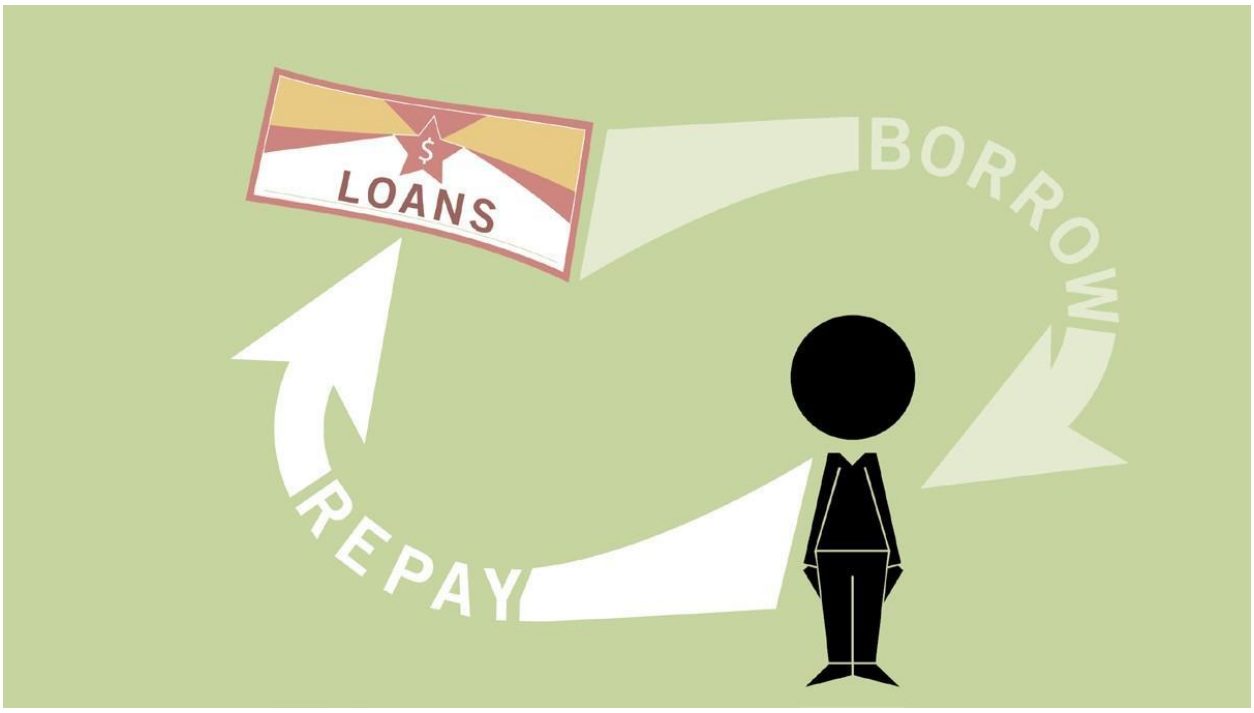
Loan Default Prediction for Dummy Irish Bank



Group 4

Dharmik Kothari

Soham Adhikari

Yash Pasar

Abstract

Problem Statement

A loan is one of the most important products of the banking. All the banks are trying to figure out effective business strategies to persuade customers to apply their loans. However, there are some customers behave negatively after their applications are approved. To prevent this situation, banks have to find some methods to predict customers' behaviors. Machine learning algorithms have a pretty good performance on this purpose, which are widely used by the banking.

The data set is based upon Lending Club Information – The Irish Dummy Banks is a peer to peer lending bank based in Ireland, in which bank provide funds for potential borrowers and bank earn a profit depending on the risk they take (the borrowers credit score). Irish Fake bank provides loan to their loyal customers.

Proposed techniques and data science methodology

We intend to perform several algorithms such as logistic regression, decision tree and random forest to classify the data into a good loan application & a bad loan application in order to predict which application to grant to. We would also like to compare the accuracy of these models to determine which would be the most suitable for this kind of prediction.

A Few Business Questions intended to analyze

1. Classifying an application as a good or a bad loan application
2. Determining the correlation between 2-year delinquent and the loan condition
3. What are the major reasons for a loan application?

Dataset

The data set is based upon Lending Club Information – The Irish Dummy Banks is a peer to peer lending bank based in Ireland, in which bank provide funds for potential borrowers and bank earn a profit depending on the risk they take (the borrowers credit score). Irish Fake bank provides loan to their loyal customers. This clean dataset has been chosen from Kaggle. This dataset has 887k rows and 30 columns.

Introduction

As a bank, you would like to give as many loan applications as possible and hope that everyone pays back the loan with a good interest amount. This is the most likely scenario in an ideal world. As we do not live in an ideal world, few customers may not pay the loan amount as per we like. So, using Machine Learning Algorithms we are trying to find if there are any factors that help us find the customers that do not repay the loan.

The data set is based upon Lending Club Information – The Irish Dummy Banks, which is a peer to peer lending bank based in Ireland, in which this bank provides funds to potential borrowers and the bank earns a profit depending on the risk it takes (the borrowers credit score). Irish Fake bank provides loan to their loyal customers. The complete data set is borrowed from Lending Club and few details might have been changed for confidential purpose. This dataset is copied and clean from Kaggle, but it has been changed by the uploader. The dataset was obtained from Kaggle.

Link

https://www.kaggle.com/mrferozi/loan-data-for-dummy-bank

Data Description

The data has over 887k observations and 30 columns. The important columns are listed below with their description:

- **LoanStatNew**: Description
- **addr_state**: The state provided by the borrower in the loan application
- **annual_inc**: The self-reported annual income provided by the borrower during registration.
- **annual_inc_joint**: The combined self-reported annual income provided by the co-borrowers during registration

- **application_type**: Indicates whether the loan is an individual application or a joint application with two co-borrowers
- **collection_recovery_fee**: post charge off collection fee
- **collections_12_mths_ex_med**: Number of collections in 12 months excluding medical collections
- **delinq_2yrs**: The number of 30+ days past-due incidences of delinquency in the borrower's credit file for the past 2 years
- **desc**: Loan description provided by the borrower
- **dti**: A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, - - - excluding mortgage and the requested LC loan, divided by the borrower's self-reported monthly income.
- **dti_joint**: A ratio calculated using the co-borrower's total monthly payments on the total debt obligations, - excluding mortgages and the requested LC loan, divided by the co-borrowers' combined self-reported monthly income
- **earliest_cr_line**: The month the borrower's earliest reported credit line was opened
- **emp_length**: Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.
- **emp_title**: The job title supplied by the Borrower when applying for the loan.*
- **fico_range_high**: The upper boundary range the borrower's FICO at loan origination belongs to.
- **fico_range_low**: The lower boundary range the borrower's FICO at loan origination belongs to.
- **funded_amnt**: The total amount committed to that loan at that point in time.
- **funded_amnt_inv**: The total amount committed by investors for that loan at that point in time.
- **grade:** LC assigned loan grade
- **home_ownership**: The home ownership status provided by the borrower during registration. Our values are: RENT, OWN, MORTGAGE, OTHER.

The datatypes of the columns

```
#List of Column names present in the dataframe and their types
loan_copy_df.dtypes
```

```
Out[7]: [('id', 'int'),
 ('year', 'int'),
 ('issue_d', 'string'),
 ('final_d', 'int'),
 ('emp_length_int', 'double'),
 ('home_ownership', 'string'),
 ('home_ownership_cat', 'int'),
 ('income_category', 'string'),
 ('annual_inc', 'int'),
 ('income_cat', 'int'),
 ('loan_amount', 'int'),
 ('term', 'string'),
 ('term_cat', 'int'),
 ('application_type', 'string'),
 ('application_type_cat', 'int'),
 ('purpose', 'string'),
 ('purpose_cat', 'int'),
 ('interest_payments', 'string'),
 ('interest_payment_cat', 'int'),
 ('loan_condition', 'string'),
 ('loan_condition_cat', 'int'),
```

```
#Descriptive Statistics applied on our Loan Dataset
loan_copy_df.toPandas().describe()
```

```
/databricks/python/lib/python3.7/site-packages/pyarrow/__init__.py:152: UserWarning: pyarrow.open_stream is deprecated, please use pyarrow.ipc.open_stream
  warnings.warn("pyarrow.open_stream is deprecated, please use "
Out[8]:
```

	id	year	final_d	emp_length_int	home_ownership_cat	annual_inc	income_cat	loan_amount	term_cat	application_type_cat	purpose_cat	interest_
count	8.873790e+05	887379.000000	8.873790e+05	887379.000000	887379.000000	8.873790e+05	887379.000000	887379.000000	887379.000000	887379.000000	887379.000000	88
mean	3.246513e+07	2014.021761	1.047089e+06	6.050564	2.099130	7.502759e+04	1.196702	14755.264605	1.300045	1.000576	4.874621	
std	2.282734e+07	1.261741	4.555149e+04	3.507405	0.944839	6.469815e+04	0.442542	8435.455601	0.458278	0.023990	2.381156	
min	5.473400e+04	2007.000000	1.012008e+06	0.500000	1.000000	0.000000e+00	1.000000	500.000000	1.000000	1.000000	1.000000	
25%	9.206643e+06	2013.000000	1.012016e+06	3.000000	1.000000	4.500000e+04	1.000000	8000.000000	1.000000	1.000000	3.000000	
50%	3.443327e+07	2014.000000	1.012016e+06	6.050000	3.000000	6.500000e+04	1.000000	13000.000000	1.000000	1.000000	6.000000	
75%	5.490814e+07	2015.000000	1.092015e+06	10.000000	3.000000	9.000000e+04	1.000000	20000.000000	2.000000	1.000000	6.000000	
max	6.861706e+07	2015.000000	1.122015e+06	10.000000	6.000000	9.500000e+06	3.000000	35000.000000	2.000000	2.000000	14.000000	

Understanding the Data

The initial data set contained data which was imbalanced and values that proved a hindrance when performing analysis. The dataset contained various data types, including integer, string, and categorical values. In order to generate a dataset that is most usable, we generated the following set of assumptions.

