Project Report on

UNDERSTANDING RISK ANALYSIS IN BANKING AND FINANCIAL SERVICES

Submitted in partial fulfillment for the award of
Post Graduate Diploma in High Performance Computing Application Programming from CDAC-ACTS (Pune)

Under the Guidance of
Mr. Prakash Sinha

Submitted By:

Aditya Nagose	220940141001
Anjali Sangode	220940141005
Pooja Jadhav	220940141017
Taksande Sandeep Ravindra	220940141028

INDEX

INTRODUCTION

- **About Lending Club**
  - LendingClub is a US peer-to-peer lending company, headquartered in San Francisco, California founded by Renaud Laplanche and Soul Htite in 2006. It was the first peer-to-peer lender to register its offerings as securities with the Securities and Exchange Commission (SEC), and to offer loan trading on a secondary market. LendingClub is the world's largest peer-to-peer lending platform.
  - Borrowers were able to arrange unsecured personal loans between $1,000 and $40,000 via LendingClub. The typical loan term was three years.
  - According to the information provided about the borrower, loan amount, loan grade, and loan purpose, investors were able to search and browse the loan listings on the LendingClub website and choose loans that they wished to invest in.
  - The interest on these loans generated profits for the investors. By charging investors a service fee and borrowers an origination fee, LendingClub generated income.
  - As the largest technology IPO in the United States of 2014, the firm raised $1 billion in its initial public offering. Despite being one of the biggest fintech companies and regarded as a pioneer in the field, LendingClub ran into issues in early 2016. These issues included difficulty attracting investors, a scandal involving some of the company's loans, and board concerns over CEO Renaud Laplanche's disclosures, which caused a significant decline in the company's share price and Laplanche's resignation.
  - The peer-to-peer lending platform of LendingClub, which had recently purchased Radius Bank, would be shutting down in 2020. There are no new loans accessible for individual investment; however, existing account holders will still be able to receive interest on their existing notes until they are all repaid or default occurs. A secondary market, which was originally used to sell existing loans, is no longer available.

- **Purpose/ Objective of the Project**
  - The goal of risk analysis is to recognise, quantify, and reduce different risk exposures or hazards that may be encountered by a project, investment, or organization.
  - Solving this case study will provide an idea about how real business problems are solved using EDA and Machine Learning. In this case study, we will also develop a basic understanding of risk analytics in banking and financial services and understand how data is used to minimize the risk of losing money while lending to customers.

- **Dataset**
  - The dataset contains all entries of Lending Club from 2007 to 2018 having 2260701 entries and 151 features.

IMPORTING ESSENTIAL LIBRARIES

3. Installing and loading the essential libraries

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import rcParams
from scipy import stats
import warnings
from sklearn import datasets
from sklearn import linear_model
from sklearn.metrics import explained_variance_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import RepeatedStratifiedKFold
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import plot_confusion_matrix, plot_roc_curve, roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.datasets import make_blobs
from sklearn import metrics
from math import sqrt
import time


sns.set_style('darkgrid')
warnings.filterwarnings('ignore')
```

READING THE DATASET

4. Reading the dataset

```
In [3]: prog_start = time.time()


        # time taken to read data
        s_time_chunk = time.time()
        chunk = pd.read_csv(r'D:\LendingClub\accepted.csv', chunksize=10000,low_memory=False)
        df = pd.concat(chunk)
        e_time_chunk = time.time()

        print("\n\nWith chunks: ", (e_time_chunk-s_time_chunk)/60, " Minutes\n\n")
        df.head()


        With chunks:  0.8419672012329101  Minutes
```

With Chunk size of 10000, the process takes 0.8149 Minutes (50.5176 seconds) to read the dataset. It is approximately 18 seconds LESS than executing without chunks.

4. Reading the dataset

```
In [5]: prog_start = time.time()


        # time taken to read data
        s_time_chunk = time.time()
        df = pd.read_csv(r'D:\LendingClub\accepted.csv',low_memory=False)
        e_time_chunk = time.time()

        print("\n\nWithout chunks: ", (e_time_chunk-s_time_chunk)/60, " Minutes\n\n")
        df.head()


        Without chunks:  1.1313332239786784  Minutes
```

Without any chunk the process takes 1.13133 Minutes (67.8798 seconds) to read the dataset.

```
In [7]: pd.options.display.max_columns = 200
        df.describe()
```

Out[7]:

	member_id	loan_amnt	funded_amnt	funded_amnt_inv	int_rate	installment	annual_inc	dti	delinq_2yrs	fico_range_low	fico_
count	0.0	2.260668e+06	2.260668e+06	2.260668e+06	2.260668e+06	2.260668e+06	2.260664e+06	2.258957e+06	2.260639e+06	2.260668e+06	2.
mean	NaN	1.504693e+04	1.504166e+04	1.502344e+04	1.309283e+01	4.458068e+02	7.799243e+04	1.882420e+01	3.068792e-01	6.985882e+02	7.
std	NaN	9.190245e+03	9.188413e+03	9.192332e+03	4.832138e+00	2.671735e+02	1.126962e+05	1.418333e+01	8.672303e-01	3.301038e+01	3.
min	NaN	5.000000e+02	5.000000e+02	0.000000e+00	5.310000e+00	4.930000e+00	0.000000e+00	-1.000000e+00	0.000000e+00	6.100000e+02	6.
25%	NaN	8.000000e+03	8.000000e+03	8.000000e+03	9.490000e+00	2.516500e+02	4.600000e+04	1.189000e+01	0.000000e+00	6.750000e+02	6.
50%	NaN	1.290000e+04	1.287500e+04	1.280000e+04	1.262000e+01	3.779900e+02	6.500000e+04	1.784000e+01	0.000000e+00	6.900000e+02	6.
75%	NaN	2.000000e+04	2.000000e+04	2.000000e+04	1.599000e+01	5.933200e+02	9.300000e+04	2.449000e+01	0.000000e+00	7.150000e+02	7.
max	NaN	4.000000e+04	4.000000e+04	4.000000e+04	3.099000e+01	1.719830e+03	1.100000e+08	9.990000e+02	5.800000e+01	8.450000e+02	8.

