CREDIT RISK ANALYSIS.

Smitesh Jadhav

PROBLEM STATEMENT

- Based on the credit history of the customers given, check whether they are capable enough to pay the loan or not. Predict whether the customer will default or not . To manage credit risk by using the past data and deciding whom to give the loan to in the future.

CREDIT RISK ANALYSIS

- Credit risk or credit default risk associated with a financial transaction is simply the expected loss of that transaction.
- It can be defined as follows:

Credit Risk = Default Probability x Exposure x Loss Rate

- WHERE:

Default Probability is the probability of a debtor reneging on his debt payments.

Exposure is the total amount the lender is supposed to get paid. In most cases, it is simply the amount borrowed by the debtor plus interest payments.

Loss Rate = 1 – Recovery Rate, where Recovery Rate is the proportion of the total amount that can be recovered if the debtor defaults. Credit risk analysts analyze each of the determinants of credit risk and try to minimize the aggregate risk faced by an organization.

CREDIT RISK ANALYSIS

- Credit risk analysis can be thought of as an extension of the credit allocation process.
- After an individual or business applies to a bank or financial institution for a loan, the lending institution analyzes the potential benefits and costs associated with the loan.
- Credit risk or credit default risk is a type of risk faced by lenders.
- Credit risk arises because a debtor can always renege on their debt payments.

DIAGRAMATIC REPRESENTATION

DATASET DISCRIPTION

1. Details about people who applied for loan from June 2007 to December 2015.

2. 73 Variables and 855969 Observations.

3. Dependent Variable: default_ind:- 0 as Not Defaulter

   1 as Defaulter

UNDERSTAND THE DATA

jupyter Credit Risk Analysis Last Checkpoint: an hour ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python

Run | Code

In [ ]:
```
#IMPORTING LIBRARIES
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
pd.set_option("display.max_rows", None)
pd.set_option("display.max_columns", None)
```

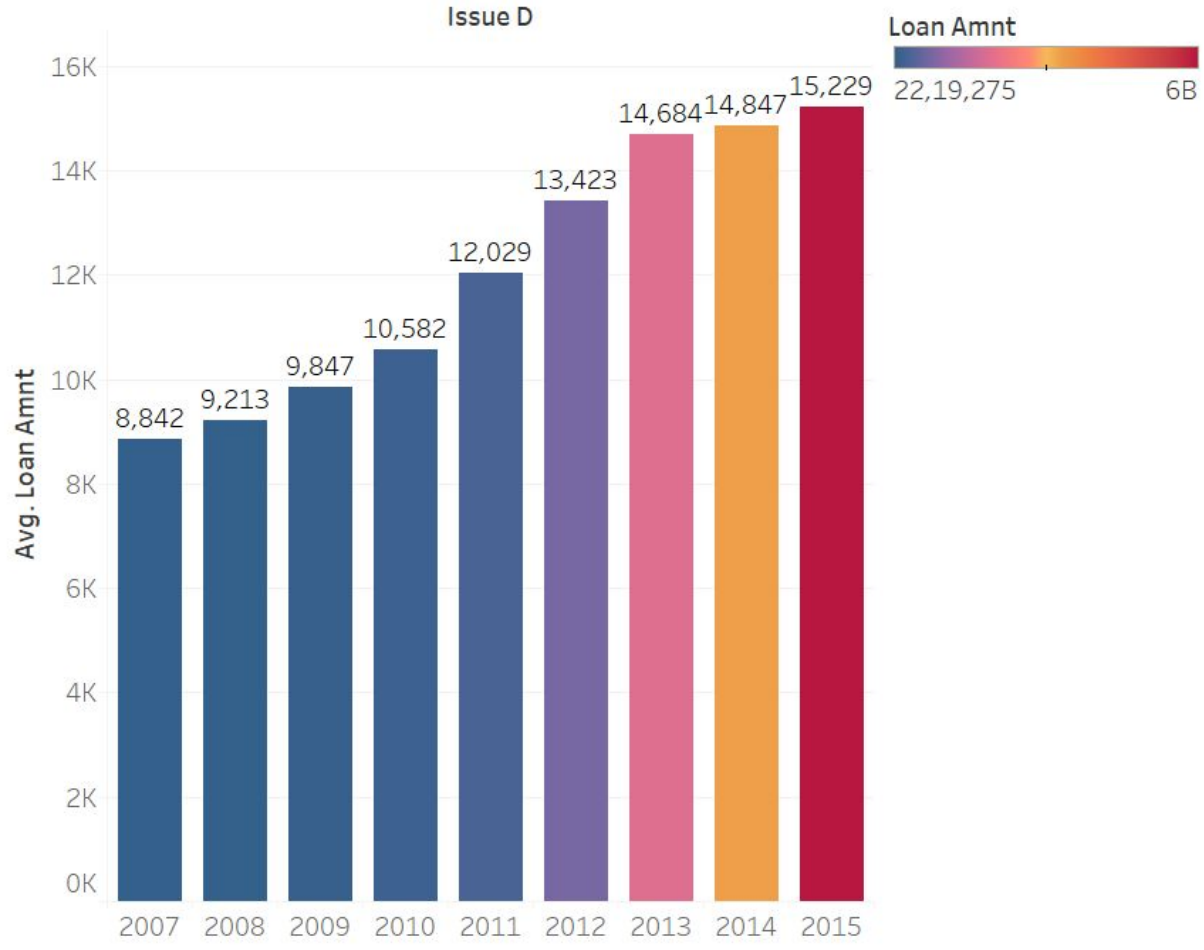
In [3]:
```
#LOADING THE DATA & READING IT
Loan_df=pd.read_csv("LoanDefaulter.txt", header=0, index_col=None, sep=" ", delimiter="\t")
```

In [4]:
```
Loan_df.head()
```

Out[4]:

