Analysis of Lending Club Data using Machine Learning

Trupti Gore | Tushar Goel
gore.t@husky.neu.edu, goel.t@husky.neu.edu
CSYE 7245, Spring 2018, Northeastern University

Abstract

The Lending loan club is a platform designed to bring together the investors and borrowers by encouraging a fair monetary transaction of lending and borrowing. This has helped to foster growth of small businesses and let people take control of their debt.

An interested applicant(borrower) is only needed to create a profile and enter his detail stating his requirements. The system is designed to check the user's eligibility for the quoted amount based on his/her financial standing and market situation. The user is intimated about his eligibility and if eligible also the amount for which the loan was sanctioned with the interest rate. This is the most critical aspect and is the business generator for Lending Club. It has to be ensured that the given interest rate is fair to both the customer and the bank from the transaction point of view.

Introduction

Our project revolves around creating the business model for the Lending Club loan. Our machine learning model is capable of delivering the prime aspect of predicting fair interest rates only when the user is eligible for the loan amount. It works in two phases.

Initial Stage :

An interested applicant creates a profile for themselves using a registered email id and password. As a part of profile creation, a user is needed to input his/financial history as asked by the system. The questionnaire starts with the basic question of whether the registrant is a lender or a borrower. Each of them have separate set of questions. Our model is only designed to handle a borrower for now.

At the end of the questionnaire, he/she is needed to quote the amount he/she is seeking loan with a reason.

First Phase :

Depending on the user fed criteria, user's eligibility is decided. If eligible, the user's request is further taken into consideration for the quoted amount for further estimation.

Second Phase :

The user lands into this phase if and only if he clears the phase 1. This phase is the most critical and it is this phase which must strike the balance between understanding the customer's financial standing keeping in mind the business aspect for the company and play fair to both the parties. This is the key to the success of the business. Our model was carefully designed to consider the important pointers of a user's financial record, and let that be the deciding factor for the interest rates. Factors like FICO score, was considered before predicting the interest rate and loan money sanctioned to the user.

Method of Approach

The data is available on Kaggle as a readily available dataset. But since we wanted to ensure that we work on the latest financial history of the customer, we designed a LUIGI pipeline which extracted the data from the Lending Club website (masking user's personal information) from 2007 until 2017.

Using LUIGI :

Since we both were new to using LUIGI, we had to first learn how it functions. Second challenge was to accommodate it into our piece of project. It took us a while to structure the pipeline. We faced issues while setting the pipeline. With help from blogposts and stack-exchange posts, we were able to get rid of the errors and successfully download data from the website.

Dataset :