(1) Null values were few in number compared to the overall data, therefore the team extracted all survey data observations that contained null values. In addition, most variables that were used while analyzing are with no null values.

(2) Where values fell within the metadata, team members used the integer values within their analysis, this provides a more complete dataset.

(3) To effectively differentiate between a good and a bad loan application, few important columns were identified and used accordingly.

Scope

(1) Improve the loan application accuracy as a bad loan

(2) Generate actionable insight for Dummy Irish Bank.

Problem Statement

Given a dataset of over 887,000 observations, what actionable insights can be understood to help Dummy Irish Bank to improve their good loan accuracy.

Purpose

We will analyze a dataset consisting of 887,000 observations within the course of 8 weeks in order to provide actionable insight for Irish Dummy Bank to identify good loan and bad loan applications.

Data Munging

Basic Spark features were used to transform our data. We had no NA values and dummy variables for few columns were created to carry out model creation.

Dummy variables were created for the following columns

- Loan condition
- Grade
- Home Ownership
- Income_Category
- Purpose
- Term_Cat

The dropped columns are:

- G
- Low
- Good Loan
- Bad Loan
- ANY
- other

Analysis

Ethics Statement: Our team shall not disclose to any third party any details regarding the Client's business, including, without limitation any information regarding any of the Client's customer information, business plans, or price points (the Confidential Information), (ii) make copies of any Confidential Information or any content based on the concepts contained within the Confidential Information for personal use or for distribution unless requested to do so by the Client, or (iii) use Confidential Information other than solely for the benefit of the Client.

To best answer and demonstrate the value of our analysis, understanding the types of customers, their previous loan history and if they have a mortgage, their grade (credit score) is of vital importance.

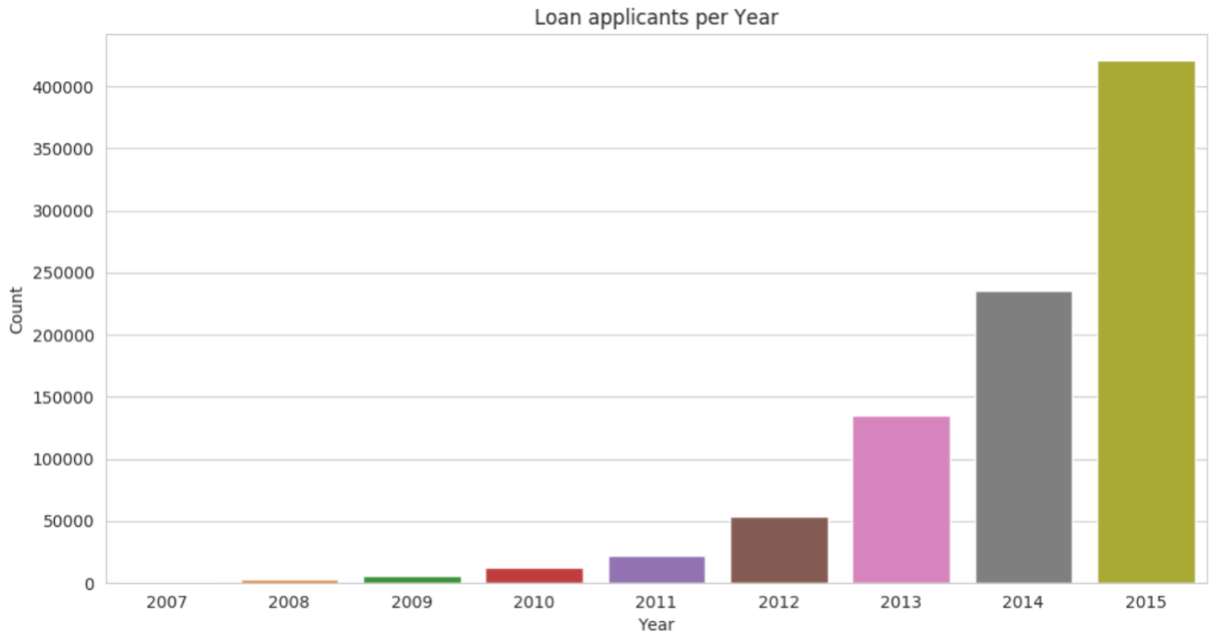
Levels of Understanding

The levels of understanding will walk through each team members processes as well as provide insightful analysis on each of the variables and models used. The end result of the levels is to assist the team in answering the business questions and providing predictive analysis to answer the problem statement listed at the beginning of this report.

Level 1: Descriptive Statistics

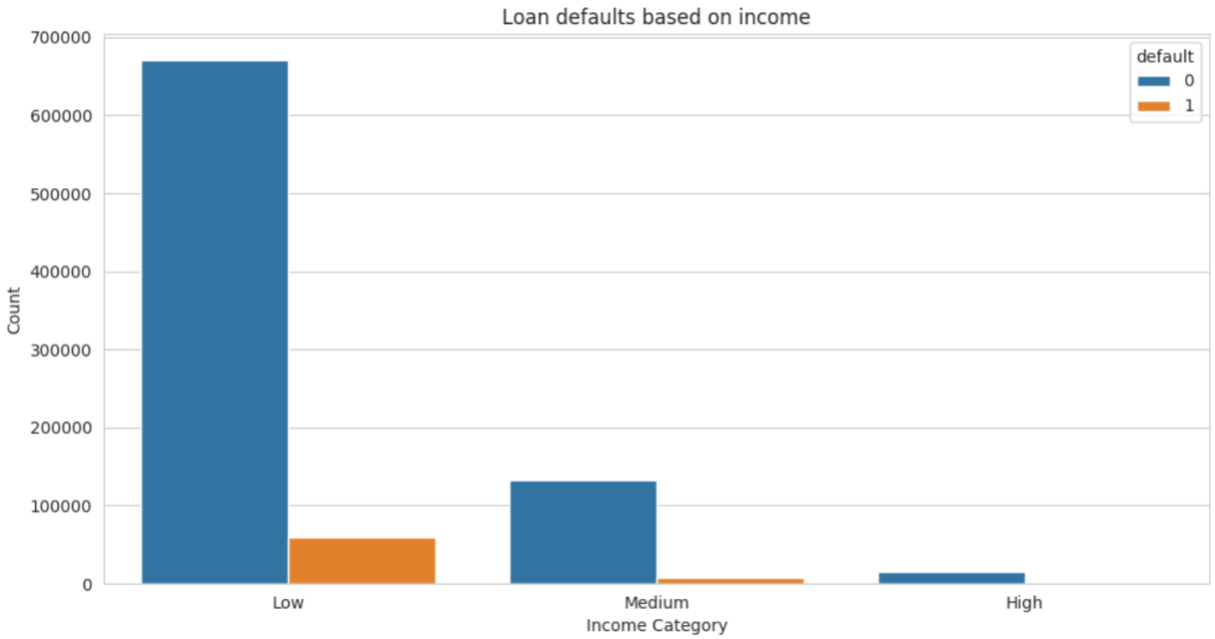
Using Visualization to find the insights

- Finding number of loan applications per year.



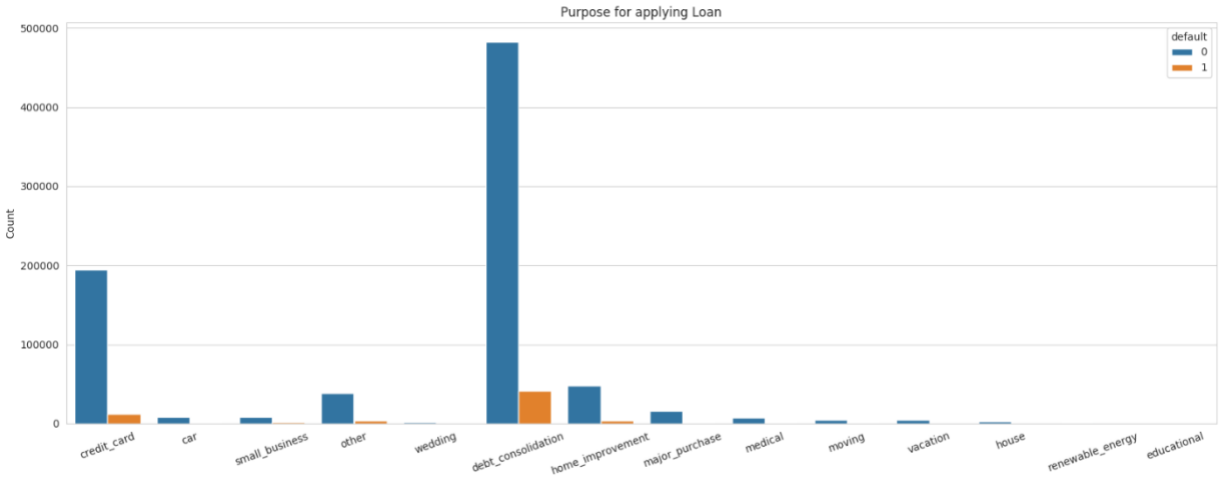
This shows that the bank has a good growth rate per year and the number of loan applications are gradually increasing.

