PROJECT REPORT

Credit Default Analysis

Submitted towards the partial fulfillment of the criteria for award of Genpact Data Science Prodegree by Imarticus

Submitted By:

Allwyn Joseph
Lalit Kacha
Rahul Dayma
Utkarsh Khemka

Course and Batch: DSP19 October'18

Acknowledgements

We are using this opportunity to express our gratitude to everyone who supported us throughout the course of this group project. We are thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the project work. We are sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the project.

Further, we were fortunate to have Ms. Nikita Tandel as our mentor. She has readily shared her immense knowledge in data analytics and guide us in a manner that the outcome resulted in enhancing our data skills.

We wish to thank, all the faculties, as this project utilized knowledge gained from every course that formed the DSP program.

We certify that the work done by us for conceptualizing and completing this project is original and authentic.

Date: January 23, 2019

Place: Mumbai

Allwyn Joseph

Lalit Kacha

Rahul Dayma

Utkarsh Khemka

Certificate of Completion

I hereby certify that the project titled "Title comes here" was undertaken and completed under my supervision by Member 1 and Member 1 from the batch of DSP (Apr 2018)

Mentor:

Date: April 1, 2018

Place – Mumbai

Content

Introduction

The history of developing credit-scoring models goes as far back as the history of borrowing and repaying. It reflects the desire to issue an appropriate rate of interest for undertaking the risk of giving away one's own money.

A credit-scoring model is a tool that is typically used in the decision-making process of accepting or rejecting a loan. A credit scoring model is the result of a statistical model which, based on information about the borrower (e.g. monthly-income, number of previous loans, etc.), allows one to distinguish between "good" and "bad" loans and give an estimate of the probability of default.

With the advent of the modern statistics era in the 20th century appropriate techniques have been developed to assess the likelihood of someone's default on the payment, *given* the resemblance of his/her characteristics to those who have already defaulted in the past.

In this document we outline one important application of advanced analytics. We showcase a solution to a common business problem in banking, namely assessing the likelihood of a client's default.

Objective

To build a model to predict default in the future based on the data that is available during loan application, which will help the company in deciding whether or not to pass the loan.

Importing the Dataset & the necessary libraries

Importing the needed libraries

```
import pandas as pd
from datetime import datetime
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
plt.rc("font", size=14)
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import train_test_split
import seaborn as sns
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,roc_curve,auc
```

Importing the data set

```
In [16]: df = pd.read_csv(r'F:\BY LALIT\XYZCorp_LendingData.txt',header=
0,delimiter='\t')
C:\Users\Admin\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:
2698: DtypeWarning: Columns (17,45,53) have mixed types. Specify dtype option
on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Data Quality

Before statistics can take over, there is an important step of preprocessing and checking the quality of the underlying data. This provides a first insight into the patterns inside the data, but also an insight on the trustworthiness of the data itself.
The investigation in this phase includes the following aspects:

What is the proportion of defaults in the data?
In order for the model to be able to make accurate forecasts it needs to see enough examples of what constitutes a default. For this reason it is important that there is a sufficiently large number of defaults in the data.

What is the frequency of values in each variable in the data?
This question provides valuable insight into the importance of each of the variables. The data can contain numerical variables or categorical ones. For some of the variables we may notice that they are dominated by one category, which will render the remaining categories hard to highlight in the model.

What is the proportion of outliers in the data?
Outliers can play an important role in the model's forecasting behavior. Although outliers represent events that occur with a very small probability but can create a high impact. That aside, outliers can be easily detected by the use of boxplots.

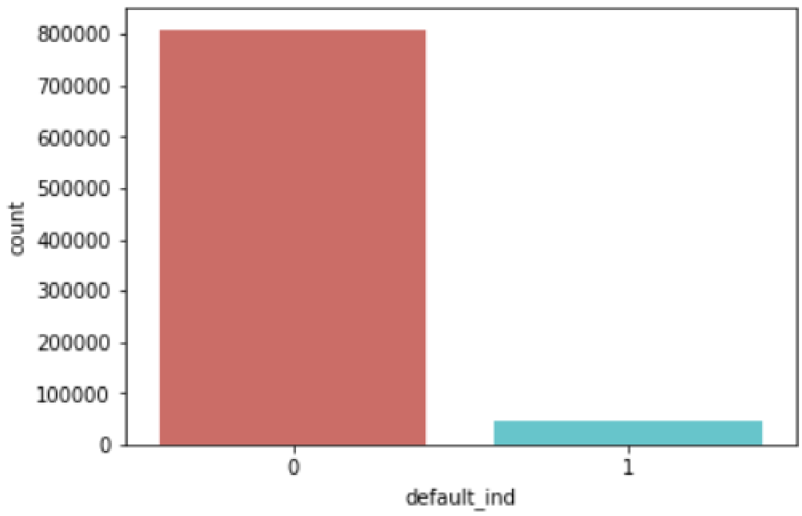
How many missing values are there and what is the reason?
Values can be missing for various reasons, which range from missing due to nonresponse, due to drop out of the clients, or due to censoring of the answers, or simply missing at random. Missing values pose the following dilemma: On one hand they refer to incomplete instances of data and therefore treatment or imputation may not reflect the exact state of affairs. However, avoiding handling missing values and simply ignoring them may lead to loss of valuable information.

What does the Data Say?

The data given consist of 8,55,969 observations, which has 73 variables focusing on the quality and quantity of the various attributes of the Individual. Most of the variables are exactly the type of information that a typical bank would want to know about a potential borrower (e.g. -The self-reported annual income provided by the borrower during registration, Interest Rate on the loan, Current status of the loan etc).

Target Variable

