elastic Net model: 0.446 * X2 + 0.414 * X0 + 0.105 * X3 + -0.051 * X1 + -0.0 * X4 + -0.0 * X5 + 0.0 * X6 + -0.0 * X7 + 0.0 * X8
Elastic Net - Root Mean Square Error: 2.74

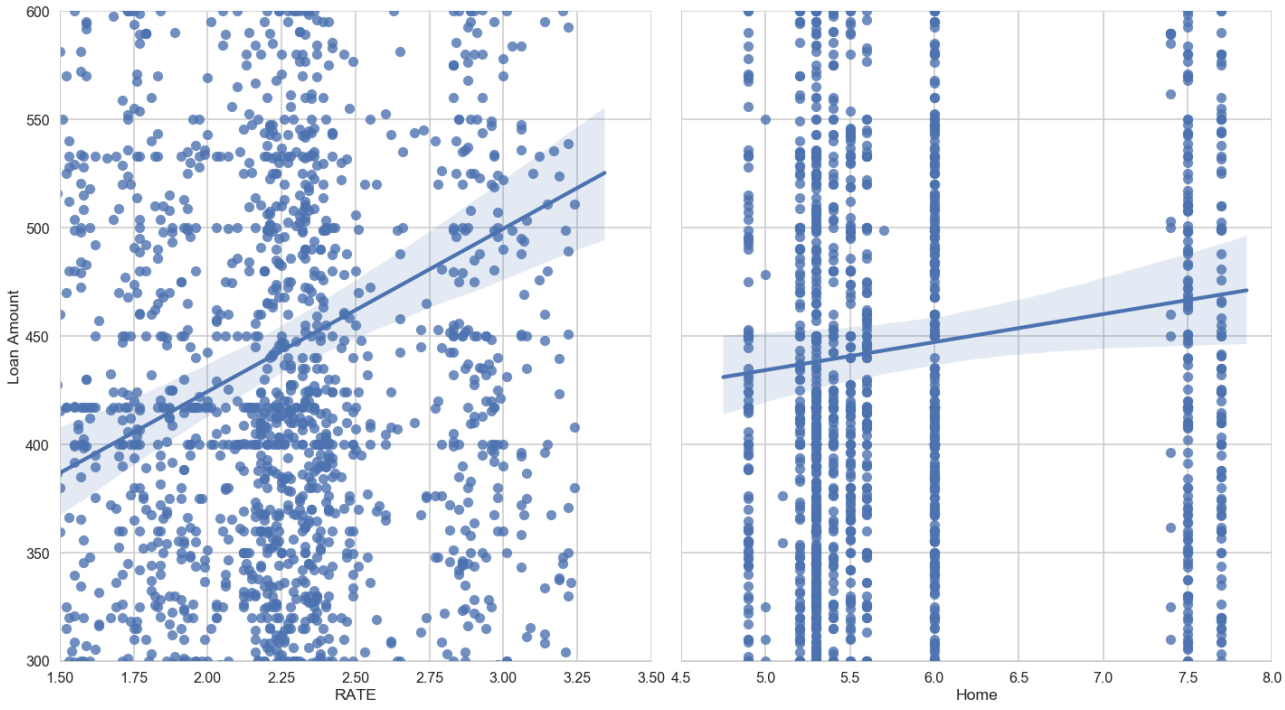
Stochastic Gradient Descent model: 0.565 * X0 + 0.541 * X2 + 0.25 * X3 + -0.144 * X1 + -0.044 * X7 + -0.034 * X5 + 0.012 * X8 + 0.012 * X6 + 0.003 * X4
Stochastic Gradient Descent - Root Mean Square Error: 2.77

Lasso and Elastic Net Regression seems to be better performing.

Now we have a pandas DataFrame called model_data containing all the data we want to use to predict Mortgage Loan prices ("Loan Amount").

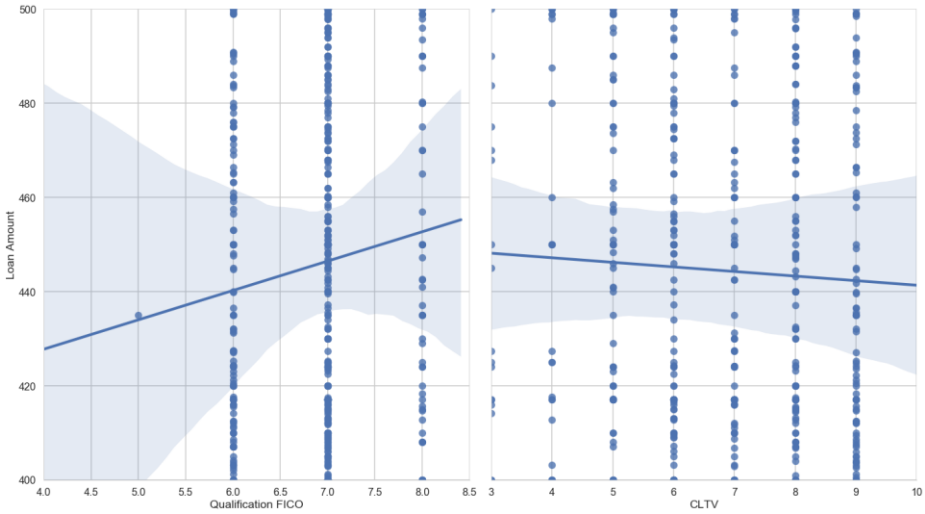
Let's create a variable called 'Loan Amount' which will contain the prices. This information is contained

```
import seaborn as sns
%matplotlib inline
g1=sns.pairplot(model_data, x_vars=['RATE', 'Home'], y_vars='Loan Amount', size=11,
aspect=0.9, kind='reg')
g1.axes[0,1].set_ylim(300,600)
g1.axes[0,0].set_xlim(1.5,3.5)
g1.axes[0,1].set_xlim(4.5,8)
plt.show()
```



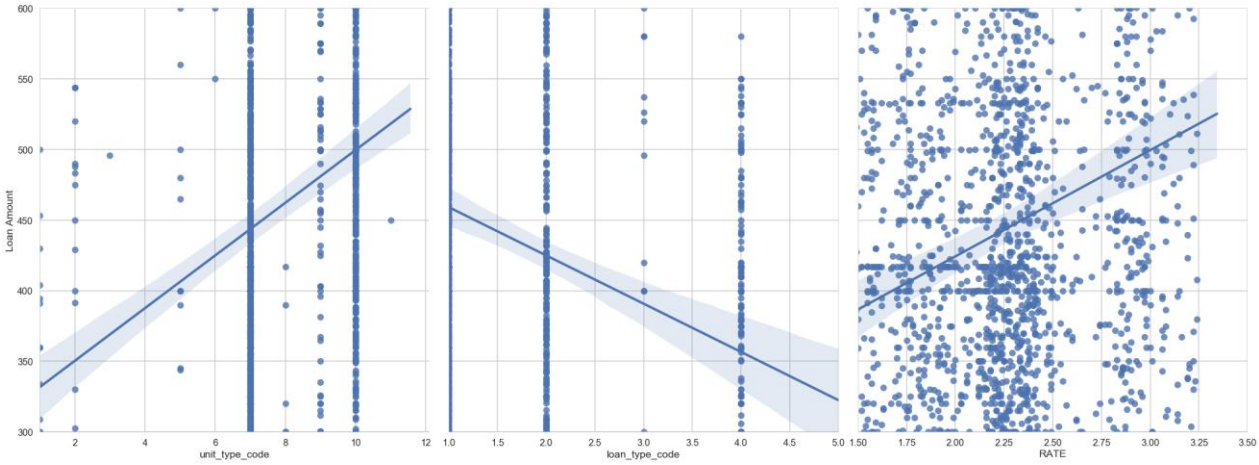
US10Y goes up loan amount slightly increases (Loan Amounts in 1000s)
Home Supply increases, loan amount also increases (Loan Amounts in 1000s)
Interest Rate Chage has bigger impact on Loan Amount compare to Home Supply

```
g=sns.pairplot(model_data, x_vars=['Qualification FICO', 'CLTV'], y_vars='Loan Amount', size=11,
aspect=0.9, kind='reg')
g.axes[0,1].set_ylim(400,500)
g.axes[0,0].set_xlim(4,8.5)
g.axes[0,1].set_xlim(3,10)
plt.show()
```