- Finding the defaults based on the income of the people



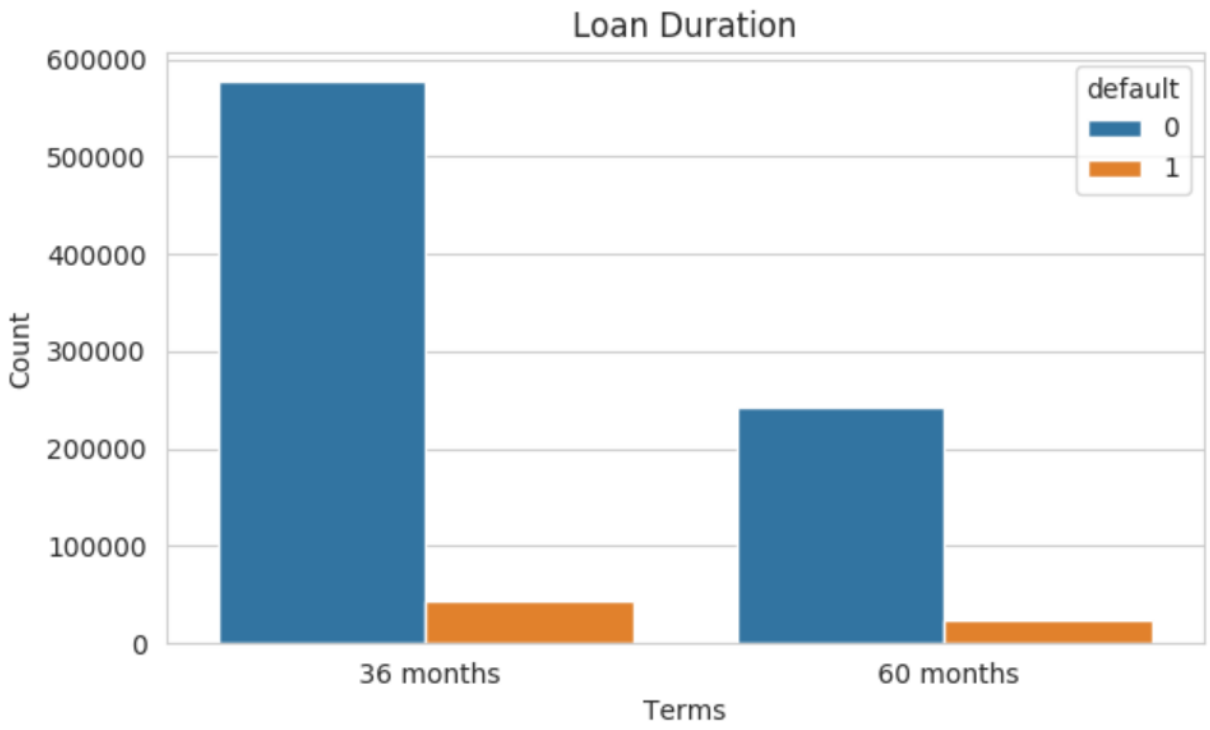
This graph shows that the number of loan applications and the defaulters were high on the low-income category. This shows that the income is an important part of the analysis.

- Finding the purpose of loan applications



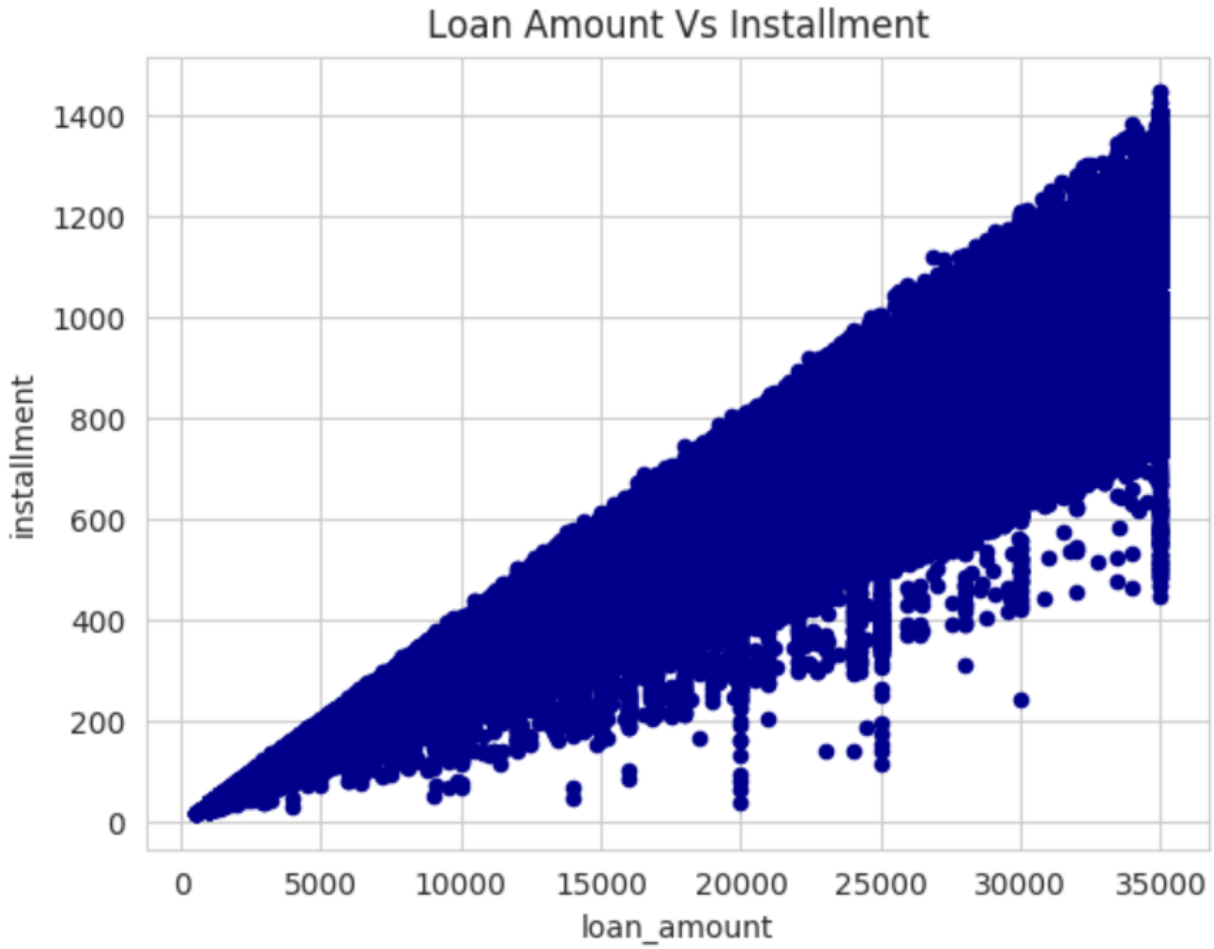
There are 13 categories for loan applications, some categories have no defaulters and some categories have considerable number of defaulters. So, knowing why people apply for loan could be helpful in identifying good loans and bad loans.

- Finding the loan term duration



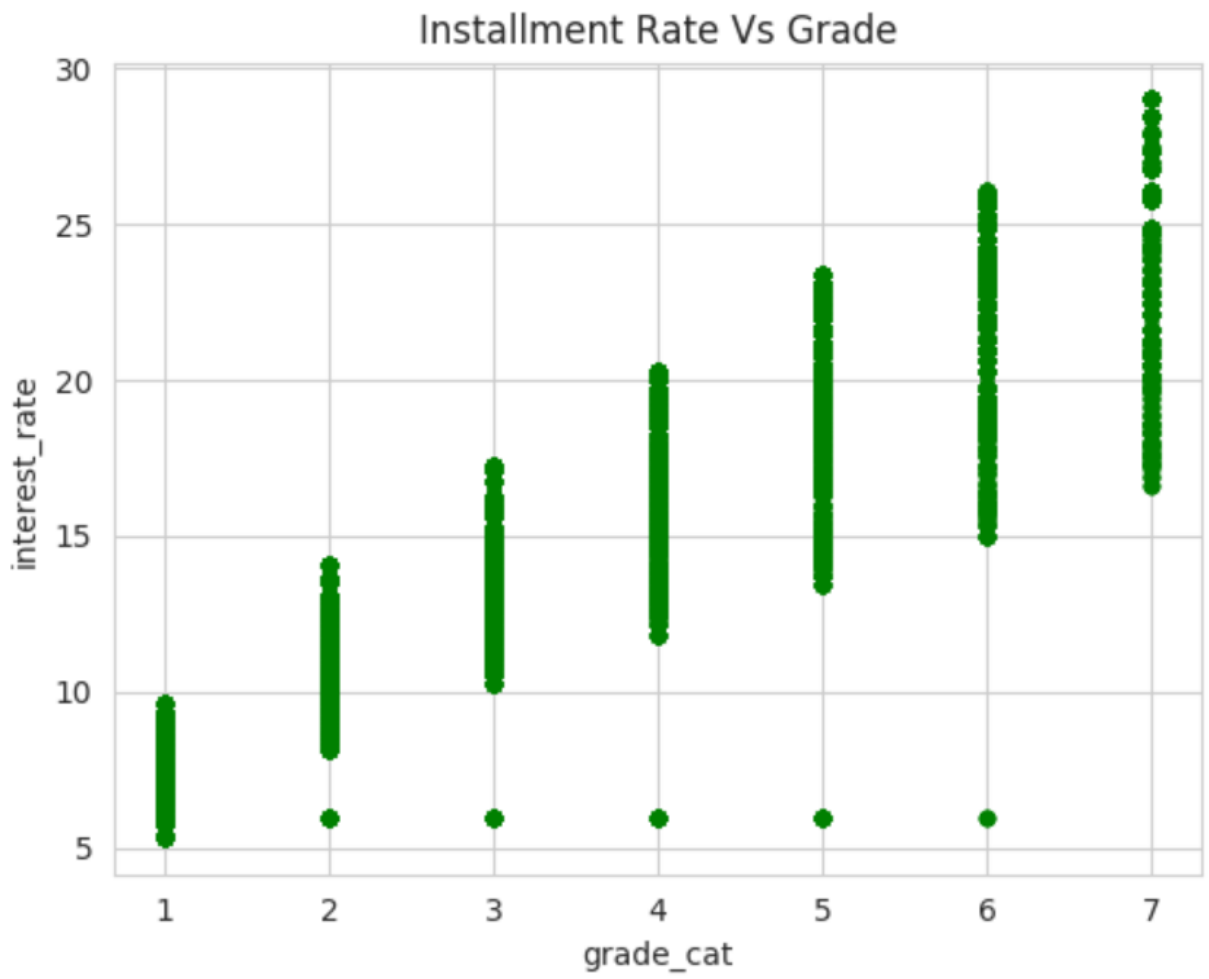
The percentage of defaulters in long term loan duration are higher than the short term. There are only terms in which the loan is being dispersed.