```
sns.countplot(x='default_ind',data=mydata4, palette='hls')
plt.show()
```



The objective of this project is to predict if an individual is going to default on his loan or not.

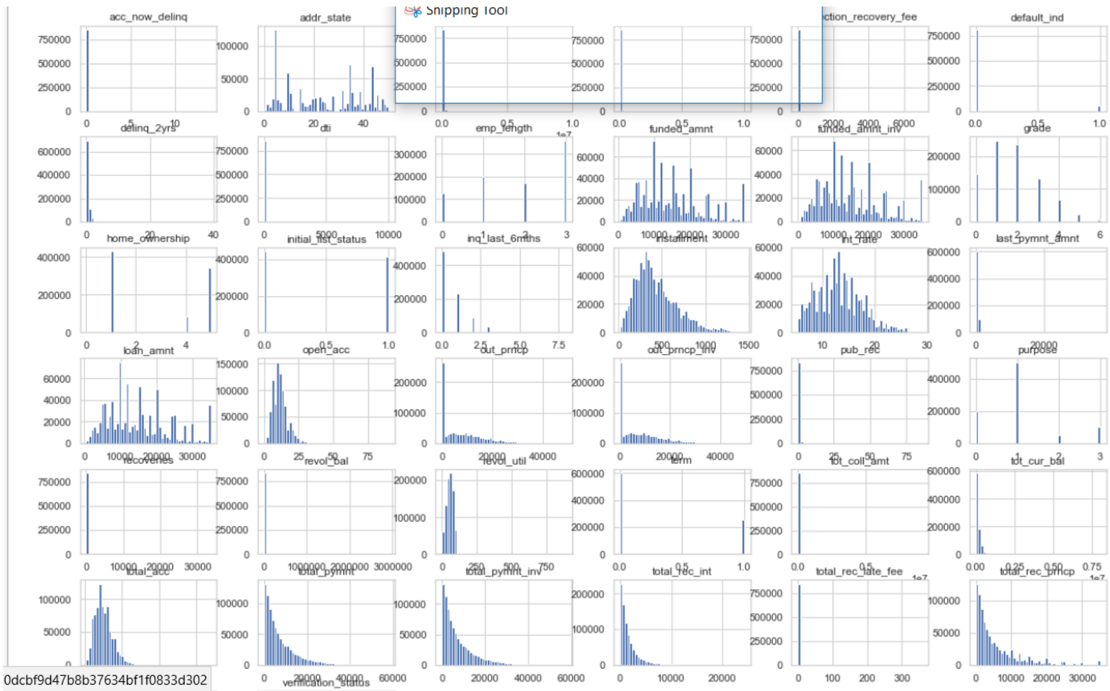
Understanding the Data

```
In [9]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 855969 entries, 0 to 855968
Data columns (total 73 columns):
id                            855969 non-null int64
member_id                     855969 non-null int64
loan_amnt                     855969 non-null float64
funded_amnt                   855969 non-null float64
funded_amnt_inv               855969 non-null float64
term                          855969 non-null object
int_rate                      855969 non-null float64
installment                   855969 non-null float64
grade                         855969 non-null object
sub_grade                     855969 non-null object
emp_title                     806530 non-null object
emp_length                    855969 non-null object
home_ownership                855969 non-null object
annual_inc                    855969 non-null float64
verification_status           855969 non-null object
issue_d                       855969 non-null object
pymnt_plan                    855969 non-null object
desc                          121813 non-null object
purpose                       855969 non-null object
title                         855937 non-null object
zip_code                      855969 non-null object
addr_state                    855969 non-null object
dti                           855969 non-null float64
delinq_2yrs                   855969 non-null float64
```

This code helps us to identify among the given 73 variables there are 21 categorical & 52 Numerical variables.

Structure and the Summary of the data