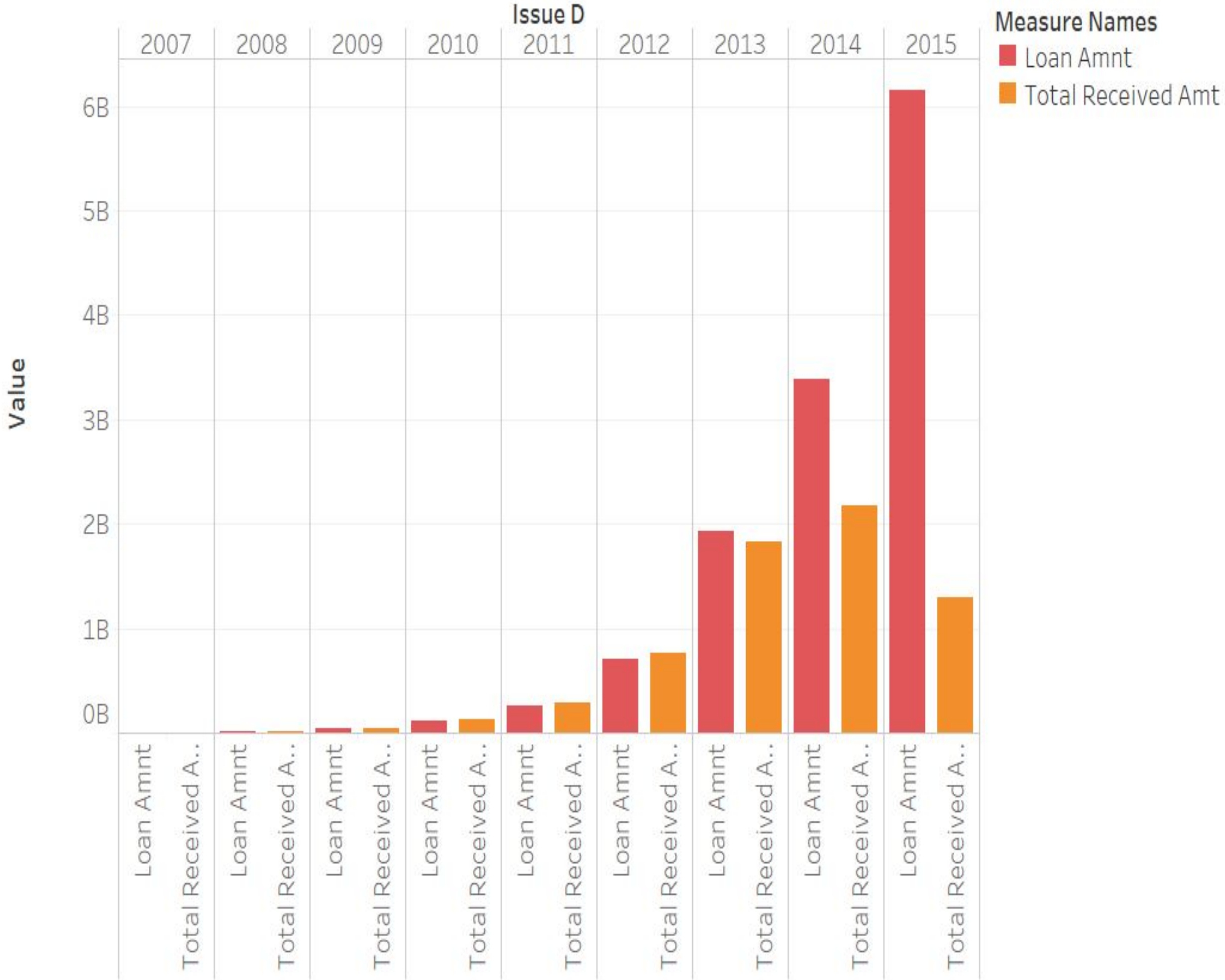
	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ow
0	1077501	1296599	5000.0	5000.0	4975.0	36 months	10.65	162.87	B	B2	NaN	10+ years	
1	1077430	1314167	2500.0	2500.0	2500.0	60 months	15.27	59.83	C	C4	Ryder	< 1 year	
2	1077175	1313524	2400.0	2400.0	2400.0	36 months	15.96	84.33	C	C5	NaN	10+ years	

AVERAGE LOAN AMOUNT GIVEN EVERY YEAR



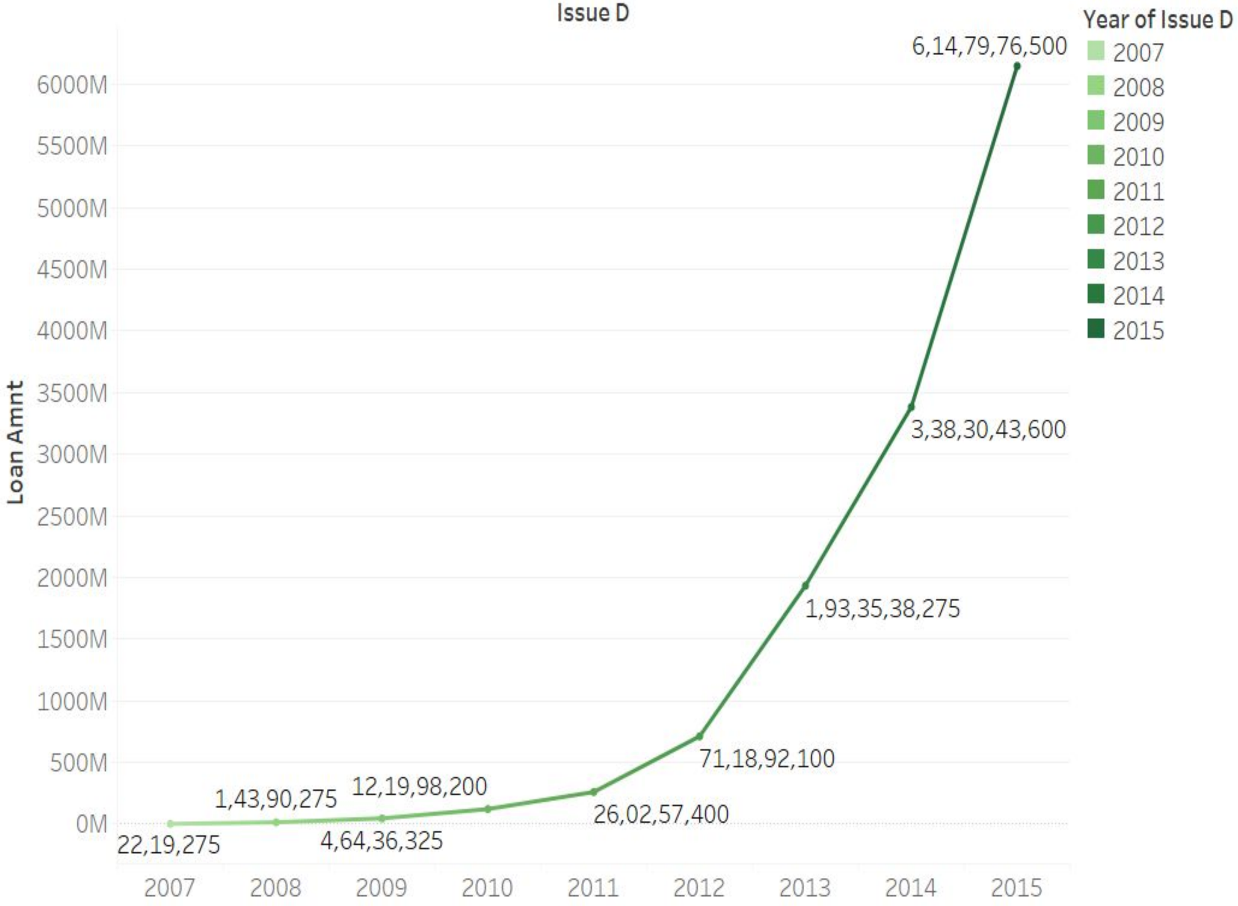
Average of Loan Amnt for each Issue D Year. Color shows sum of Loan Amnt. The marks are labeled by average of Loan Amnt.