- Finding out the correlation between the loan amount and the installment amount



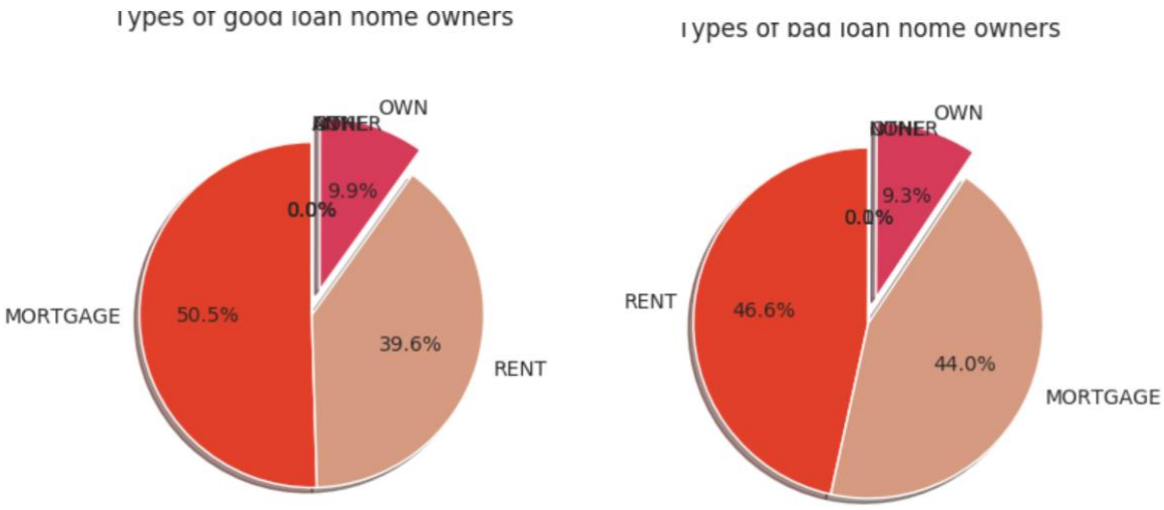
As one would think, they have a strong correlation and we can see that the variety of installment rates offered per loan amount.

- Finding out what kind of interest rates are given based on the grades.



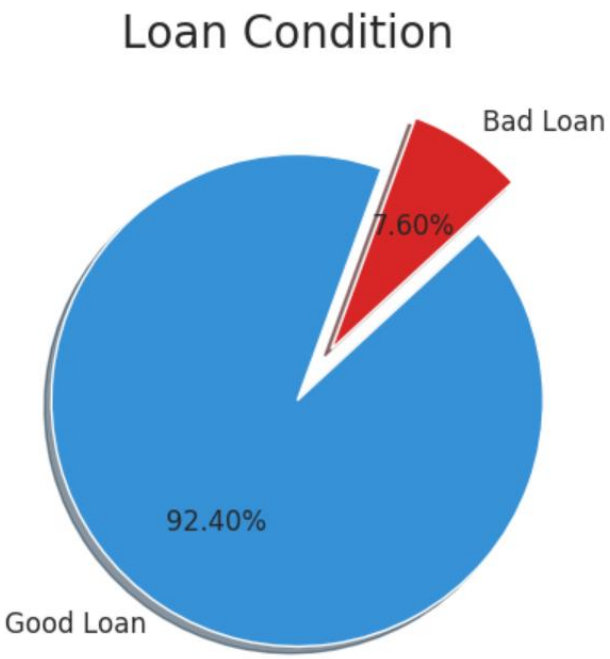
People with better grades are guaranteed lower interest rates and there are some exceptions with worse grades, we can see few outliers where they have been given lower grades albeit a bad grade.

- Finding Homeowner type based on a good loan and a bad loan



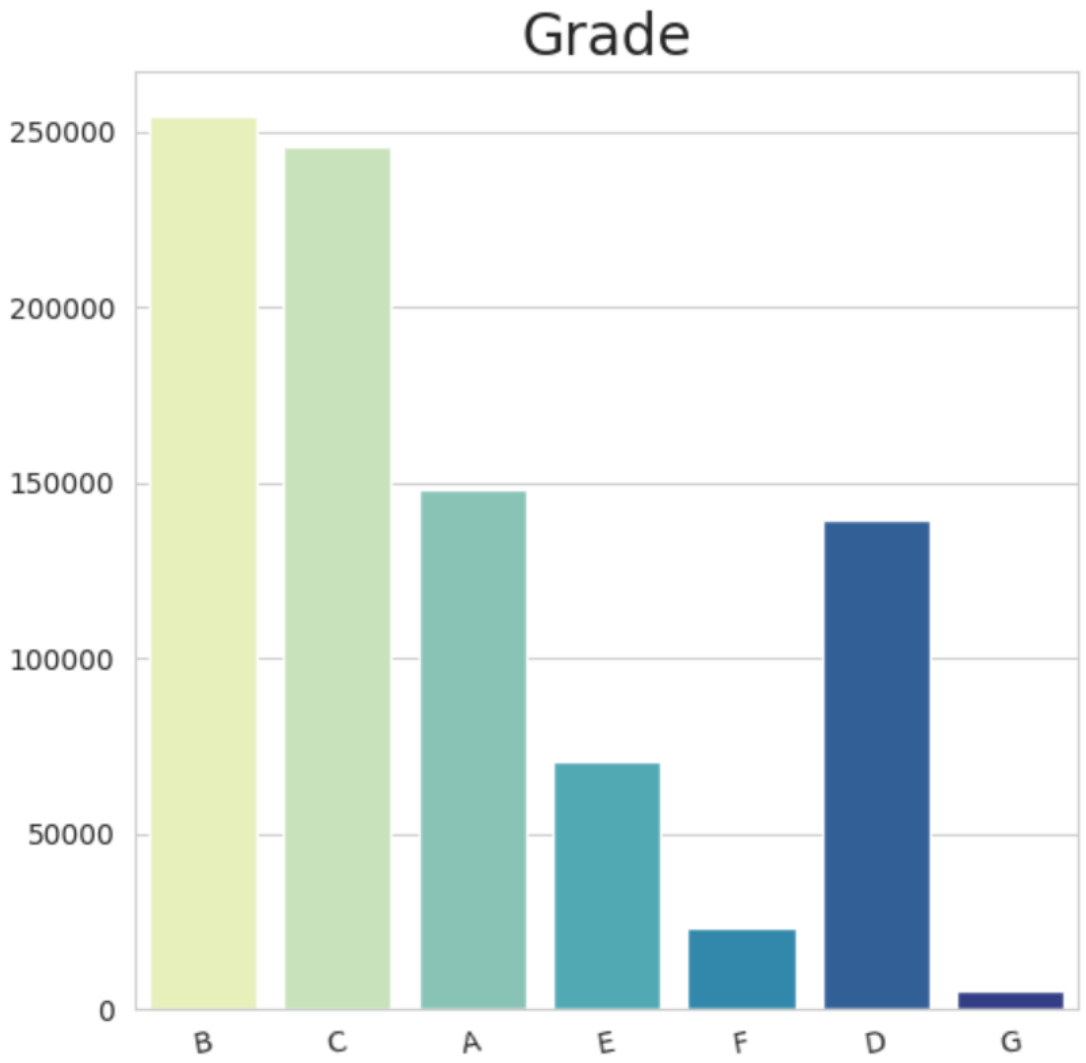
We can see that most people with a bad loan have a rented house and Most people with a good loan have a mortgage. This could indicate people with collateral have better chance of repaying the loan.