```
In [12]: df.describe()
Out[12]:
                 id     member_id      loan_amnt    funded_amnt  \
count  8.559690e+05  8.559690e+05  855969.000000  855969.000000
mean   3.224073e+07  3.476269e+07   14745.571335   14732.378305
std    2.271969e+07  2.399418e+07    8425.340005    8419.471653
min    5.473400e+04  7.069900e+04     500.000000     500.000000
25%    9.067986e+06  1.079273e+07    8000.000000    8000.000000
50%    3.431355e+07  3.697532e+07   13000.000000   13000.000000
75%    5.446311e+07  5.803559e+07   20000.000000   20000.000000
max    6.861687e+07  7.351969e+07   35000.000000   35000.000000

       funded_amnt_inv       int_rate    installment    annual_inc  \
count    855969.000000  855969.000000  855969.000000  8.559690e+05
mean      14700.061226      13.192320     436.238072  7.507119e+04
std        8425.805478       4.368365     243.726876  6.426447e+04
min           0.000000       5.320000      15.690000  0.000000e+00
25%        8000.000000       9.990000     260.550000  4.500000e+04
50%       13000.000000      12.990000     382.550000  6.500000e+04
75%       20000.000000      15.990000     571.560000  9.000000e+04
max       35000.000000      28.990000    1445.460000  9.500000e+06

                 dti    delinq_2yrs  inq_last_6mths  mths_since_last_delinq  \
count  855969.000000  855969.000000   855969.000000           416157.000000
mean       18.122165       0.311621        0.680915               34.149943
std        17.423629       0.857189        0.964033               21.868500
min         0.000000       0.000000        0.000000                0.000000
25%        11.880000       0.000000        0.000000               15.000000
50%        17.610000       0.000000        0.000000               31.000000
75%        23.900000       0.000000        1.000000               50.000000
max      9999.000000      39.000000        8.000000              188.000000
```



Checking if there exist Missing Values

```
In [17]: df.isnull().sum()
Out[17]:
id                              0
member_id                       0
loan_amnt                       0
funded_amnt                     0
funded_amnt_inv                 0
term                            0
int_rate                        0
installment                     0
grade                           0
sub_grade                       0
emp_title                   49439
emp_length                      0
home_ownership                  0
annual_inc                      0
verification_status             0
issue_d                         0
pymnt_plan                      0
desc                       734156
purpose                         0
title                          32
zip_code                        0
addr_state                      0
dti                             0
delinq_2yrs                     0
earliest_cr_line                0
inq_last_6mths                  0
mths_since_last_delinq     439812
mths_since_last_record     724785
```

As we can infer that there are many missing values.

So we will have to treat them accordingly.

Dropping Variables as most values are Missing

LoanStatNew	Description	NULL VA
annual_inc_joint	The combined self-reported annual income provided by the co-borrowers during registration	855527
desc	Loan description provided by the borrower	734157
dti_joint	A ratio calculated using the co-borrowers' total monthly payments on the total debt obligations, excluding mortgag	855529
mths_since_last_major_dero	Months since most recent 90-day or worse rating	642830
mths_since_last_record	The number of months since the last public record.	724785
verified_status_joint	Indicates if the co-borrowers' joint income was verified by XYZ corp., not verified, or if the income source was v	855527
open_acc_6m	Number of open trades in last 6 months	842681
open_il_6m	Number of currently active installment trades	842681
open_il_12m	Number of installment accounts opened in past 12 months	842681
open_il_24m	Number of installment accounts opened in past 24 months	842681
mths_since_rcnt_il	Months since most recent installment accounts opened	842681
total_bal_il	Total current balance of all installment accounts	842681
il_util	Ratio of total current balance to high credit/credit limit on all install acct	842681
open_rv_12m	Number of revolving trades opened in past 12 months	842681
open_rv_24m	Number of revolving trades opened in past 24 months	842681
max_bal_bc	Maximum current balance owed on all revolving accounts	842681
all_util	Balance to credit limit on all trades	842681
inq_fi	Number of personal finance inquiries	842681
total_cu_tl	Number of finance trades	842681
inq_last_12m	Number of credit inquiries in past 12 months	842681

Dropping these variables because they are insignificant as the information is missing for most of the observations.