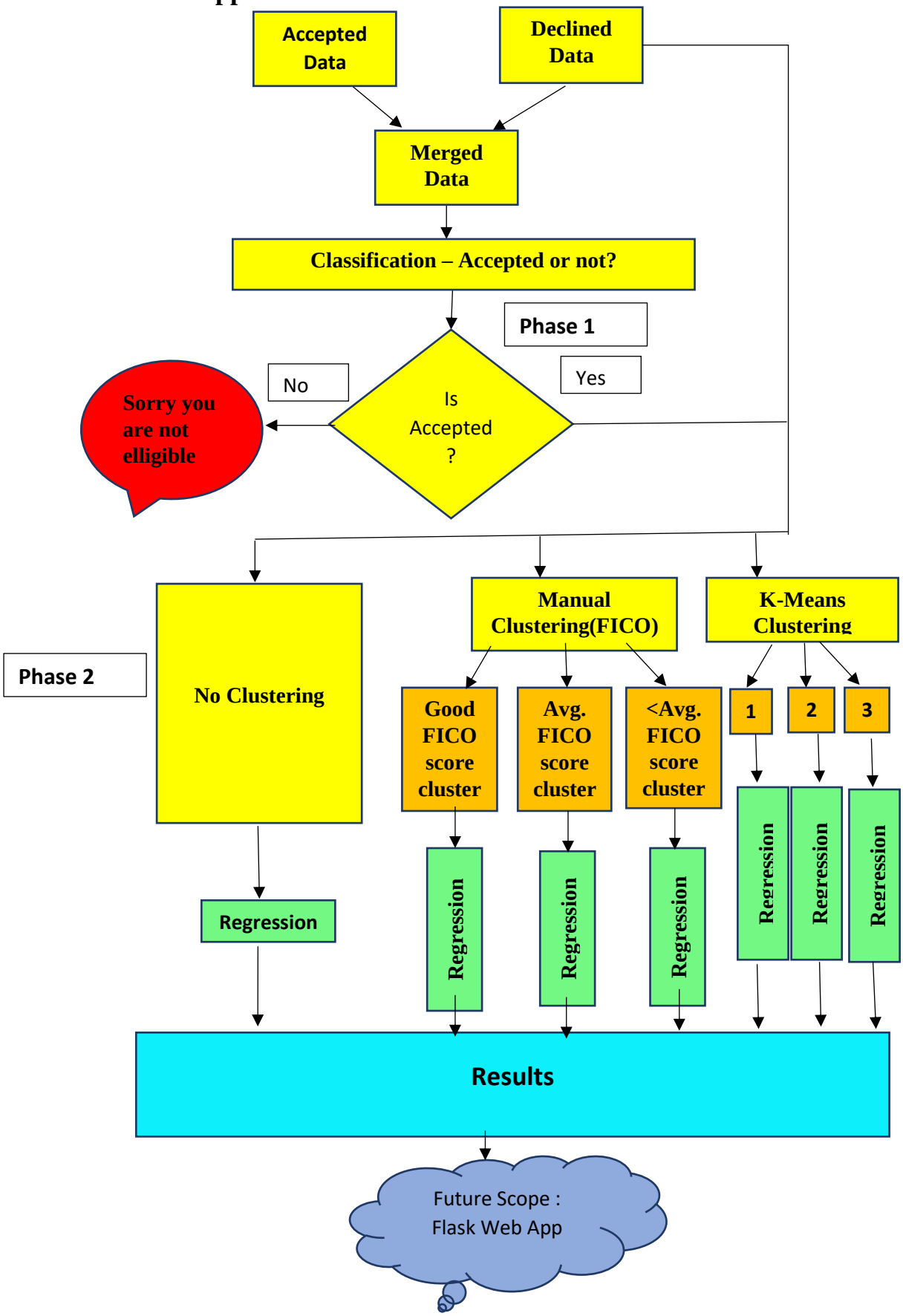
We created two .py files to extract the loaned data, and the declined for loan data through two separate pipelines. For the columns with more than 80% of data missing, we deleted them since these would not add value to our model. Secondly, we removed the junk like punctuations and repeated words which did not add value. We also standardized some columns. For example, we converted the month column in the range 1-12 instead of their names. We filled missing values for the remainder of columns with missing values either by using mean, median or max values of that column. The original downloaded data had 146 columns and over 10 million rows. After data cleaning we were able to delete 37 columns, which landed us with 109 columns.

Using 109 columns to predict the eligibility of a customer would only give us inaccurate results. Hence, we feature engineered our data to extrapolate only the value adding columns for our Phase 1 of the project.

Feature Selection

In [12]:
```
import pandas as pd
model_features = pd.read_csv('/Users/Trupti/Final Project - BDSIA/CleanedLoanData.csv', encoding="ISO-8859-1"
    #Begining feature selection
#We have more than 100 features but in our model we can only use the columns that a user can eneter
#From the dictionary we got an idea of the features that a user can enter and we build our model on that info
Y = model_features.int_rate
model_features.drop('int_rate', axis=1, inplace=True)
user_entered_cols = ['loan_amnt', 'term', 'emp_length', 'home_ownership', 'annual_inc',
                                'verification_status', 'purpose', 'dti', 'delinq_2yrs',
                                'inq_last_6mths', 'open_acc', 'revol_bal', 'revol_util', 'total_acc',
                                'mths_since_last_major_derog', 'funded_amnt_inv', 'installment', 'application
                                'addr_state']
model_features = model_features[user_entered_cols]
model_features=pd.get_dummies(model_features, columns=["purpose"])
model_features=pd.get_dummies(model_features,columns=["application_type"])
X = model_features._get_numeric_data()
ranks={}
def rank_to_dict(ranks, names, order=1):
            minmax = MinMaxScaler()
            ranks = minmax.fit_transform(order*np.array([ranks]).T).T[0]
            ranks = map(lambda x: round(x, 2), ranks)
            return dict(zip(names, ranks ))
```