```
In [8]: df.shape
Out[8]: (2260701, 151)
```

4.1 Looking at Columns

The columns are mentioned along with the datatype, missing values, unique values, first, second and third values and finally the entropy of the column. The entropy is defined as the measure of disorder or uncertainity. If the entropy is 0.88, this is considered a high entropy , a high level of disorder (meaning low level of purity).

In [9]:
```
def resumetable(df):
    print(f"Dataset Shape: {df.shape}")
    summary = pd.DataFrame(df.dtypes,columns=['dtypes'])
    summary = summary.reset_index()
    summary['Name'] = summary['index']
    summary = summary[['Name','dtypes']]
    summary['Missing'] = df.isnull().sum().values
    summary['Uniques'] = df.nunique().values
    summary['First Value'] = df.loc[0].values
    summary['Second Value'] = df.loc[1].values
    summary['Third Value'] = df.loc[2].values

    for name in summary['Name'].value_counts().index:
        summary.loc[summary['Name'] == name, 'Entropy'] = round(stats.entropy(df

    return summary
```

We will look at the columns in three installments. We will look at the first 60 columns first, then the next 60, and then the final 31 columns.

In [11]:
```
resumetable(df[:100000])[60:121]
```

```
Dataset Shape: (100000, 151)
```

Out[11]:

	Name	dtypes	Missing	Uniques	First Value	Second Value	Third Value	Entropy
60	acc_now_delinq	float64	0	5	0.0	0.0	0.0	0.05
61	tot_coll_amt	float64	0	4010	722.0	0.0	0.0	2.40
62	tot_cur_bal	float64	0	82713	144904.0	204396.0	189699.0	16.24
63	open_acc_6m	float64	78628	13	2.0	1.0	0.0	2.06
64	open_act_il	float64	78628	35	2.0	1.0	1.0	3.05
...	...	...	...	...	...	...	...	...
116	sec_app_fico_range_low	float64	100000	0	NaN	NaN	NaN	0.00
117	sec_app_fico_range_high	float64	100000	0	NaN	NaN	NaN	0.00
118	sec_app_earliest_cr_line	object	100000	0	NaN	NaN	NaN	0.00
119	sec_app_inq_last_6mths	float64	100000	0	NaN	NaN	NaN	0.00
120	sec_app_mort_acc	float64	100000	0	NaN	NaN	NaN	0.00

61 rows × 8 columns

Accroding to the missing values, unique values and the entropy, we have removed the following columns. These columns are excluded from the analysis.

In [9]:
```
df.drop(columns=['member_id','desc','mths_since_last_delinq','mths_since_last_record','mths_since_last_major_derog',
                 'annual_inc_joint','dti_joint','total_bal_il','total_bal_il','max_bal_bc',
                 'bc_open_to_buy','revol_bal_joint','sec_app_fico_range_low','sec_app_fico_range_high',
                 'sec_app_earliest_cr_line','sec_app_inq_last_6mths','sec_app_mort_acc','sec_app_open_acc',
                 'sec_app_revol_util','sec_app_open_act_il','sec_app_num_rev_accts','sec_app_chargeoff_within_12_mths',