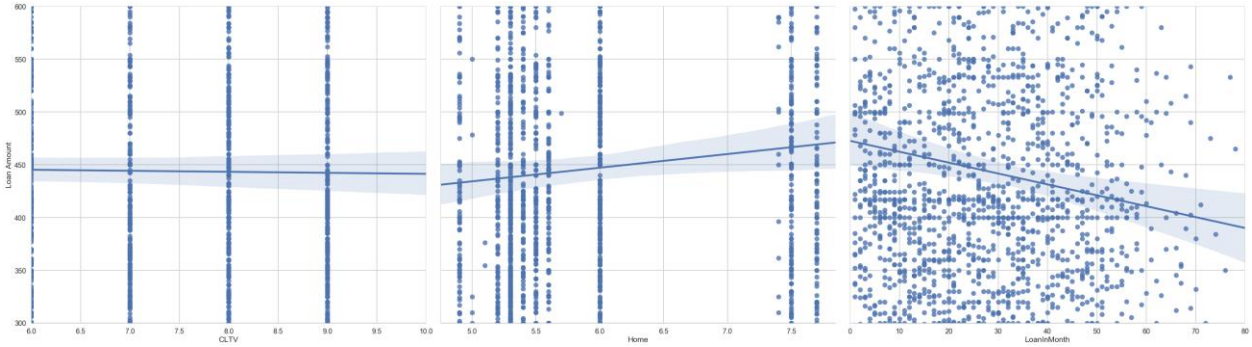
Majority of the FICO scores between 600 and 820 (Loan Amounts in 1000s)
Majority of the CLTV scores between 30 and 100 (Loan Amounts in 1000s)
FICO goes up, Loan Amount goes up
CLTV goes up, Loan Amount Goes down

```
g2=sns.pairplot(model_data, x_vars=['unit_type_code',
    'loan_type_code', 'RATE'], y_vars='Loan Amount', size=11, aspect=0.9, kind='reg')
g2.axes[0,1].set_ylim(300,600)
g2.axes[0,0].set_xlim(1.0,12.1)
g2.axes[0,1].set_xlim(1,5)
g2.axes[0,2].set_xlim(1.5,3.5)
plt.show()
```



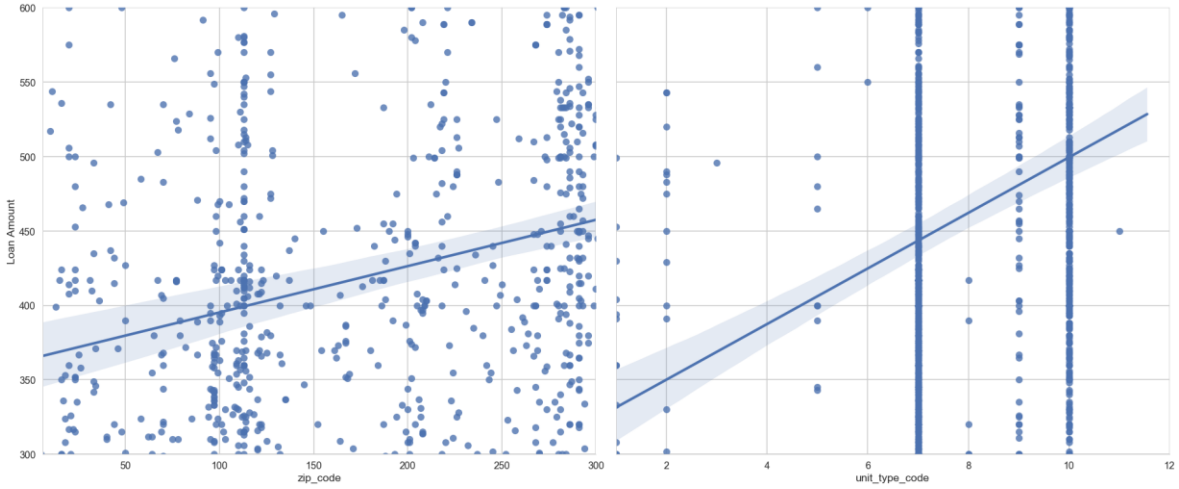
As num of unit decreases ave Loan Amount also decreases
loan_type_code: Conventional is high volume but Slightly low average loan amount
Two Family (Code = 10) has higher Loan Amount that three Family (Code = 9)
Conventional Mortgage has Higher Loan Amount FHA
Loan Amount increases with 10 Years Treasury Rate.

```
g3=sns.pairplot(model_data, x_vars=[ 'CLTV', 'Home', 'LoanInMonth'], y_vars='Loan Amount',
size=11, aspect=1.2, kind='reg')
g3.axes[0,1].set_ylim(300,600)
g3.axes[0,0].set_xlim(6,10)
g2.axes[0,1].set_xlim(4.5,7.5)
g3.axes[0,2].set_xlim(0,80)
plt.show()
```



CLTV does not have much impact on Loan Amount
As number of loans per month increases, loan amount decreases
As Home Supply increases, loan amount also increases

```
g4=sns.pairplot(fit_data, x_vars=[ 'zip_code', 'unit_type_code'], y_vars='Loan Amount', size=11,
aspect=1.2, kind='reg')
g4.axes[0,1].set_ylim(300,600)
g4.axes[0,0].set_xlim(6,300)
g4.axes[0,1].set_xlim(1,12)
plt.show()
```



NYC NJ (zip_code 1st digit starting with 7 & 11) seems to be closing more loans and generation more revenues for the Mortgage Bank. City shows similar results. Population density plays bigger role on Loan Amount. As number of loans per month increases, loan amount decreases

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier

X = np.array(fit_data.drop(['loan_type_code'],1))
y = np.array(fit_data['loan_type_code'])
X_train, X_test, y_train, y_test_knn = cross_validation.train_test_split(X, y, test_size = 0.2)

# Create the DataFrame: numeric_data_only
numeric_data_only = model_data[0:10].fillna(-1000)
# Create training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=21)