```
df1=df[['id','member_id','loan_amnt','funded_amnt','funded_amnt_inv','term','int_rate','installment','grade','sub_grade',
        'emp_title','emp_length','home_ownership','annual_inc','verification_status','issue_d','pymnt_plan','desc','purpose',
        'title','zip_code','addr_state','dti','delinq_2yrs','earliest_cr_line','inq_last_6mths','mths_since_last_delinq',
        'mths_since_last_record','open_acc','pub_rec','revol_bal','revol_util','total_acc','initial_list_status','out_prncp',
        'out_prncp_inv','total_pymnt','total_pymnt_inv','total_rec_prncp','total_rec_int','total_rec_late_fee','recoveries',
        'collection_recovery_fee','last_pymnt_d','last_pymnt_amnt','next_pymnt_d','last_credit_pull_d',
        'collections_12_mths_ex_med','mths_since_last_major_derog','policy_code','application_type','annual_inc_joint',
        'dti_joint','verification_status_joint','acc_now_delinq','tot_coll_amt','tot_cur_bal','open_acc_6m','open_il_6m',
        'open_il_12m','open_il_24m','mths_since_rcnt_il','total_bal_il','il_util','open_rv_12m','open_rv_24m','max_bal_bc',
        'all_util','total_rev_hi_lim','inq_fi','total_cu_tl','inq_last_12m','default_ind']]


df2=df[['loan_amnt','funded_amnt','funded_amnt_inv','term','int_rate','installment','grade','emp_length','home_ownership',
        'annual_inc','verification_status','issue_d','purpose','addr_state','dti','delinq_2yrs','earliest_cr_line',
        'inq_last_6mths','open_acc','pub_rec','revol_bal','revol_util','total_acc','initial_list_status','out_prncp',
        'out_prncp_inv','total_pymnt','total_pymnt_inv','total_rec_prncp','total_rec_int','total_rec_late_fee','recoveries',
        'collection_recovery_fee','last_pymnt_d','last_pymnt_amnt','last_credit_pull_d','application_type','acc_now_delinq',
        'tot_coll_amt','tot_cur_bal','total_rev_hi_lim','default_ind']]
```

Variable with only a few missing values

LoanStatNew	Description	NULL VA
collections_12_mths_ex_med	Number of collections in 12 months excluding medical collections	56
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.	43061
emp_title	The job title supplied by the Borrower when applying for the loan.	49443
last_credit_pull_d	The most recent month XYZ corp. pulled credit for this loan	50
last_pymnt_d	Last month payment was received	8862
revol_util	Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.	446
title	The loan title provided by the borrower	33
total_rev_hi_lim	Total revolving high credit/credit limit	67313
tot_coll_amt	Total collection amounts ever owed	67313
tot_cur_bal	Total current balance of all accounts	67313

Variables where missing values should be treated before proceeding further.

Emplength - 43,061(missing Values)

To impute the missing values we tried to predict them using regression.

```
#separating into two dataframes based on null and non null emp_length
regress3_NON=regress2[regress2.emp_length.notnull()]
regress3_NON.emp_length.isnull().sum()colname1=['emp_length']
```

```
0
```

```
regress3_NULL=regress2[regress2.emp_length.isnull()]
regress3_NULL.emp_length.isnull().sum()
```

```
43061
```

Separated the data set into 2 parts.

One does not have any missing values and the other only has missing values in order to predict using regression model.

```
# Linear regression with sklearn
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()  #alt1
regressor.fit(X_train, Y_train) #alt1
#regr = linear_model.LinearRegression() #alt2
#regr.fit(X_train, Y_train)            #alt2
#regr.score(X_train,Y_train) ###RETURNS R squared value #alt2
#print('Intercept: \n', regr.intercept_)
#print('Coefficients: \n', regr.coef_)
#print("Done! We now have a working Linear Regression model.")
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
#Calculate R squared: #ALT1

y_pred = regressor.predict(X_test)
print('Liner Regression R squared: %.4f' % regressor.score(X_test, Y_test))
#So, in our model, 0.74% of the variability in Y can be explained using X. This is not that exciting.
```