                 'sec_app_collections_12_mths_ex_med','sec_app_mths_since_last_major_derog',
                 'hardship_type','deferral_term','hardship_length'], inplace=True)
```

In [10]:
```
df.drop(columns=['id'], inplace=True)
```

The shape of the dataset after removing the columns is:

In [11]:
```
df.shape
```

Out[11]:
```
(2260701, 124)
```

In [16]:
```
df.dtypes
```

Out[16]:
```
loan_amnt               float64
funded_amnt             float64
funded_amnt_inv         float64
term                     object
int_rate                float64
                          ...
settlement_status        object
settlement_date          object
settlement_amount       float64
settlement_percentage   float64
settlement_term         float64
Length: 124, dtype: object
```

The Program is divided into 3 stages:

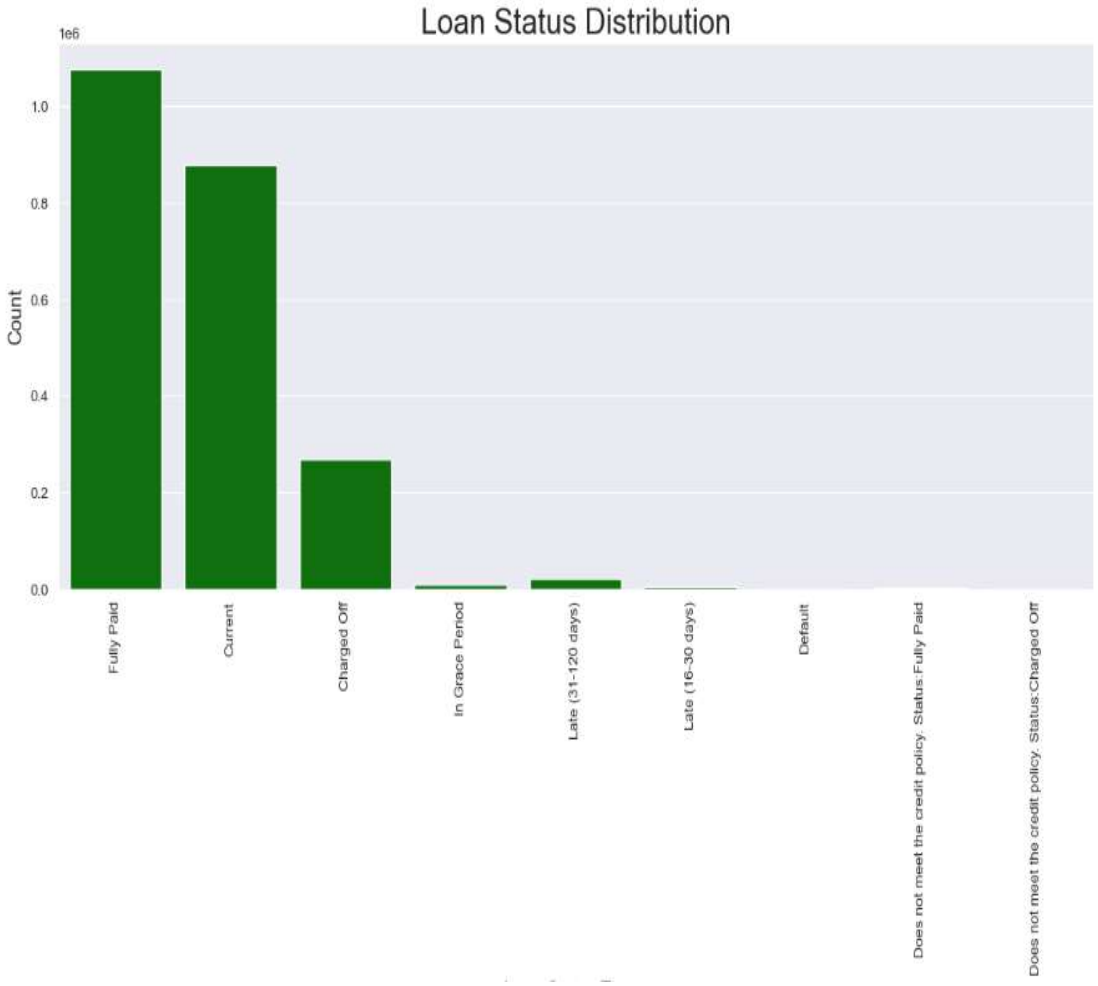
1. **Exploratory Data Analysis / Data Visualization**
2. **Model Designing by splitting dataset in to Train and Test**
3. **Model Diagnostics**

EXPLORATORY DATA ANALYSIS/ DATA VISUALIZATION

5. Exploratory Data Analysis (Data Visualization) of Lending Club Dataset

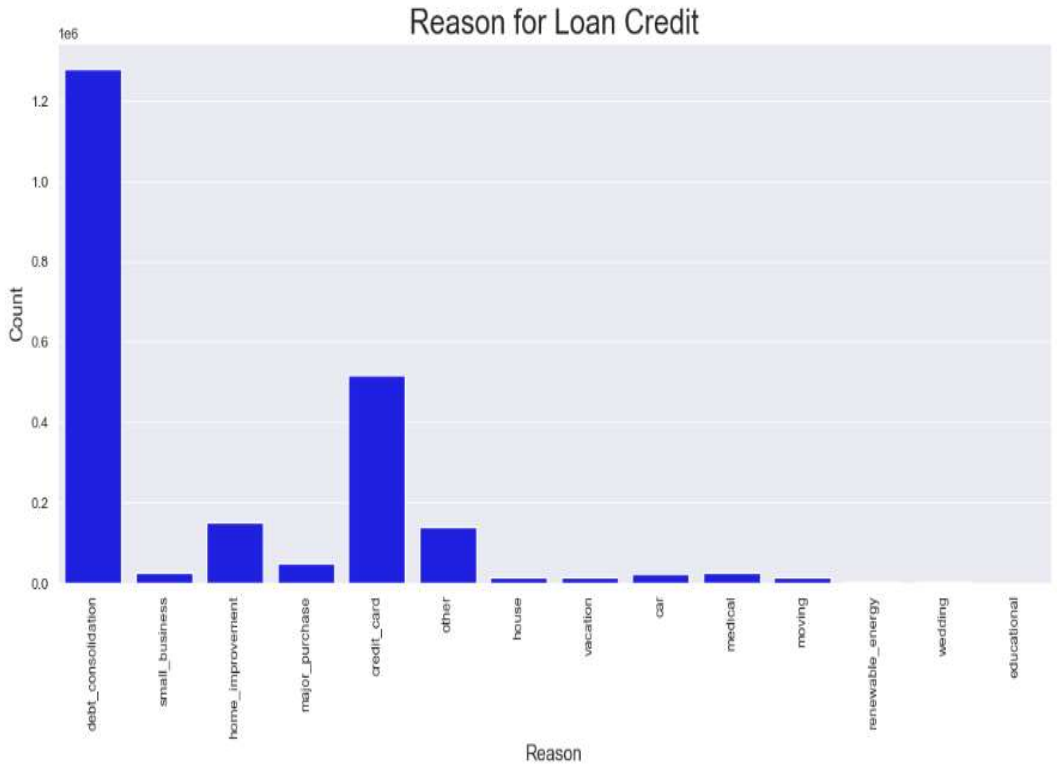
For our EDA, we first want to see the different loan categories in the dataset.

In [24]:
```
plt.figure(figsize = (14,6))
g = sns.countplot(x="loan_status", data=df, color='green')
g.set_xticklabels(g.get_xticklabels(),rotation=90)
g.set_xlabel("Loan Status Types", fontsize=13)
g.set_ylabel("Count", fontsize=13)
g.set_title("Loan Status Distribution", fontsize=22)
plt.show()
```



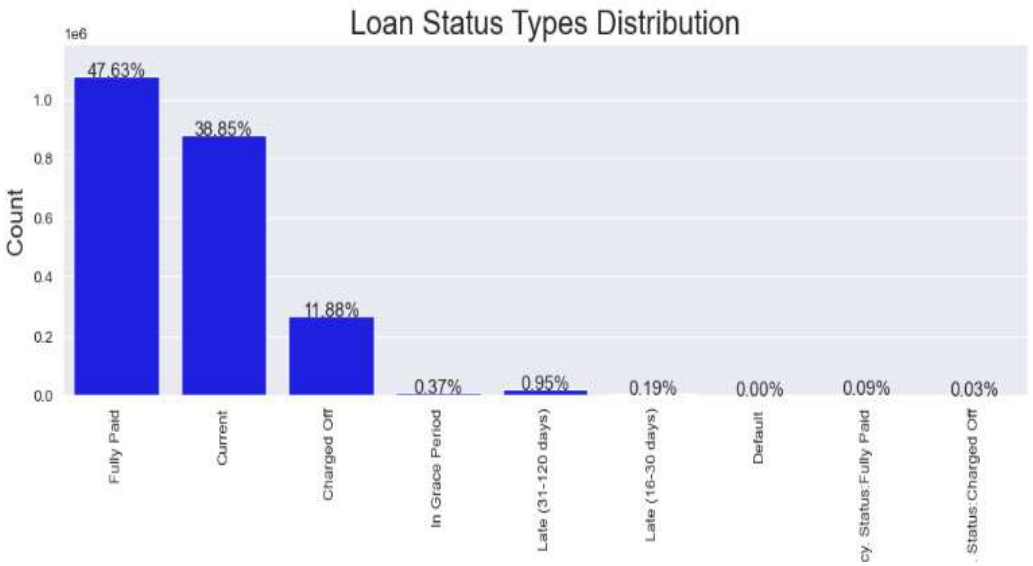
Next, it would be interesting to understand the reason for borrowing the loan. For this we have taken the column 'purpose'.

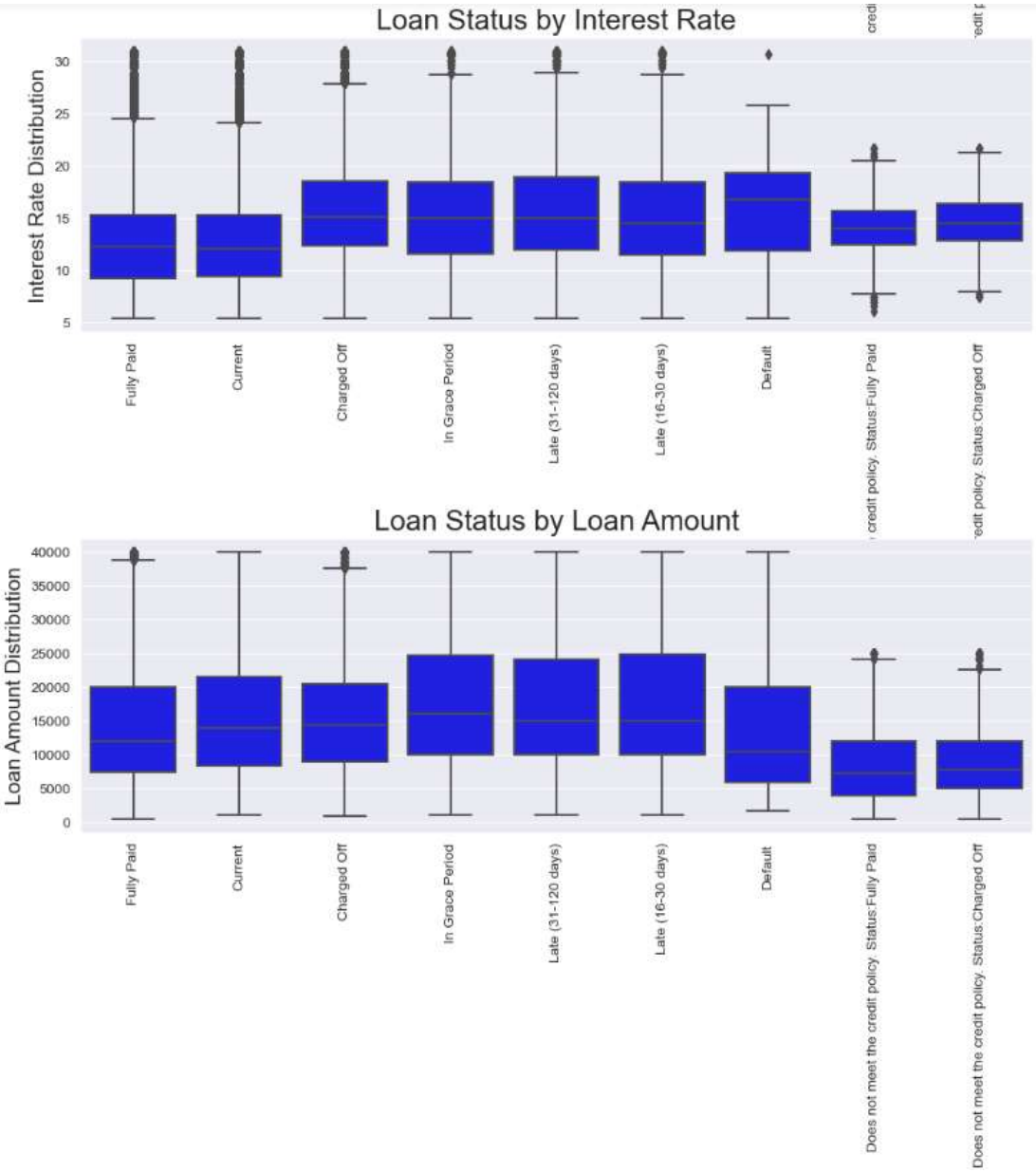
In [25]:
```
plt.figure(figsize=(14,6))
g = sns.countplot(x='purpose', data=df, color='blue')
g.set_xticklabels(g.get_xticklabels(),rotation=90)
g.set_title("Reason for Loan Credit", fontsize=22)
g.set_xlabel("Reason", fontsize=13)
g.set_ylabel('Count', fontsize=13)
plt.show()
```



Debt consolidation and credit card are the main reasons for borrowing a loan.

In [26]:
```
plt.figure(figsize = (12,16))
total = len(df)
plt.subplot(311)
g = sns.countplot(x="loan_status", data=df,
                  color='blue')
g.set_xticklabels(g.get_xticklabels(),rotation=90)
g.set_xlabel("Loan Status Categories", fontsize=12)
g.set_ylabel("Count", fontsize=15)
g.set_title("Loan Status Types Distribution", fontsize=20)
sizes=[]
for p in g.patches:
    height = p.get_height()
    sizes.append(height)
    g.text(p.get_x()+p.get_width()/2.,
           height + 3,
           '{:1.2f}%'.format(height/total*100),
           ha="center", fontsize=12)
g.set_ylim(0, max(sizes) * 1.10)

