Lending Club Loan Analysis and Default Loan/Rating Prediction

CS570 Big Data Processing & Analytics
Guided by: Prof. Nooshin Nabizadeh
13th November, 2021

Presented by
Sai Harshinee Roopakula
19577

Table of Contents

Introduction - Lending Club & Business Problem

Lending Club is a peer-to-peer lending company that utilizes group of private investors to fund loan requests. Lending Club assigns each borrower a grade and subgrades based on their credit history.

Investors are presented with a list of loan requests along with their grades and borrower details. Then they select loan request they will fund/partially fund.

Lending Club makes money by charging borrowers an origination fee and a service fee to investors.

The business problem is to build a model which will give a more comprehensive assessment of borrowers than what is presented by Lending Club in order to reduce investment risk.

Data Description

- Lending Club has provided historical data since its origination (2007-2015) under open source license.
- This dataset contained information pertaining to the borrower's past credit history, employment, income details and Lending Club loan information. The total dataset consisted of 80+ features and over 850,000 records.
- The variables which are used in this data provide ample amount of information which we could use to predict loan default likelihood.
- We only required variables which have direct response to borrower's potential to default. We used feature selection techniques and business knowledge for choosing relevant variables.
- This data is structured data with lot of missing/null values. Includes continuous, ordinal and nominal feature types.

Design - Importing necessary libraries

```python
import findspark
findspark.init("/Users/hemanthharshinee/Downloads/spark-3.1.2-bin-hadoop3.2")

from pyspark import SparkConf
from pyspark.sql import SparkSession
from pyspark.sql import SQLContext
from pyspark.sql import functions as F
from pyspark.sql.functions import isnan, when, count, col, year, quarter, lit, to_date, to_timestamp, concat, avg
from pyspark.sql.types import DateType, TimestampType
from pyspark import SparkContext
from pyspark import SparkConf
from pyspark.ml.feature import Imputer
from pyspark.sql import DataFrameStatFunctions as statFunc
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import IndexToString
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.ml.classification import GBTClassifier
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.classification import LinearSVC
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.mllib.evaluation import BinaryClassificationMetrics
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.feature import PCA


from pyspark.ml.classification import LogisticRegression
from pyspark.mllib.classification import LogisticRegressionWithLBFGS

from pyspark.ml import Pipeline
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

from pyspark.mllib.classification import SVMWithSGD, SVMModel
from pyspark.mllib.regression import LabeledPoint

from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

from sklearn.metrics import roc_curve, auc


%matplotlib inline
```

```python
import datetime
import numpy as np
import pandas as pd
from pandas import DataFrame as df
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(color_codes=True)
from scipy import stats

from chart_studio import plotly as py

import plotly.graph_objs as go
from plotly.offline import init_notebook_mode,iplot
init_notebook_mode(connected=True)

import os
memory = '4g'
pyspark_submit_args = ' --driver-memory ' + memory + ' pyspark-shell'
os.environ["PYSPARK_SUBMIT_ARGS"] = pyspark_submit_args

SparkContext.setSystemProperty('spark.executor.memory', '4g')
SparkContext.setSystemProperty('spark.driver.memory', '4g')

spark_conf = SparkConf().setAll(pairs = [('spark.executor.memory', '4g'), ('spark.executor.cores', '3'),
                                         ('spark.cores.max', '3'), ('spark.driver.memory','4g')])

spark = SparkSession.builder.master("local[*]").config(conf = spark_conf)
.appName("Lending-Club Loan Analysis using Pyspark").getOrCreate()
sqlContext = SQLContext(spark)

spark.sparkContext.setLogLevel('ERROR')

import warnings
warnings.filterwarnings('ignore')
```

Design - Loading the data to Spark DataFrame

Load Data to Spark DataFrame

In [2]:
```
loanDF = spark.read.csv("loan.csv", header=True, mode="DROPMALFORMED")

loanDF.printSchema()

loanDF_Pandas = pd.read_csv("loan.csv", low_memory=False)
```

```
root
 |-- id: string (nullable = true)
 |-- member_id: string (nullable = true)
 |-- loan_amnt: string (nullable = true)
 |-- funded_amnt: string (nullable = true)
 |-- funded_amnt_inv: string (nullable = true)
 |-- term: string (nullable = true)
 |-- int_rate: string (nullable = true)
 |-- installment: string (nullable = true)
 |-- grade: string (nullable = true)
 |-- sub_grade: string (nullable = true)
 |-- emp_title: string (nullable = true)
 |-- emp_length: string (nullable = true)
 |-- home_ownership: string (nullable = true)
 |-- annual_inc: string (nullable = true)
 |-- verification_status: string (nullable = true)
 |-- issue_d: string (nullable = true)
 |-- loan_status: string (nullable = true)
 |-- pymnt_plan: string (nullable = true)
 |-- url: string (nullable = true)
 |-- desc: string (nullable = true)
 |-- purpose: string (nullable = true)
 |-- title: string (nullable = true)
 |-- zip_code: string (nullable = true)
 |-- addr_state: string (nullable = true)
 |-- dti: string (nullable = true)
 |-- delinq_2yrs: string (nullable = true)
 |-- earliest_cr_line: string (nullable = true)
 |-- inq_last_6mths: string (nullable = true)
 |-- mths_since_last_delinq: string (nullable = true)
 |-- mths_since_last_record: string (nullable = true)
 |-- open_acc: string (nullable = true)
 |-- pub_rec: string (nullable = true)
 |-- revol_bal: string (nullable = true)
 |-- revol_util: string (nullable = true)
 |-- total_acc: string (nullable = true)
 |-- initial_list_status: string (nullable = true)
 |-- out_prncp: string (nullable = true)
 |-- out_prncp_inv: string (nullable = true)
 |-- total_pymnt: string (nullable = true)
 |-- total_pymnt_inv: string (nullable = true)
 |-- total_rec_prncp: string (nullable = true)
 |-- total_rec_int: string (nullable = true)
 |-- total_rec_late_fee: string (nullable = true)
 |-- recoveries: string (nullable = true)
 |-- collection_recovery_fee: string (nullable = true)
 |-- last_pymnt_d: string (nullable = true)
 |-- last_pymnt_amnt: string (nullable = true)
 |-- next_pymnt_d: string (nullable = true)
 |-- last_credit_pull_d: string (nullable = true)
 |-- collections_12_mths_ex_med: string (nullable = true)
 |-- mths_since_last_major_derog: string (nullable = true)
 |-- policy_code: string (nullable = true)
 |-- application_type: string (nullable = true)
 |-- annual_inc_joint: string (nullable = true)
 |-- dti_joint: string (nullable = true)
 |-- verification_status_joint: string (nullable = true)
 |-- acc_now_delinq: string (nullable = true)
 |-- tot_coll_amt: string (nullable = true)
 |-- tot_cur_bal: string (nullable = true)
 |-- open_acc_6m: string (nullable = true)
 |-- open_il_6m: string (nullable = true)
 |-- open_il_12m: string (nullable = true)
 |-- open_il_24m: string (nullable = true)
 |-- mths_since_rcnt_il: string (nullable = true)
 |-- total_bal_il: string (nullable = true)
 |-- il_util: string (nullable = true)
 |-- open_rv_12m: string (nullable = true)
 |-- open_rv_24m: string (nullable = true)
 |-- max_bal_bc: string (nullable = true)
 |-- all_util: string (nullable = true)
 |-- total_rev_hi_lim: string (nullable = true)
 |-- inq_fi: string (nullable = true)
 |-- total_cu_tl: string (nullable = true)
 |-- inq_last_12m: string (nullable = true)
```

Design - Exploratory Data Analysis

The data used for this project is the structured data with few missing/null values.

This data consists of 80+ features of three distinct types: continuous, categorical, ordinal.

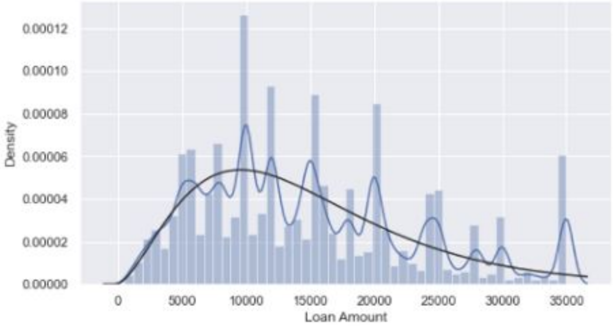
I have gathered business domain knowledge about the data to deal with data cleaning and missing data imputation.

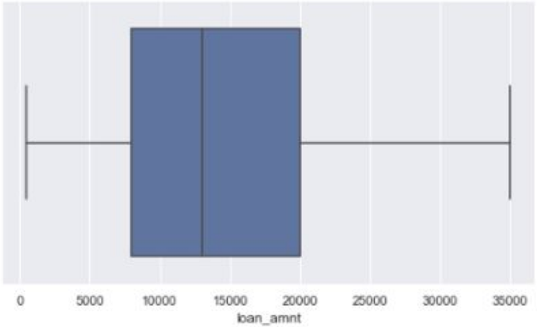
During the preliminary exploration of data, it was noticed that some of the features with more than 50% of the missing data, I have planned to drop these features while data processing for learning model.