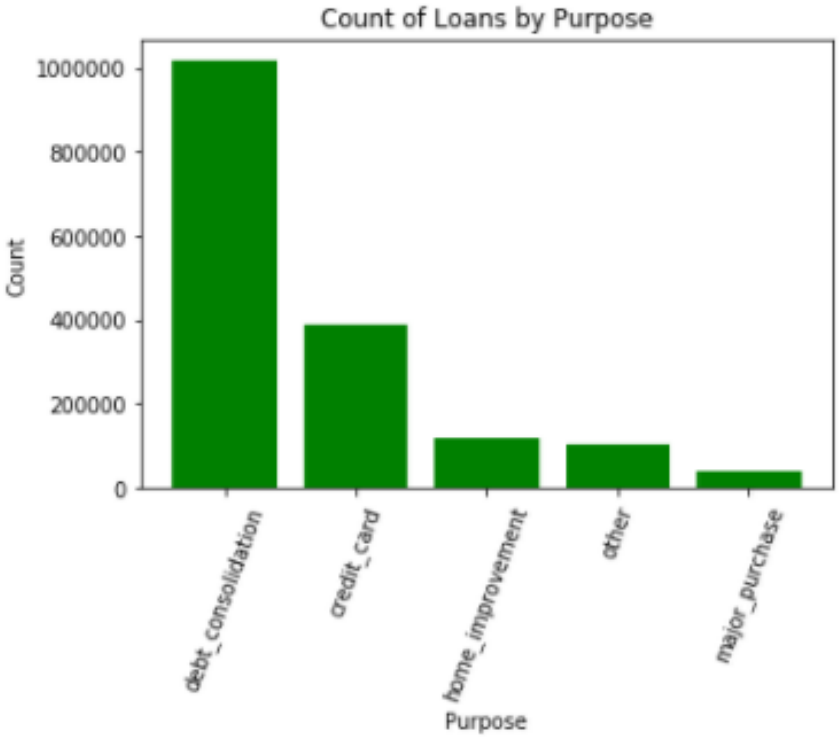
Structure of our Approach :



Exploratory Data Analysis/Data Mining

We plotted some features to understand how the data was trending

Purpose of Loan :

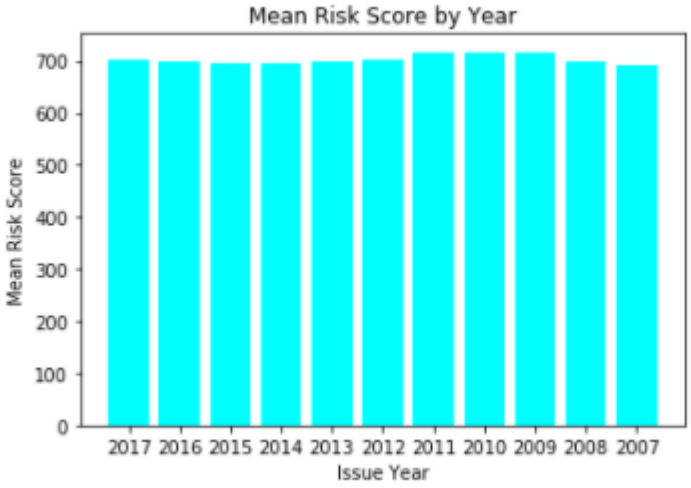


Observation : Loans are higher for paying off consolidated debt

Interest Rate Year by Year :

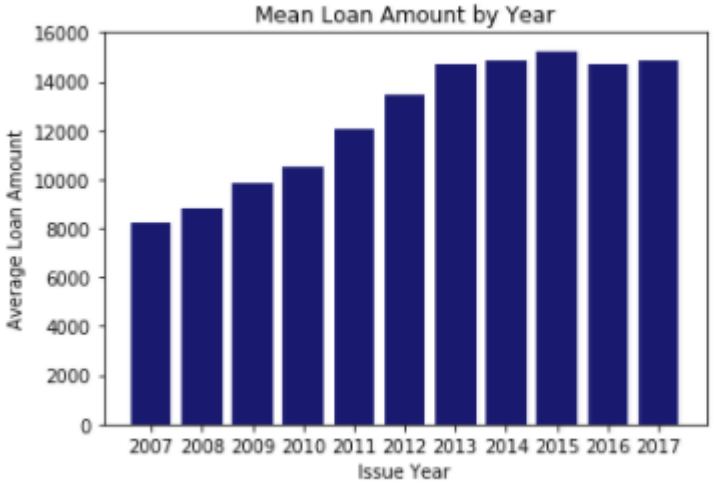
Observation : The mean interest rate was highest in the year 2013 and dropped in 2015 only to increase but not as high as 2013