# Instantiate the classifier: clf
clf = OneVsRestClassifier(LogisticRegression())

# Fit the classifier to the training data
clf.fit(X_train, y_train)

# Print the accuracy
print("Logistic Regression Accuracy: {}".format(clf.score(X_test, y_test)))
```

Logistic Regression Accuracy: 0.6906187624750499

Centering, Scaling and Logistic Regression

Import necessary packages

```
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('ggplot')
from sklearn import datasets
from sklearn import linear_model
import numpy as np
X = np.array(fit_data.drop(['loan_type_code'],1))
y = np.array(fit_data['loan_type_code'])
# Split the data into test and training sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train logistic regression model and print performance on the test set
lr = linear_model.LogisticRegression()
```

```
# fit the model
lr = lr.fit(X_train, y_train)
print('Logistic Regression score for training set: %f' % lr.score(X_train, y_train))
from sklearn.metrics import classification_report
y_true, y_pred = y_test, lr.predict(X_test)
print(classification_report(y_true, y_pred))
```

Logistic Regression score for training set: 0.707000 or 70%

	precision	***recall***	***f1-score***	***support***
0	***0.33***	***0.10***	***0.15***	***20***
1	***0.68***	***0.99***	***0.80***	***339***
2	***0.50***	***0.01***	***0.02***	***96***
3	***0.00***	***0.00***	***0.00***	***5***
4	***0.00***	***0.00***	***0.00***	***41***
avg / total	***0.57***	***0.67***	***0.55***	***501***

Out of the box, this logistic regression performs better than K-NN (with or without scaling). Let's now scale our data and perform logistic regression:

Scaled Logistic Regression score for test set: 0.682635 or 68%

	precision	***recall***	***f1-score***	***support***
0	***0.00***	***0.00***	***0.00***	***30***
1	***0.68***	***1.00***	***0.81***	***338***
2	***0.80***	***0.05***	***0.09***	***82***
3	***0.00***	***0.00***	***0.00***	***7***
4	***0.00***	***0.00***	***0.00***	***44***
avg / total	***0.59***	***0.68***	***0.56***	***501***

This is very interesting! The performance of **logistic regression did not improve** with data scaling. Why not, particularly when we saw that **k-Nearest Neigbours** performance **improved** substantially with scaling? The reason is that, if there predictor variables with large ranges that do not effect the target variable, a regression algorithm will make the corresponding coefficients ai small so that they do not effect predictions so much. K-nearest neighbours does not have such an inbuilt strategy and so we very much needed to scale the data.'''

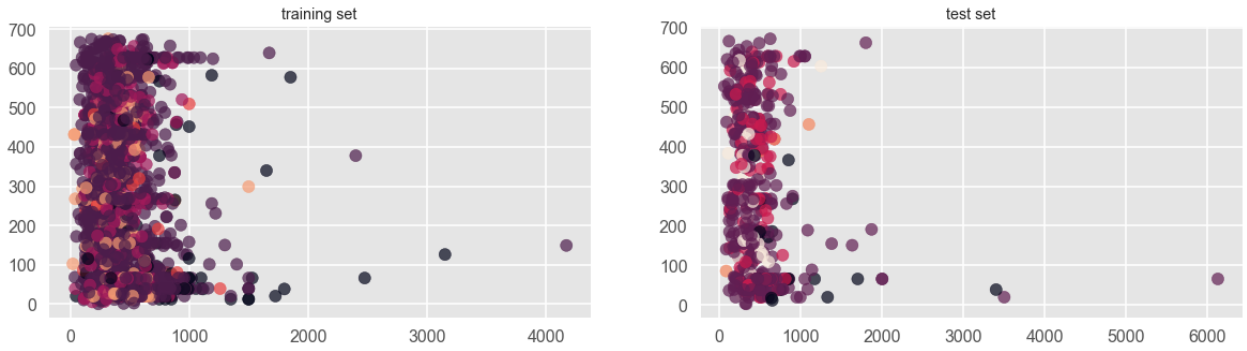
Scaling Synthesized Data

Scaling numerical data (that is, multiplying all instances of a variable by a constant in order to change that variable's range) has two related purposes: i) if your measurements are in different currencies and, then, if we both scale our data, they end up being the same & ii) if two variables have vastly different ranges, the one with the **larger range may dominate your predictive model**, even though it may be less

important to our target variable than the variable with the smaller range. What we saw is that this problem identified in ii) occurs with **k-NN**, which explicitly looks at **how close data are to one another** but **not in logistic regression** which, when being trained, will shrink the relevant coefficient to account for the lack of scaling. We can see was how the models performed before and after scaling.

Let's now split into testing & training sets & plot both sets:

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
plt.figure(figsize=(20,5));
plt.subplot(1, 2, 1 );
plt.title('training set')
plt.scatter(X_train[:,0] , X_train[:,1],  c = y_train, alpha = 0.7);
plt.subplot(1, 2, 2);
plt.scatter(X_test[:,0] , X_test[:,1],  c = y_test, alpha = 0.7);
plt.title('test set')
plt.show()
```



Looking good! Now let's instantiate a k-Nearest Neighbors voting classifier & train it on our training set

```
from sklearn import neighbors, linear_model
knn = neighbors.KNeighborsClassifier()
knn_model = knn.fit(X_train, y_train)

print('k-NN score for test set: %f' % knn_model.score(X_test, y_test))
print('k-NN score for training set: %f' % knn_model.score(X_train, y_train))

from sklearn.metrics import classification_report
y_true, y_pred = y_test, knn_model.predict(X_test)
print(classification_report(y_true, y_pred))
```

k-NN score for test set: 0.646707 or 65%
k-NN score for training set: 0.734000 or 73%

	precision	***recall***	***f1-score***	***support***
0	***0.27***	***0.20***	***0.23***	***20***
1	***0.69***	***0.90***	***0.78***	***339***
2	***0.42***	***0.16***	***0.23***	***96***

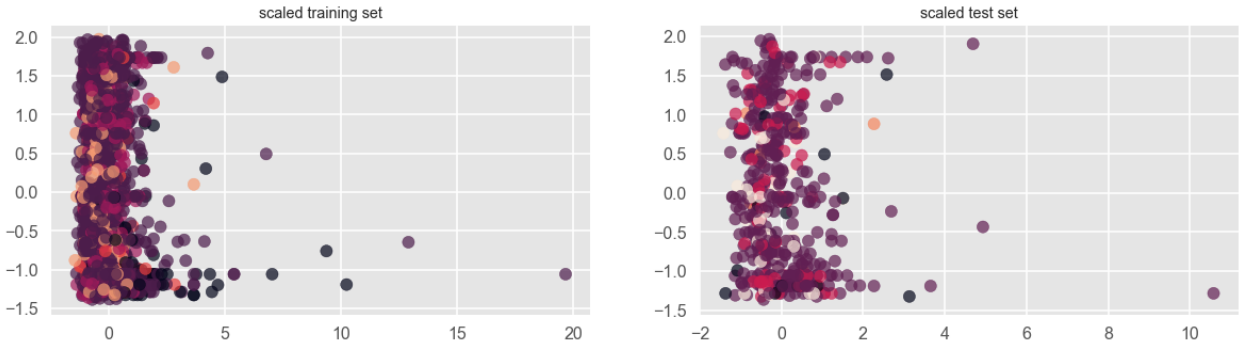
3 0.00 0.00 0.00 5
4 0.00 0.00 0.00 41

avg / total 0.56 0.65 0.58 501
We can notice the improvement for KNN compare to Logistic Regression