- Finding the percentage of bad loan



We can see there is an imbalance in the dataset, only 7.6% of the total loan is bad loan, which is good for the bank. We are trying to bring it down as much as possible based on the available information.

- Finding the number of loan applications based on the grade



Level 2: Machine Learning Algorithms

Handling Class Imbalance

Since the percentage of ones in the dataset is just 7.6 % surely there is an imbalance in the dataset. Thankfully, in the case of logistic regression we have a technique called “Class Weighing”. Since negatives are in a majority. Therefore, **logistic loss** objective function should treat the positive class (Outcome == 0) with higher weight. For this purpose, we calculate the BalancingRatio as follows:

BalancingRatio= numNegatives/dataset_size

Then against every Outcome == 1, we put BalancingRatio in column “classWeights”, and against every Outcome == 0, we put 1-BalancingRatio in column “classWeights”

In this way, we assign higher weightage to the minority class (i.e. positive class)

```
dataset_size=float(training_df.select("default").count())
numPositives=training_df.select("default").where('default == 0').count()
per_ones=(float(numPositives)/float(dataset_size))*100
numNegatives=float(dataset_size-numPositives)
print('The number of ones are {}'.format(numPositives))
print('Percentage of ones are {}'.format(per_ones))
```

```
The number of ones are 491972
Percentage of ones are 92.3981308972895
```

```
training_df=training_df.withColumn("classWeights", fn.when(training_df.default == 0,BalancingRatio).otherwise(1-BalancingRatio))
training_df.select("classWeights").show(5)
```

```
+------------------+
|     classWeights|
+------------------+
|0.076018691027105|
|0.076018691027105|
|0.923981308972895|
|0.076018691027105|
|0.076018691027105|
+------------------+
only showing top 5 rows
```

For **Decision Trees** and **Random Forest,** we don’t have the ease of setting the weight column at the moment, so we try different sampling techniques in order to deal with the underlying problem. The techniques have been shown in more detail in their respective sections.

Logistic Regression

In our problem, we are trying to solve a classification problem, i.e. whether a loan is good or bad. Logistic regression is basically a supervised classification algorithm. In a classification problem, the target variable (or output), y, can take only discrete values for given set of features (or inputs), X.

Contrary to popular belief, logistic regression is a regression model. The model builds a regression model to predict the probability that a given data entry belongs to the category

numbered as “1”. Just like Linear regression assumes that the data follows a linear function, Logistic regression models the data using the sigmoid function.

Logistic regression becomes a classification technique only when a decision threshold is brought into the picture. The decision for the value of the threshold value is majorly affected by the values of precision and recall.

The model was built as such:

```
cols = ['total_pymnt','loan_amount', 'dti', 'installment', 'total_rec_prncp', 'interest_rate', 'A', 'B', 'C', 'D', 'E', 'F',
'MORTGAGE','NONE','OTHER','OWN','RENT',
'car','credit_card','debt_consolidation','educational','home_improvement','house','major_purchase','medical','moving','renewable_energy','small_business','va
cation','wedding','1']

va = VectorAssembler(inputCols=cols, outputCol='features')
sc = StandardScaler(withMean=True, withStd=False, inputCol='features', outputCol='std_features')
lr = LogisticRegression().setLabelCol('default').setFeaturesCol('std_features').setWeightCol('classWeights')

lr_Model = Pipeline(stages=[va,sc, lr]).fit(training_df)
lr_prediction = lr_Model.transform(testing_df)
```

Hyper Parameter Tuning

“hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm”

In our problem, we are using ParamGridBuilder to tune the parameters using a three-fold cross validation, from there we choose the best model. Using the best model, we train our model and got an accuracy of 79%

```
# Create ParamGrid for Cross Validation
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
paramGrid_lr = (ParamGridBuilder()
                    .addGrid(lr.regParam, [0.05, 0.01, 0.2])
                .addGrid(lr.elasticNetParam, [0.3, 0.1])
                .build())
```

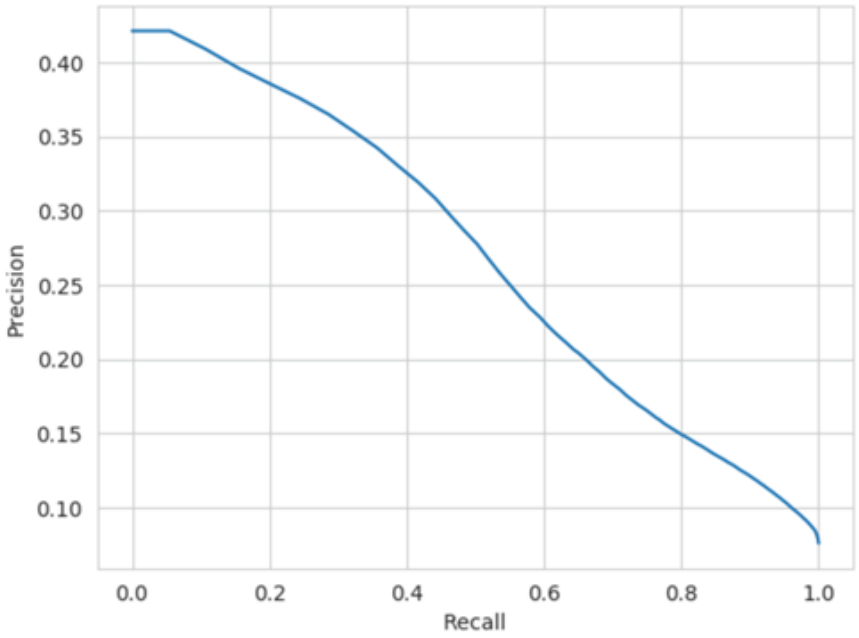
Results

Precision is the ratio of the number of true positives divided by the sum of the true positives and false positives. It describes how good a model is at predicting the positive class. Precision is referred to as the positive predictive value.

Recall is calculated as the ratio of the number of true positives divided by the sum of the true positives and the false negatives. Recall is the same as sensitivity.

Reviewing both precision and recall is useful in cases where there is an imbalance in the observations between the two classes. The reason for this is that typically the large number of class 0 examples means we are less interested in the skill of the model at predicting class 0 correctly, e.g. high true negatives.

The precision-recall curve plot is then created showing the precision/recall for each threshold for below logistic regression model:



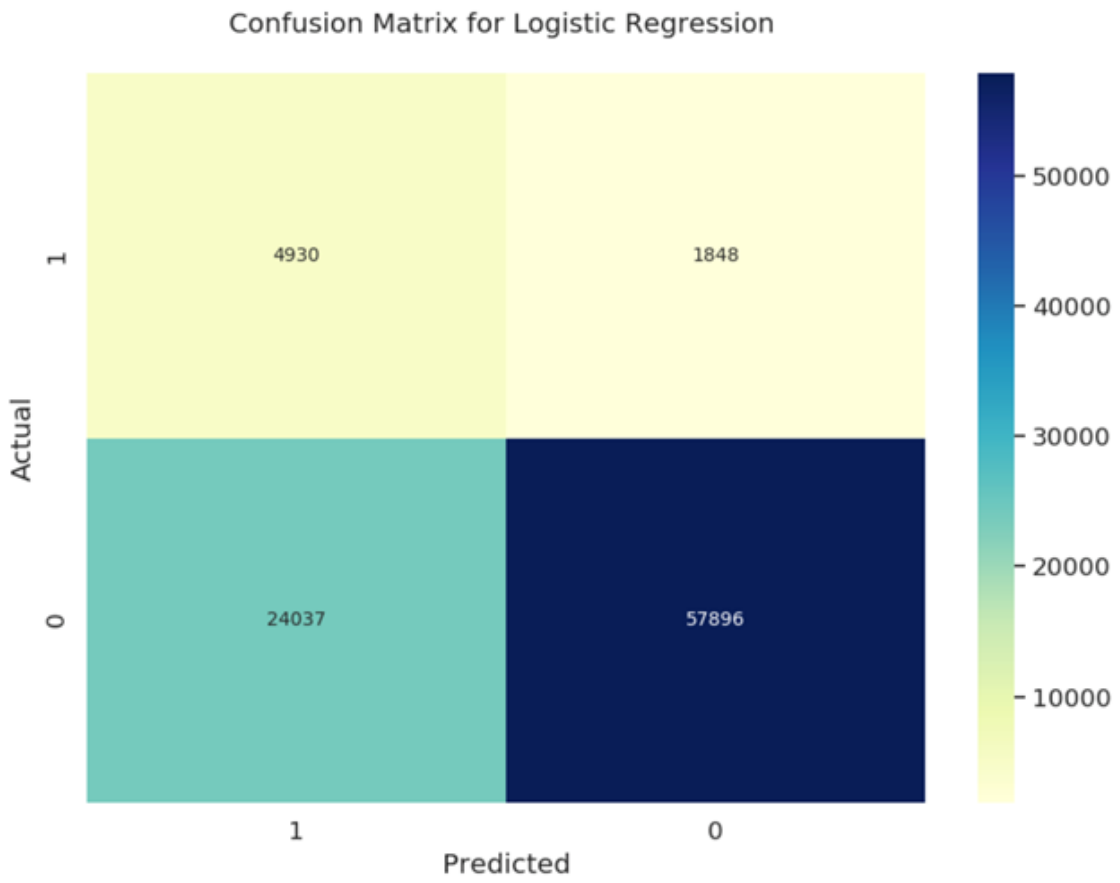
Confusion Matrix:

1. **True Positives:** 57896
2. **True Negatives:** 4930
3. **False Positives:** 1848
4. **False Negatives:** 24037

The classifier made a total of 88711 predictions.