```
   Liner Regression R squared: 0.0074
```

The model performed poorly and indicated that it is unable to explain the variability of the response data around its mean.

Imputing the missing values

```
#replacing null 8k values with mode..tyhere are 4l repeating values of mode

df2["last_pymnt_d"].fillna("Jan-2016", inplace = True)
#mydata3.last_pymnt_d.isnull().sum()

df2["last_credit_pull_d"].fillna("Jan-2016", inplace = True)
#mydata3.last_credit_pull_d.isnull().sum()

#replacing with mean values

df2["total_rev_hi_lim"].fillna(df2.total_rev_hi_lim.mean(), inplace = True)
#mydata3.total_rev_hi_lim.isnull().sum()

df2["revol_util"].fillna(df2.revol_util.mean(), inplace = True)
#mydata3.revol_util.isnull().sum()

df2["tot_coll_amt"].fillna(df2.tot_coll_amt.mean(), inplace = True)
#mydata3.tot_coll_amt.isnull().sum()

df2["tot_cur_bal"].fillna(df2.tot_cur_bal.mean(), inplace = True)
#mydata3.tot_cur_bal.isnull().sum()

df2["emp_length"].fillna("10+ years", inplace = True)
df2.emp_length.isnull().sum()
```

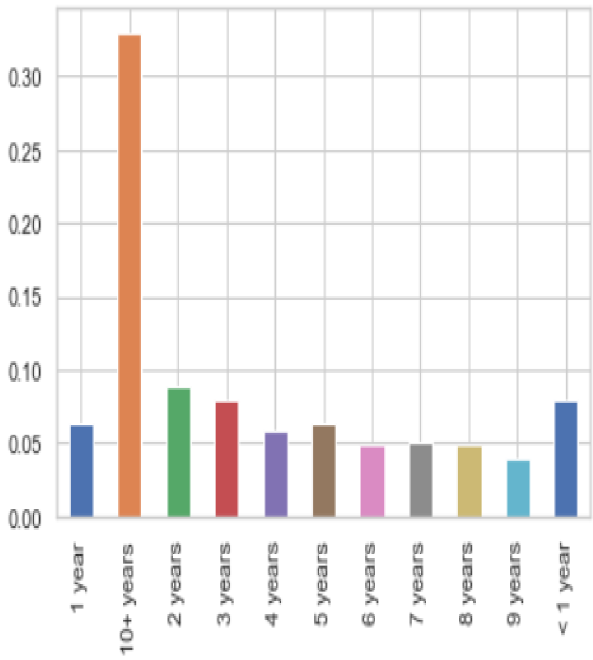
Imputing using Mode

Imputing using mean

```
C:\Users\GLADY\Anaconda3\lib\site-packages\pandas\core\generic.py:5434: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  self._update_inplace(new_data)

0
```

```
# have a look at the distribution of the employment length
(df['emp_length'].value_counts().sort_index()/len(df)).plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e1665af860>
```



As the missing values are finally treated we can now actually plot graphs to get a better understanding of the variables.

Label Encoding

```
##CREATING LIST OF ALL CATEGORICAL VARIABLES.

colname=['emp_length','addr_state','application_type','grade', 'home_ownership','initial_list_status','purpose','term','verificat

#colname
```

```
# for preprocessing the data

from sklearn import preprocessing

le={}##creating empty dictionary
##LABEL ENCODING IS USED TO CONVERT CATEGORICAL TO NUMERICAL DATA
#creating dictionary in first for loop && transforming in second
for x in colname:
    le[x]=preprocessing.LabelEncoder()
for x in colname:
    df2[x]=le[x].fit_transform(df2[x])##firtransform to replace original values
```

```
C:\Users\GLADY\Anaconda3\lib\site-packages\ipykernel_launcher.py:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  # This is added back by InteractiveShellApp.init_path()
```

Converting the categories into numbers.

Reducing Levels

```
f2['emp_length']=df2.emp_length.replace(['< 1 year','1 year','2 years','3 years','4 years','5 years','6 years','7 years','8 years'
                                        ,'9 years','10+ years'],
                                       ['low','low','low','medium','medium','medium','high','high','high','v.high','v.high'])
```