Now with scaling KNN: I'll now scale the predictor variables and then use k-NN again:

from sklearn.preprocessing import scale

```
Xs = scale(X)
Xs_train, Xs_test, y_train, y_test = train_test_split(Xs, y, test_size=0.2)
plt.figure(figsize=(20,5));
plt.subplot(1, 2, 1 );
plt.scatter(Xs_train[:,0] , Xs_train[:,1],  c = y_train, alpha = 0.7);
plt.title('scaled training set')
plt.subplot(1, 2, 2);
plt.scatter(Xs_test[:,0] , Xs_test[:,1],  c = y_test, alpha = 0.7);
plt.title('scaled test set')
plt.show()
knn_model_s = knn.fit(Xs_train, y_train)
print('k-NN score for test set: %f' % knn_model_s.score(Xs_test, y_test))
```



k-NN score for test set: 0.698603 or 70%

It doesn't perform any better with scaling! This is most likely because both features were already around the same range. It really makes sense to scale when variables have widely varying ranges. To see this in action, we're going to add another feature. Moreover, this feature will bear no relevance to the target variable: it will be mere noise.

Adding Gaussian noise to the signal:

We add a third variable of Gaussian noise with mean 0 and variable standard deviation σ. We'll call σ the strength of the noise and we'll see that the stronger the noise, the worse the performance of k-Nearest Neighbours

```
X = np.array(fit_data.drop(['loan_type_code'],1))
y = np.array(fit_data['loan_type_code'])
# Generate some clustered data (blobs!)
import numpy as np
from sklearn.datasets.samples_generator import make_blobs
n_samples=2000
X, y = make_blobs(n_samples, centers=4, n_features=2,
                  random_state=0)

# Add noise column to predictor variables
ns = 10**(3) # Strength of noise term
newcol = np.transpose([ns*np.random.randn(n_samples)])
Xn = np.concatenate((X, newcol), axis = 1)

from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(15,10));
ax = fig.add_subplot(111, projection='3d' , alpha = 0.5);
ax.scatter(Xn[:,0], Xn[:,1], Xn[:,2], c = y);

Xn_train, Xn_test, y_train, y_test = train_test_split(Xn, y, test_size=0.2, random_state=42)
knn = neighbors.KNeighborsClassifier(n_neighbors=6)
knn_model = knn.fit(Xn_train, y_train)
print('k-NN score for test set: %f' % knn_model.score(Xn_test, y_test))
type(Xn_train)
```

k-NN score for test set: 0.275000 or 28%

This is a **horrible model**! How about we **scale** and check out performance?

```
Xns = scale(Xn)
s = int(.2*n_samples)
Xns_train = Xns[s:]
y_train = y[s:]
Xns_test = Xns[:s]
y_test = y[:s]
knn_models = knn.fit(Xns_train, y_train)
knn_accuracy = knn_models.score(Xns_test, y_test)
print('knn_accuracy for test set: ' , knn_accuracy)

knn_prediction = knn.predict(Xns[1515:1535])
print('KNN : - Output of Real Data : Conv=5, FHA=4, Res=3, Comm=2: ', (y[1515:1535]))
print('KNN : - Output of prediction: Conv=5, FHA=4, Res=3, Comm=2: ',knn_prediction)
```

knn_accuracy for test set: 0.93 or 93%
KNN : - Output of Real Data : Conv=5, FHA=4, Res=3, Comm=2: [3 3 1 2 3 1 1 2 0 3 2 3 3 1 3 2 2 1 0 0]
KNN : - Output of prediction: Conv=5, FHA=4, Res=3, Comm=2: [3 3 1 2 3 1 1 2 0 3 2 3 3 1 3 2 2 1 0 2]

With Scale and Synthesize the data we can see huge improvement on the model accuracy.. ***28% to 93%***
Let's do same for Logistic Regression and check out the performance.

```
#Import packages
import numpy as np
from sklearn.cross_validation import train_test_split
from sklearn import neighbors, linear_model
from sklearn.preprocessing import scale
from sklearn.datasets.samples_generator import make_blobs

#Generate some data
n_samples=2000
X, y = make_blobs(n_samples, centers=4, n_features=2, random_state=0)
# Add noise column to predictor variables
newcol = np.transpose([ns*np.random.randn(n_samples)])
Xn = np.concatenate((X, newcol), axis = 1)

#Scale if desired
if sc == True:
        Xn = scale(Xn)

#Train model and test after splitting
Xn_train, Xn_test, y_train, y_test = train_test_split(Xn, y, test_size=0.2, random_state=42)
lr = linear_model.LogisticRegression()
lr_model = lr.fit(Xn_train, y_train)
print('logistic regression score for test set: %f' % lr_model.score(Xn_test, y_test))
```

logistic regression score for test set: 0.925000 or 92.5%

We can see big improvement. We have seen the essential place in the data scientific pipeline by preprocessing, in its scaling and centering incarnation, and we have done so to promote a holistic approach to minimize the challenges of machine learning.

Random Forests (RF)

Random forests is a supervised ***learning algorithm***. It can be used both for classification and regression. It is also the most flexible and easy to use algorithm. A forest is comprised of trees. It is said that the more trees it has, the more robust a forest is. Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance. Random forests has a variety of applications, such as recommendation engines, image classification and feature selection. It can be used to classify loyal loan applicants, identify fraudulent activity and predict diseases.

- Random forests is considered as a highly accurate and robust method because of the number of decision trees participating in the process.
- It does not suffer from the over fitting problem. The main reason is that it takes the average of all the predictions, which cancels out the biases.
- The algorithm can be used in both classification and regression problems.
- Random forests can also handle missing values

```
import pandas as pd
import numpy as np
from sklearn import preprocessing, cross_validation, neighbors, svm
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor

X = np.array(fit_data.drop(['loan_type_code'],1))
y = np.array(fit_data['loan_type_code'])
X_train, X_test, y_train, y_test_rfc = cross_validation.train_test_split(X, y, test_size = 0.2)

rfc = RandomForestClassifier(n_jobs=-1,max_features= 'auto' ,n_estimators=200, oob_score = True)
rfc.fit(X_train, y_train)
rfc_accuracy = rfc.score(X_test, y_test_rfc)
print('Unscalled: Random Forest Classifier Accuracy : ', rfc_accuracy)

X = np.array(fit_data.drop(['loan_type_code'],1))
y = np.array(fit_data['loan_type_code'])

from sklearn.preprocessing import scale
X_scaled = scale(X)

X_train, X_test, y_train, y_test_rfc = cross_validation.train_test_split(X, y, test_size = 0.2)
rfc = RandomForestClassifier(n_jobs=-1,max_features= 'auto' ,n_estimators=200, oob_score = True)
rfc.fit(X_train, y_train)
rfc_accuracy = rfc.score(X_test, y_test_rfc)
print('Scalled: Random Forest Classifier Accuracy   : ', rfc_accuracy)

# Look for the confussion Matrix
from sklearn.metrics import confusion_matrix
#confusion_matrix?
```

```
rfc.predict(X_test)
y_pred_rfc = rfc.predict(X_test)
y_pred_rfc_out = rfc.predict(X[975:995,])
print('RFC Confussion Matrix: ')
print(confusion_matrix(y_test_rfc, y_pred_rfc, labels=None, sample_weight=None))

print('KNeighborsClassifier Accuracy : ', rfc_accuracy)
print('RandomForestClassifier :Input Real Data : Conv=5, FHA=4, Res=3, Comm=2: ', X[975:995,])
print('RandomForestClassifier :Output of X[975:995]:Conv=5, FHA=4, Res=3, Comm=0:' , y[975:995])
print('RandomForestClassifier :Output of prediction: Conv=5, FHA=4, Res=3, Comm=:', y_pred_rfc_out)
print('RandomForestClassifier : - Output of prediction: Fix_True =1 & ARM =0:', y_pred_rfc)
```