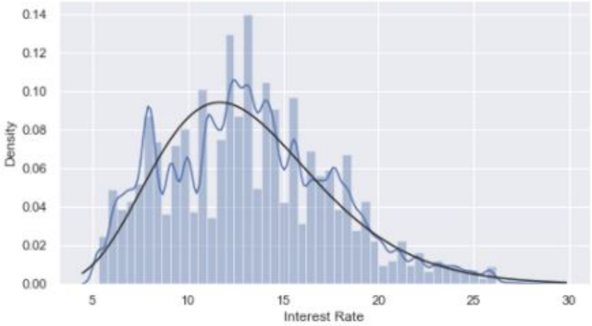
For data analysis phase, I have converted some of the feature type to relevant primitive types. I have handled data cleaning and imputation part while preparing data for learning model. For preliminary data cleanup, spark data frame transformation APIs was used to convert the feature data types.
As part of exploratory data analysis, I have tried to find any interesting fact and findings about the loans from the historical loan data. This analysis helped to develop my understanding about data and its distribution patterns. In addition, this assisted to select the most effecting features and develop business rules for missing data imputation.

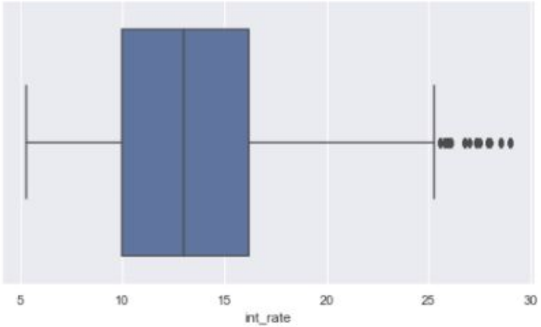