LOAN AMOUNT GIVEN VS RECEIVED EVERY YEAR



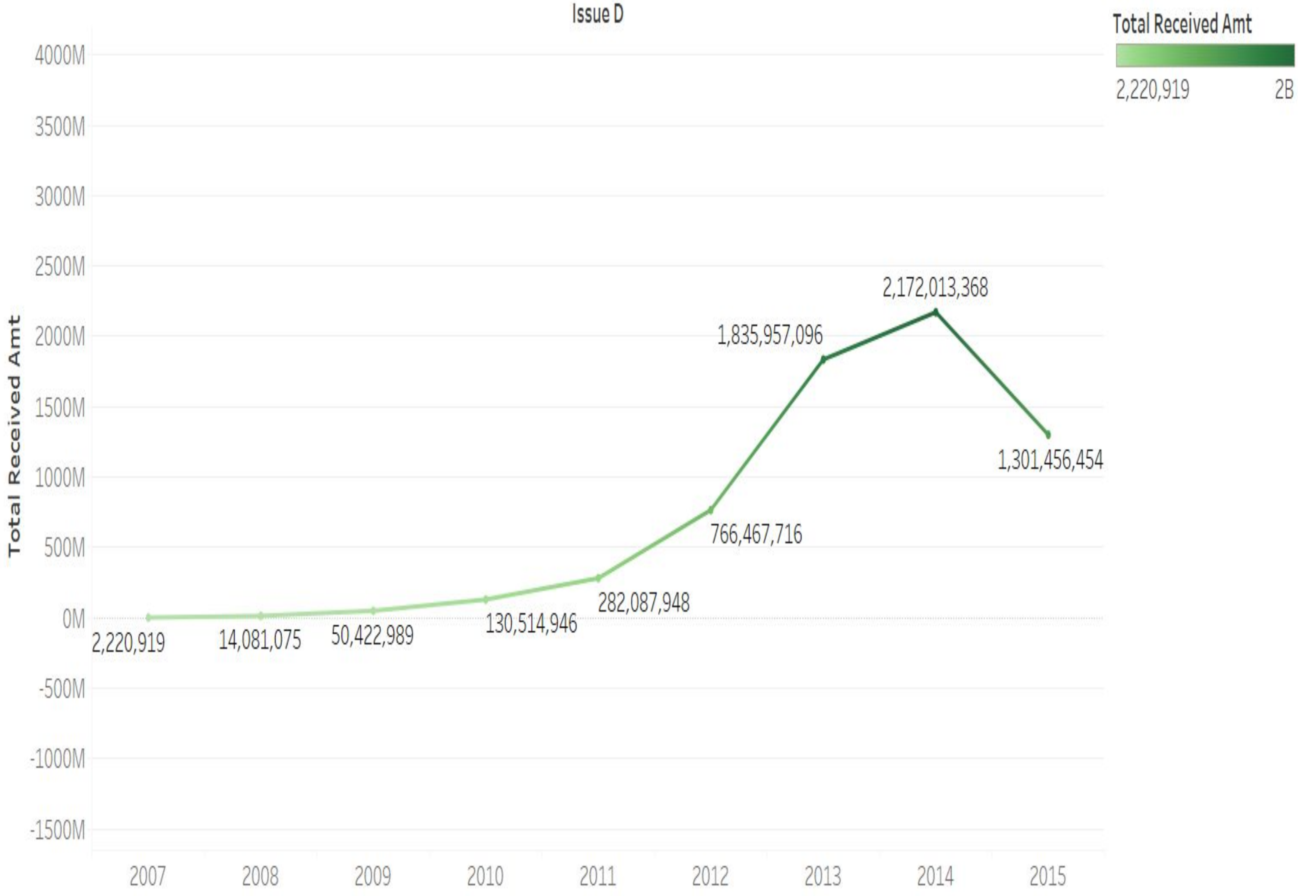
Loan Amnt and Total Received Amt for each Issue D Year. Color shows details about Loan Amnt and Total Received Amt.

TOTAL AMOUNT OF LOAN GIVEN EVERY YEAR

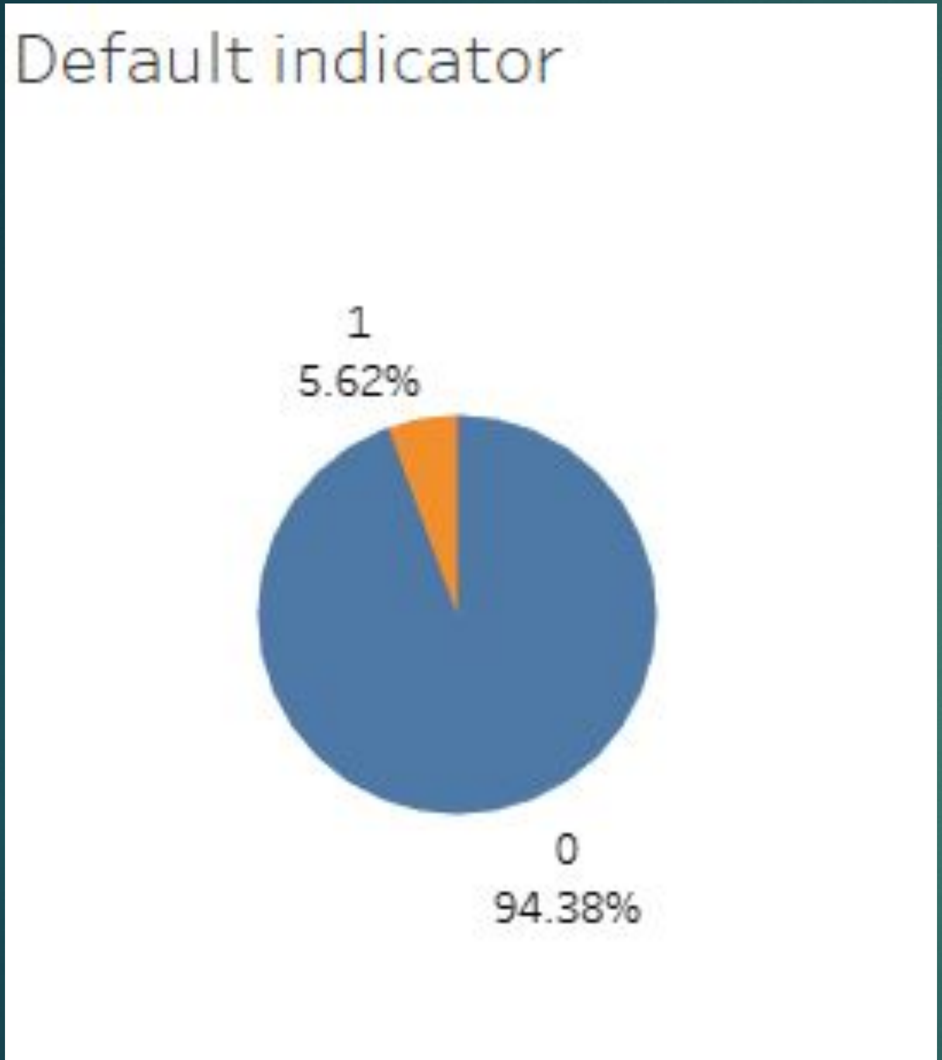


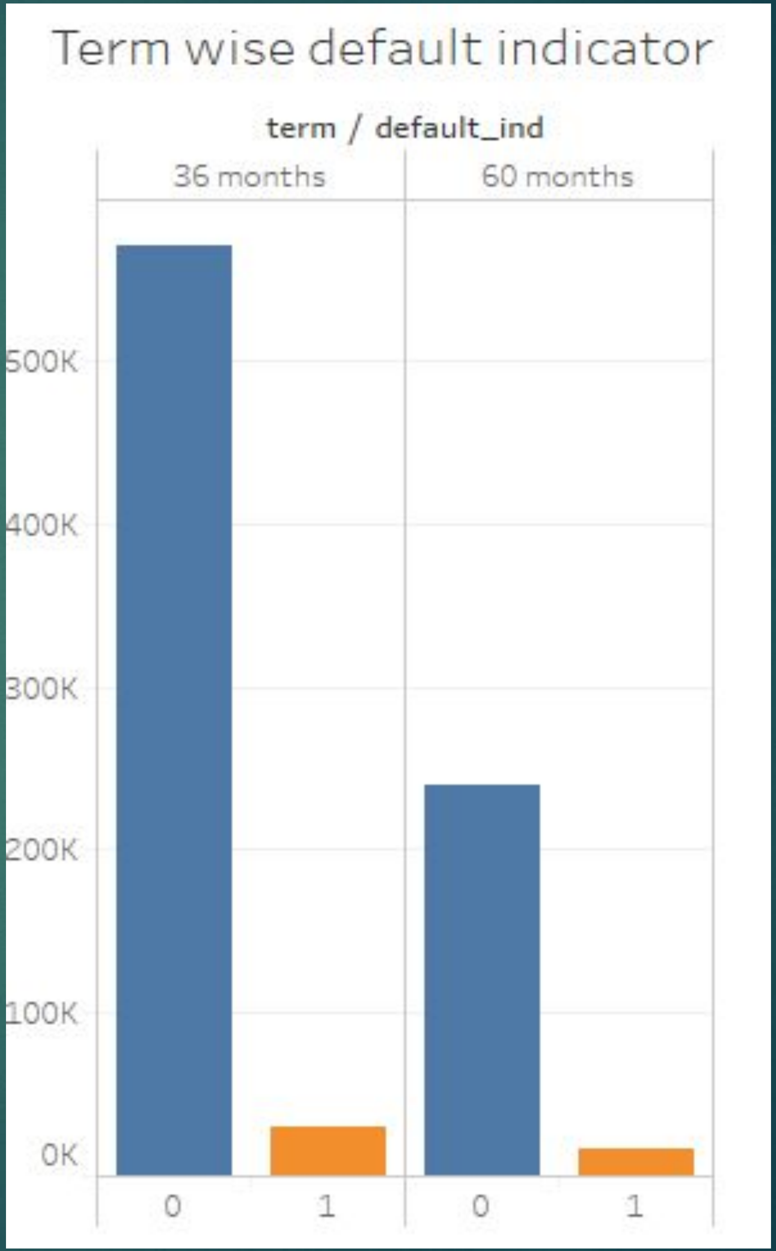
The trend of sum of Loan Amnt for Issue D Year. Color shows details about Issue D Year. The marks are labeled by sum of Loan Amnt.