plt.subplot(312)
g1 = sns.boxplot(x="loan_status", y="int_rate", data=df, color='blue')
g1.set_xticklabels(g1.get_xticklabels(),rotation=90)
g1.set_xlabel("Loan Status Categories", fontsize=12)
g1.set_ylabel("Interest Rate Distribution", fontsize=15)
g1.set_title("Loan Status by Interest Rate", fontsize=20)
plt.subplot(313)
g2 = sns.boxplot(x="loan_status", y="loan_amnt", data=df,
                 color='blue')
g2.set_xticklabels(g2.get_xticklabels(),rotation=90)
g2.set_xlabel("Loan Status Categories", fontsize=15)
g2.set_ylabel("Loan Amount Distribution", fontsize=15)
g2.set_title("Loan Status by Loan Amount", fontsize=20)

plt.subplots_adjust(hspace = 0.7,top = 0.9)

plt.show()
```




Loan Status by Interest Rate
Interest Rate Distribution
30
25
20
15
10
5
Fully Paid
Current
Charged Off
In Grace Period
Late (31-120 days)
Late (16-30 days)
Default
credi
edit p
credit policy. Status:Fully Paid
edit policy. Status:Charged Off

Loan Status by Loan Amount
Loan Amount Distribution
40000
35000
30000
25000
20000
15000
10000
5000
0
Fully Paid
Current
Charged Off
In Grace Period
Late (31-120 days)
Late (16-30 days)
Default
Does not meet the credit policy. Status:Fully Paid
Does not meet the credit policy. Status:Charged Off

5.1 Crosstabs

Purpose by Loan Status

In [28]:
```
purp_loan= ['purpose', 'loan_status']
cm = sns.light_palette("green", as_cmap=True)
(round(pd.crosstab(df[purp_loan[0]], df[purp_loan[1]],
                   normalize='columns') * 100,2)).style.background_gradient(cmap = cm)
```

Out[28]:

loan_status	Charged Off	Current	Default	Does not meet the credit policy. Status:Charged Off	Does not meet the credit policy. Status:Fully Paid	Fully Paid	In Grace Period	Late (16-30 days)	Late (31-120 days)
purpose									
car	0.800000	1.030000	7.500000	1.710000	2.570000	1.160000	0.890000	0.690000	0.890000
credit_card	18.610000	24.460000	15.000000	9.070000	13.630000	22.780000	19.820000	18.850000	18.700000
debt_consolidation	61.440000	54.240000	52.500000	38.370000	40.640000	57.140000	59.250000	57.990000	58.110000
educational	0.020000	0.000000	0.000000	4.200000	3.270000	0.030000	0.000000	0.000000	0.000000
home_improvement	5.770000	6.880000	7.500000	9.330000	7.190000	6.690000	6.610000	7.470000	6.710000
house	0.590000	0.750000	2.500000	1.450000	1.660000	0.530000	0.720000	0.970000	0.870000
major_purchase	2.040000	2.280000	5.000000	3.020000	5.030000	2.220000	2.230000	2.460000	2.650000
medical	1.260000	1.300000	5.000000	2.890000	1.810000	1.130000	1.260000	1.130000	1.370000
moving	0.820000	0.640000	0.000000	1.970000	1.560000	0.670000	0.700000	0.990000	0.830000
other	6.100000	6.680000	5.000000	15.900000	15.240000	5.710000	6.760000	7.130000	7.470000
renewable_energy	0.080000	0.060000	0.000000	0.130000	0.100000	0.070000	0.010000	0.090000	0.070000
small_business	1.710000	0.980000	0.000000	9.460000	4.480000	1.010000	1.320000	1.630000	1.580000
vacation	0.650000	0.710000	0.000000	0.790000	0.650000	0.680000	0.440000	0.600000	0.750000
wedding	0.100000	0.000000	0.000000	1.710000	2.160000	0.190000	0.000000	0.000000	0.000000

Grade by Loan Status

In [29]:
```
purp_loan= ['grade', 'loan_status']
cm = sns.light_palette("green", as_cmap=True)
(round(pd.crosstab(df[purp_loan[0]], df[purp_loan[1]],
                   normalize='columns') * 100,2)).style.background_gradient(cmap = cm)
```

Out[29]:

loan_status	Charged Off	Current	Default	Does not meet the credit policy. Status:Charged Off	Does not meet the credit policy. Status:Fully Paid	Fully Paid	In Grace Period	Late (16-30 days)	Late (31-120 days)
grade									
A	5.290000	22.260000	12.500000	1.050000	4.530000	20.510000	7.490000	7.840000	6.160000
B	19.570000	29.930000	17.500000	11.170000	13.530000	31.590000	20.990000	22.830000	22.300000
C	31.890000	29.110000	20.000000	19.450000	24.200000	27.490000	35.480000	33.940000	35.190000
D	22.730000	13.110000	32.500000	25.890000	24.850000	12.990000	22.510000	22.280000	22.180000
E	13.420000	4.340000	15.000000	20.760000	19.010000	5.350000	9.470000	9.240000	10.090000
F	5.400000	0.970000	2.500000	12.220000	7.750000	1.630000	2.840000	2.970000	2.980000
G	1.700000	0.280000	0.000000	9.460000	6.140000	0.420000	1.210000	0.900000	1.100000

Interest Rate by Grade and Loan Status

In [30]:
```
loan_grade = ['loan_status', 'grade']
cm = sns.light_palette("green", as_cmap=True)
round(pd.crosstab(df[loan_grade[0]], df[loan_grade[1]],
            values=df['int_rate'], aggfunc='mean'),2).fillna(0).style.background_gradient(cmap = cm)
```

Out[30]:

grade	A	B	C	D	E	F	G
loan_status							
Charged Off	7.390000	10.790000	14.080000	17.790000	21.200000	25.100000	27.950000
Current	7.050000	10.670000	14.320000	18.850000	23.460000	27.380000	29.790000
Default	7.200000	11.040000	13.870000	18.860000	24.520000	30.740000	0.000000
Does not meet the credit policy. Status:Charged Off	8.300000	10.760000	12.610000	14.480000	15.530000	17.180000	18.860000
Does not meet the credit policy. Status:Fully Paid	8.090000	10.760000	12.870000	14.440000	15.570000	17.190000	18.930000
Fully Paid	7.100000	10.660000	14.000000	17.690000	21.100000	24.800000	27.510000
In Grace Period	7.270000	10.700000	14.350000	18.750000	23.390000	27.500000	29.620000
Late (16-30 days)	7.190000	10.790000	14.380000	18.910000	23.740000	27.850000	30.140000
Late (31-120 days)	7.250000	10.730000	14.360000	18.910000	23.670000	27.920000	30.130000

Verification Status

This indicates if the income was verified by LC, not verified, or if the income source was verified.

In [31]:
```
plt.figure(figsize = (13,6))
g = sns.countplot(x="verification_status", data=df, color='yellow')
g.set_xlabel("Loan Status", fontsize=13, color='blue')
g.set_ylabel("Count", fontsize=13, color='blue')
g.set_title("Loan Status Distribution", fontsize=22)
sizes=[]
for p in g.patches:
    height = p.get_height()
    sizes.append(height)
    g.text(p.get_x()+p.get_width()/2.,
           height + 3,
           '{:1.2f}%'.format(height/total*100),
           ha="center", fontsize=14)
g.set_ylim(0, max(sizes) * 1.10)
plt.show()
```

Source verified status has the highest distribution.

Loan Status by Verification Status

In [32]:
```
loan_verification = ['loan_status', 'verification_status']
cm = sns.light_palette("green", as_cmap=True)
pd.crosstab(df[loan_verification[0]], df[loan_verification[1]],
            normalize='index').style.background_gradient(cmap = cm)