```
C:\Users\GLADY\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

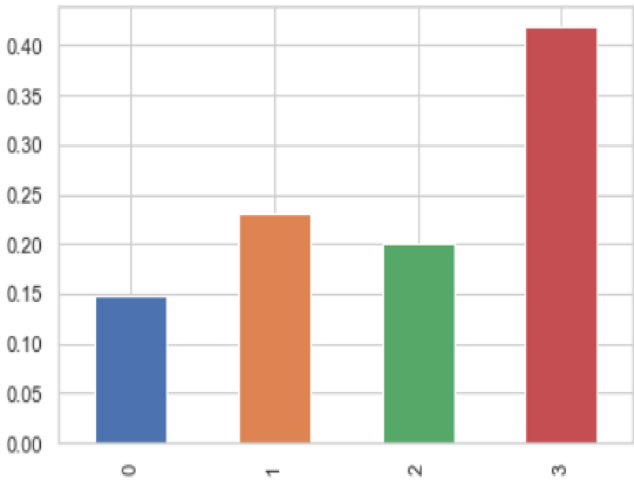
```
df2['purpose']=df2.purpose.replace(['debt_consolidation','credit_card','home_improvement','other','major_purchase','small_busines
                                    'moving','vacation','house','wedding','renewable_energy','educational'],
                                   ['debt_consolidation1','credit_card','home_improvement','others','others','others','o
```

```
C:\Users\GLADY\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

Here we reduce the levels by combining them.

```python
# have a look at the distribution of the employment length after reducing levels
(df2['emp_length'].value_counts().sort_index()/len(df)).plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e1742afbe0>
```



Converting date objects to datetime format

```
##converting date objects in dataframe to datetime format
df3['issue_d']=pd.to_datetime(df3['issue_d'])
df3['earliest_cr_line']=pd.to_datetime(df3['earliest_cr_line'])
df3['last_pymnt_d']=pd.to_datetime(df3['last_pymnt_d'])
df3['last_credit_pull_d']=pd.to_datetime(df3['last_credit_pull_d'])
```

```
C:\Users\GLADY\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
C:\Users\GLADY\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
```

Splitting the Data-Set into Test & Train

```
#splitting the dataframe into 2 different frames based on date.
traindf=df3[df3['issue_d']<='2015-05-01']
testdf=df3[df3['issue_d']>'2015-05-01']
```

As our data is now clean and can hence be used for building a model in order to make predictions.

Model Development

Default Definition: -

Before the analysis begins it is important to clearly state out what defines a default. Different choices will have an impact on what the model predicts.

Classification: -

The aim of the model is to perform a classification: To distinguish the "good" applicants from the "bad" ones.

Reject inference: -

Apart from this, there is an additional difficulty in the development of a credit scorecard for which there is no solution. For clients that were declined in the past the bank cannot possibly know what would have happened if they would have been accepted. In other words, the data that the bank has refers only to the customers that were initially accepted for a loan. This means, that the data is already biased towards a lower default-rate. This implies that the model is not truly representative for a through-the-door client. This problem is often termed "reject inference".

Logistic Regression

Performing binary classification to “good” and “bad” via a logistic function.
For each of the existing data points it is known whether the client has gone into default or not.
Logistic model uses a logistic function to model a binary dependent variable.

```
#scaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

sc = MinMaxScaler()
clf = LogisticRegression(penalty='l1', C=0.1)

pipe_lr = Pipeline([('scaler', sc), ('clf', clf)])
```

```
pipe_lr.fit(X_traindf, Y_traindf)
```

```
Pipeline(memory=None,
     steps=[('scaler', MinMaxScaler(copy=True, feature_range=(0, 1))), ('clf', LogisticRegression(C=0.1, class_weight=None, dua
l=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False))])
```

Results and Interpretations

1.Compute precision, recall, F-measure and support
2.The precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier to not label a sample as positive if it is negative.
3.The recall is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.
4.The F-beta score can be interpreted as a weighted harmonic mean of the precision and recall, where an F-beta score reaches its best value at 1 and worst score at 0.
5.The F-beta score weights the recall more than the precision by a factor of beta. beta = 1.0 means recall and precision are equally important.
6.The support is the number of occurrences of each class in y_test.