Design - Exploratory Data Analysis

Analyzing Loan amount and Interest rates



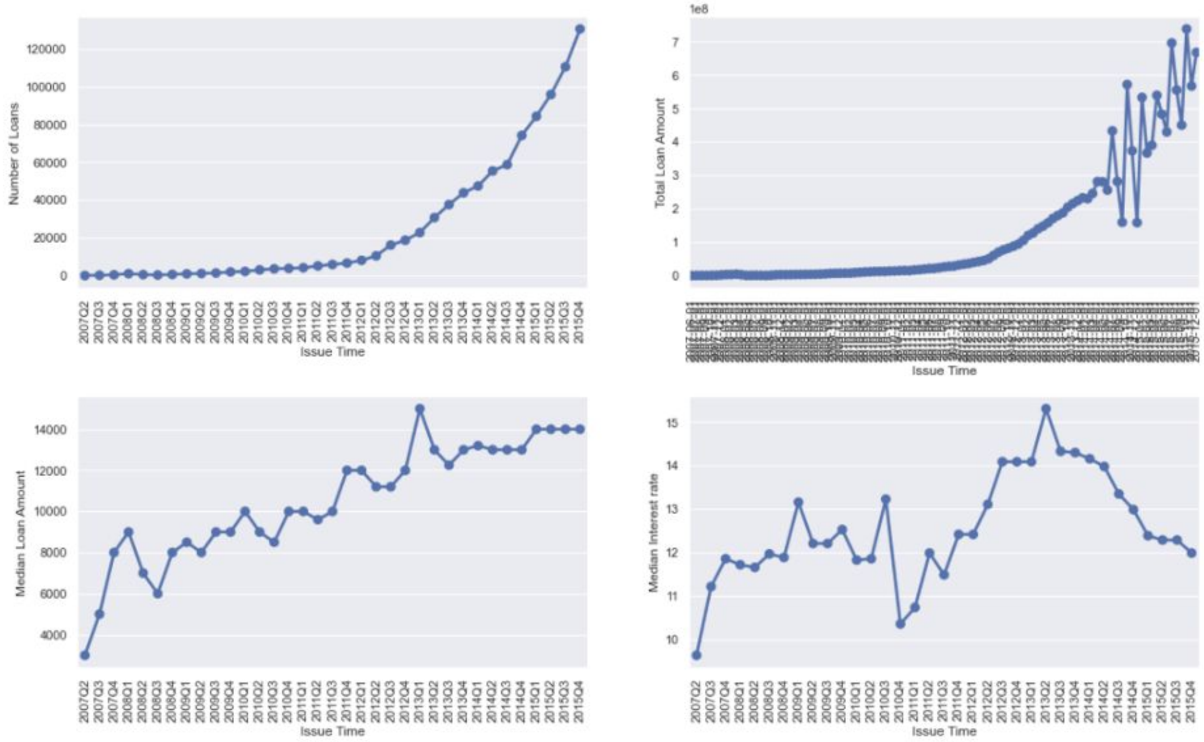






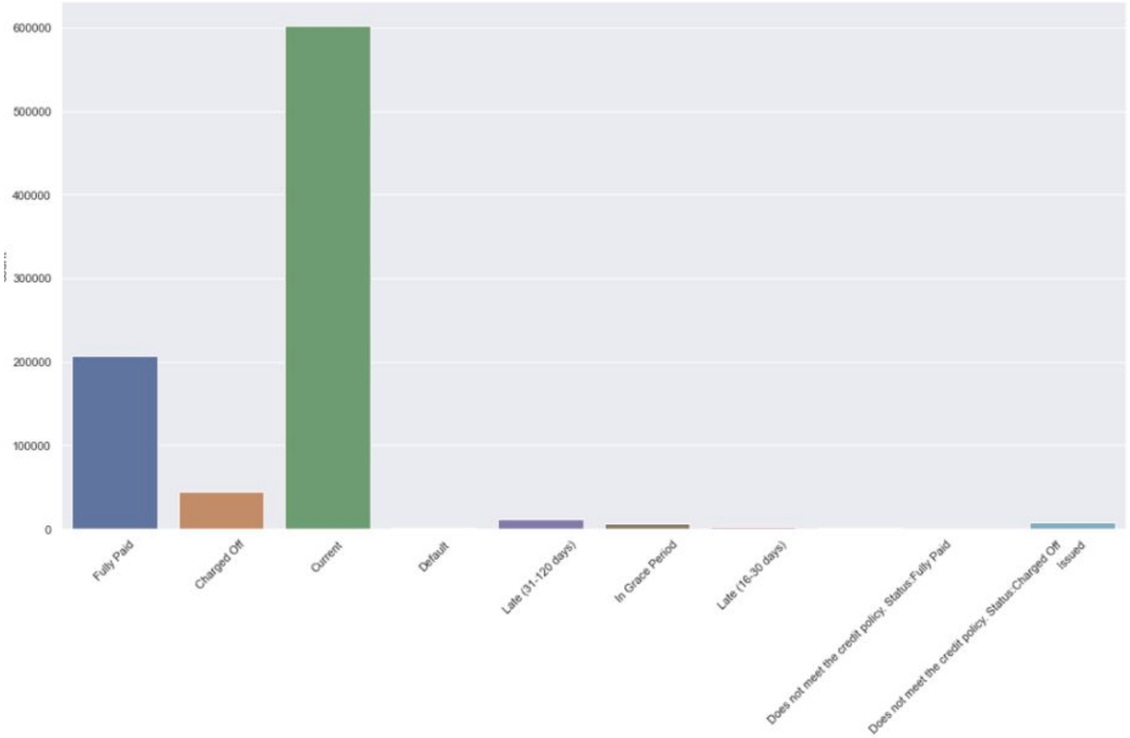
Design - Exploratory Data Analysis

Analyzing Loans Interest rates over time



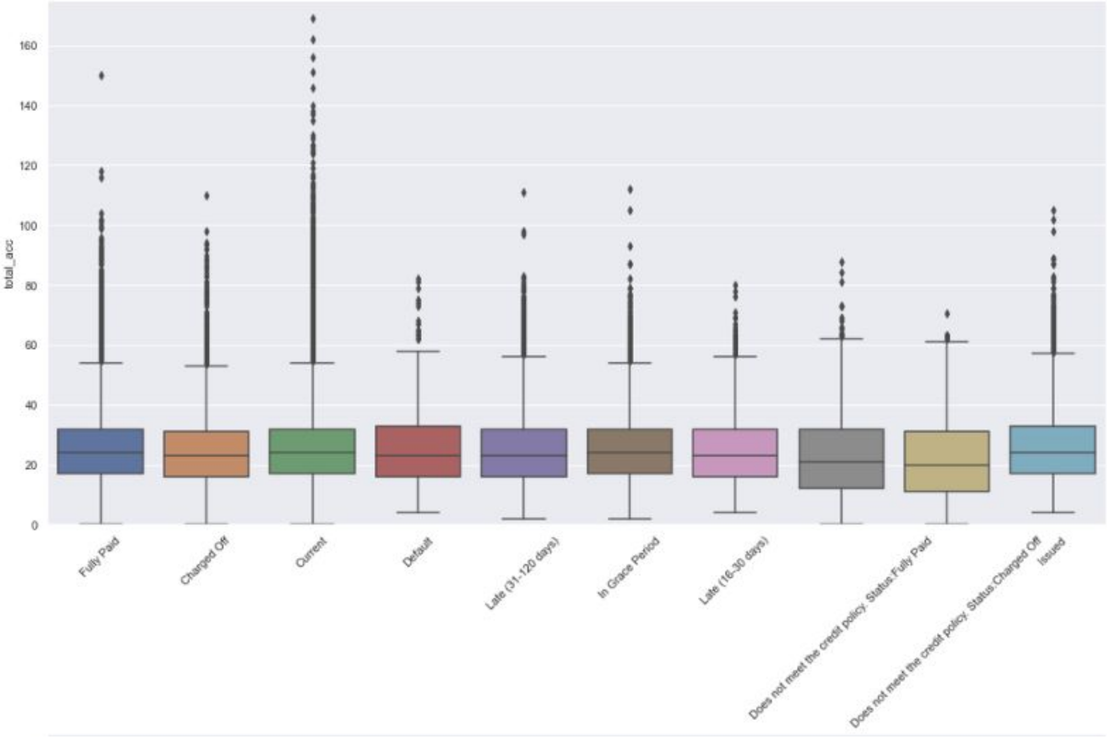
Design - Exploratory Data Analysis

Analyzing Loans over loan status



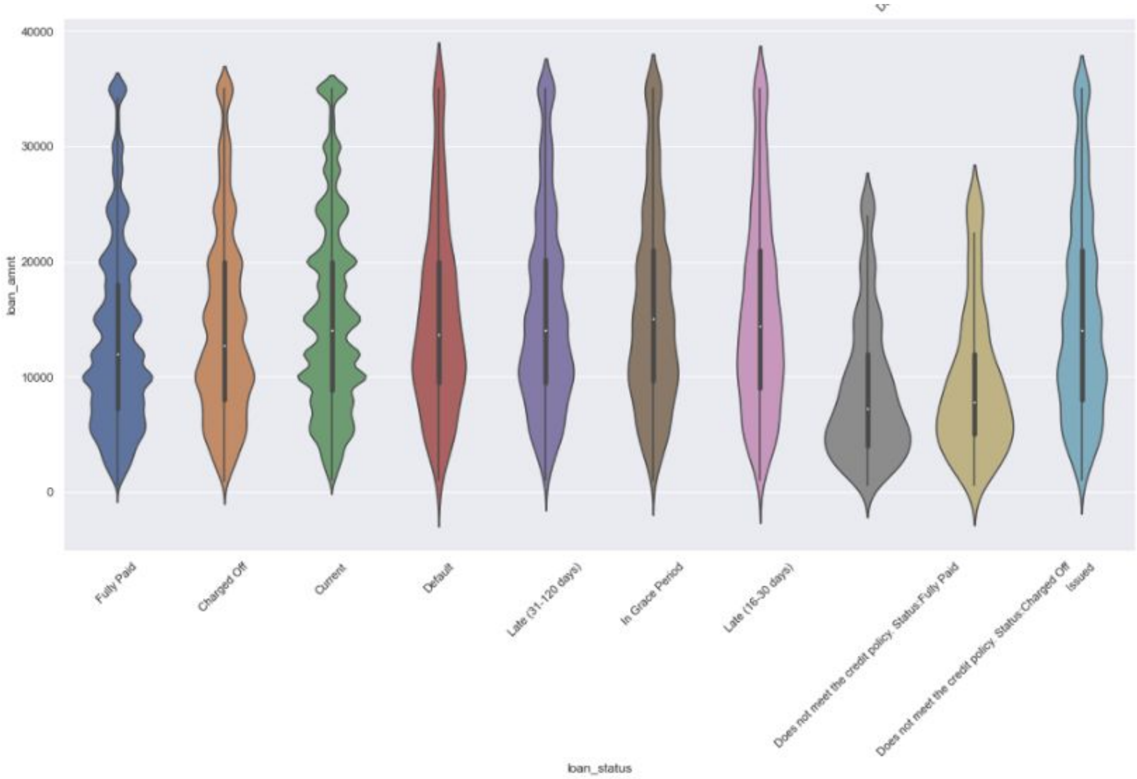
Design - Exploratory Data Analysis

Analyzing Loans over loan status



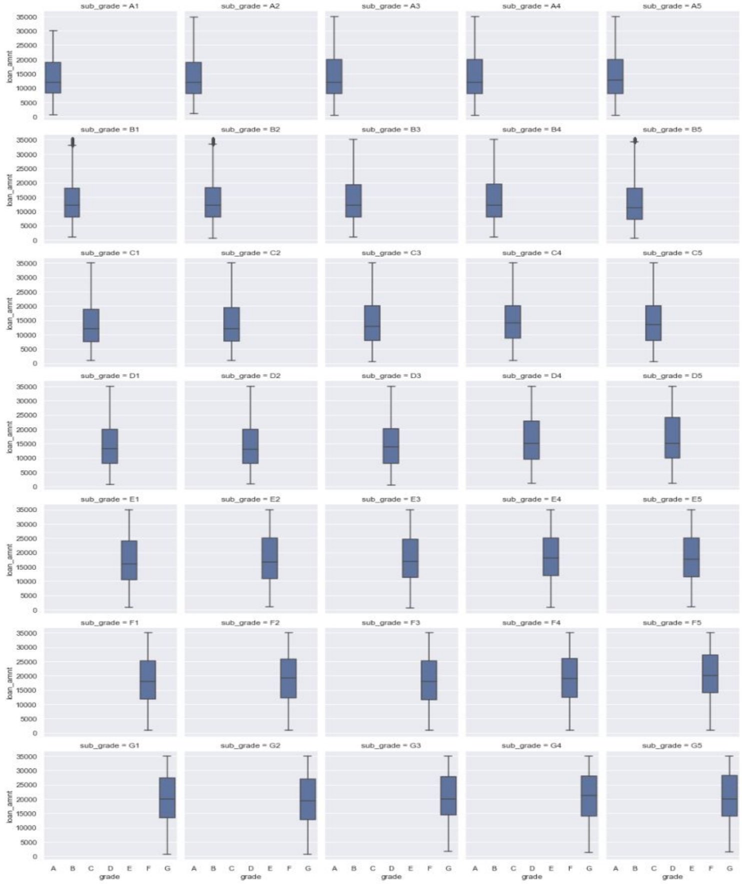
Design - Exploratory Data Analysis

Analyzing Loans over loan status



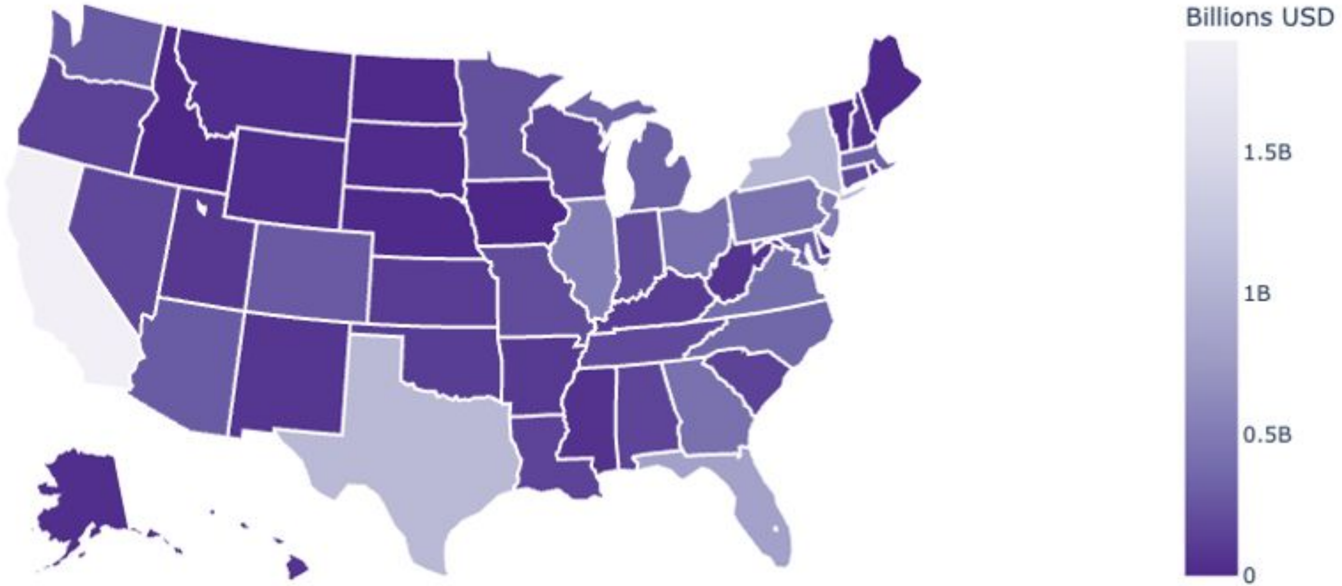
Design - Exploratory Data Analysis

Analyzing loan amount and interest rate quantile summary for each grade, factored over subgrade