Mean Risk Score



Observation : Mean Risk Score has almost remained at par with the past few years

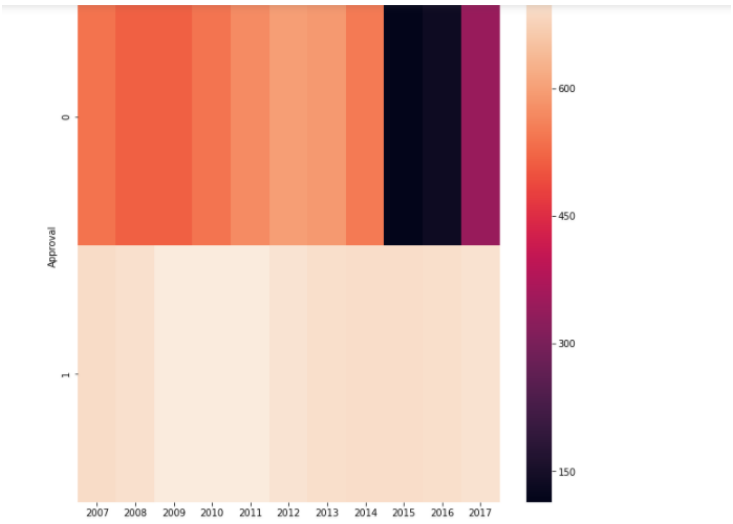
Mean Loan Amount year by year :



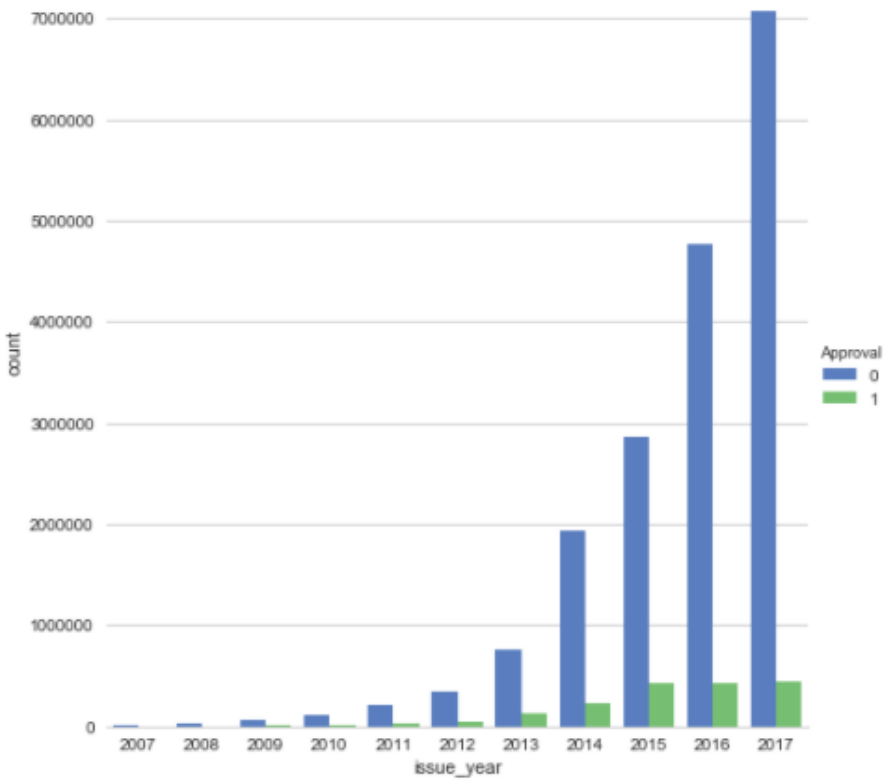
Observation : The mean loan amount increased until 2015 and then a sudden drop was noticed in 2016.

Mean Amount loaned by State

Heatmap for approval based on Risk Score



Trend of Approvals to Declined :



Observation : The approval rate increased year by year

Let's now play with the data !

Our exploratory Data Analysis gave us successful insights of the trend of the data. Now let us deep dive into the focus of the task – Leveraging Machine Learning to build a model to serve end to end purpose of handling a borrower.

Phase 1 :

For the phase 1, we first needed to extract important feature extracted columns of top importance from the data cleansed files of loaned data and declined data. We merged them into one as a final dataset.