```
#Predicting the test set results and creating confusion matrix

Y_preddf = pipe_lr.predict(X_testdf)
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(Y_testdf, Y_preddf)
print(confusion_matrix)
#Y_pred
```

```
[[256627     53]
 [    64    247]]
```

Confusion Matrix 1

```
#Accuracy
pipe_lr.score(X_testdf, Y_testdf)
#print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(classifier.score(X_testdf, Y_testdf)))
```

```
0.9995447311384446
```

Accuracy score 1

```
from sklearn.metrics import classification_report
print(classification_report(Y_testdf, Y_preddf))

#Classifier visualization playground
```

```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00    256680
          1       0.82      0.79      0.81       311

avg / total       1.00      1.00      1.00    256991
```

Classification Report 1

Model Accuracy=0.999544

Gradient Boosting model

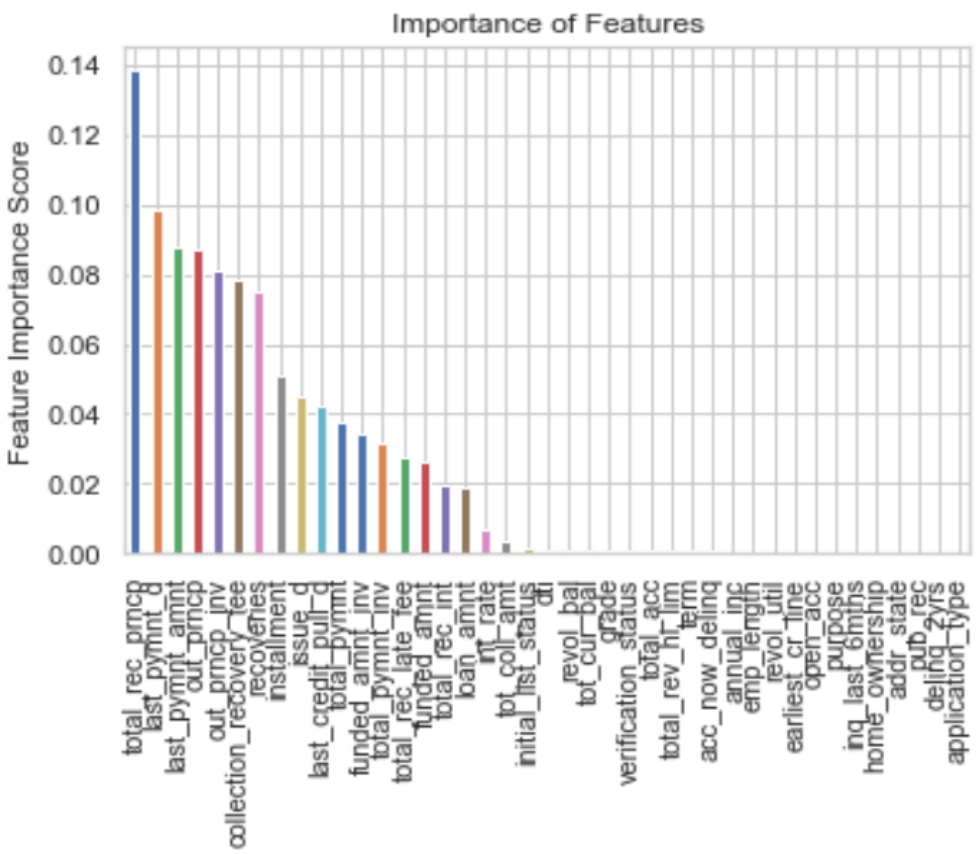
Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

1. GBM Model Building

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report
from sklearn.grid_search import GridSearchCV


baseline = GradientBoostingClassifier(learning_rate=0.1, n_estimators=100,max_depth=3, min_samples_split=2, min_samples_leaf=1, s
baseline.fit(X_traindf,Y_traindf)
predictors=list(X_traindf)
feat_imp = pd.Series(baseline.feature_importances_, predictors).sort_values(ascending=False)
feat_imp.plot(kind='bar', title='Importance of Features')
plt.ylabel('Feature Importance Score')
print('Accuracy of the GBM on test set: {:.3f}'.format(baseline.score(X_testdf, Y_testdf)))
pred=baseline.predict(X_testdf)
print(classification_report(Y_testdf, pred))
plt.savefig('map1.png')
```

```
Accuracy of the GBM on test set: 0.967
             precision    recall  f1-score   support

          0       1.00      0.97      0.98    256680
          1       0.04      0.99      0.07       311

avg / total       1.00      0.97      0.98    256991
```

Confusion Matrix 2



Feature Importance 1

The plot displays the importance of the feature: The number of words in capital and bang seem to have 4 the highest predictive power.

With this first model, we obtain a rate of 0.04 of true positives (positive meaning spam) and 1.00 true negatives and an accuracy of 0.967.