```

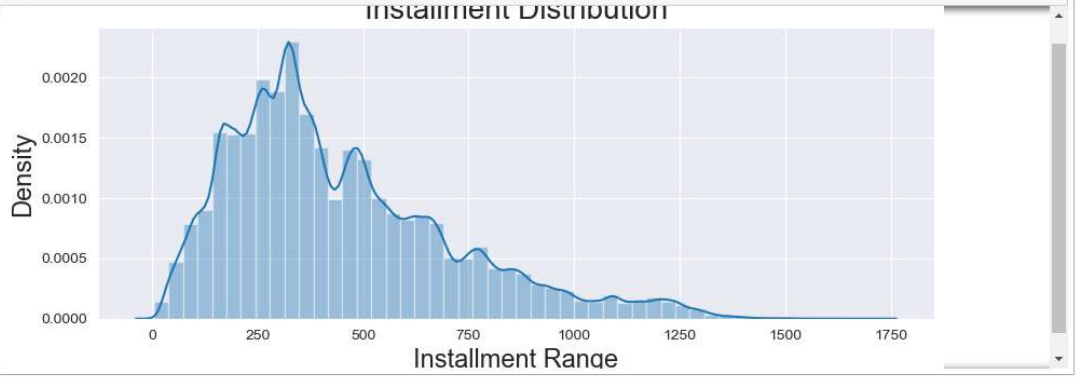
Out[32]:

verification_status	Not Verified	Source Verified	Verified
loan_status			
Charged Off	0.221683	0.406734	0.371583
Current	0.373559	0.398717	0.227724
Default	0.200000	0.400000	0.400000
Does not meet the credit policy. Status:Charged Off	0.671485	0.107753	0.220762
Does not meet the credit policy. Status:Fully Paid	0.664487	0.104628	0.230885
Fully Paid	0.321491	0.382671	0.295838
In Grace Period	0.266240	0.423661	0.310100
Late (16-30 days)	0.275236	0.420556	0.304208
Late (31-120 days)	0.266409	0.421531	0.312060

Installment

It is the monthly payment owed by the borrower if the loan originates.

In [39]:
```
plt.figure(figsize=(10,3.5))
sns.distplot(df['installment'])
plt.title("Installment Distribution", fontsize=20)
plt.xlabel("Installment Range", fontsize=17)
plt.ylabel("Density", fontsize=17)
plt.show()
```



We can see the peak of the installments are at monthly ~300USD.

Loan Status by Application Type

In [41]:
```
loan_application = ['loan_status', 'application_type']
cm = sns.light_palette("green", as_cmap=True)
pd.crosstab(df[loan_application[0]], df[loan_application[1]]).style.background_gradient(cmap = cm)
```

Out[41]:

application_type	Individual	Joint App
loan_status		
Charged Off	262215	6344
Current	787191	91126
Default	34	6
Does not meet the credit policy. Status:Charged Off	761	0
Does not meet the credit policy. Status:Fully Paid	1988	0
Fully Paid	1057295	19456
In Grace Period	7342	1094
Late (16-30 days)	3843	506
Late (31-120 days)	19289	2178

Loan Status by Home Ownership

In [42]:
```
loan_home = ['loan_status', 'home_ownership']
cm = sns.light_palette("green", as_cmap=True)
round(pd.crosstab(df[loan_home[0]], df[loan_home[1]],
            normalize='index'),2).fillna(0).style.background_gradient(cmap = cm)
```

Out[42]:

home_ownership	ANY	MORTGAGE	NONE	OTHER	OWN	RENT
loan_status						
Charged Off	0.000000	0.430000	0.000000	0.000000	0.110000	0.460000
Current	0.000000	0.490000	0.000000	0.000000	0.120000	0.390000
Default	0.000000	0.420000	0.000000	0.000000	0.200000	0.380000
Does not meet the credit policy. Status:Charged Off	0.000000	0.460000	0.000000	0.010000	0.060000	0.460000
Does not meet the credit policy. Status:Fully Paid	0.000000	0.460000	0.000000	0.010000	0.070000	0.460000
Fully Paid	0.000000	0.510000	0.000000	0.000000	0.110000	0.380000
In Grace Period	0.000000	0.480000	0.000000	0.000000	0.110000	0.410000
Late (16-30 days)	0.000000	0.440000	0.000000	0.000000	0.130000	0.430000
Late (31-120 days)	0.000000	0.430000	0.000000	0.000000	0.130000	0.440000

Mean Interest Rate by Grade and Sub-Grade

In [43]:
```
loan_grade = ['sub_grade', 'grade']
cm = sns.light_palette("green", as_cmap=True)
round(pd.crosstab(df[loan_grade[0]], df[loan_grade[1]],
            values=df['int_rate'], aggfunc='mean'),2).fillna(0).style.background_gradient(cmap = cm)
```

Out[43]:

grade	A	B	C	D	E	F	G
sub_grade							
A1	5.600000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
A2	6.550000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
A3	7.090000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
A4	7.560000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
A5	8.200000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
B1	0.000000	9.080000	0.000000	0.000000	0.000000	0.000000	0.000000
B2	0.000000	9.970000	0.000000	0.000000	0.000000	0.000000	0.000000
B3	0.000000	10.710000	0.000000	0.000000	0.000000	0.000000	0.000000
B4	0.000000	11.370000	0.000000	0.000000	0.000000	0.000000	0.000000
B5	0.000000	12.010000	0.000000	0.000000	0.000000	0.000000	0.000000
C1	0.000000	0.000000	12.780000	0.000000	0.000000	0.000000	0.000000
C2	0.000000	0.000000	13.540000	0.000000	0.000000	0.000000	0.000000
C3	0.000000	0.000000	14.100000	0.000000	0.000000	0.000000	0.000000
C4	0.000000	0.000000	14.880000	0.000000	0.000000	0.000000	0.000000
C5	0.000000	0.000000	15.770000	0.000000	0.000000	0.000000	0.000000
D1	0.000000	0.000000	0.000000	16.660000	0.000000	0.000000	0.000000
D2	0.000000	0.000000	0.000000	17.600000	0.000000	0.000000	0.000000
D3	0.000000	0.000000	0.000000	18.390000	0.000000	0.000000	0.000000
D4	0.000000	0.000000	0.000000	19.070000	0.000000	0.000000	0.000000
D5	0.000000	0.000000	0.000000	20.060000	0.000000	0.000000	0.000000
E1	0.000000	0.000000	0.000000	0.000000	20.330000	0.000000	0.000000
E2	0.000000	0.000000	0.000000	0.000000	21.010000	0.000000	0.000000
E3	0.000000	0.000000	0.000000	0.000000	21.880000	0.000000	0.000000
E4	0.000000	0.000000	0.000000	0.000000	22.780000	0.000000	0.000000
E5	0.000000	0.000000	0.000000	0.000000	24.110000	0.000000	0.000000
F1	0.000000	0.000000	0.000000	0.000000	0.000000	24.550000	0.000000
F2	0.000000	0.000000	0.000000	0.000000	0.000000	24.990000	0.000000
F3	0.000000	0.000000	0.000000	0.000000	0.000000	25.780000	0.000000
F4	0.000000	0.000000	0.000000	0.000000	0.000000	26.350000	0.000000
F5	0.000000	0.000000	0.000000	0.000000	0.000000	27.100000	0.000000
G1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	27.850000
G2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	27.650000
G3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	28.110000
G4	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	28.560000
G5	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	28.810000

Grades for each employee title

In [44]:
```
title_mask = df.emp_title.value_counts()[:20].index.values
cm = sns.light_palette("green", as_cmap=True)
round(pd.crosstab(df[df['emp_title'].isin(title_mask)]['emp_title'],
                  df[df['emp_title'].isin(title_mask)]['sub_grade'],
                  normalize='index') * 100,2).style.background_gradient(cmap = cm)
```

Out[44]:

emp_title \ sub_grade	A1	A2	A3	A4	A5	B1	B2	B3	B4	B5	C1	C2	C3	C4
Accountant	3.430000	3.410000	3.450000	4.810000	5.220000	6.340000	6.180000	5.880000	6.180000	6.490000	6.590000	5.650000	5.560000	5.400000	4.8400
Director	6.830000	3.970000	3.970000	5.200000	5.380000	6.150000	5.740000	6.010000	6.350000	5.930000	6.600000	5.080000	5.340000	5.450000	4.2400
Driver	2.450000	2.280000	2.960000	3.750000	3.940000	5.420000	5.420000	5.440000	6.190000	6.670000	6.740000	6.320000	5.880000	5.960000	5.8400
Engineer	6.430000	4.440000	4.490000	5.260000	5.630000	5.980000	5.890000	6.120000	5.890000	5.500000	6.020000	5.600000	4.810000	4.830000	4.6500