Unscalled: Random Forest Classifier Accuracy : 0.7844311377245509
Scalled: Random Forest Classifier Accuracy : 0.8023952095808383
RFC Confussion Matrix:
[[7 19 0 0 0]
[0 348 12 0 2]
[0 37 33 0 1]
[1 5 0 0 0]
[0 19 3 0 14]]

'''With Random Forest Classifier we have height Accuracy of 78%, with scaling (Loan Types : Conv=5, FHA=4, Res=3, Comm=2:) prediction accuracy has ***improved to 80%*** '''

Let's analyze Interest Rate types using Random Forest Classifier. We have achieved greater accuracy compare to Loan Types. We have Fixed Rate mortgage where Fix_True = 1 & Fix_True = 0 for ARM (Adjustable Rate Mortgage)

RandomForestClassifier : Output of X[975:995]: Fix_True =1 & ARM =0: [1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
RandomForestClassifier : Output of prediction: Fix_True =1 & ARM =0: [1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
Random Forest Classifier Accuracy : 0.8862275449101796 or 89%
RFC Confussion Matrix:
[[13 44]
[13 431]]

Finding Important Features in Scikit-learn

Here, we are finding important features or selecting features in the Mortgage Loan dataset. In scikit-learn, we can perform this task in the following steps:

- First, we need to create a random forests model.
- Second, use the feature importance variable to see feature importance scores.
- Third, visualize these scores using the seaborn library.'''

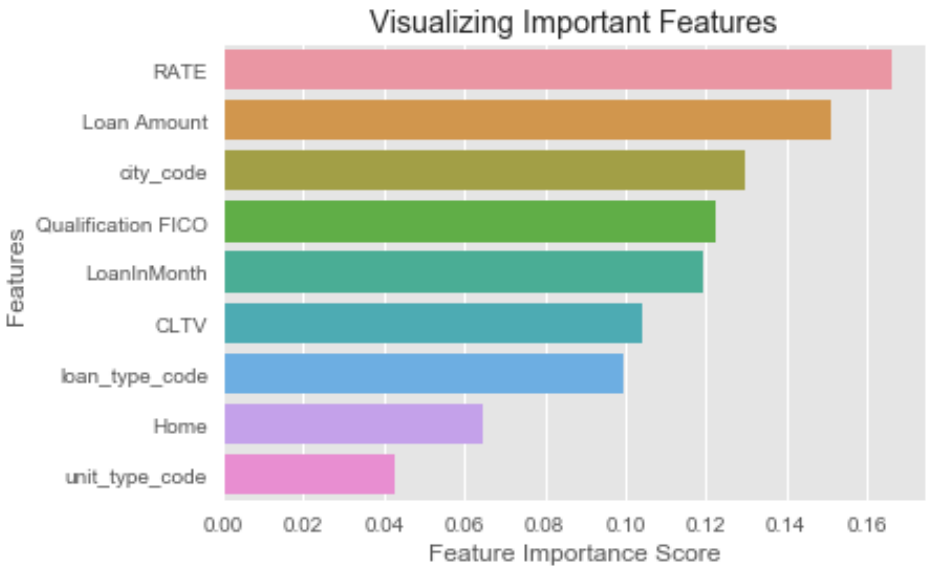
from sklearn.ensemble import RandomForestClassifier

```
#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
        max_depth=None, max_features='auto', max_leaf_nodes=None,
        min_impurity_decrease=0.0, min_impurity_split=None,
        min_samples_leaf=1, min_samples_split=2,
        min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
        oob_score=False, random_state=None, verbose=0,
        warm_start=False)

import pandas as pd

X = (model_data.drop(['loan_purpose_code'],1))
y = (model_data['loan_purpose_code'])

feature_cols = X.columns
fit_data.columns
feature_names = feature_cols
feature_imp = pd.Series(clf.feature_importances_,index=X.columns).sort_values(ascending=False)
feature_imp
```

'''We can also visualize the feature importance. Visualizations are easy to understand and interpretable. For visualization, we have used a combination of matplotlib and seaborn. Higher the value, greater the feature importance'''

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# Creating a bar plot
sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.legend()
plt.show()
```



Support Vector Regression (SVR)

SUPPORT VECTOR REGRESSION. Those who are in Machine Learning or Data Science are quite familiar with the term SVM or Support Vector Machine. But SVR is a bit different from SVM. As the name suggest the SVR is an regression algorithm, so we can use SVR for working with continuous Values instead of Classification which is SVM.

```
from sklearn import preprocessing, cross_validation, svm
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from matplotlib import style

clf = svm.SVR()
clf.fit(X_train, y_train)
accuracy = clf.score(X_test, y_test)
print('SVM-SVR accuracy: ', accuracy)
```

SVM-SVR accuracy: -0.0026638040886672876 which is unacceptable for our Mortgage Loan Data Sets

Support Vector Machine (SVM)

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

```
from sklearn import preprocessing, cross_validation, neighbors, svm

X_train, X_test, y_train, y_test_svm = cross_validation.train_test_split(X, y, test_size = 0.2)
#Define SVM support vector classifier
svmc = svm.SVC(kernel='rbf', C=10, gamma=0.001)
svmc.fit(X_train, y_train)
svm_accuracy = svmc.score(X_test, y_test_svm)
print('Support Vector Classifier Accuracy : ', svm_accuracy)

# Look for the confussion Matrix
from sklearn.metrics import confusion_matrix

svmc.predict(X_test)
y_pred_svm = svmc.predict(X_test)
print('SVC Confussion Matrix: ')
print(confusion_matrix(y_test_svm, y_pred_svm, labels=None, sample_weight=None))

from sklearn.cross_validation import cross_val_score
svm_scores = cross_val_score(svmc, X, y, cv=7, scoring='accuracy')
print('SVM: cross_val_score accuracy : ', svm_scores)
```

Support Vector Classifier Accuracy : 0.6447105788423154 or 64%
SVC Confussion Matrix:
[[2 17 1 0 1]
[9 290 32 2 22]
[2 53 24 0 3]
[0 1 0 1 0]
[1 31 3 0 6]]

Tunning Parameter for SVM

```
import pandas as pd
import numpy as np
from sklearn import preprocessing, cross_validation, neighbors, svm