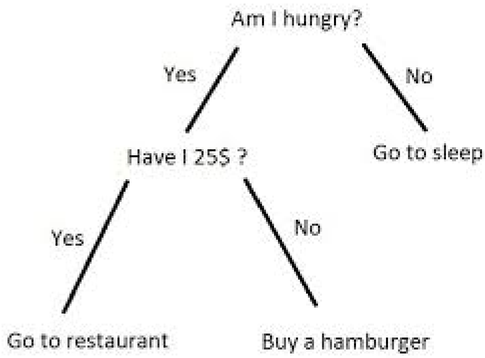
2. Tuning the model

```
learning_rates = [0.05,0.1,0.25,0.5,0.75,1]
for learning_rate in learning_rates:
    gb=GradientBoostingClassifier(n_estimators=20,learning_rate=learning_rate, max_features=2,max_depth=2, random_state=0)
    gb.fit(X_train,Y_train)
    print("Learning rate: ",learning_rate)
    print("accuracy score training:{0:3f}".format(gb.score(X_train,Y_train)))
    print("accuracy score testing:{0:3f}".format(gb.score(X_test,Y_test)))
    print()
```

```
Learning rate:  0.05
accuracy score training:0.923027
accuracy score testing:0.998066

Learning rate:  0.1
accuracy score training:0.932371
accuracy score testing:0.968263

Learning rate:  0.25
accuracy score training:0.962561
accuracy score testing:0.927962

Learning rate:  0.5
accuracy score training:0.969667
accuracy score testing:0.906837

Learning rate:  0.75
accuracy score training:0.971274
accuracy score testing:0.901825

Learning rate:  1
accuracy score training:0.974276
accuracy score testing:0.883844
```

Decision Tree

Decision-tree Algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.



Building the model

```
In [131]: treemodel=tree.DecisionTreeClassifier()
     ...: treemodel.fit(X_train, Y_train)
Out[131]:
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')

In [132]:
```

Model Accuracy

```
In [132]: treemodel.score(X_test, Y_test)
Out[132]: 0.8963271087314342
```


Confusion Matrix

```
In [133]: y_pred=treemodel.predict(X_test)
     ...:
     ...: cfm=confusion_matrix(Y_test,y_pred)
     ...: cfm
Out[133]:
array([[230038,  26642],
       [     1,    310]], dtype=int64)
```

Classification Report

```
In [135]: print(classification_report(Y_test,Y_pred))
             precision    recall  f1-score   support

        0.0       1.00      1.00      1.00    256680
        1.0       0.98      0.57      0.72       311

avg / total       1.00      1.00      1.00    256991
```

Random Forests

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean prediction of the individual trees.

Model Implementation

```
In [131]: treemodel=tree.DecisionTreeClassifier()
     ...: treemodel.fit(X_train, Y_train)
Out[131]:
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')

In [132]:
```

Accuracy of the Model

```
In [140]: model.score(X_test, Y_test)
Out[140]: 0.9497842336891175
```


Confusion matrix & Classification Report

```
In [142]: y_pred=model.predict(X_test)
     ...:
     ...: cfm=confusion_matrix(Y_test,y_pred)
     ...: print(cfm)
     ...: print(classification_report(Y_test,y_pred))
[[243776  12904]
 [     1    310]]
             precision    recall  f1-score   support

        0.0       1.00      0.95      0.97    256680
        1.0       0.02      1.00      0.05       311

avg / total       1.00      0.95      0.97    256991
```

Feature Ranking

```
std = np.std([tree.feature_importances_ for tree in rf1.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(X_traindf.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

# Plot the feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(range(X_traindf.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X_traindf.shape[1]), indices)
plt.xlim([-1, X_traindf.shape[1]])
plt.show()
plt.savefig('map34.png')
fig=plt.figure(figsize=(18, 16), dpi= 80, facecolor='w', edgecolor='k')
```

```
Feature ranking:
1. feature 32 (0.300684)
2. feature 31 (0.239183)
3. feature 28 (0.089988)
4. feature 33 (0.074961)
5. feature 24 (0.064142)
6. feature 25 (0.063236)
7. feature 27 (0.037696)
8. feature 34 (0.036759)
9. feature 35 (0.016935)
10. feature 26 (0.016089)
11. feature 11 (0.011299)
```

Conclusion

Key Findings:
Some of the most significant variables from our findings are as below: -
total_rec_prncp
last_pymnt_amnt
last_pymnt_d
Outstanding Principal
Recoveries
Interest Rate

Sr. No.	Model Name	Accuracy
1	Logistic Regression	0.995
2	GBM	0.967
3	Decision Tree	0.896
4	Random Forest	0.949