Out of those cases, the classifier predicted a good loan 59744 times, and bad loan, 28967 times.

In reality, 81933 loans in the sample were good loans, and 6778 were not.



Decision Tree

As the same in logistic regression, we did the same under sampling in decision tree

```
#Undersampling the dataset with good loans to take care of the Imbalanced dataset
good_loans = loan_linear_reg.filter(loan_linear_reg.default == 0)
bad_loans = loan_linear_reg.filter(loan_linear_reg.default == 1)
sampleRatio = bad_loans.count() / loan_linear_reg.count()
good_loansSampleDf = good_loans.sample(False, sampleRatio)
loan_linear_reg1 = bad_loans.unionAll(good_loansSampleDf)
```

```
loan_linear_reg1.groupby('default').count().show()
```

```
+-------+-----+
|default|count|
+-------+-----+
|      1|67429|
|      0|62572|
+-------+-----+
```

So we got an even count of observations in both good and bad loans.

The data was split into three categories:

- training - 60%
- validation - 30%
- testing - 10%

In decision tree, the root and the attribute nodes contain the test conditions to separate the records that contain different characteristics. All the terminal node is assigned a class label Yes or No.

The model was built the following way:

```
vaD = VectorAssembler(inputCols=['total_pymnt','loan_amount','installment','total_rec_prncp','interest_rate','A', 'B', 'C','D','E','F',
'MORTGAGE','NONE','OTHER','OWN','RENT','car','dti','credit_card','debt_consolidation','educational','home_improvement','house','major_purchase','medical','mo
ving','renewable_energy','small_business','vacation','wedding','1'], outputCol='features')
sc = StandardScaler(withMean=True, withStd=False, inputCol='features', outputCol='std_features')
dt = DecisionTreeClassifier(featuresCol='std_features', labelCol='default')

dt_model = Pipeline(stages=[vaD, sc, dt]).fit(training_df1)
```

```
evaluator = BinaryClassificationEvaluator(labelCol="default", rawPredictionCol="rawPrediction", metricName="areaUnderROC")
validation_accuracy = evaluator.evaluate(dt_model.transform(validation_df1))
print("Validation Accuracy = %g " % (validation_accuracy))
print("Validation Error = %g " % (1.0 - validation_accuracy))

Validation Accuracy = 0.694245
Validation Error = 0.305755
```

We got an accuracy of about 69% which was less than the logistic regression.

The tree had depth of 5 and 35 nodes.

Hyper Parameter Tuning

Using Parameter Grid and cross validation, we tuned the parameters.

We did three-fold cross validation and a depth of 20, we selected the best model of it.


```
evaluator = BinaryClassificationEvaluator(labelCol="default", metricName="areaUnderPR")
validation_accuracy = evaluator.evaluate(tuned_dt_Model.transform(validation_df1))
print("Validation Accuracy = %g " % (validation_accuracy))
print("Validation Error = %g " % (1.0 - validation_accuracy))

Validation Accuracy = 0.711733
Validation Error = 0.288267
```

We improved the accuracy to 71%

Stratified Sampling

Trying Startified Sampling

```
stratified_data = loan_linear_reg.sampleBy('default', fractions={1: 0.9, 0: 0.40}).cache()

stratified_data.groupby('default').count().show()
```

```
+-------+-----+
|default|count|
+-------+-----+
|      1|60800|
|      0|81849|
+-------+-----+
```

```
vaD = VectorAssembler(inputCols=['total_pymnt','loan_amount','installment','total_rec_prncp','interest_rate','A', 'B', 'C','D','E','F',
'MORTGAGE','NONE','OTHER','OWN','RENT','car','dti','credit_card','debt_consolidation','educational','home_improvement','house','major_purchase','medical','mo
ving','renewable_energy','small_business','vacation','wedding','1'], outputCol='features')
sc = StandardScaler(withMean=True, withStd=False, inputCol='features', outputCol='std_features')
dt = DecisionTreeClassifier(featuresCol='std_features', labelCol='default')

dt_model = Pipeline(stages=[vaD, sc, dt]).fit(training_df1)
```

```
evaluator = BinaryClassificationEvaluator(labelCol="default", rawPredictionCol="rawPrediction", metricName="areaUnderROC")
validation_accuracy = evaluator.evaluate(dt_model.transform(validation_df1))
print("Validation Accuracy = %g " % (validation_accuracy))
print("Validation Error = %g " % (1.0 - validation_accuracy))

Validation Accuracy = 0.609479
Validation Error = 0.390521
```

The same columns were used and the accuracy dropped to 61 %

Hyper Parameter Tuning

The same technique is used to tune the parameters.

The best model gave out of about 72% accuracy.

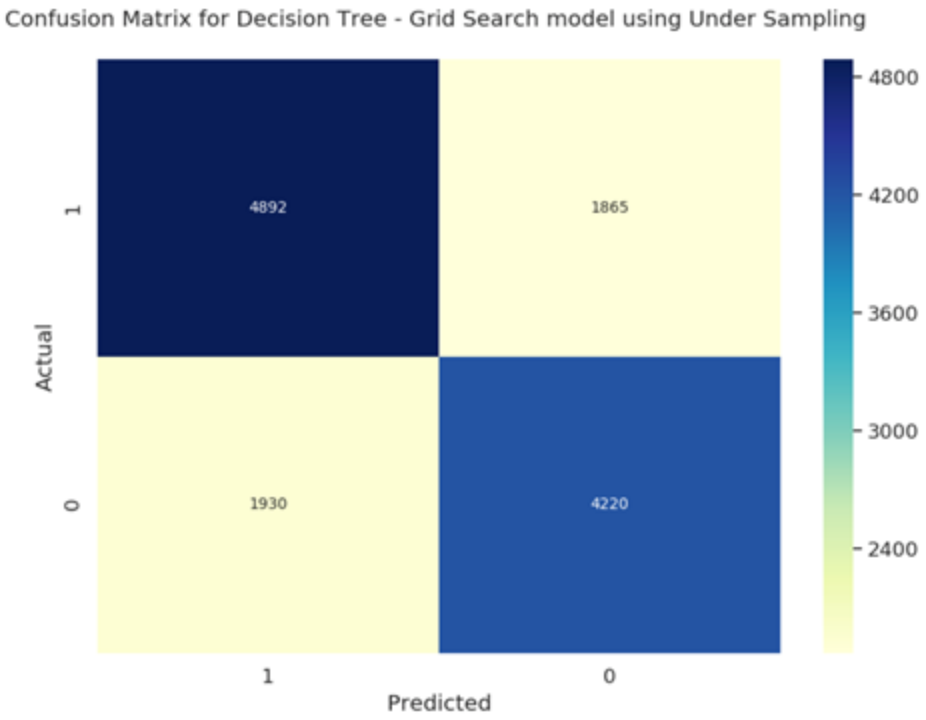
Confusion Matrix for grid search with under-sampling:

1. **True Positives:** 4220
2. **True Negatives:** 4892
3. **False Positives:** 1865
4. **False Negatives:** 1930

The classifier made a total of 12907 predictions.

Out of those cases, the classifier predicted a good loan 6085 times, and bad loan, 6822 times.

In reality, 6150 loans in the sample were good loans, and 6757 were not.



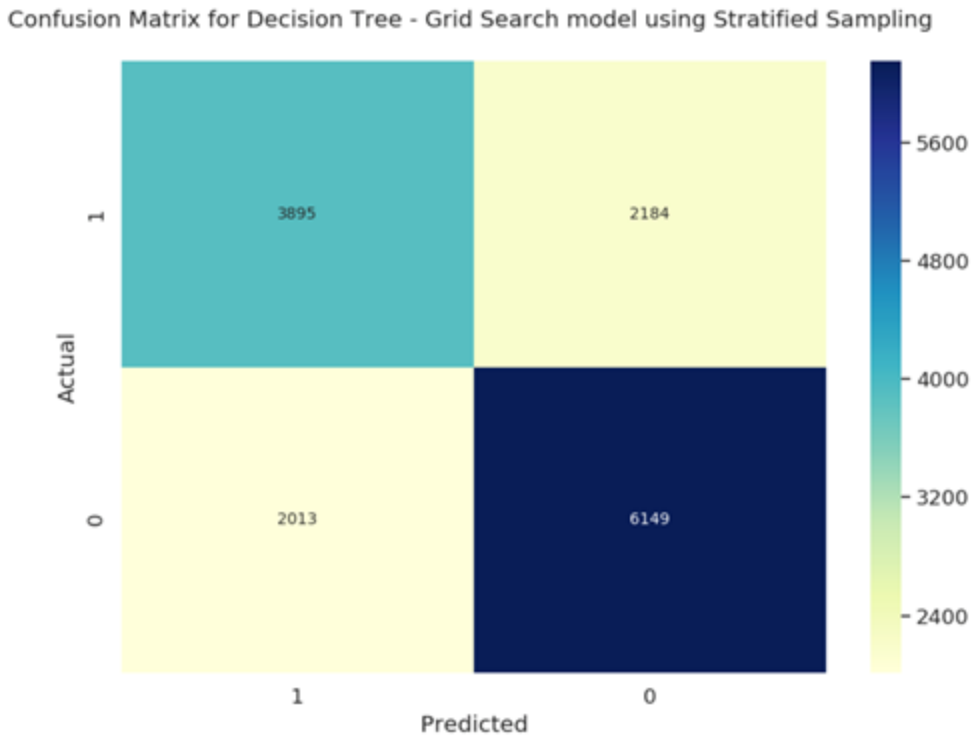
Confusion Matrix for grid search with stratified sampling:

1. **True Positives:** 6149
2. **True Negatives:** 3895
3. **False Positives:** 2184
4. **False Negatives:** 2013

The classifier made a total of 14241 predictions.