TOTAL AMOUNT RECOVERED EVERY YEAR



The trend of sum of Total Received Amt for Issue D Year. Color shows sum of Total Received Amt. The marks are labeled by sum of Total Received Amt.


Default indicator
1
5.62%
0
94.38%


Term wise default indicator
term / default_ind
36 months
60 months
500K
400K
300K
200K
100K
0K
0
1
0
1

DATA PREPROCESSING

- Removing variables based on domain knowledge.

```
df=df.drop(["id","member_id","desc","zip_code","addr_state","earliest_cr_line"], axis=1)
df.shape
```

Out[6]: (855969, 67)

In [21]:
```
df=df.drop(["emp_title","title","next_pymnt_d"],axis=1)
df.shape
```

Out[21]: (855969, 44)

In [26]:
```
df=df.drop(["last_pymnt_d","last_credit_pull_d"],axis=1)
df.shape
```

Out[26]: (855969, 42)

LIST OF NULLS

In [9]:
```
#HANDLING THE MISSING VALUES
df.isnull().sum()
```

Out[9]:
```
loan_amnt                          0
funded_amnt                        0
funded_amnt_inv                    0
term                               0
int_rate                           0
installment                        0
grade                              0
sub_grade                          0
emp_length                     43061
home_ownership                     0
annual_inc                         0
verification_status                0
issue_d                            0
pymnt_plan                         0
desc                          734157
purpose                            0
dti                                0
delinq_2yrs                        0
earliest_cr_line                   0
inq_last_6mths                     0
mths_since_last_delinq        439812
mths_since_last_record        724785
open_acc                           0
pub_rec                            0
revol_bal                          0
revol_util                       446
total_acc                          0
initial_list_status                0
out_prncp                          0
out_prncp_inv                      0
total_pymnt                        0
total_pymnt_inv                    0
total_rec_prncp                    0
total_rec_int                      0
total_rec_late_fee                 0
recoveries                         0
collection_recovery_fee            0
last_pymnt_d                    8862
last_pymnt_amnt                    0
```

jupyter LoanDefaulterFinalFile (1) (autosaved)

File Edit View Insert Cell Kernel Widgets Help

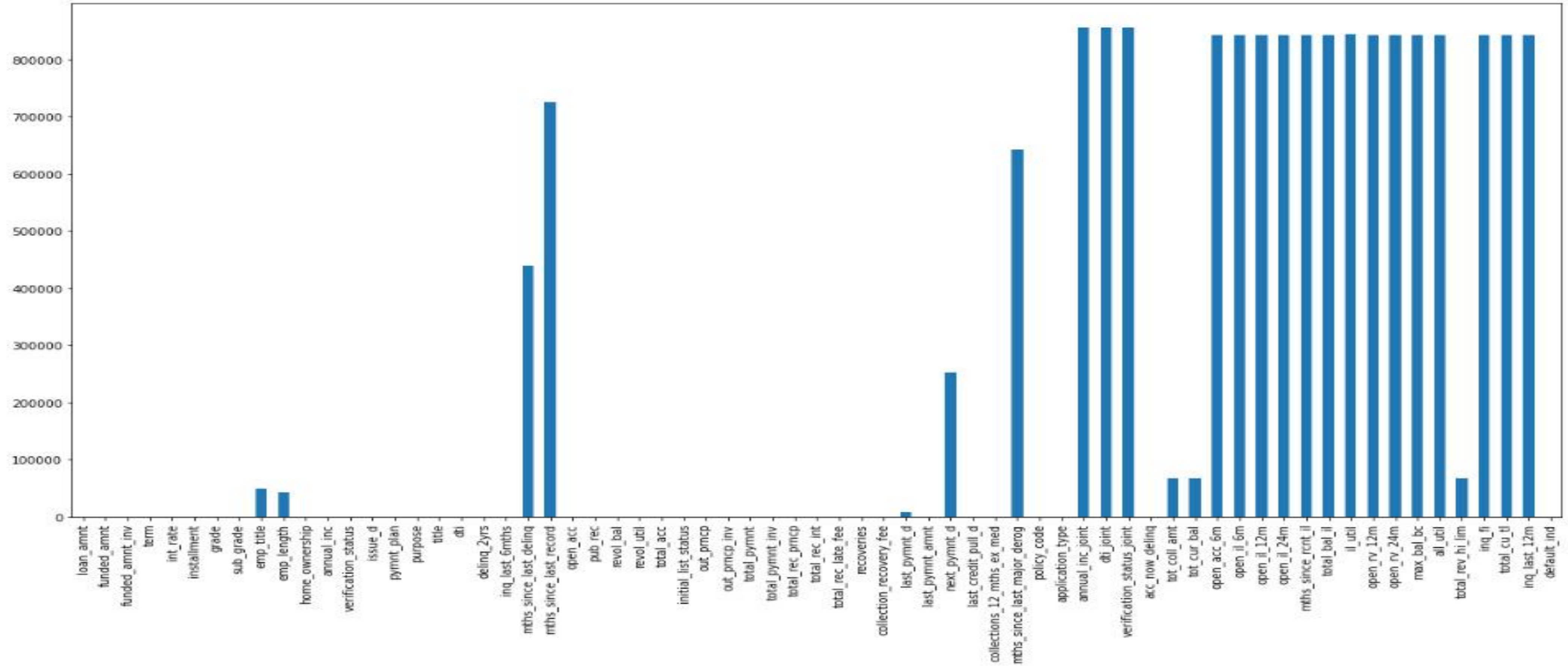
Run Code

```
total_acc                          0
initial_list_status                0
out_prncp                          0
out_prncp_inv                      0
total_pymnt                        0
total_pymnt_inv                    0
total_rec_prncp                    0
total_rec_int                      0
total_rec_late_fee                 0
recoveries                         0
collection_recovery_fee            0
last_pymnt_d                    8862
last_pymnt_amnt                    0
next_pymnt_d                  252971
last_credit_pull_d                50
collections_12_mths_ex_med        56
mths_since_last_major_derog   642830
policy_code                        0
application_type                   0
annual_inc_joint              855527
dti_joint                     855529
verification_status_joint     855527
acc_now_delinq                     0
tot_coll_amt                   67313
tot_cur_bal                    67313
open_acc_6m                   842681
open_il_6m                    842681
open_il_12m                   842681
open_il_24m                   842681
mths_since_rcnt_il            843035
total_bal_il                  842681
il_util                       844360
open_rv_12m                   842681
open_rv_24m                   842681
max_bal_bc                    842681
all_util                      842681
total_rev_hi_lim               67313
inq_fi                        842681
total_cu_tl                   842681
inq_last_12m                  842681
default_ind                        0
dtype: int64
```