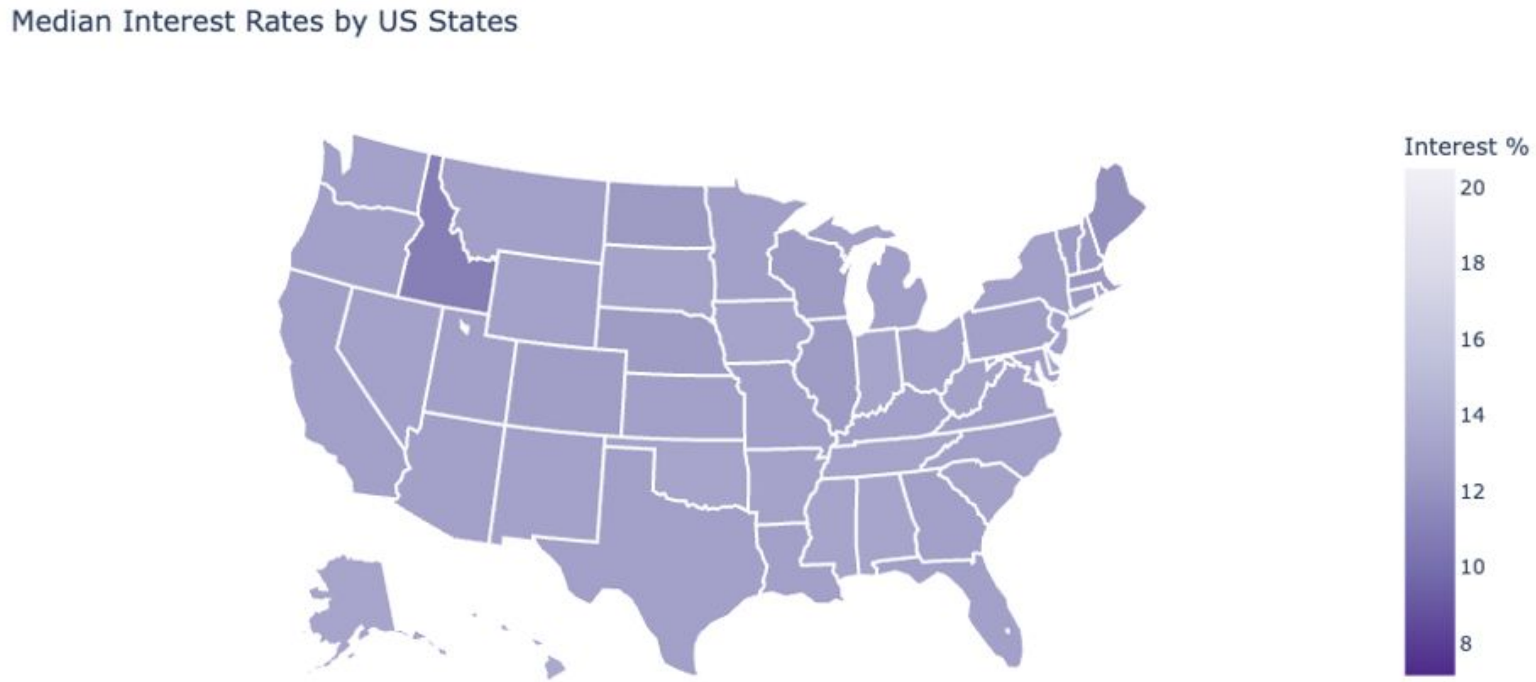
Design - Exploratory Data Analysis

US States map with the total loan amount



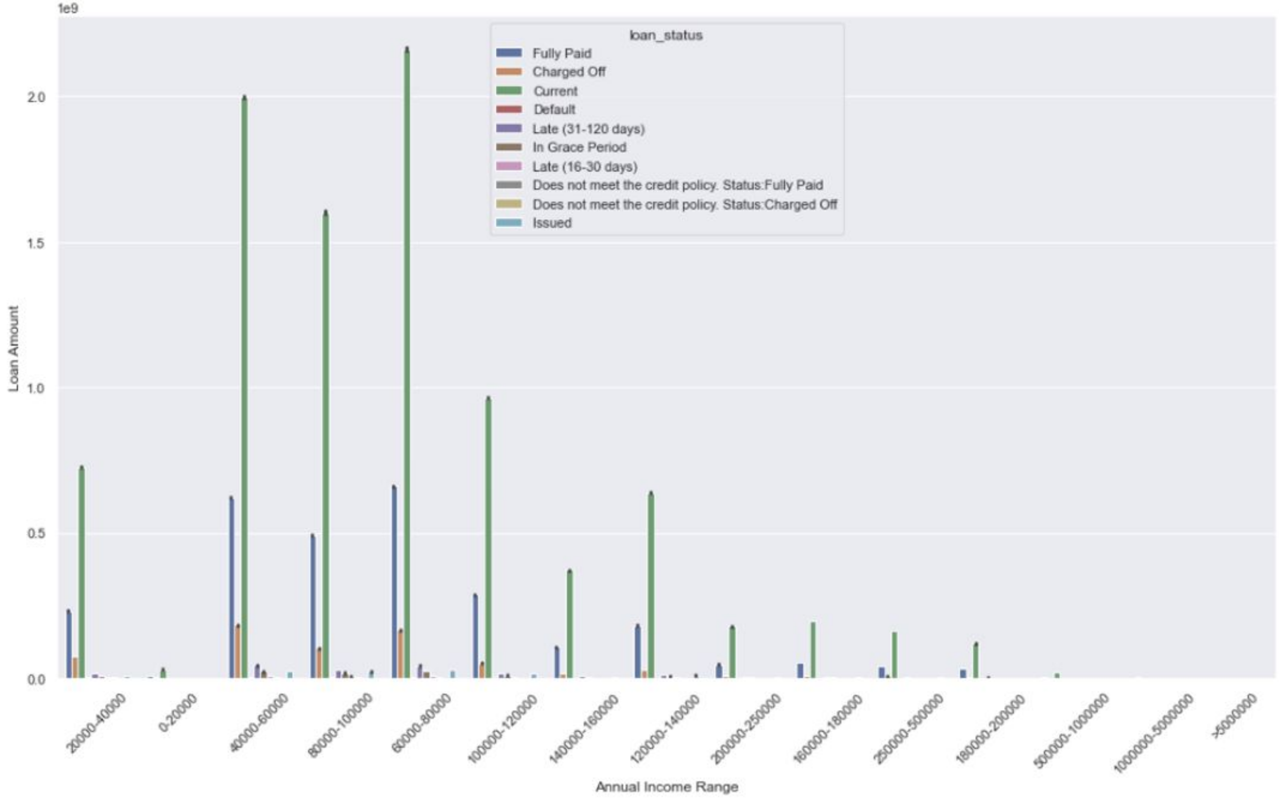
Design - Exploratory Data Analysis

US States map with median interest rates



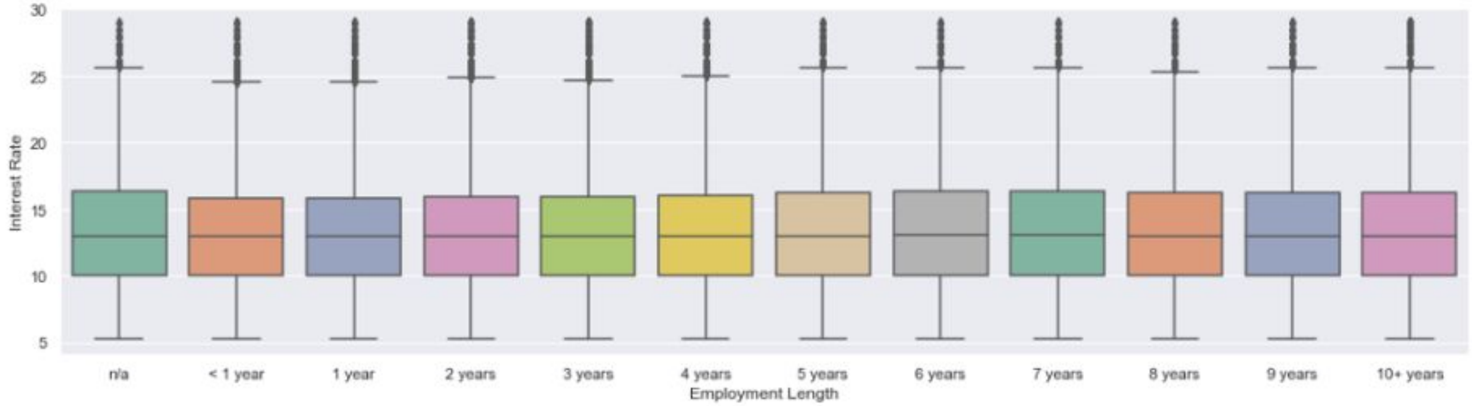
Design - Exploratory Data Analysis

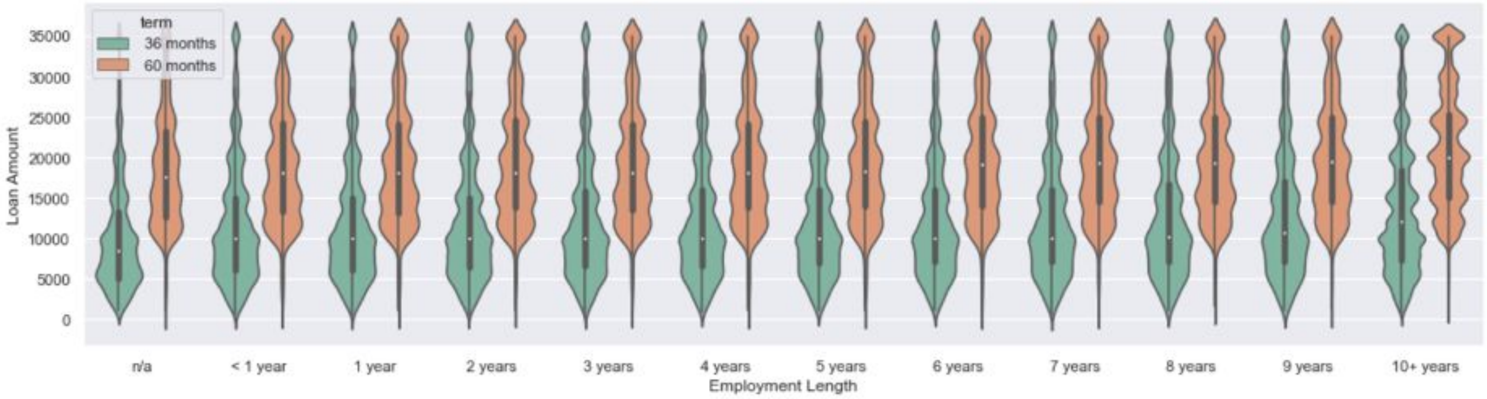
Total loan amount by income range and loan status