Out of those cases, the classifier predicted a good loan 8333 times, and bad loan, 5908 times.

In reality, 8162 loans in the sample were good loans, and 6079 were not.



Random Forest Classifier

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.

We used stratified sampling technique in Random Forest Classifier.

The random forest classifier was built as follows:

```
rf_assembler = VectorAssembler(inputCols=
['loan_amount','dti','total_pymnt','total_rec_prncp','recoveries','installment','interest_rate','A','B','C','D','E','F','MORTGAGE','NONE','OTHER','OWN','RENT
','High','Medium','car','credit_card','debt_consolidation','educational','home_improvement','house','major_purchase','medical','moving','renewable_energy','s
mall_business','vacation','wedding','1'], outputCol="features")
rf = classification.RandomForestClassifier(featuresCol = "features", labelCol = "default")
```

```
pipe_rf = Pipeline(stages = [rf_assembler,rf]).fit(training_df1)
```

```
evaluator = evaluation.BinaryClassificationEvaluator(labelCol='default', metricName='areaUnderROC')
```

The default model gave us accuracy of about 81 %

Hyper Parameter Tuning

The same technique was used as previous models.

The best model gave an accuracy of 85 %

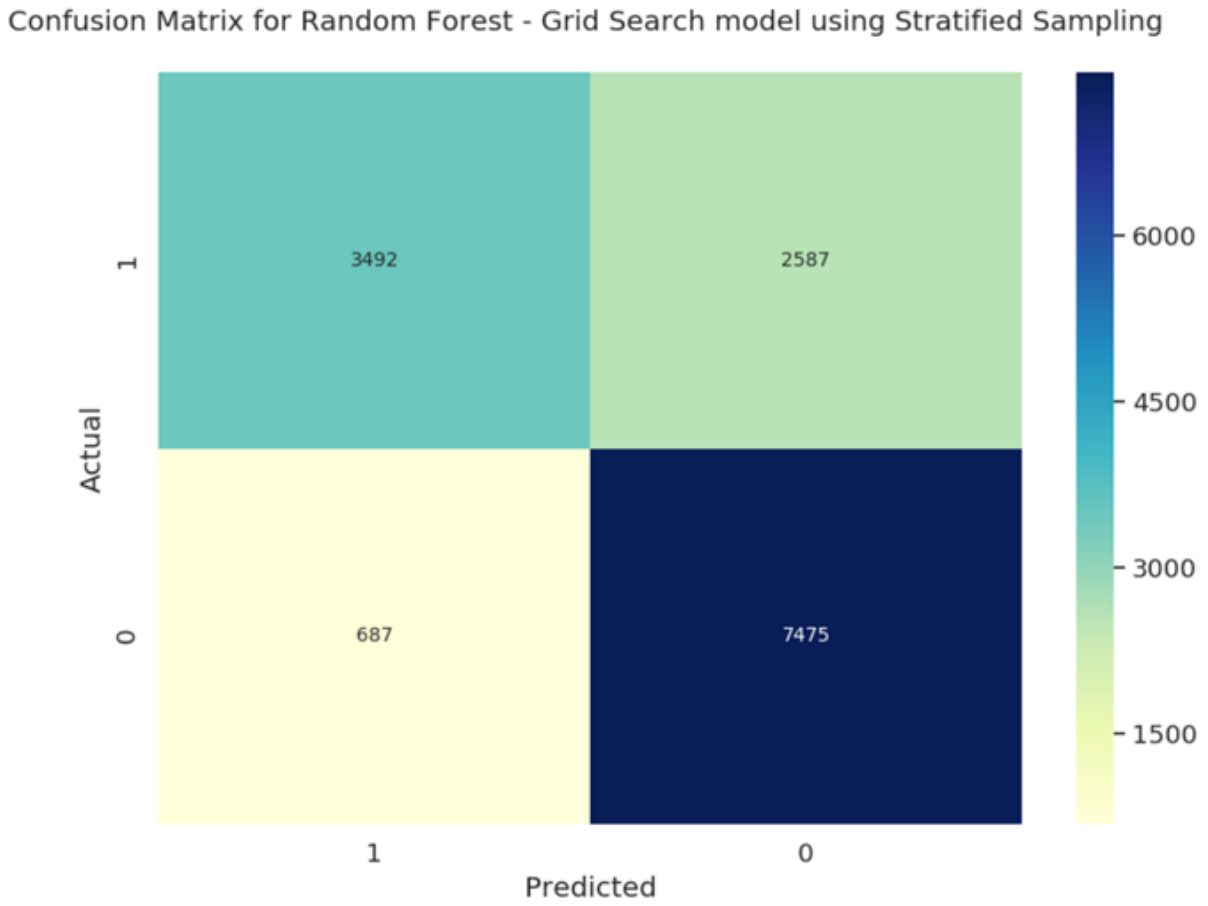
Confusion Matrix for grid search with stratified sampling:

1. **True Positives:** 7475
2. **True Negatives:** 3492
3. **False Positives:** 2587
4. **False Negatives:** 687

The classifier made a total of 14241 predictions.

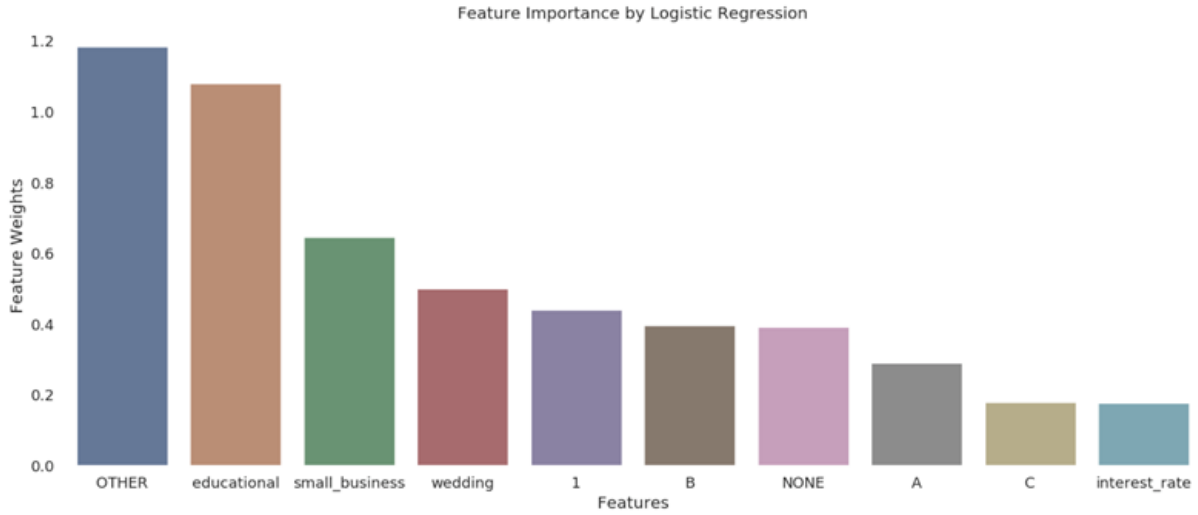
Out of those cases, the classifier predicted a good loan 10062 times, and bad loan, 4179 times.

In reality, 8162 loans in the sample were good loans, and 6079 were not.