ELIMINATING THE VARIABLES THAT HAVE MORE THAN 50% NA'S

In [11]:

```
#PLOTTING A GRAPH TO SEE THE MISSING VALUES IN THE DATA
cols = df.isnull().sum()
plt.figure(figsize=(20,10))
cols.plot(kind='bar')
plt.show()
```



In [12]:
```python
#ELIMINATING THE VARIABLES THAT HAVE MORE THAN 50% NA'S
null_cols=[]
for x in df.columns:
    if df[x].isnull().sum()>=0.5*len(df):
        null_cols.append(x)
print(null_cols)
```

```
['mths_since_last_delinq', 'mths_since_last_record', 'mths_since_last_major_derog', 'annual_inc_joint', 'dti_joint', 'verifi
cation_status_joint', 'open_acc_6m', 'open_il_6m', 'open_il_12m', 'open_il_24m', 'mths_since_rcnt_il', 'total_bal_il', 'il_u
til', 'open_rv_12m', 'open_rv_24m', 'max_bal_bc', 'all_util', 'inq_fi', 'total_cu_tl', 'inq_last_12m']
```

In [13]:
```python
print(len(null_cols))
```

```
20
```

In [15]:
```python
df=df.drop(columns=null_cols,axis=1)
df.shape
```

Out[15]:
```
(855969, 47)
```

Treating null values

```
#TREATING THE NULL VALUES OF SOME COLUMNS WITH MEAN BASED ON DOMAIN KNOWLEGDE
for value in ['revol_util','tot_coll_amt','tot_cur_bal','total_rev_hi_lim']:
    df[value].fillna(df[value].mean(),inplace=True)
```

```
#TREATING THE NULL VALUES OF SOME COLUMNS WITH MEDIAN BASED ON DOMAIN KNOWLEGDE
df['collections_12_mths_ex_med'].fillna(df['collections_12_mths_ex_med'].median(),inplace=True)
```

Checking for any null values

```
In [27]:  df.isnull().sum()

Out[27]: loan_amnt                     0
         funded_amnt                   0
         funded_amnt_inv               0
         term                          0
         int_rate                      0
         installment                   0
         grade                         0
         sub_grade                     0
         emp_length                    0
         home_ownership                0
         annual_inc                    0
         verification_status           0
         issue_d                       0
         pymnt_plan                    0
         purpose                       0
         dti                           0
         delinq_2yrs                   0
         inq_last_6mths                0
         open_acc                      0
         pub_rec                       0
         revol_bal                     0
         revol_util                    0
         total_acc                     0
         initial_list_status           0
         out_prncp                     0
         out_prncp_inv                 0
         total_pymnt                   0
         total_pymnt_inv               0
         total_rec_prncp               0
         total_rec_int                 0
         total_rec_late_fee            0
         recoveries                    0
         collection_recovery_fee       0
         last_pymnt_amnt               0
         collections_12_mths_ex_med    0
         policy_code                   0
         application_type              0
         acc_now_delinq                0
         tot_coll_amt                  0
         tot_cur_bal                   0
         total_rev_hi_lim              0
         default_ind                   0
         dtype: int64
```

Converting categorical values into numeric values.

In [36]:
```
#collecting the categorical variables to perform transformation
char_vars=[]
for x in df.columns:
    if df[x].dtype=="object":
        char_vars.append(x)
print(char_vars)
```

```
['term', 'grade', 'home_ownership', 'verification_status', 'pymnt_plan', 'purpose', 'initial_list_status']
```

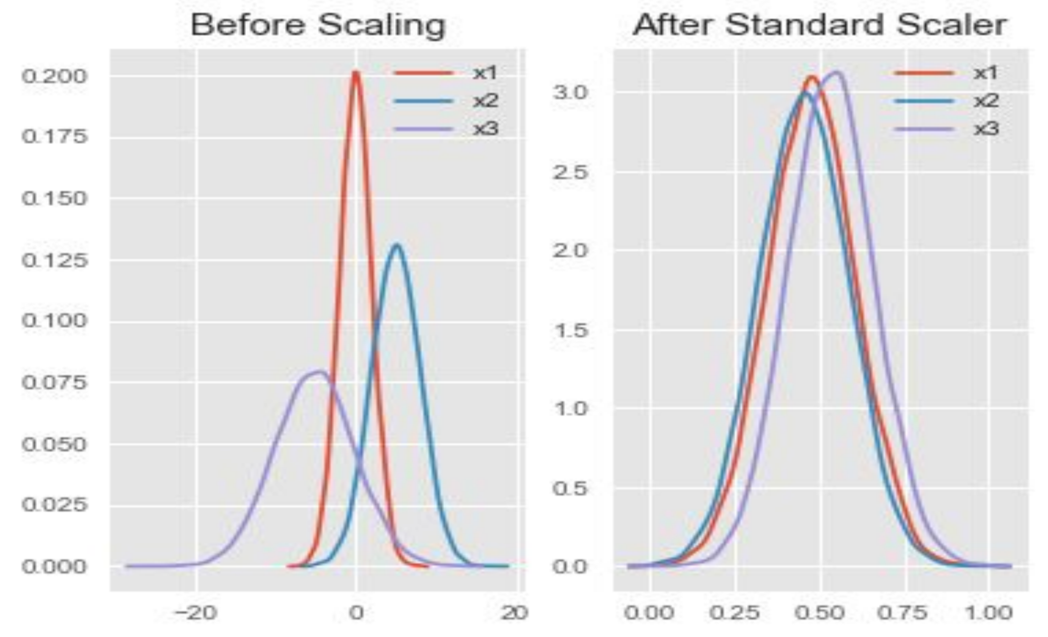
In [37]:
```
#converting the categorical variables into numeric variables using LabelEncoder()
from sklearn import preprocessing
#create an object
le=preprocessing.LabelEncoder()
for x in char_vars:
    df[x]=le.fit_transform(df[x])
```

Splitting the data.

```
In [38]: #SPLITTING THE DATASET BASED ON THE DATA VARIABLE
         split_date = pd.datetime(2015,5,1)

In [39]: #SPLITTING INTO TRAIN & TEST
         Train = df.loc[df['issue_d']<=split_date]
         Test = df.loc[df['issue_d']>split_date]

In [40]: '''ELIMINATING ISSUE_D SINCE WE DON'T NEED IT'''
         Train.drop(columns='issue_d',inplace=True)
         Test.drop(columns='issue_d',inplace=True)

In [41]: TrainX = Train.drop(columns='default_ind')
         Y_train = Train['default_ind']

In [42]: TestX = Test.drop(columns='default_ind')
         Y_test = Test['default_ind']
```

Scaling the data

```
In [43]: #SCALING THE DATA
         from sklearn.preprocessing import StandardScaler
         scaler = StandardScaler()
         scaler.fit(TrainX)
         X_train= scaler.transform(TrainX)
         X_test= scaler.transform(TestX)
```




```
In [40]: df.shape

Out[40]: (855969, 37)
```

MODEL SELECTION ,TRAINING,TUNNING AND EVALUATION.