X_train, X_test, y_train, y_test_svm = cross_validation.train_test_split(X, y, test_size = 0.2)
#Define SVM support vector classifier

from sklearn.grid_search import GridSearchCV
svc = svm.SVC()

param_grid = {
        "kernel" : ['linear', 'rbf', 'sigmoid'],
        "gamma" : [.1, 1, 10],
        "C" : [1, 5, 10]}

#print('SVM GridSearchCV : ', CV_svc.best_params_)
svmc = svm.SVC(kernel='rbf', C=1, gamma=1)
svmc.fit(X_train, y_train)
svm_accuracy = svmc.score(X_test, y_test_svm)

print('Support Vector Classifier Accuracy : ', svm_accuracy)
```

Support Vector Classifier Accuracy : 0.6866267465069861 or 69%

Because of tuning the parameter SVM accuracy has ***improved 8%***

k-Nearest Neighbors (KNN)

k-Nearest Neighbors: FIT

Having explored the Congressional mortgage records dataset, we have build our classifier. Here, we will fit a k-Nearest Neighbors classifier to the mortgage dataset. The features need to be in an array where each column is a feature and each row a different observation or data point. The target needs to be a single column with the same number of observations as the feature data. Notice we named the feature array X and response variable y: This is in accordance with the common scikit-learn practice.

We need create an instance of a k-NN classifier with 6 neighbors (by specifying the n_neighbors parameter) and then fit it to the data.

k-Nearest Neighbors: Predict

Having fit a k-NN classifier, we can use it to predict the label of a new data point. However, there is no unlabeled data available since all of it was used to fit the model! We will use your classifier to predict the label for this new data point, as well as on the training data X that the model has already seen.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import preprocessing, cross_validation, neighbors
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split # for train and test set split
from sklearn.model_selection import cross_val_score

X = np.array(fit_data.drop(['loan_type_code'],1))
y = np.array(fit_data['loan_type_code'])
X_train, X_test, y_train, y_test_knn = cross_validation.train_test_split(X, y, test_size = 0.2)

from sklearn.neighbors import KNeighborsClassifier
print("size of the training feature set is",X_train.shape)
print("size of the test feature set is",X_test.shape)
print("size of the training Target set is",y_train.shape)
print("size of the test Target set is",y_test.shape)

# Import scale
from sklearn.preprocessing import scale

# Scale the features: X_scaled
X_scaled = scale(X)

# Print the mean and standard deviation of the unscaled features
print("\n\nMean of Unscaled Features: {}".format(np.mean(X)))
print("Standard Deviation of Unscaled Features: {}".format(np.std(X)))

# Print the mean and standard deviation of the scaled features
print("\n\nMean of Scaled Features: {}".format(np.mean(X_scaled)))
print("Standard Deviation of Scaled Features: {}".format(np.std(X_scaled)))
knn = neighbors.KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
knn_accuracy = knn.score(X_test, y_test_knn)
print('\nKNeighborsClassifier Accuracy : ', knn_accuracy)
knn_prediction = knn.predict(X[1200:1220,])
print('\nKNN : - Input of Real Data :: ', X[1200:1220,])
print('KNN : - Output of Real Data :: ', y[1200:1220,])
print('KNN : - Output of prediction:: ', knn_prediction)
```

size of the training feature set is (2000, 9)

size of the test feature set is (501, 9)
size of the training Target set is (2000,)
size of the test Target set is (751,)

Mean of Unscaled Features: 114.7400015416056
Standard Deviation of Unscaled Features: 200.86870126441687
Mean of Scaled Features: -4.6087893474154617e-17
Standard Deviation of Scaled Features: 1.0

KNeighborsClassifier Accuracy for Loan Types: 0.6826347305389222 or 68%
KNN : - Output of Real Data : Conv=1, FHA=2, Res=4, Comm=0: [1 0 0 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 0 2]
KNN : - Output of prediction: Conv=1, FHA=2, Res=4, Comm=0: [1 1 0 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2]

Preprocessing: scaling

Here below I (i) scale the data, (ii) use k-Nearest Neighbors and (iii) check the model performance. I'll use scikit-learn's scale function, which standardizes all features (columns) in the array passed to it.'''

```
from sklearn.preprocessing import scale
Xs = scale(X)
from sklearn.cross_validation import train_test_split
Xs_train, Xs_test, y_train, y_test = train_test_split(Xs, y, test_size=0.2)
knn_model_2 = knn.fit(Xs_train, y_train)
print('k-NN score for test set: %f' % knn_model_2.score(Xs_test, y_test))
print('k-NN score for training set: %f' % knn_model_2.score(Xs_train, y_train))
y_true, y_pred = y_test, knn_model_2.predict(Xs_test)
print(classification_report(y_true, y_pred))
```

k-NN score for test set: 0.678643
k-NN score for training set: 0.770000 or 77%

	precision	recall	f1-score	support
0	0.46	0.38	0.41	16
1	0.73	0.88	0.80	347
2	0.41	0.21	0.27	87
3	0.00	0.00	0.00	2
4	0.48	0.21	0.29	48
5	0.00	0.00	0.00	1
avg / total	0.64	0.68	0.64	501

All these measures improved by 0.1325, which is a **13.25% improvement and significant**! As hinted at above, before scaling there were a number of predictor variables with ranges of different order of magnitudes, meaning that one or two of them could dominate in the context of an algorithm such as k-NN. The two main reasons for scaling our data are

Our predictor variables may have significantly different ranges and, in certain situations, such as when implementing k-NN, this needs to be mitigated so that certain features do not dominate the algorithm; We want our features to be unit-independent, that is, not reliant on the scale of the measurement involved. If we both scale our respective data, this feature will be the same for each of us.

Decision Tree Classifier

Using Scikit-learn, optimization of decision tree classifier performed by only pre-pruning. Maximum depth of the tree can be used as a control variable for pre-pruning. In the following the example, we can plot a decision tree on the same data with max_depth=4. Other than pre-pruning parameters, We have also tried other attribute selection measure such as entropy

This pruned model is less complex, explainable, and easy to understand than the previous decision tree model plot.

Pros
Decision trees are easy to interpret and visualize.
It can easily capture Non-linear patterns.
It requires fewer data preprocessing from the user, for example, there is no need to normalize columns.
It can be used for feature engineering such as predicting missing values, suitable for variable selection.
The decision tree has no assumptions about distribution because of the non-parametric nature of the algorithm.
Cons
Sensitive to noisy data. It can overfit noisy data.
The small variation(or variance) in data can result in the different decision tree.
Decision trees are biased with imbalance dataset, so we can balance out the dataset before creating the decision tree.