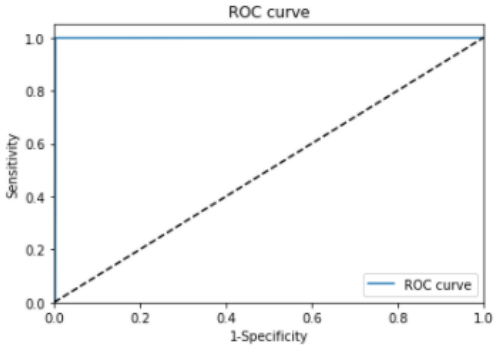
```
In [9]: resultdata.columns

Out[9]: Index(['amount_requested', 'Risk_Score', 'dti', 'State', 'emp_length',
               'policy_code', 'app_month', 'status'],
              dtype='object')
```

Dividing the data into test and train in the proportion of 30-70,

```
In [46]: train, test = train_test_split(resultdata, train_size = 0.7)
```

```
In [47]: train_y = train['status']
         test_y = test['status']
         train_x = train.loc[:, train.columns != 'status']
         test_x = test.loc[:, test.columns != 'status']
```

we trained our classifier using Logistic Regression, Random Forest Classifier for an accuracy of 99.7% and 100%. We have also built a Multi-Layer Perceptron classifier but only on a sample of the data, since it threw a memory error when used on the above entire dataset. This gave us an accuracy of 99.92%.



Classification	Linear Regression	MLP Regressor	Random Forest Regressor
Test Accuracy	99.73	99.92	100

Phase 2 :

A customer lands into this zone if and only if he is eligible for the loan. The data used here is the loaned dataset. We wanted to have an unbiased approach to provide the best interest rate to our clients. Hence we used a pre-processing here before we dived into predicting interest rates. It was necessary to bucket clients since the data was immense. We built 3 models to do this :

Clustering on basis of :

Risk Score(Manual Clustering)
KNN Clustering
No Clustering

Lets us now study our methodology of approach for each of these models and the resultant output of the models :

Risk Score Clustering

Which is nothing but the FICO Score appended. We created this column which was nothing but the average of the max and min FICO score.

We have used Linear Regression, MLP Regressor, KNN Regressor and Random Forest Regressor for clustering our data. We got best results for Random Forest Regressor

In [5]:
```
#Multple Linear Regression Case 1:
reg=linear_model.LinearRegression()
reg.fit(x_train,y_train)
predicted_train=reg.predict(x_train)
predicted_test=reg.predict(x_test)
print("Testing mean absolute error : %f" %mean_absolute_error(y_test,predicted_test))
print("Training mean absolute error: %f" %mean_absolute_error(y_train,predicted_train))
rmse=np.sqrt(mean_squared_error(y_test,predicted_test))
print("RMSE Value for Testing")
print(rmse)
rmse=np.sqrt(mean_squared_error(y_train,predicted_train))
print("RMSE Value for Training")
print(rmse)
print("MAPE Value for Testing")
print(mean_absolute_percentage_error(y_test, predicted_test))
print("MAPE Value for Training")
print(mean_absolute_percentage_error(y_train, predicted_train))
```

```
Testing mean absolute error : 2.226257
Training mean absolute error: 2.237564
RMSE Value for Testing
2.855180635495495
RMSE Value for Training
2.874954543765836
MAPE Value for Testing
15.420866306236158
MAPE Value for Training
```

In [9]:
```
#RandomForest Regressor
from sklearn.ensemble import RandomForestRegressor

z='auto'
x=100
y=1
model=RandomForestRegressor(n_estimators=x,max_features=z,oob_score=True,n_jobs=-1,random_state=50,min_samples_leaf=y)
model.fit(x_train,y_train)
predicted_train=model.predict(x_train)
predicted_test=model.predict(x_test)
print("Tuning Combination")
print(x," ",y ," ", z,"")
print("Testing mean absolute error : %f" % mean_absolute_error(y_test,predicted_test))
print("Training mean absolute error: %f" %mean_absolute_error(y_train,predicted_train))
rmse=np.sqrt(mean_squared_error(y_test,predicted_test))
print("RMSE Value for testing")
print(rmse)
rmse=np.sqrt(mean_squared_error(y_train,predicted_train))
print("RMSE Value for training")
print(rmse)
print("MAPE Value for Testing")
print(mean_absolute_percentage_error(y_test, predicted_test))
print("MAPE Value for Training")
print(mean_absolute_percentage_error(y_train, predicted_train))
```

```
Tuning Combination
100   1   auto
Testing mean absolute error : 0.673678
Training mean absolute error: 0.253242
```