LOGISTIC REGRESSION

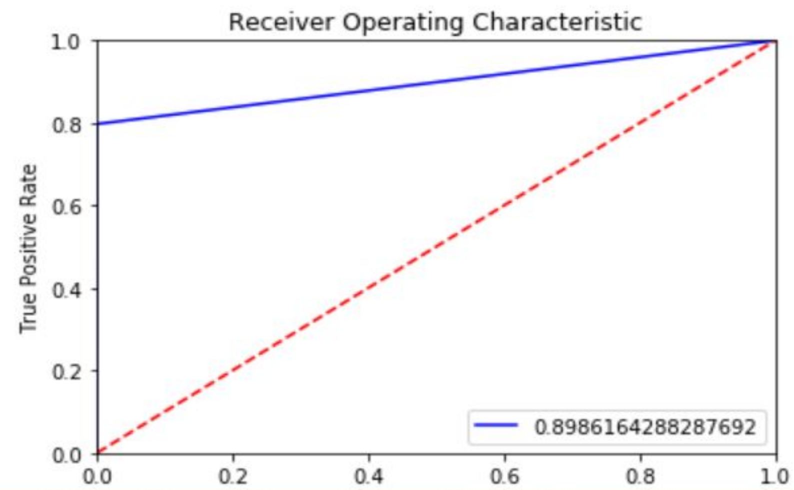
LOGISTIC REGRESSION

```
#MODEL BUILDING USING LOGISTIC REGRESSION
from sklearn.linear_model import LogisticRegression
#create the model object
lg=LogisticRegression()
#train the object
lg.fit(X_train,Y_train)
#predict
Y_pred=lg.predict(X_test)
```

```
#EVALUATION METRICS
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
print(confusion_matrix(Y_test,Y_pred))
print(classification_report(Y_test,Y_pred))
print(accuracy_score(Y_test,Y_pred))
```

```
[[256630     50]
 [    63    248]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    256680
           1       0.83      0.80      0.81       311

    accuracy                           1.00    256991
   macro avg       0.92      0.90      0.91    256991
weighted avg       1.00      1.00      1.00    256991

0.9995602958858482
```

```
#PLOTTING THE ROC CURVE
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```




```
#ROC CURVE
from sklearn import metrics
fpr, tpr, z = metrics.roc_curve(Y_test, Y_pred)
auc = metrics.auc(fpr, tpr)
print(auc)
```

```
0.898164288287692
```

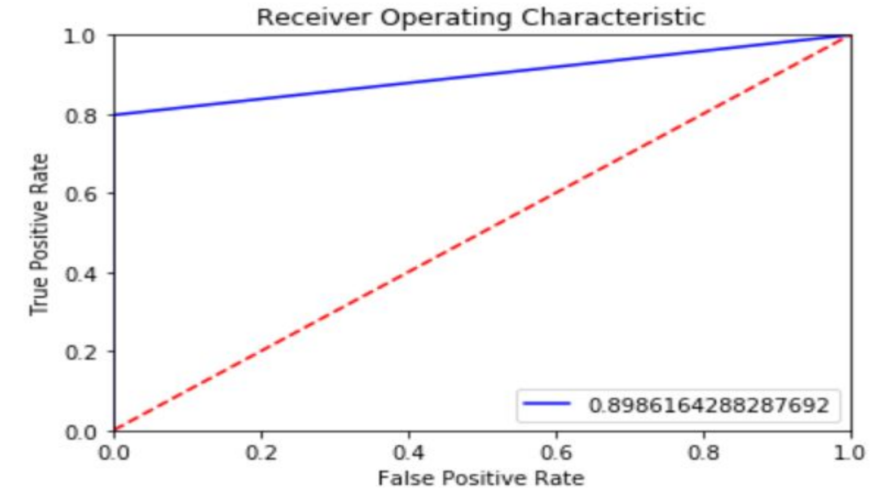
TUNNED LOGISTIC REGRESSION.

```
'''HERE WE CAN SEE THAT 0.73 IS THE BEST THRESHOLD'''
y_pred_class=[]
for value in y_pred_prob[:,1]:
    if value > 0.73:
        y_pred_class.append(1)
    else:
        y_pred_class.append(0)
#print(y_pred_class)
```

```
#EVALUATION METRICS AFTER TUNNING
cfm=confusion_matrix(Y_test, y_pred_class)
print(cfm)
print('Classification report: ')
print(classification_report(Y_test,y_pred_class))
acc= accuracy_score(Y_test, y_pred_class)
print("Accuracy of the model: ", acc)
```

```
[[256646     34]
 [    63    248]]
Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    256680
           1       0.88      0.80      0.84       311

    accuracy                           1.00    256991
   macro avg       0.94      0.90      0.92    256991
weighted avg       1.00      1.00      1.00    256991

Accuracy of the model:  0.999622548754626
```

```
#PLOTTING THE ROC CURVE
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

plt.show()
```




```
#ROC CURVE
from sklearn import metrics
fpr, tpr, z = metrics.roc_curve(Y_test, Y_pred)
auc = metrics.auc(fpr, tpr)
print(auc)
```

```
0.8986164288287692
```

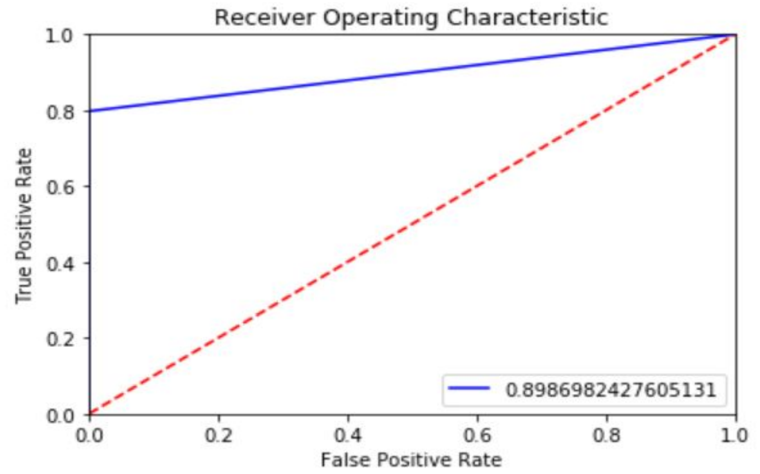
SGD CLASSIFIER.

```
from sklearn.linear_model import SGDClassifier
#create a model
classifier=SGDClassifier()
#fitting training data to the model
classifier.fit(X_train,Y_train)
Y_pred=classifier.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
confusion_matrix = confusion_matrix(Y_test,Y_pred)
print(confusion_matrix)
print("CLASSIFICATION MATRIX:")
print(classification_report(Y_test,Y_pred))
accuracy_score = accuracy_score(Y_test,Y_pred)
print("ACCURACY OF THE MODEL:",accuracy_score)
```

```
[[256672      8]
 [    63    248]]
CLASSIFICATION MATRIX:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    256680
           1       0.97      0.80      0.87       311

    accuracy                           1.00    256991
   macro avg       0.98      0.90      0.94    256991
weighted avg       1.00      1.00      1.00    256991

ACCURACY OF THE MODEL: 0.999723725733586
```

```
#PLOTTING THE ROC CURVE
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```




```
#ROC CURVE
from sklearn import metrics
fpr, tpr, z = metrics.roc_curve(Y_test, Y_pred)
auc = metrics.auc(fpr, tpr)
print(auc)
```

```
0.8986982427605131
```