General Manager	4.180000	3.190000	2.940000	3.970000	5.250000	5.880000	5.960000	5.730000	5.820000	6.380000	6.540000	6.120000	5.620000	5.560000	5.3400
Manager	3.850000	3.020000	3.280000	4.200000	4.950000	5.530000	5.490000	5.530000	5.990000	6.250000	6.480000	5.880000	5.990000	5.720000	5.3900
Office Manager	3.190000	3.150000	3.260000	4.270000	4.870000	5.430000	5.430000	5.270000	6.000000	6.640000	6.440000	5.580000	6.170000	5.710000	5.5600
Operations Manager	3.850000	3.080000	3.380000	4.110000	5.120000	5.890000	5.790000	5.320000	5.780000	6.020000	6.580000	6.200000	5.830000	5.740000	4.9900
Owner	4.560000	3.840000	3.580000	4.460000	4.810000	6.770000	6.240000	5.970000	6.140000	6.040000	6.550000	5.950000	5.200000	5.280000	4.8100
President	7.060000	4.560000	4.670000	5.850000	5.470000	6.480000	6.200000	5.590000	5.740000	5.840000	6.780000	5.180000	4.840000	4.800000	4.2000
Project Manager	5.660000	4.010000	3.500000	4.880000	5.700000	6.220000	5.840000	5.690000	5.840000	6.440000	6.260000	5.650000	5.340000	5.300000	4.7800
RN	4.680000	3.450000	3.520000	4.800000	5.220000	5.970000	6.000000	6.130000	6.200000	6.280000	6.050000	5.930000	5.460000	5.680000	4.8000
Registered Nurse	4.330000	3.570000	3.730000	4.700000	5.310000	5.780000	5.970000	6.030000	6.280000	6.270000	6.150000	5.400000	5.500000	5.770000	5.2800
Sales	3.770000	2.920000	3.210000	3.990000	4.880000	6.110000	5.820000	5.640000	6.440000	6.610000	6.610000	5.850000	6.000000	5.800000	5.1400
Supervisor	2.910000	2.530000	2.630000	3.550000	3.980000	4.960000	5.150000	5.510000	5.680000	6.780000	7.070000	5.710000	6.310000	5.840000	5.7700
Teacher	3.970000	3.280000	3.520000	4.420000	4.990000	6.100000	5.910000	5.970000	6.250000	6.390000	6.460000	5.770000	5.530000	5.610000	5.2100
Vice President	7.410000	4.720000	4.220000	5.410000	5.750000	5.990000	5.360000	5.790000	5.640000	5.750000	6.150000	5.460000	5.580000	4.720000	4.1900
manager	2.540000	2.320000	2.900000	3.460000	3.800000	5.570000	5.040000	5.080000	6.290000	6.630000	6.710000	6.360000	6.270000	5.580000	5.4800
owner	4.230000	3.360000	3.740000	4.110000	4.810000	6.620000	6.240000	5.800000	6.320000	5.910000	7.040000	5.780000	5.550000	5.170000	5.2700
teacher	3.650000	3.330000	2.970000	4.210000	5.330000	6.200000	6.220000	6.140000	6.230000	6.690000	6.680000	5.530000	5.900000	5.380000	5.2300

We can see that Director, Engineer, Vice President are the categories with highest incidence in Grade A. Analyzing this table we can get some insights about the profile of grades and professionals.

Interest rates according to employee lengths

In [45]:
```
df.groupby(["emp_length"])['int_rate'].mean().reset_index().T
```

Out[45]:

	0	1	2	3	4	5	6	7	8	9	10
emp_length	1 year	10+ years	2 years	3 years	4 years	5 years	6 years	7 years	8 years	9 years	< 1 year
int_rate	13.189791	13.002788	13.127222	13.120526	13.13112	13.141375	13.176806	13.19482	13.097373	13.07509	13.073689

The interest rate for employee lengths of 2 years is the highest and the lowest interest rate is for 9 years.

DTI

DTI is a ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower's self-reported monthly income.

In [46]:
```
cm = sns.light_palette("green", as_cmap=True)
round((pd.crosstab(df['loan_status'], df['purpose'],
                   values=df['dti'], aggfunc='mean')).fillna(0),
      2).style.background_gradient(cmap = cm)
```

Out[46]:

purpose loan_status	car	credit_card	debt_consolidation	educational	home_improvement	house	major_purchase	medical	moving	other	renew
Charged Off	17.470000	20.510000	20.820000	11.050000	17.790000	16.770000	16.400000	19.570000	18.440000	18.630000	
Current	16.430000	19.760000	20.280000	17.070000	18.270000	15.460000	16.460000	19.390000	18.500000	17.850000	
Default	13.930000	11.090000	21.580000	0.000000	7.500000	8.800000	12.250000	22.740000	0.000000	13.820000	
Does not meet the credit policy. Status:Charged Off	11.540000	15.140000	15.500000	13.660000	13.350000	8.610000	12.460000	14.960000	8.030000	14.120000	
Does not meet the credit policy. Status:Fully Paid	9.920000	15.700000	15.700000	10.190000	13.370000	10.240000	11.540000	15.160000	12.440000	12.680000	
Fully Paid	14.500000	18.060000	18.450000	11.270000	15.650000	14.080000	14.480000	17.080000	16.000000	16.550000	
In Grace Period	15.070000	21.100000	21.470000	0.000000	17.570000	15.140000	15.000000	18.250000	18.630000	18.130000	
Late (16-30 days)	18.010000	19.840000	20.110000	0.000000	18.550000	15.560000	15.800000	17.730000	18.370000	17.860000	
Late (31-120 days)	16.650000	20.730000	20.890000	0.000000	18.190000	14.220000	16.240000	19.500000	22.390000	18.050000	

DEALING WITH MISSING VALUES

6. Dealing with the missing values

In [47]:
```
df.isnull().sum()
```

Out[47]:
```
loan_amnt                   33
funded_amnt                 33
funded_amnt_inv             33
term                        33
int_rate                    33
                          ...
settlement_status        2226455
settlement_date          2226455
settlement_amount        2226455
settlement_percentage    2226455
settlement_term          2226455
Length: 124, dtype: int64
```

In [48]:
```
df.isnull().mean()
```

Out[48]:
```
loan_amnt               0.000015
funded_amnt             0.000015
funded_amnt_inv         0.000015
term                    0.000015
int_rate                0.000015
                            ...
settlement_status       0.984852
settlement_date         0.984852
settlement_amount       0.984852
settlement_percentage   0.984852
settlement_term         0.984852
Length: 124, dtype: float64
```

In [49]:
```
df.head()
```

Out[49]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verificatio
0	3600.0	3600.0	3600.0	36 months	13.99	123.03	C	C4	leadman	10+ years	MORTGAGE	55000.0	No
1	24700.0	24700.0	24700.0	36 months	11.99	820.28	C	C1	Engineer	10+ years	MORTGAGE	65000.0	No
2	20000.0	20000.0	20000.0	60 months	10.78	432.66	B	B4	truck driver	10+ years	MORTGAGE	63000.0	No
3	35000.0	35000.0	35000.0	60 months	14.85	829.90	C	C5	Information Systems Officer	10+ years	MORTGAGE	110000.0	Source
4	10400.0	10400.0	10400.0	60 months	22.45	289.91	F	F1	Contract Specialist	3 years	MORTGAGE	104433.0	Source

Dropping columns containing either 40% or more than 40% NaN values.

In [50]:
```
perc = 40.0
min_count =  int(((100-perc)/100)*df.shape[0] + 1)
mod_df = df.dropna( axis=1,
                    thresh=min_count)
print("Modified Dataframe : ")
print(mod_df)
```