In [6]:

```
#KNN Regressor
# Create the knn model.
# Look at the five closest neighbors.
knn = KNeighborsRegressor(n_neighbors=5)
# Fit the model on the training data.
knn.fit(x_train, y_train)
# Make point predictions on the test set using the fit model.
predicted_train = knn.predict(x_train)
predicted_test = knn.predict(x_test)
print("Testing mean absolute error : %f" %mean_absolute_error(y_test,predicted_test))
print("Training mean absolute error: %f" %mean_absolute_error(y_train,predicted_train))
rmse=np.sqrt(mean_squared_error(y_test,predicted_test))
print("RMSE Value for Testing")
print(rmse)
rmse=np.sqrt(mean_squared_error(y_train,predicted_train))
print("RMSE Value for Training")
print(rmse)
print("MAPE Value for Testing")
print(mean_absolute_percentage_error(y_test, predicted_test))
print("MAPE Value for Training")
print(mean_absolute_percentage_error(y_train, predicted_train))
```

```
Testing mean absolute error : 3.557643
Training mean absolute error: 2.881304
RMSE Value for Testing
4.611642756926346
```

Cluster 1	Linear Regressor	KNN Regressor	MLP Regressor	Random Forest Regressor
Test rmse	2.22	3.54	4.87	0.46
Train rmse	2.23	2.89	4.87	0.17

Cluster 2	Linear Regressor	KNN Regressor	MLP Regressor	Random Forest Regressor
Test rmse	2.22	3.55	54.50	0.67
Train rmse	2.22	2.88	53.45	0.25

Cluster 3	Linear Regressor	KNN Regressor	MLP Regressor	Random Forest Regressor
Test rmse	1.83	2.70	23.22	1.02
Train rmse	1.83	2.20	23.79	0.39

Cluster 4	Linear Regressor	KNN Regressor	MLP Regressor	Random Forest Regressor
Test rmse	1.47	2.1	25.43	0.92
Train rmse	1.45	1.68	24.23	0.40

K-Means Clustering Model

We have used Linear Regression, MLP Regressor, KNN Regressor and Random Forest Regressor for clustering our data. We got best results for Random Forest Regressor

```
#Multple Linear Regression Case 1:
reg=linear_model.LinearRegression()
reg.fit(x_train,y_train)
predicted_train=reg.predict(x_train)
predicted_test=reg.predict(x_test)
print("Testing mean absolute error : %f" %mean_absolute_error(y_test,predicted_test))
print("Training mean absolute error: %f" %mean_absolute_error(y_train,predicted_train))
rmse=np.sqrt(mean_squared_error(y_test,predicted_test))
print("RMSE Value for Testing")
print(rmse)
rmse=np.sqrt(mean_squared_error(y_train,predicted_train))
print("RMSE Value for Training")
print(rmse)
print("MAPE Value for Testing")
print(mean_absolute_percentage_error(y_test, predicted_test))
print("MAPE Value for Training")
print(mean_absolute_percentage_error(y_train, predicted_train))
```

```
Testing mean absolute error : 1.019415
Training mean absolute error: 1.015879
RMSE Value for Testing
1.7188914537
RMSE Value for Training
1.73668565301
```

```
#KNN Regressor


# Create the knn model.
# Look at the five closest neighbors.
knn = KNeighborsRegressor(n_neighbors=5)
# Fit the model on the training data.
knn.fit(x_train, y_train)
# Make point predictions on the test set using the fit model.
predicted_train = knn.predict(x_train)
predicted_test = knn.predict(x_test)
print("Testing mean absolute error : %f" %mean_absolute_error(y_test,predicted_test))
print("Training mean absolute error: %f" %mean_absolute_error(y_train,predicted_train))
rmse=np.sqrt(mean_squared_error(y_test,predicted_test))
print("RMSE Value for Testing")
print(rmse)
rmse=np.sqrt(mean_squared_error(y_train,predicted_train))
print("RMSE Value for Training")
print(rmse)
print("MAPE Value for Testing")
print(mean_absolute_percentage_error(y_test, predicted_test))
print("MAPE Value for Training")
print(mean_absolute_percentage_error(y_train, predicted_train))
```

```
Testing mean absolute error : 3.273344
Training mean absolute error: 2.653485
```

```
#Neural Network Case 1:
#List of Alpha Values
from sklearn.neural_network import MLPRegressor
hidden_layers=[]
alpha=[10]


for a in alpha:
    nn = MLPRegressor(hidden_layer_sizes=(100,),  activation='relu', solver='adam', alpha=a,batch_size='auto',learning_rate='cons
    learning_rate_init=0.001, max_iter=1000, shuffle=True,random_state=50, tol=0.0001, verbose=False, warm_start=False,
    early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
    n = nn.fit(x_train,y_train)
    predicted_trainnn=n.predict(x_train)
    predicted_testnn=n.predict(x_test)
    print("Testing mean absolute error : %f" % mean_absolute_error(y_test,predicted_testnn))
    print("training mean absolute error: %f"% mean_absolute_error(y_train,predicted_trainnn))
    rmse=np.sqrt(mean_squared_error(y_test,predicted_testnn))
    print("RMSE Value For Testing")
    print(rmse)
    rmse=np.sqrt(mean_squared_error(y_train,predicted_trainnn))
    print("RMSE Value For Training")
    print(rmse)
    print("MAPE Value for Testing")
    print(mean_absolute_percentage_error(y_test, predicted_testnn))
    print("MAPE Value for Training")
    print(mean_absolute_percentage_error(y_train, predicted_trainnn))
```

```
Testing mean absolute error : 26.322620
training mean absolute error: 26.355846
```