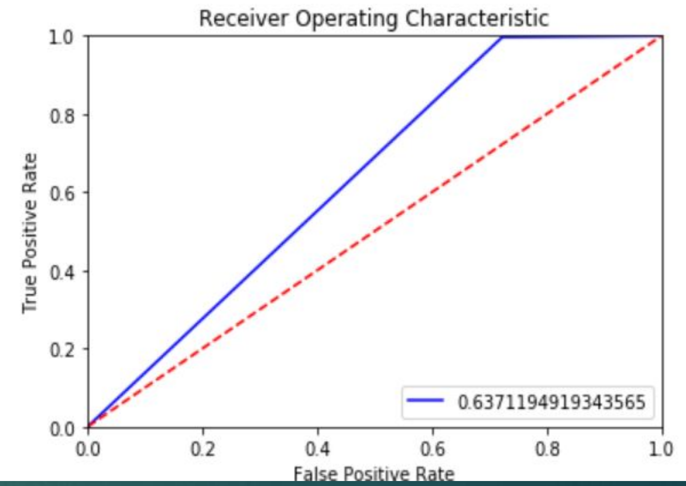
DECISION TREE

```
#MODEL BUILDING USING DECISION TREE
from sklearn.tree import DecisionTreeClassifier


model_DT=DecisionTreeClassifier()

##Train the model
model_DT.fit(X_train,Y_train)
Y_pred=model_DT.predict(X_test)
```

```
#EVALUATION METRICS
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
print(confusion_matrix(Y_test,Y_pred))
print(classification_report(Y_test,Y_pred))
print(accuracy_score(Y_test,Y_pred))
```

```
[[ 71217 185463]
 [     1    310]]
              precision    recall  f1-score   support

           0       1.00      0.28      0.43    256680
           1       0.00      1.00      0.00       311

    accuracy                           0.28    256991
   macro avg       0.50      0.64      0.22    256991
weighted avg       1.00      0.28      0.43    256991

0.278324921884424
```

```
#PLOTTING THE ROC CURVE
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

plt.show()
```




```
#ROC CURVE
from sklearn import metrics
fpr, tpr, z = metrics.roc_curve(Y_test, Y_pred)
auc = metrics.auc(fpr, tpr)
print(auc)
```

```
0.6371194919343565
```

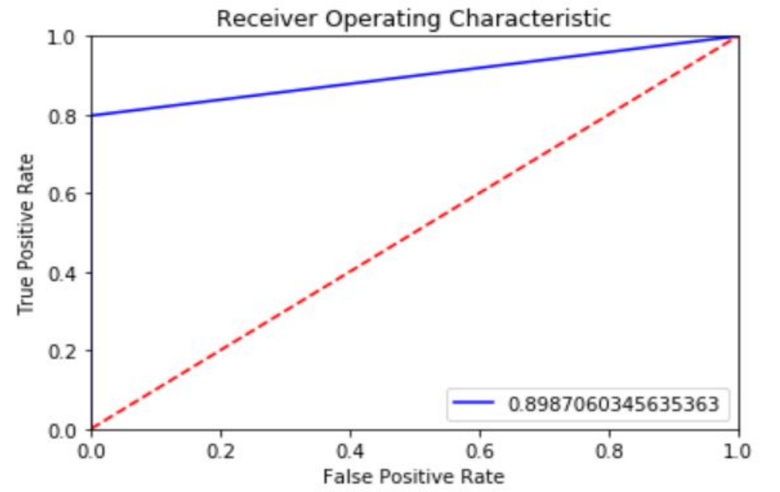
TUNNED DECISION TREE.

```
#MODEL BUILDING USING PRUNED DECISION TREE AFTER VARIOUS NUMBER OF ITERATIONS
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier(criterion="gini",random_state=10,max_depth=50,min_samples_leaf=100)
dt.fit(X_train,Y_train)
Y_pred=dt.predict(X_test)
```

```
#EVALUATION METRICS
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
print(confusion_matrix(Y_test,Y_pred))
print(classification_report(Y_test,Y_pred))
print(accuracy_score(Y_test,Y_pred))
```

```
[[256676      4]
 [    63    248]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    256680
           1       0.98      0.80      0.88       311

    accuracy                           1.00    256991
   macro avg       0.99      0.90      0.94    256991
weighted avg       1.00      1.00      1.00    256991

0.9997392904809896
```

```
#PLOTTING THE ROC CURVE
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

plt.show()
```




```
#ROC CURVE
from sklearn import metrics
fpr, tpr, z = metrics.roc_curve(Y_test, Y_pred)
auc = metrics.auc(fpr, tpr)
print(auc)
```

```
0.8987060345635363
```

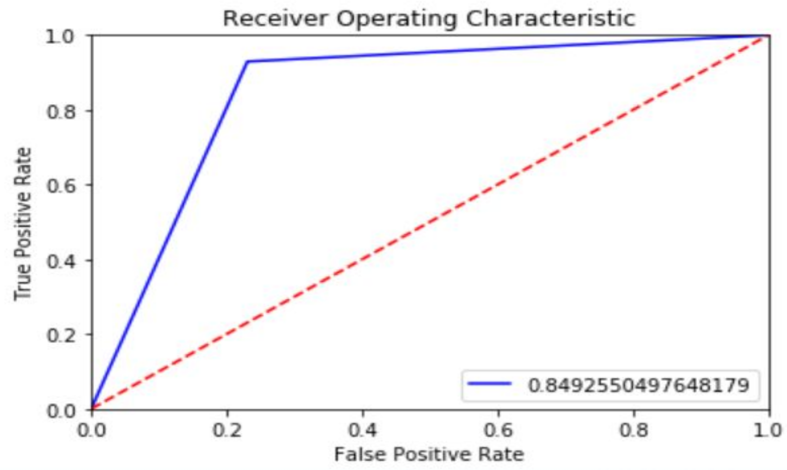
EXTRA TREE.

```python
#MODEL BUILDING USING EXTRA TREE CLASSIFIER
from sklearn.ensemble import ExtraTreesClassifier
model=ExtraTreesClassifier(30,random_state=10,bootstrap=True)
#fit the model on the data and predict the values
model=model.fit(X_train,Y_train)
Y_pred=model.predict(X_test)
```

```python
#EVALUATION METRICS
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
print(confusion_matrix(Y_test,Y_pred))
print(classification_report(Y_test,Y_pred))
print(accuracy_score(Y_test,Y_pred))
```

```
[[197451  59229]
 [    22    289]]
              precision    recall  f1-score   support

           0       1.00      0.77      0.87    256680
           1       0.00      0.93      0.01       311

    accuracy                           0.77    256991
   macro avg       0.50      0.85      0.44    256991
weighted avg       1.00      0.77      0.87    256991

0.7694432878972416
```

```python
#PLOTTING THE ROC CURVE
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

plt.show()
```




```python
#ROC CURVE
from sklearn import metrics
fpr, tpr, z = metrics.roc_curve(Y_test, Y_pred)
auc = metrics.auc(fpr, tpr)
print(auc)
```

```
0.8492550497648179
```