```
Modified Dataframe :
         loan_amnt  funded_amnt  funded_amnt_inv        term  int_rate  \
0           3600.0       3600.0           3600.0   36 months     13.99
1          24700.0      24700.0          24700.0   36 months     11.99
2          20000.0      20000.0          20000.0   60 months     10.78
3          35000.0      35000.0          35000.0   60 months     14.85
4          10400.0      10400.0          10400.0   60 months     22.45
...            ...          ...              ...         ...       ...
2260696    40000.0      40000.0          40000.0   60 months     10.49
2260697    24000.0      24000.0          24000.0   60 months     14.49
2260698    14000.0      14000.0          14000.0   60 months     14.49
2260699        NaN          NaN              NaN         NaN       NaN
2260700        NaN          NaN              NaN         NaN       NaN

         installment grade sub_grade                  emp_title emp_length  \
0             123.03     C        C4                    leadman  10+ years
1             820.28     C        C1                   Engineer  10+ years
2             432.66     B        B4                truck driver  10+ years
3             829.90     C        C5  Information Systems Officer  10+ years
```

In [51]:
```
mod_df.isnull().mean()
```

Out[51]:
```
loan_amnt                   0.000015
funded_amnt                 0.000015
funded_amnt_inv             0.000015
term                        0.000015
int_rate                    0.000015
                              ...
total_bc_limit              0.022145
total_il_high_credit_limit  0.031101
hardship_flag               0.000015
disbursement_method         0.000015
debt_settlement_flag        0.000015
```