Cluster 0	Linear Regressor	KNN Regressor	MLP Regressor	Random Forest Regressor

Test rmse	**1.02**	**3.27**	**26.32**	**0.02**
Train rmse	**1.06**	**2.65**	**26.35**	**0.009**

Cluster 1	**Linear Regressor**	**KNN Regressor**	**MLP Regressor**	**Random Forest Regressor**
Test rmse	**0.93**	**3.94**	**12.29**	**0.02**
Train rmse	**0.93**	**3.17**	**12.44**	**0.009**

Cluster 2	**Linear Regressor**	**KNN Regressor**	**MLP Regressor**	**Random Forest Regressor**
Test rmse	**0.78**	**4.20**	**375**	**0.608**
Train rmse	**0.781**	**3.39**	**373**	**2.29**

Cluster 3	**Linear Regressor**	**KNN Regressor**	**MLP Regressor**	**Random Forest Regressor**
Test rmse	**1.16**	**3.58**	**20.7**	**0.029**
Train rmse	**1.16**	**2.89**	**20.86**	**0.012**

No Clustering Method

Here we apply Linear Regression, MLP Regressor, KNN Regressor and Random Forest Regressor for clustering our data. We got best results for Random Forest Regressor

```
reg=linear_model.LinearRegression()
reg.fit(x_train,y_train)
predicted_train=reg.predict(x_train)
predicted_test=reg.predict(x_test)
print("Testing mean absolute error : %f" %mean_absolute_error(y_test,predicted_test))
print("Training mean absolute error: %f" %mean_absolute_error(y_train,predicted_train))
rmse=np.sqrt(mean_squared_error(y_test,predicted_test))
print("RMSE Value for Testing")
print(rmse)
rmse=np.sqrt(mean_squared_error(y_train,predicted_train))
print("RMSE Value for Training")
print(rmse)
print("MAPE Value for Testing")
print(mean_absolute_percentage_error(y_test, predicted_test))
print("MAPE Value for Training")
print(mean_absolute_percentage_error(y_train, predicted_train))
```

```
Testing mean absolute error : 2.215362
Training mean absolute error: 2.214613
RMSE Value for Testing
2.898319029932848
```

```
#KNN Regressor


# Create the knn model.
# Look at the five closest neighbors.
knn = KNeighborsRegressor(n_neighbors=5)
# Fit the model on the training data.
knn.fit(x_train, y_train)
# Make point predictions on the test set using the fit model.
predicted_train = knn.predict(x_train)
predicted_test = knn.predict(x_test)
print("Testing mean absolute error : %f" %mean_absolute_error(y_test,predicted_test))
print("Training mean absolute error: %f" %mean_absolute_error(y_train,predicted_train))
rmse=np.sqrt(mean_squared_error(y_test,predicted_test))
print("RMSE Value for Testing")
print(rmse)
rmse=np.sqrt(mean_squared_error(y_train,predicted_train))
print("RMSE Value for Training")
print(rmse)
print("MAPE Value for Testing")
print(mean_absolute_percentage_error(y_test, predicted_test))
print("MAPE Value for Training")
print(mean_absolute_percentage_error(y_train, predicted_train))
```

```
Testing mean absolute error : 3.658064
Training mean absolute error: 2.962697
RMSE Value for Testing
```

```
#RandomForest Regressor
from sklearn.ensemble import RandomForestRegressor

z='auto'
x=100
y=1
model=RandomForestRegressor(n_estimators=x,max_features=z,oob_score=True,n_jobs=-1,random_state=50,min_samples_leaf=y)
model.fit(x_train,y_train)
predicted_train=model.predict(x_train)
predicted_test=model.predict(x_test)
print("Tuning Combination")
print(x," ",y ," ", z,"")
print("Testing mean absolute error : %f" % mean_absolute_error(y_test,predicted_test))