Design - Exploratory Data Analysis

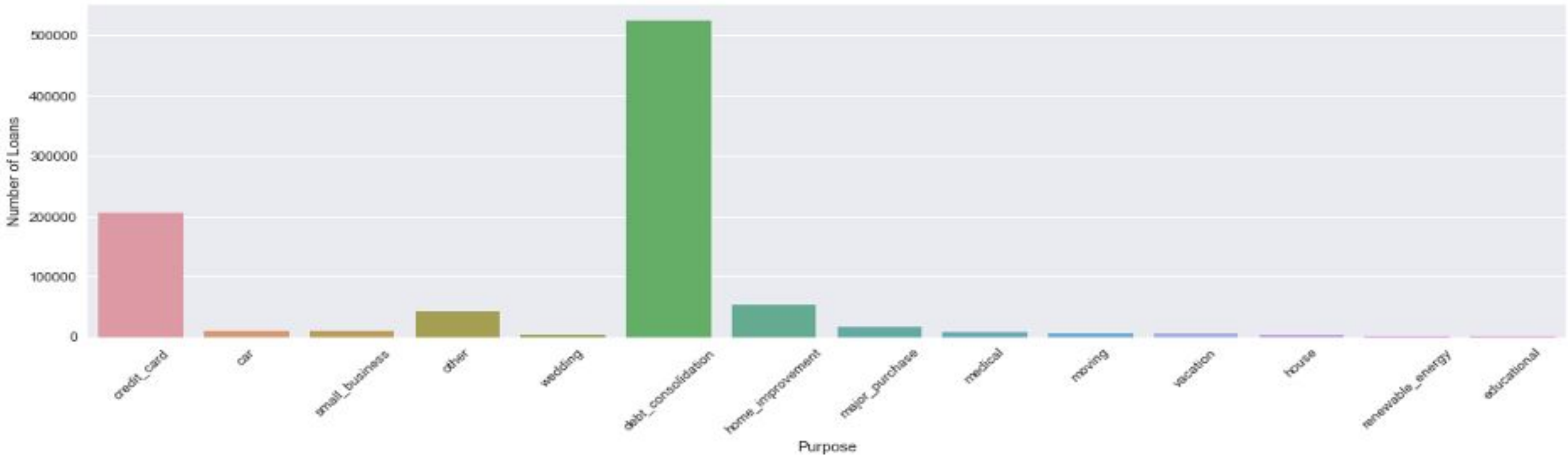
Analyzing Loan Amount and interest rate over customers employment length (With the loan term).