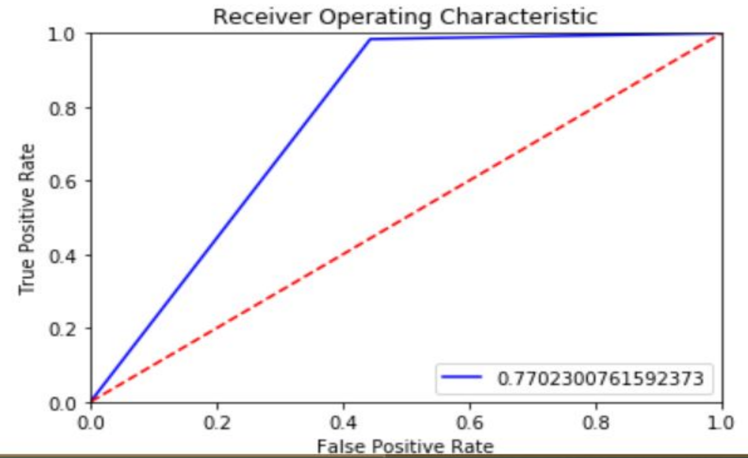
RANDOM FOREST

```python
# MODEL BUILDING USING RANDOM FOREST
from sklearn.ensemble import RandomForestClassifier
model_RandomForest=RandomForestClassifier(criterion="entropy",n_estimators=50,random_state=10,
                                          max_depth=50,min_samples_leaf=100)
#fit the model on the data and predict the values
model_RandomForest.fit(X_train,Y_train)
Y_pred=model_RandomForest.predict(X_test)
```

```python
#EVALUATION METRICS
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
print(confusion_matrix(Y_test,Y_pred))
print(classification_report(Y_test,Y_pred))
print(accuracy_score(Y_test,Y_pred))
```

```
[[256615     65]
 [    63    248]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    256680
           1       0.79      0.80      0.79       311

    accuracy                           1.00    256991
   macro avg       0.90      0.90      0.90    256991
weighted avg       1.00      1.00      1.00    256991

0.9995019280830846
```

```python
#PLOTTING THE ROC CURVE
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```




```python
#ROC CURVE
from sklearn import metrics
fpr, tpr, z = metrics.roc_curve(Y_test, Y_pred)
auc = metrics.auc(fpr, tpr)
print(auc)
```

```
0.7702300761592373
```

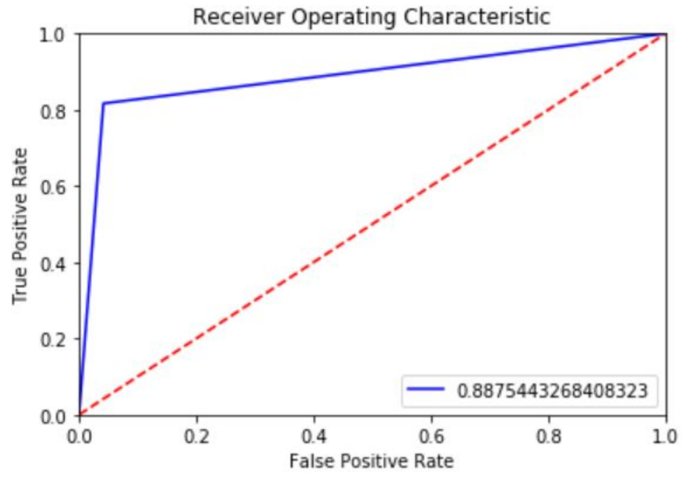
GRADIENT BOOSTING

```
#MODEL BUILDING USING GRADIENT BOOST
from sklearn.ensemble import GradientBoostingClassifier
model_GradientBoosting=GradientBoostingClassifier(loss="exponential")
#fit the model on the data and predict the values
model_GradientBoosting.fit(X_train,Y_train)
Y_pred=model_GradientBoosting.predict(X_test)
```

```
#EVALUATION METRICS
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
print(confusion_matrix(Y_test,Y_pred))
print(classification_report(Y_test,Y_pred))
print(accuracy_score(Y_test,Y_pred))
```

```
[[245994  10686]
 [    57    254]]
              precision    recall  f1-score   support

           0       1.00      0.96      0.98    256680
           1       0.02      0.82      0.05       311

    accuracy                           0.96    256991
   macro avg       0.51      0.89      0.51    256991
weighted avg       1.00      0.96      0.98    256991

0.9581969796607663
```

```
#PLOTTING OF THE ROC CURVE
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```




```
#ROC CURVE
from sklearn import metrics
fpr, tpr, z = metrics.roc_curve(Y_test, Y_pred)
auc = metrics.auc(fpr, tpr)
print(auc)
```

```
0.8875443268408323
```

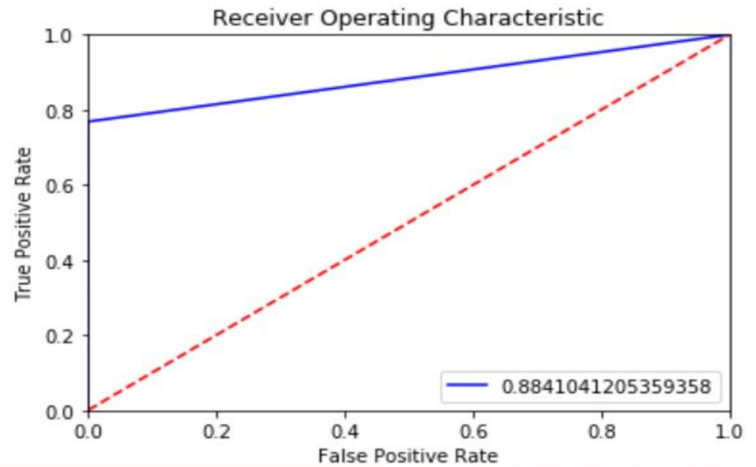
ADA BOOSTING

```
#MODEL BUILDING USING ADA BOOST
from sklearn.ensemble import AdaBoostClassifier
model_AdaBoost=AdaBoostClassifier(n_estimators=50,algorithm="SAMME.R")
#fit the model on the data and predict the values
model_AdaBoost.fit(X_train,Y_train)
Y_pred=model_AdaBoost.predict(X_test)
```

```
#EVALUATION METRICS
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
print(confusion_matrix(Y_test,Y_pred))
print(classification_report(Y_test,Y_pred))
print(accuracy_score(Y_test,Y_pred))
```

```
[[245616  11064]
 [    62    249]]
              precision    recall  f1-score   support

           0       1.00      0.96      0.98    256680
           1       0.02      0.80      0.04       311

    accuracy                           0.96    256991
   macro avg       0.51      0.88      0.51    256991
weighted avg       1.00      0.96      0.98    256991

0.9567066550968711
```

```
#PLOTTING THE ROC CURVE
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

plt.show()
```




```
#ROC CURVE
from sklearn import metrics
fpr, tpr, z = metrics.roc_curve(Y_test, Y_pred)
auc = metrics.auc(fpr, tpr)
print(auc)
```

```
0.8841041205359358
```

COMPARING ALL OUTPUTS

Model Name	Accuracy Score	Type 1 Error	Type 2 Error	Recall Class 0	Recall Class 1	Precision Class 0	Precision Class 1
LOGISTIC	99.95	50	63	1.00	0.80	1.00	0.83
TUNED LOGISTIC	99.96	34	63	1.00	0.80	1.00	0.83
DECISION TREE	27.83	185463	1	0.28	1.00	1.00	0.00
PRUNED DECISION TREE	99.97	4	63	1.00	0.80	1.00	0.98
EXTRA TREES	76.94	59229	22	0.77	0.93	1.00	0.00
RANDOM FOREST	99.95	65	63	1.00	0.80	1.00	0.79
GRADIENT BOOSTING	95.81	10686	57	0.96	0.82	1.00	0.02
ADA	95.67	11064	62	0.96	0.80	1.00	0.02
SGD	99.97	8	63	1.00	0.80	1.00	0.97

CONCLUSION

- Using analysing techniques one can predict or analyse that a customer applying for the loan will repay the loan or not. So using multiple algorithms we can be able to analyse a defaulter. The best model selected out of all is Pruned Decision Tree with accuracy 99.97%.