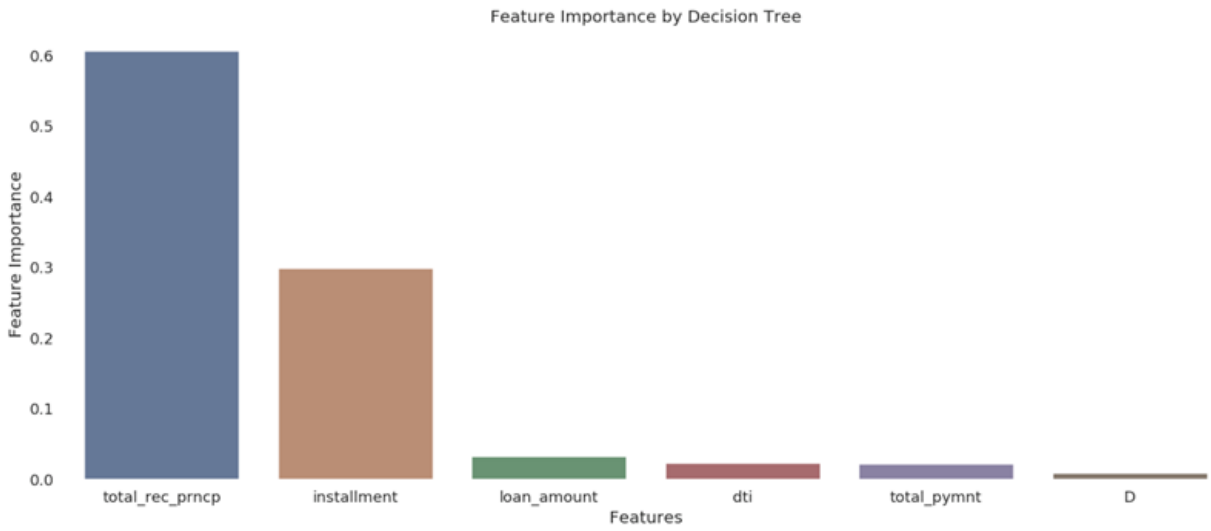
Conclusion

Important features by Logistic Regression



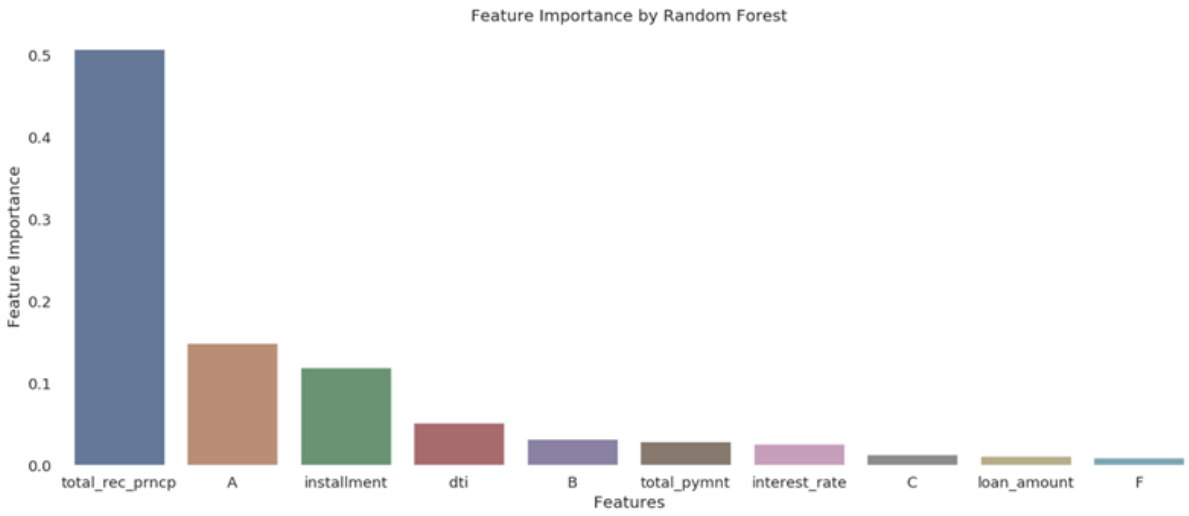
From the above graph, we see that the other category of feature is rated the most important and educational purpose of getting loan is close behind. This may not be as meaningful as we would expect.

Important Features by Decision Tree



In this we see that total recovery principal plays an important feature and the next important feature is the installment amount which could say that based on the installment rate someone may not be able to repay the loan. Then a few other key features are loan amount, dti, total payment and D (grade).

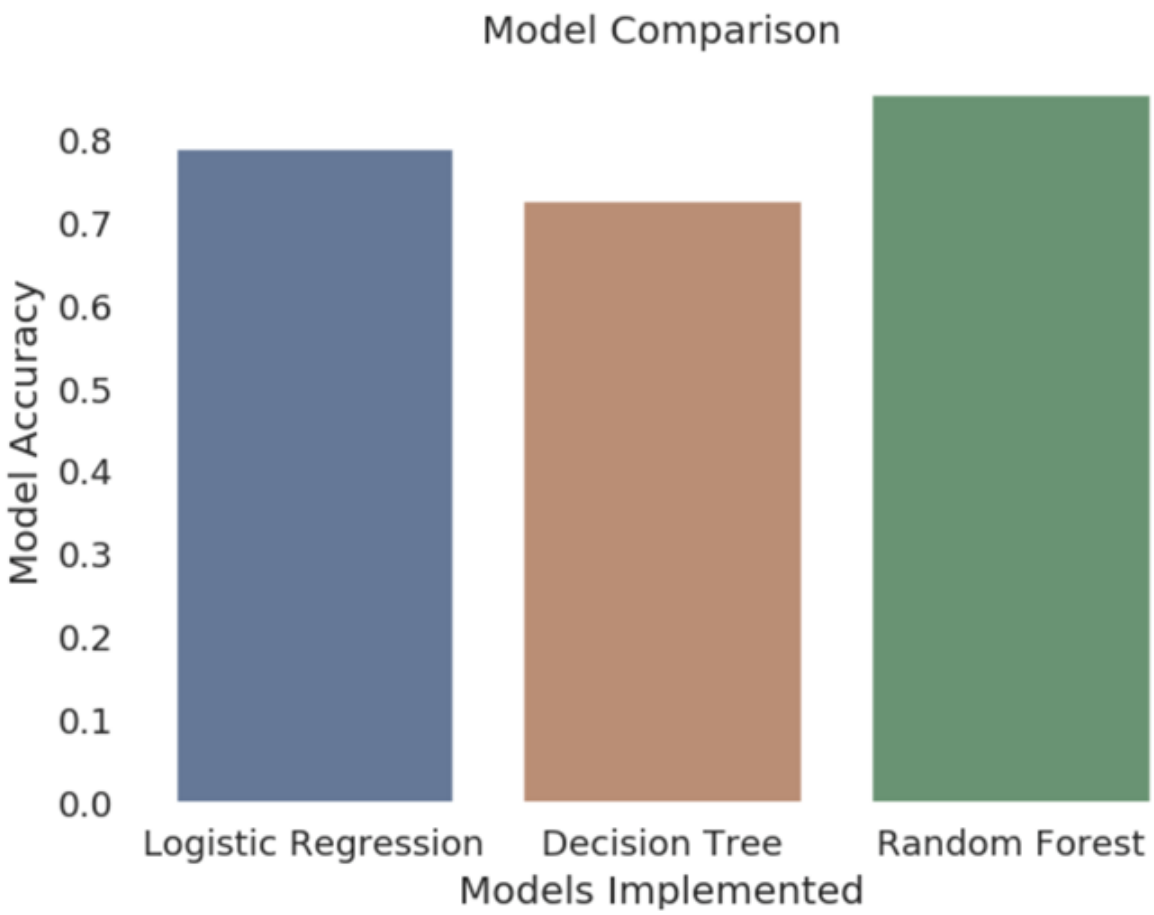
Important Features by Random Forest Classifier



By Random Forest Classifier, we get the most important key feature as that total recovery principal which resonate with the features determined by the decision tree model. The other key features are grade type A, installment amount, dti, grade type B, total payment, interest rate, grade type C, loan amount and grade type F.

Model Comparison

By comparing the accuracy of various models, we infer that the Random Forest Classifier gave the best accuracy as well as took the shortest time to run. The second best was Logistic regression, followed by Decision Trees with the least accuracy.



Appendix

Section1: References

1. Official seaborn tutorial¶. (n.d.). Retrieved from https://seaborn.pydata.org/tutorial.html.
2. Classification and regression. (n.d.). Retrieved from https://spark.apache.org/docs/latest/ml-classification-regression.html.
3. Random Forests Classifiers in Python. (n.d.). Retrieved from https://www.datacamp.com/community/tutorials/random-forests-classifier-python.
4. ML - Decision Trees. (n.d.). Retrieved from https://spark.apache.org/docs/1.5.2/ml-decision-tree.html.

5. dbakrdbakr 28311 gold badge33 silver badges44 bronze badges, SerendipitySerendipity 1, kanielckanielc 59577 silver badges99 bronze badges, & PSAfrancePSAfrance 1122 bronze badges. (1965, December 1). Dealing with unbalanced datasets in Spark MLlib. Retrieved from https://stackoverflow.com/questions/33372838/dealing-with-unbalanced-datasets-in-spark-mllib.

Section2: Runtime Environment

- Databricks

Section3: Code Description

Packages used

```
# Load the packages needed for this part
# create spark and sparkcontext objects
from pyspark.sql import SparkSession
import numpy as np

spark = SparkSession.builder.getOrCreate()
sc = spark.sparkContext

import pyspark
from pyspark.ml import feature, regression, Pipeline, classification, pipeline, evaluation
from pyspark.ml.classification import LogisticRegression
from pyspark.sql import functions as fn, Row
from pyspark.sql.functions import when, regexp_extract, col
from pyspark import sql
from pyspark.sql.functions import *

from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.feature import VectorIndexer

import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import sys
```

The code is as follows:

- Load the packages
- Load the data
- Exploratory data Analysis (Visualization and tabular)

  - Data description
  - Data grouping
  - Correlation Matrix
  - Bar plot of Loan applications
  - Bar plot on Loan Defaults
  - Bar plot on Loan Purpose
  - Bar plot of Loan Duration
  - Scatter plot of Loan Amount Vs Installment
  - Scatter plot of Interest rate Vs Grade
  - Scatter plot of Interest rate Vs Grade with Linear Regression
  - Box plot on Default vs Annual Income
  - Box plot on Default vs Installment
  - Box plot on Purpose of applying loan
  - Box plot on Purpose Vs Loan Amount
  - Box plot on Purpose Vs Annual Income
  - Pie chart for type of house owners for good loan
  - Pie chart for type of house owners for bad loan
  - Barplot on Grade
  - Plot on issue date
  - Pie chart on Loan condition
  - Violin plot on Loan amount vs Grade
  - Box plot on Grade and total Payment
- Feature Engineering
- Model Creation
  - Logistic Regression
  - Decision Tree
  - Random Forest Classifier
- Inference
- Model Comparison