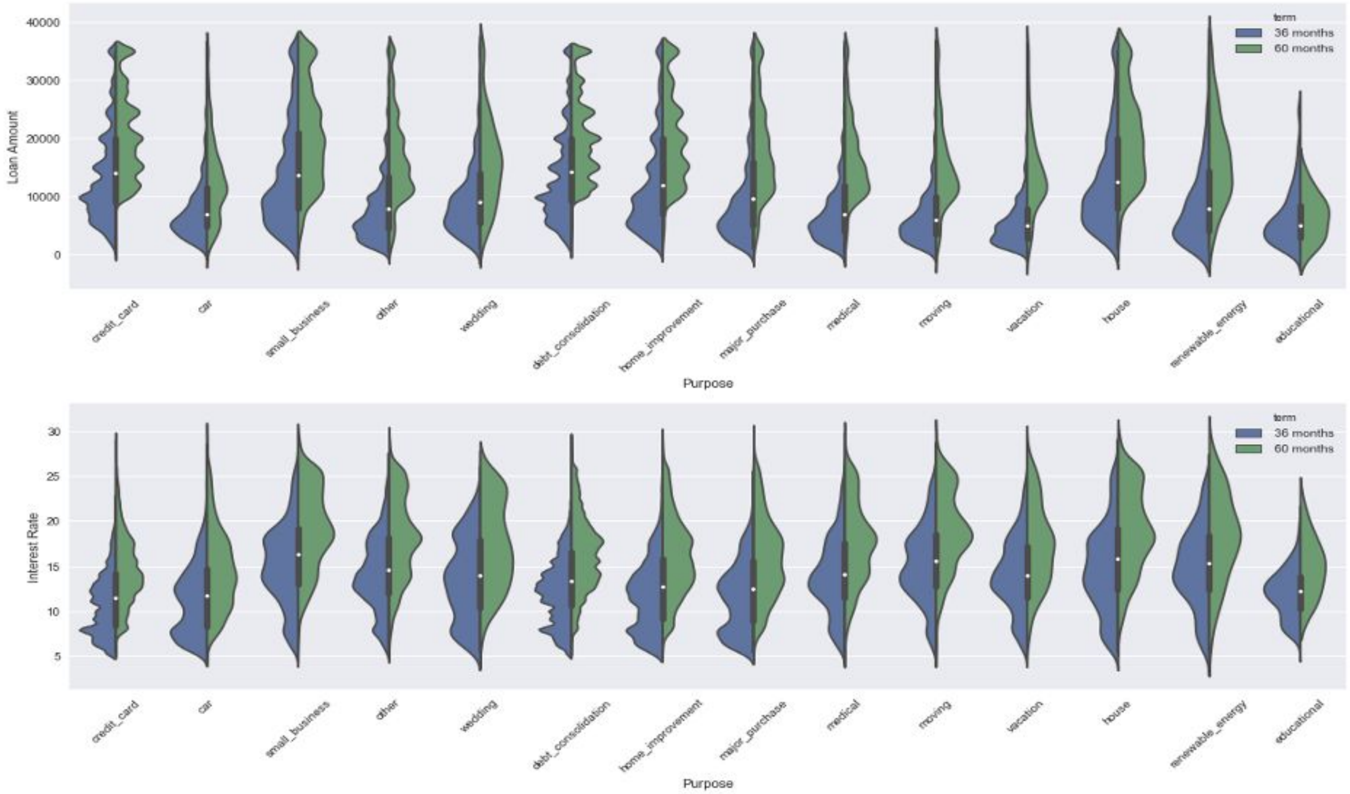
Design - Exploratory Data Analysis

Analyzing loans by its purpose



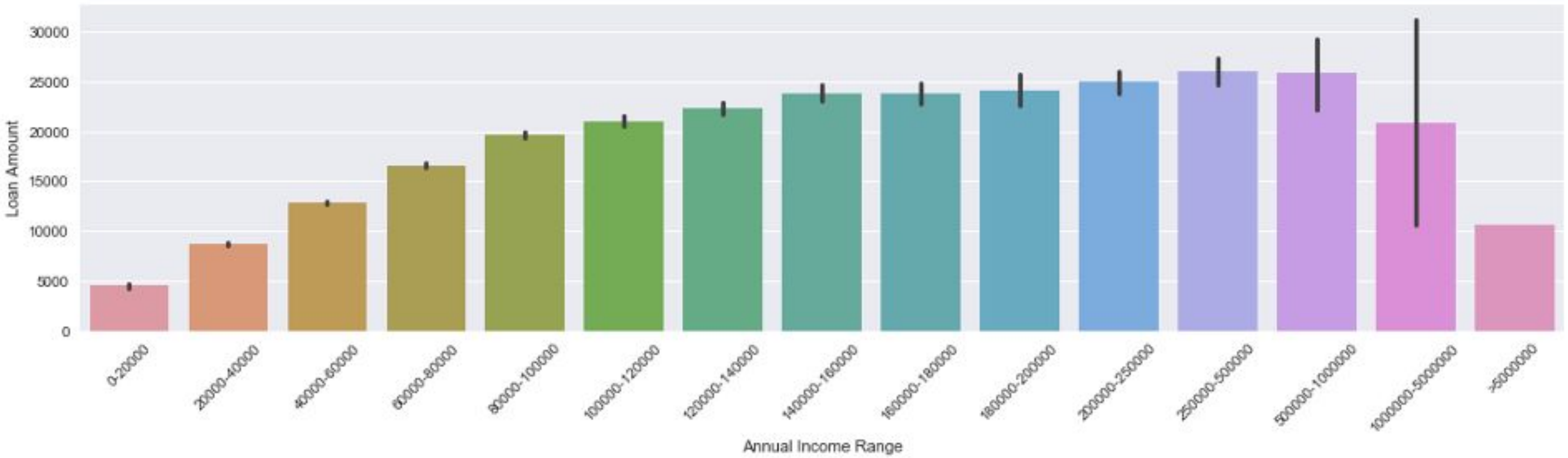
Design - Exploratory Data Analysis

Analyzing loans by its purpose



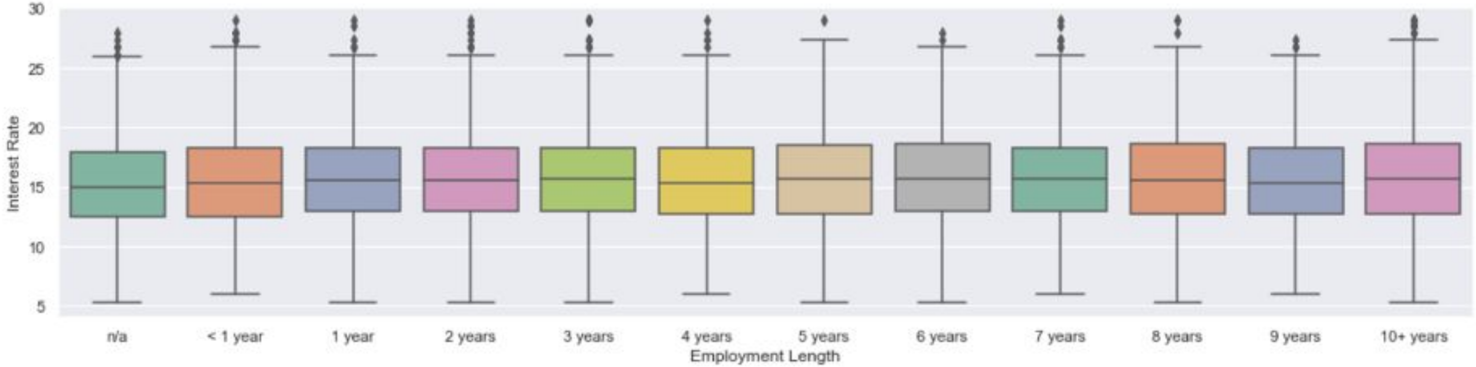
Design - Exploratory Data Analysis

Analyzing Default Loans



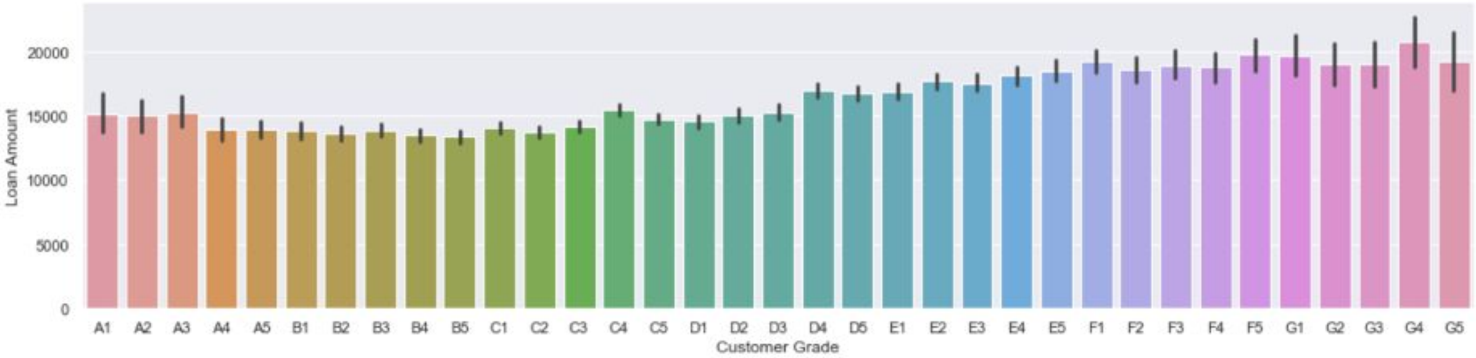
Design - Exploratory Data Analysis

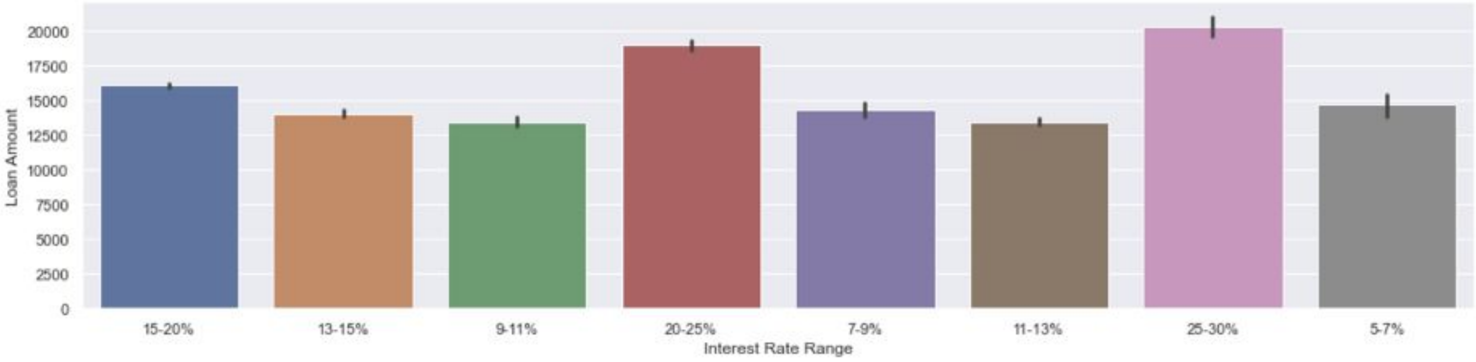
Analyzing Default Loans



Design - Exploratory Data Analysis

Analyzing Default Loans





Design - Data Preparation and Processing

Removed the columns which had more than 50% of missing data. Also removed the columns that has no direct relation to our analysis such as ID's and other unique identifiers.

Removed the variables which has only one category such as policy_code , pymnt_plan and initial_list_status, application_type.

Three types of categorical features: ordinal, nominal and binary. We encode these with built-in Spark APIs using StringIndex API for ordinal feature, VectorAssembler API for nominal feature and perform custom binary encoding for binary feature.

For our classification model we needed to add a class label variable "isDefault" to our data set. isDefault =1 if the loan is defaulted and 0 if not defaulted.

We labeled our data based on the features provided in data. "Default", "Charged Off", "Late (31-120 days)", "Late (16-30 days)", "Does not meet the credit policy. Status:Charged Off" were
considered as defaulted loans.

Design - Algorithm Implementation

Developed four classifier models namely,

- Logistic Regression
- Naïve Bayes
- Random Forest
- Gradient Boosting Classifier

to determine and compared their predictive capabilities. Multiple assessment metrics as given below were used to determine the most successful model.

- Confusion Matrix
- Sensitivity VS Specificity
- ROC curve
- Maximum Likelihood
- Hyper Parameter Learning

Design - Algorithm Implementation

Logistic Regression

Logistic regression is a popular method to predict a categorical/binary response.

The main intention behind using this learning algorithm is to handle the class imbalance. Logistic regression is one of the implementation of linear models, for which spark provides ability to handle the class imbalance by adding a class weights.

We used pyspark.ml.classifier package to implement the model with optimized hyper parameter. We also implemented “elastic net” regularization since it solves the limitations of both L1 and L2 regularization.

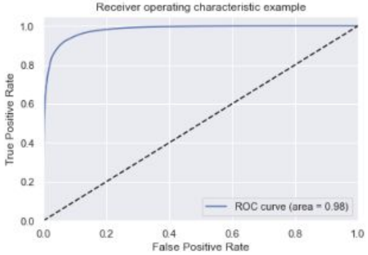
Design - Algorithm Implementation

Logistic Regression

Logistic Regression Classifier

```
print("**** Running Logistic Regression Classifier with best parameter found using ML pipeline **** ")

# Create initial LogisticRegression model
lr_classifier = LogisticRegression(labelCol="label", featuresCol="features", maxIter=3, weightCol="weightColumn")

# Train model with Training Data
lrModel = lr_classifier.fit(trainingSetDF)

# Make predictions on test data using the transform() method.
# LogisticRegression.transform() will only use the 'features' column.
predictions = lrModel.transform(testSetDF)

# Evaluate model
evaluator = MulticlassClassificationEvaluator( labelCol="label")
lr_accuracy = evaluator.evaluate(predictions)

getEvaluationMatrix(predictions)
```

```
**** Running Logistic Regression Classifier with best parameter found using ML pipeline ****
totalCount   -  263654
correctCount -  251846
wrongCount   -  11808
trueP        -  235938
trueN        -  15908
falseN       -  2226
falseP       -  9582
ratioWrong   -  0.04478596949031685
ratioCorrect -  0.9552140305096831
Accuracy     -  0.9552140305096831
Precision    -  96.09726295210166
Recall       -  99.06534992694111
F-1 Score    -  97.5587366958593
Sensitivity  -  99.06534992694111
Specificity  -  62.40878775990585
ROC score is -  0.9800881762583983
```