print("Training mean absolute error: %f" %mean_absolute_error(y_train,predicted_train))
rmse=np.sqrt(mean_squared_error(y_test,predicted_test))
print("RMSE Value for testing")
print(rmse)
rmse=np.sqrt(mean_squared_error(y_train,predicted_train))
print("RMSE Value for training")
print(rmse)
print("MAPE Value for Testing")
print(mean_absolute_percentage_error(y_test, predicted_test))
print("MAPE Value for Training")
print(mean_absolute_percentage_error(y_train, predicted_train))
```

```
Tuning Combination
100   1   auto
Testing mean absolute error : 0.635242
Training mean absolute error: 0.239902
RMSE Value for testing
```

```
#List of Alpha Values
from sklearn.neural_network import MLPRegressor
hidden_layers=[]
alpha=[10]


for a in alpha:
    nn = MLPRegressor(hidden_layer_sizes=(100,),  activation='relu', solver='adam', alpha=a,batch_size='auto',learning_rate='cons
    learning_rate_init=0.001, max_iter=1000, shuffle=True,random_state=50, tol=0.0001, verbose=False, warm_start=False,
    early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
    n = nn.fit(x_train,y_train)
    predicted_trainnn=n.predict(x_train)
    predicted_testnn=n.predict(x_test)
    print("Testing mean absolute error : %f" % mean_absolute_error(y_test,predicted_testnn))
    print("training mean absolute error: %f"% mean_absolute_error(y_train,predicted_trainnn))
    rmse=np.sqrt(mean_squared_error(y_test,predicted_testnn))
    print("RMSE Value For Testing")
    print(rmse)
    rmse=np.sqrt(mean_squared_error(y_train,predicted_trainnn))
    print("RMSE Value For Training")
    print(rmse)
    print("MAPE Value for Testing")
    print(mean_absolute_percentage_error(y_test, predicted_testnn))
    print("MAPE Value for Training")
    print(mean_absolute_percentage_error(y_train, predicted_trainnn))
```

```
Testing mean absolute error : 20.558064
training mean absolute error: 20.533011
RMSE Value For Testing
58.243702780993836
RMSE Value For Training
```

Regression (No Clustering)	Linear Regressor	KNN Regressor	MLP Regressor	Random Forest Regressor
Test rmse	2.215	3.65	20.55	0.635
Train rmse	2.21	2.96	20.53	0.239

Discussion

We have uniformly applied four classifiers Linear Regressor, KNN Regressor, MLP Regressor and Random Forest Regressor to our each case. It is observed that Random Forest is the best classifier in every case and produces the best result everytime.

Results

We observe that **Random Forest Regressor** works best suited for our data for both classification as observed in Phase 1 with accuracy of 100% and in Phase 2 for prediction of interest rates in each of our models in varied cases as below :

Best Model Name and Stage	Train MAE	Test MAE	Train MAPE	Test MAPE	Train RMSE	Test RMSE
Manual Cluster 1 (Random Forest)	0.177	0.46	1.3	3.3	0.38	0.99
Manual Cluster 2 (Random Forest)	0.25	0.67	2.3	6.15	0.51	1.3
Manual Cluster 3 (Random Forest)	0.38	1.02	4.49	11.86	0.67	1.76
Manual Cluster 4 (Random Forest)	0.4	0.92	5.0	11.7	0.7	1.5
KMeans Cluster 1 (Random Forest)	0.009	0.02	0.05	0.12	0.07	0.17
KMeans Cluster 2 (Random Forest)	0.009	0.02	0.04	0.12	0.07	0.18
KMeans Cluster 3 (Random Forest)	0.22	0.6	0.49	1.3	2.11	5.6
KMeans Cluster 4 (Random Forest)	0.01	0.029	0.05	0.13	0.09	0.23
No Clustering Regression (Random Forest)	0.23	0.63	2.24	5.9	0.49	1.31