Decision Tree Algorithm
A decision tree is a flowchart-like tree structure where an internal node represents feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning. This flowchart-like structure helps you in decision making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
# Split dataset into training set and test set
X = np.array(fit_data.drop(['Fix_True'],1))
y = np.array(fit_data['Fix_True'])
```

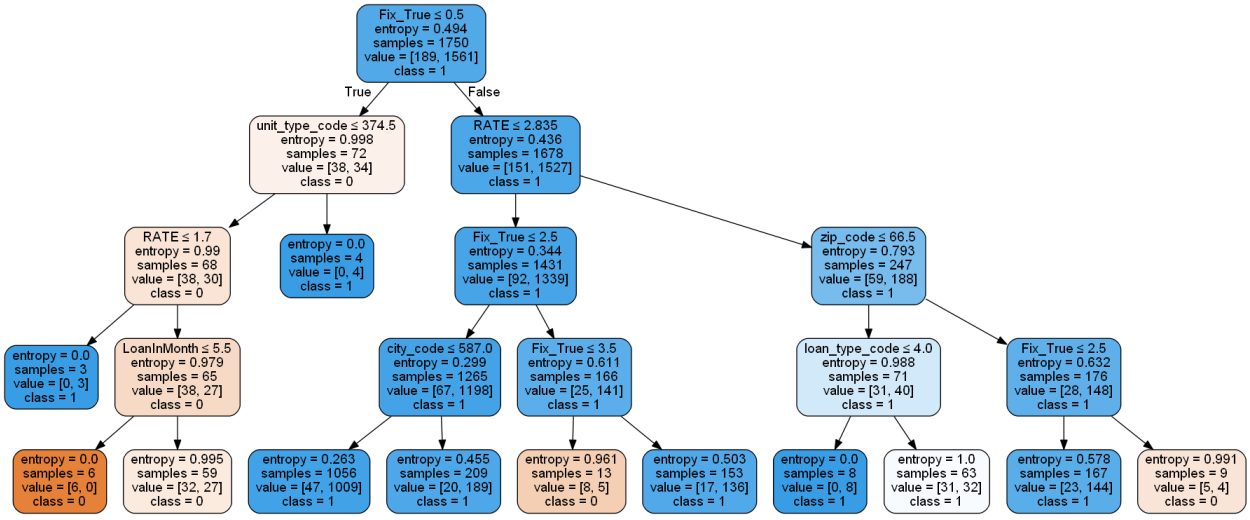
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70% training and 30% test

#Building Decision Tree Model Let's create a Decision Tree Model using Scikit-learn.
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
# Import scale
from sklearn.preprocessing import scale
# Scale the features: X_scaled
X_scaled = scale(X)

clf.predict(X_test)
y_pred_dt = clf.predict(X_test)
y_pred_dt_out = clf.predict(X[975:995,])
print('DecisionTre Confussion Matrix: ')
print(confusion_matrix(y_test, y_pred_dt, labels=None, sample_weight=None))

dt_accuracy = clf.score(X_test, y_test)
print('Decision Tree Classifier Accuracy : ', dt_accuracy)
```

print('Decision Tree Classifier : - Input of Real Data : Fix_True =1 & ARM =0: ', X[975:995,])
print('Decision Tree Classifier : - Output of X[975:995]: Fix_True =1 & ARM =0:' , y[975:995])
print('Decision Tree Classifier : - Output of prediction: Fix_True =1 & ARM =0:', y_pred_dt_out)

Decision Tree Classifier Accuracy : 0.833555259653795 or 83.4%
Decision Tree Classifier: - Output of X[975:995]: Fix_True =1 & ARM =0: [1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
Decision Tree Classifier : - Output of prediction: Fix_True =1 & ARM =0: [1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1]

Visualizing Decision Trees

We have used Scikit-learn's export_graphviz function for display the tree within a Jupyter notebook. For plotting tree, you also need to install graphviz and pydotplus. export_graphviz function converts decision tree classifier into dot file and pydotplus convert this dot file to png or displayable form on Jupyter Notebook.



The most important features of our model are as follows:

- **The financial institution's interest rate**
- **Changes in the financial institution's interest rates**
- **The amount of mortgage applications on the previous day**

By analyzing the results of our model, we can see that in particular the 'outliers' (i.e. the days with an extremely high amount of mortgage applications) are consistently under-predicted. These outliers are often influenced by changes in mortgage interest rates, which implies that there is still room for Improvement in our model. As the changes in interest rate is one of the most important features of our model and are influenced by many factors, and hence hard to predict, a dynamic dashboard is proposed. In this dashboard, interest rate changes can be entered manually, so that their impact on the amount of mortgage applications is shown real-time.

A predictive model was created using the SVM, KNN, Random Forest, and Deaccession Tree technique, which predicts the amount of mortgage applications per day(LoanInMonth) with a mean absolute error of mortgage applications per day. This can directly be converted to the amount of personnel needed at the mortgage application department of the financial institutions, by dividing it by the amount of mortgage applications handled per person per day. Based on mortgage per months, company can manage core human resources, such as Loan Processors, Mortgage Loan Originators, Underwriters, secondary market analyst, lock desk personal, compliance personals, & others.

The mortgage interest rates have the biggest impact on our model, but are difficult to predict.
Several features (**10 Years US Treasury Rate, Home Supply Index**) can be added to the model in order to improve its predictive power. Furthermore, there is still improvement in the feature regarding mortgage interest rate changes, as a significant part of the error of our model is caused by under-prediction of the outliers.

7.3 MODEL VALIDATION

Since our models are trained on historical data, we are not sure how these models will perform during the forecasting. In general, the explanatory power of the model on historical data is almost always higher than then predictive power of the model on new data. If this difference is too big, we talk about over fitting. This happens when the model is too complex and starts capturing noise as well. In order to prevent over fitting and accurately estimate the predictive power of our model, we use repeated 10-fold cross-validation. 10-fold cross-validation is one of the most widely used methods for estimating the prediction error. Using 10-fold cross-validation, we are able to split the dataset in out-of-sample data and in-sample data. The dataset is randomly split in 10 equally sized subsets, and the model is run 10 times where each of the 10 subsets is used as the validation set once. The other 9 sets are used as training sets. By changing the validation set at every run, every data point has been used for validation exactly once. After 10 observations, the results of the observations are averaged. This entire process is repeated five times, in order to account for outliers affecting the cross-validation outcome. Afterwards, the performance of the five repeats is averaged to calculate the final performance of the models. Using this method, we can get a feeling of how the model will perform on 'unknown' data. This is still not a guarantee that the model will perform the same on live data, but it will give a decent approximation.

8 EVALUATION

In the Evaluation stage, the models are evaluated using several evaluation criteria, and the results of the models are discussed. A final selection of the best model is made, using an independent t-test to check if the best model performs significantly better than the other models. Also the most important features for our models are selected.

8.1 MODEL EVALUATION

In order to measure the model's performance we use four different metrics: Root Mean Square Error (RMSE), Mean Absolute Error (MAE), MAE/Mean and R2. These are some of the most common metrics used for the evaluation of regression models. Each of these metrics use the residuals (i.e. the differences between the observed values and the predicted values) in order to measure the performance of the model.

RMSE uses a squared value of the absolute error, hence giving more emphasis on higher residuals. This metric is particularly useful because of this last characteristic. One of the key elements of the model is that it should be able to predict 'outliers', i.e. the non-standard days, with an unusually high amount of mortgage applications. RMSE takes this into account by giving more

8.2 DISCUSSION OF RESULTS

Using these metrics to evaluate the performance of the models, we can already see differences between the models. We can see that the Random Forest model performed best, not only in terms of RMSE, but also in MAE, MAE/Mean and R2. SVM slightly worse model. We can also see that the Decisions Tree model performs better, which was expected. This is a basic decision tree, whereas RF ensemble methods of multiple decision trees. In order to test whether the difference in RMSE between RF are statistically significant, an independent t-test can be conducted ($p < 0.05$). For the t-test, the out-of-fold performances (i.e. the predictions on the 10 different folds that were used as validation set in our cross-validation) of the five repeats of the RF and KNN models are compared

using the . Using a 95% significance level, there appears to be a significant difference between the average RMSE of the RF and Decision Tree models.