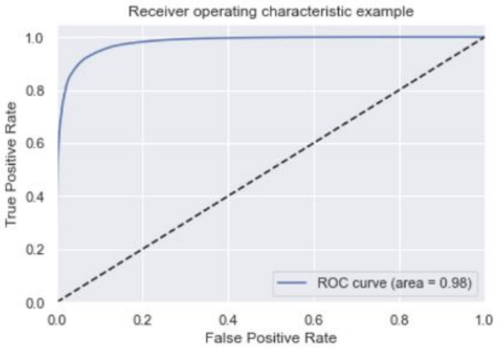
Design - Algorithm Implementation

Logistic Regression using Cross Validation

Logistic Regression Classifier with ML Pipeline to find the best hyper parameters Using Cross Validation

```
paramGrid = ParamGridBuilder().addGrid(lr_classifier.regParam, [0.01, 0.1, 1.0])
.addGrid(lr_classifier.elasticNetParam, [0.0, 0.5, 1.0]).addGrid(lr_classifier.maxIter, [1, 3, 10]).build()

pipeline = Pipeline(stages=[ lr_classifier ])

evaluator = MulticlassClassificationEvaluator( labelCol = "label" )

crossval_lr = CrossValidator( estimator = pipeline, estimatorParamMaps = paramGrid,
                              evaluator = evaluator, numFolds = 10)

# Run cross-validation, and choose the best set of parameters.
cvModel_lr = crossval_lr.fit( trainingSetDF )

cvLR_predictions = cvModel_lr.transform(testSetDF)
cvLR_accuracy = evaluator.evaluate(cvLR_predictions)

bestModel = cvModel_lr.bestModel
print(cvModel_lr.avgMetrics)
print(list(zip(cvModel_lr.avgMetrics, paramGrid)))

print(bestModel.stages[0]._java_obj.getRegParam())
print(bestModel.stages[0]._java_obj.getElasticNetParam())
print(bestModel.stages[0]._java_obj.getMaxIter())

print(cvLR_accuracy)

getEvaluationMatrix(cvLR_predictions)
```

```
totalCount   -  263654
correctCount -  263653
wrongCount   -  1
trueP        -  245520
trueN        -  18133
falseN       -  1
falseP       -  0
ratioWrong   -  3.792849719708406e-06
ratioCorrect -  0.9999962071502803
Accuracy     -  0.9999962071502803
Precision    -  100.0
Recall       -  99.99959270286453
F-1 Score    -  99.99979635101754
Sensitivity  -  99.99959270286453
Specificity  -  100.0
ROC score is -  0.9800872919883237
```



Design - Algorithm Implementation

Naive Bayes

To extend scope of our model building, we wanted to include a generative machine learning model (probabilistic classifier) in our trails.

The intention behind using this classifier was to handle the class imbalance issue by adding the default prior belief (beta distribution function) and calculate the max posterior probability.

We used pyspark.ml.classifier package to implement this algorithm with only one optimized hyper parameter for Laplace smoothing.

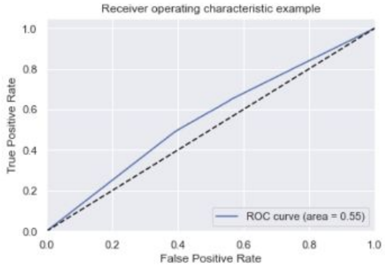
Design - Algorithm Implementation

Naive Bayes

NaiveBayes Classifier

```
print("**** Running NaiveBayes Classifier with best parameter found using ML pipeline **** ")
# Create initial NaiveBayes model
nb_classifier = NaiveBayes(labelCol="label", featuresCol="features", smoothing=50, weightCol="weightColumn")

# Train model with Training Data
nbModel = nb_classifier.fit(trainingSetDF)

# Make predictions on test data using the transform() method.
# NaiveBayes.transform() will only use the 'features' column.
predictions = nbModel.transform(testSetDF)

# Evaluate model
evaluator = MulticlassClassificationEvaluator( labelCol="label", predictionCol="prediction", metricName="accuracy")
nb_accuracy = evaluator.evaluate(predictions)

getEvaluationMatrix(predictions)
```

```
**** Running NaiveBayes Classifier with best parameter found using ML pipeline ****
totalCount    -  263654
correctCount  -  133092
wrongCount    -  130562
trueP         -  122204
trueN         -  10888
falseN        -  7246
falseP        -  123316
ratioWrong    -  0.4952020451045689
ratioCorrect  -  0.5047979548954311
Accuracy      -  0.5047979548954311
Precision     -  49.773541870316066
Recall        -  94.40247199691001
F-1 Score     -  65.18068112115637
Sensitivity   -  94.40247199691001
Specificity   -  8.113021966558374
ROC score is  -  0.5541658747092667
```



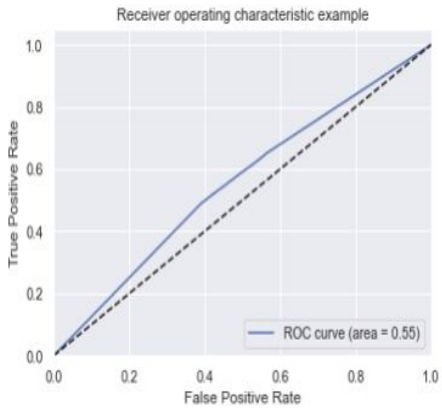
Design - Algorithm Implementation

Naive Bayes using Cross Validation

NaiveBayes Classifier with ML Pipeline to find the best hyper parameters Using Cross Validation¶

```
paramGrid = ParamGridBuilder().addGrid(nb_classifier.smoothing, [1.0, 2.0, 3.0]).build()

pipeline = Pipeline(stages=[ nb_classifier ])

evaluator = MulticlassClassificationEvaluator( labelCol="label", predictionCol="prediction", metricName="accuracy")

crossval_nb = CrossValidator( estimator = pipeline, estimatorParamMaps = paramGrid,
                              evaluator = evaluator, numFolds = 10)

# Run cross-validation, and choose the best set of parameters.
cvModel_nb = crossval_nb.fit( trainingSetDF )

cvNB_predictions = cvModel_nb.transform(testSetDF)
cvNB_accuracy = evaluator.evaluate(cvNB_predictions)

bestModel = cvModel_nb.bestModel
print(cvModel_nb.avgMetrics)
print(list(zip(cvModel_nb.avgMetrics, paramGrid)))

print(bestModel.stages[0]._java_obj.getSmoothing())

print(cvNB_accuracy)

getEvaluationMatrix(cvNB_predictions)
```

```
totalCount   - 263654
correctCount - 133091
wrongCount   - 130563
trueP        - 122203
trueN        - 10888
falseN       - 7246
falseP       - 123317
ratioWrong   - 0.4952058379542886
ratioCorrect - 0.5047941620457114
Accuracy     - 0.5047941620457114
Precision    - 49.773134571521666
Recall       - 94.40242875572619
F-1 Score    - 65.18032157325005
Sensitivity  - 94.40242875572619
Specificity  - 8.11296151410156
ROC score is - 0.554159276711371
```



Design - Algorithm Implementation

Random Forest Classifier

To further deal with the data imbalance issue, we tried random forest, ensembles of decision trees.

Random forest is the aggregation of multiple decision trees which usages entropy/Gini to find the impurities, which makes it less sensitive to the class imbalance.

The spark implementation supports random forest for binary as well as multiclass classifications.

We used pyspark.ml.classifier package to implement this algorithm with optimized hyper parameter. We also tested our model with different bin sizes and tree debts.

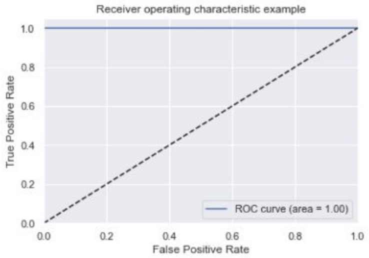
Design - Algorithm Implementation

Random Forest Classifier

Random Forest Classifier

```
# Create initial Random Forest Classifier model
print("**** Running Random Forest Classifier with best parameter found using ML pipeline **** ")
rf_classifier = RandomForestClassifier( impurity="gini", maxDepth=12,
                                        numTrees=10, featureSubsetStrategy="auto", seed=1395)

# Train model with Training Data
rf_model = rf_classifier.fit(trainingSetDF)

# Print the Forest tree rules.
#rf_model.toDebugString

# Make predictions on test data using the transform() method.
# RandomForest.transform() will only use the 'features' column.
predictions = rf_model.transform(testSetDF)

evaluator = MulticlassClassificationEvaluator( labelCol = "label" )
rf_accuracy = evaluator.evaluate(predictions)

getEvaluationMatrix(predictions)
```

```
**** Running Random Forest Classifier with best parameter found using ML pipeline ****
totalCount   -  263654
correctCount -  257868
wrongCount   -  5786
trueP        -  245520
trueN        -  12348
falseN       -  5786
falseP       -  0
ratioWrong   -  0.021945428478232835
ratioCorrect -  0.9780545715217671
Accuracy     -  0.9780545715217671
Precision    -  100.0
Recall       -  97.69762759345181
F-1 Score    -  98.8354071646814
Sensitivity  -  97.69762759345181
Specificity  -  100.0
ROC score is -  0.9999137780525865
```