We have also compared Logistic Regression and KNN. We can see big improvement on model accuracy rate with centering scaling and data synthesizing. We have seen the essential place in the data scientific pipeline by preprocessing, in its scaling and centering incarnation, and we have done so to promote a holistic approach to minimize the challenges of machine learning.

We have also compared five different types of Linear Regression model. LASSO Regression and Elastic Net Regression seems to be top performers.

NYC NJ (zip_code 1st digit between 7 & 11) seems to be closing more loans and generation more revenues for the Mortgage Bank. City shows similar results. Population density plays bigger role on Loan Amount.

Loan Amount increases with 10 Years Treasury Rate and Home Supply. As number of loans per month increases, loan amount decreases. As interest Rate goes up, monthly number of loans goes down but average Loan Amount goes up.

Majority of Loans Amount ranges between $400000 to $600000. Majority of the CLTV scores between 30 and 100 (Loan Amounts in 1000s). CLTV goes up, Loan Amount slightly goes down, but CLTV does not have much impact on the Loan Amount.

Majority of the FICO scores between 600 and 820 (Loan Amounts in 1000s). FICO goes up, Loan Amount goes up, which make sense; people with good credit score have more borrowing power. People with the lower credit score prefer FHA Loan Types over Conventional Loan Types. Conventional Loan is the leader among the loan types. Conventional Mortgage has Loan volume compare to FHA; Conventional is high volume but slightly low average loan amount

As number of unit decreases average Loan Amount also decreases. One and Two family houses seem to be more popular among the borrowers. Two Family (Code = 10) has higher Loan Amount that three Family (Code = 9).

10 CONCLUSIONS, LIMITATIONS AND FURTHER RESEARCH

10.1 CONCLUSIONS

Q1. What are the variables that influence the amount of mortgage applications?

We have shown a list of 24 features for our model is developed, for a total of 5 categories.. Eventually, a total of 12 variables is used for our model, which can roughly be grouped in 5 categories: variables related to seasonality (e.g. Loan Type, Unit Type, Loan Purpose, Loan origination City, Zip), mortgage interest rates (e.g. the US10Y & Home Supply Rate) and Loan Amount, CLTV, Loan Estimates, FICO Scores. These 12 variables are used for making the predictions.

Q2. Which techniques and algorithms can be used to create a model that is able to predict the amount of mortgage applications?

As discussed, there are two main type of prediction problems: classification problems and regression problems. Our research contains a regression problem, for which a specific set of machine learning algorithms can be used. We selected a list of 5 algorithms that can be used in our research: Support Vector Machine, KNN, Classification and decision Trees, Linear or Logistic Regression and Random Forests.

Q3. Which technique performs best on our dataset?

Each of the models mentioned above was trained on the dataset, and using a t-test with a 95% significance level we can conclude that the Decision Trees and Random Forest produces the best results in terms of Root Mean Square Error (RMSE) and Model Accuracy. With Centering, Scaling and Data Synthesizing we have seen great improvement for KNN and Logistic Regression. SVM seems to be performing poorly bust SVR is the worse model.

Q4. How can we use this model to determine the amount of personnel needed during the next week?

By incorporating the Random Forest model into a dashboard, visualization can be made of the predicted amount of mortgage applications for the next week. Since we do not have data available upfront regarding mortgage interest rate changes, a dynamic dashboard is proposed. In this dynamic dashboard, mortgage interest rate changes can be entered manually, and the effect of changing the mortgage interest rate on the amount of mortgage applications can be seen real-time. This dashboard can be used to make a prediction of the amount of personnel needed, by linking the amount of mortgage applications with the amount of personnel needed to handle these mortgage applications.

Q5. How can Historical mortgage application data be used to predict the amount of mortgage applications per day for the next week?

In our models we used historical mortgage data to build a machine learning model that predicts the amount of mortgage applications for the next month, using several machine learning techniques discussed above. Our best model has a Mean Absolute Error of across a repeated 10-fold cross-validation, which indicates that the model's predictions of the amount of mortgage applications per day are on average applications higher or lower than the actual amount. By using historical Training and Test data on our predictive model, we can predict number of the mortgage applications next month or even next years with over 80% accuracy.

By using this model to predict the amount of personnel needed to handle these mortgage applications, the financial institutions can save personnel costs and reduce the throughput time of the mortgage applications. A dynamic dashboard is proposed to visualize the amount of mortgage applications, in which interest rate changes can be manually entered in the dashboard.

10.2 LIMITATIONS

First, not all of the features could be incorporated in our model, due to the sensitivity and limited availability of external data. For example, for the profit margin, loan origination cost, MLO commissions, loan processing delay, loan denial rate, and income growth the data was either unavailable or aggregated on a yearly level, which makes these predictors useless for our model. Also the historical interest rates of competitors in the US mortgage market could not be included, due to the unavailability of data. Although the actual interest rates of the competitors are known, there was no overview of the historical interest rates.

Another limitation is the lack of historical data that was available regarding mortgage applications. We have only used data from October 2014, which means we only had a certain amount of data available for events that occur on a yearly level, such as the changes in regulations, housing bubbles, stock market crash, interest rate fluctuations, inflation data, unemployment data, gdp data, consumer credit cards debt data. By having more historical data we would be able to capture the impact of a change in regulations more accurately, and make trends over the last few years better identifiable.

Furthermore, several non-repeating events at the financial institutions that do not follow a regular pattern but still have impact on the amount of mortgage applications (e.g. downtime of services or system failure, changes in acceptation criteria at the financial institutions, special offers at the financial institutions) were not included in our model due to scope and time restrictions. By including these events, the predictive power of our model should slightly increase, as the model will be better at explaining the outliers in the dataset.

10.3 RECOMMENDATIONS FOR FURTHER RESEARCH

Based on the limitations mentioned above, a number of recommendations for further research are defined. These recommendations can roughly be grouped into three categories: follow-up research, improving the predictive model and validating the dynamic dashboard.

Regarding the follow-up research, a next step would be to use the predictions of our model for determining the amount of personnel needed and predicting monthly revenue with more dependent variable and higher accuracy. This can be done in multiple ways, either by using a mathematical formula, or by using the predictions in a new model that predicts the amount of personnel needed for the next month. Other factors can be included in this new model, such as the expected processing delay and denial rate of mortgage applications. This could be the subject of a follow-up research.

The economic situation, the housing market or mortgage interest rates influence the amount of mortgage applications on the short term. By analyzing news messages from multiple news websites, this effect could be captured and implemented in our model. Further research could be focused on extracting information from multiple media sources, to see if the model's predictive power can be improved. Next to this, data regarding the marketing budget of the financial institution could be added in our model, to see if the size of the marketing budget has impact on the amount of mortgage applications and can be used to improve the predictions.

Furthermore, the model was unable to accurately predict the outliers in our dataset. The outliers were generally under-predicted, due to interest rate changes not being captured completely by our model. For further research we would suggest to look into this feature and investigate if there are any other factors that influence the amount of mortgage applications during an interest rate change.

.