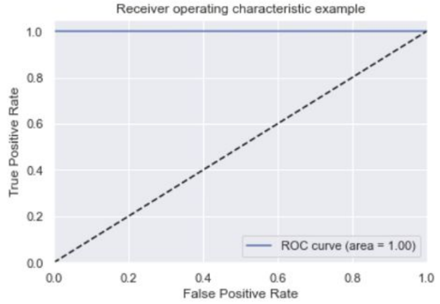
Design - Algorithm Implementation

Random Forest Classifier using Cross Validation

Random Forest Classifier with ML Pipeline to find the best hyper parameters Using Cross Validation

```
paramGrid = ParamGridBuilder().addGrid(rf_classifier.maxBins,
            [25, 28, 31, 34]).addGrid(rf_classifier.maxDepth, [4, 6, 8, 12]).addGrid(rf_classifier.impurity,
                                                                          ["entropy", "gini"]).build()

pipeline = Pipeline(stages=[ rf_classifier ])

evaluator = MulticlassClassificationEvaluator( labelCol = "label" )

crossval = CrossValidator( estimator = pipeline, estimatorParamMaps = paramGrid, evaluator = evaluator, numFolds = 10)

# Run cross-validation, and choose the best set of parameters.
cvModel = crossval.fit( trainingSetDF )

cv_predictions = cvModel.transform(testSetDF)
cv_accuracy = evaluator.evaluate(cv_predictions)

bestModel = cvModel.bestModel
print(cvModel.avgMetrics)
print(list(zip(cvModel.avgMetrics, paramGrid)))

print(bestModel.stages[0]._java_obj.getMaxBins())
print(bestModel.stages[0]._java_obj.getMaxDepth())
print(bestModel.stages[0]._java_obj.getImpurity())

print(cv_accuracy)

getEvaluationMatrix(cv_predictions)
```

```
totalCount    -  263654
correctCount  -  261772
wrongCount    -  1882
trueP         -  245520
trueN         -  16252
falseN        -  1882
falseP        -  0
ratioWrong    -  0.00713814317249122
ratioCorrect  -  0.9928618568275088
Accuracy      -  0.9928618568275088
Precision     -  100.0
Recall        -  99.23929475105294
F-1 Score     -  99.61819517083838
Sensitivity   -  99.23929475105294
Specificity   -  100.0
ROC score is  -  0.9999136312731876
```



Design - Algorithm Implementation

Gradient Boosted Trees

In contrary to random forest, which tried to minimize the error by reducing the variance, we tested the opposite way by reducing the bias.

Boosting reduces error mainly by reducing bias and also to some extent variance, by aggregating the output from many models.

GBM is a boosting method, which builds on weak classifiers. The idea is to add a classifier at a time, so that the next classifier is trained to improve the already trained ensemble in sequential order.

We used pyspark.ml.classifier package to implement this algorithm with optimized hyper parameter. We also tested our model with different step sizes for gradient descent and tree depth.

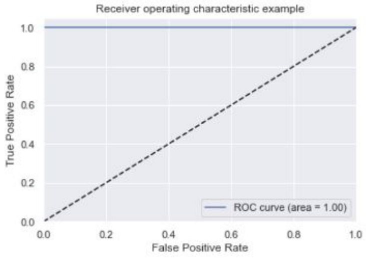
Design - Algorithm Implementation

Gradient Boosted Trees

Gradient Boosting Classifier

```
print("**** Running Gradient Boosting Classifier with best parameter found using ML pipeline **** ")

# Create initial Gradient Boosting Classifier model
gb_classifier = GBTClassifier(labelCol="label", featuresCol="features", maxDepth=5,
                              maxBins=5, lossType="logistic", maxIter=10, stepSize=.00000001)

# Train model with Training Data
gbModel = gb_classifier.fit(trainingSetDF)

# Make predictions on test data using the transform() method.
# NaiveBayes.transform() will only use the 'features' column.
predictions = gbModel.transform(testSetDF)

# Evaluate model
evaluator = MulticlassClassificationEvaluator( labelCol="label", predictionCol="prediction", metricName="accuracy")
gb_accuracy = evaluator.evaluate(predictions)

getEvaluationMatrix(predictions)
```

```
**** Running Gradient Boosting Classifier with best parameter found using ML pipeline ****
totalCount   -  263654
correctCount -  263654
wrongCount   -  0
trueP        -  245520
trueN        -  18134
falseN       -  0
falseP       -  0
ratioWrong   -  0.0
ratioCorrect -  1.0
Accuracy     -  1.0
Precision    -  100.0
Recall       -  100.0
F-1 Score    -  100.0
Sensitivity  -  100.0
Specificity  -  100.0
ROC score is -  1.0
```



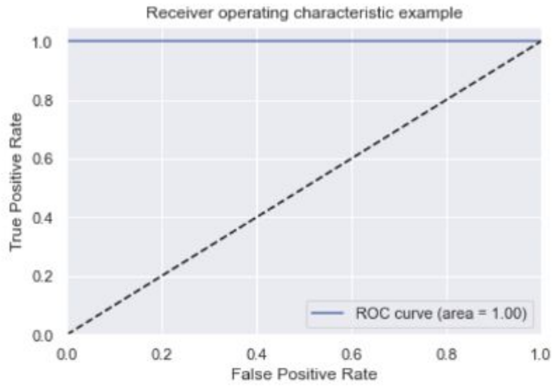
Design - Algorithm Implementation

Gradient Boosted Trees using Cross Validation

Gradient Boosting Classifier with ML Pipeline to find the best hyper parameters Using Cross Validation

```
paramGrid = ParamGridBuilder().addGrid(gb_classifier.maxDepth,
            [3, 5, 10]).addGrid(gb_classifier.maxIter, [5, 10, 15]).
            addGrid(gb_classifier.stepSize, [0.01, 0.1, 1.0]).build()

pipeline = Pipeline(stages=[ gb_classifier ])

evaluator = MulticlassClassificationEvaluator( labelCol="label", predictionCol="prediction", metricName="accuracy")

crossval_gb = CrossValidator( estimator = pipeline, estimatorParamMaps = paramGrid, evaluator = evaluator,
                              numFolds = 10)

# Run cross-validation, and choose the best set of parameters.
cvModel_gb = crossval_gb.fit( trainingSetDF )

cvGB_predictions = cvModel_gb.transform(testSetDF)
cvGB_accuracy = evaluator.evaluate(cvGB_predictions)

bestModel = cvModel_gb.bestModel
print(cvModel_gb.avgMetrics)
print(list(zip(cvModel_gb.avgMetrics, paramGrid)))

print(bestModel.stages[0]._java_obj.getMaxDepth())
print(bestModel.stages[0]._java_obj.getMaxIter())
print(bestModel.stages[0]._java_obj.getStepSize())

print(cvGB_accuracy)

getEvaluationMatrix(cvGB_predictions)
```

```
totalCount   -  263654
correctCount -  263654
wrongCount   -  0
trueP        -  245520
trueN        -  18134
falseN       -  0
falseP       -  0
ratioWrong   -  0.0
ratioCorrect -  1.0
Accuracy     -  1.0
Precision    -  100.0
Recall       -  100.0
F-1 Score    -  100.0
Sensitivity  -  100.0
Specificity  -  100.0
ROC score is -  1.0
```



Design - Usefulness of the model

As part of the Lending Club's revenue generation model, it charges an origination fee to the borrowers and service fee to the investors. This model can be useful to provide comprehensive analysis of the historical data as well as a smart prediction about the investor's money to lower their risk. By developing a nearly perfect prediction model, we would hope to reduce the number of delinquencies in the investment and helps genuine borrowers to maintain their credit ratings. This would help Lending Club to engage more investors and borrowers in their platform, hence increasing the revenue growth.

Furthermore, I would like to highlight few recommendations noticed from the exploratory data analysis:

1. The higher the loan amount, the higher the likelihood of default. Investors should invest in loans that are approximation $9000 or less.
2. Loans with term of 36 months tended to be defaulted a lot more than loans with term of 60 months. Investor should invest in long terms.
3. Certain sub-grades were almost likely to default compared to other sub-grades. Selecting loans of subgrade B5 and higher will result in a 90% chance of repayment.

Design - Summary

If Lending Club must provide the more precise information to the investors to reduce their investment risk, they need to provide the default likeliness of borrower more accurately. Should this model be used in real life, the goal of this project is to retain more investors for Lending Club. Therefore, more focus is put on precision and specificity of the model.

	Accuracy (%)	**Precision**	**Recall**	**F1 Score**
Naïve Bayes	50	49.96	94.31	65.32
Logistic Regression	95	96.05	99.0	97.50
Random Forest	**99**	**100.0**	**99.9**	**99.99**
Gradient Boosting Classifier	**100**	**100.0**	**100.0**	**100.00**

References

https://www.lendingclub.com/
https://www.kaggle.com/wendykan
https://data.world/lpetrocelli/lendingclub-loan-data-2017-q-1
https://spark.apache.org/docs/2.2.0/ml-guide.html
https://spark.apache.org/docs/2.3.0/api/python/pyspark.html

Thank you :)