```
debt_settlement_flag          0.000015
Length: 101, dtype: float64
```

In [52]:
```
mod_df.head()
```

Out[52]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verificatio
0	3600.0	3600.0	3600.0	36 months	13.99	123.03	C	C4	leadman	10+ years	MORTGAGE	55000.0	No
1	24700.0	24700.0	24700.0	36 months	11.99	820.28	C	C1	Engineer	10+ years	MORTGAGE	65000.0	No
2	20000.0	20000.0	20000.0	60 months	10.78	432.66	B	B4	truck driver	10+ years	MORTGAGE	63000.0	No
3	35000.0	35000.0	35000.0	60 months	14.85	829.90	C	C5	Information Systems Officer	10+ years	MORTGAGE	110000.0	Source
4	10400.0	10400.0	10400.0	60 months	22.45	289.91	F	F1	Contract Specialist	3 years	MORTGAGE	104433.0	Source

BUSINESS CASE 1

Problem Statement: Build a model to predict the probability of default for the applicants on the lending club app. Follow the ML pipeline of data cleaning, feature engineering, model training, K fold validation, Model diagnostics, and model selection.Try out the whole suite of classifier models that are available in scikit-learn.
To predict if a borrower will clear his/ her loan, we chose the following variables:

Independent Variables:

- home_ownership : status provided by the borrower during registration or obtained from the credit report
- emp_length:Employment length in Years
- loan_amnt: the listed amount of loan applied for by the borrower,
- grade
- annual_inc: the self reported annual income provided by the borrower during registration
- int_rate:Interest Rate on the loan
- term: Number of payments on the loan (in months)
- fico_range_high: The upper boundry range the borowwer's FICO at loan originates belongs to

Dependent Variable:

- loan_status: Current state of the loan

Summary: In 2012, LendingClub reached a $1 billion loan milestone; the company facilitated these loans through three different vehicles: notes, certificates, and whole loan sales. Through its online standard program loan division, it offered these three or five-year unsecured personal loans to borrowers with a FICO range of at least 660 who met the other credit criteria.Applicants specified the amounts they wanted to borrow and provided information about their financial situation, employment, and intended use of the loans. Using the online platform, investors could browse notes and decide which ones to fund. The most common reasons for borrowing were debt consolidation, credit card debt refinance, and home renovation.LendingClub assigned grades ranging from A to G, which are further subdivided into sub-grades 1 to 5. For each entry, the data includes over 100 descriptive features such as application ID, credit score, loan amount, purpose, and so on.To predict new data with a reassuring level of confidence, the target variable is "loan status." Several models have been trained to classify a new loan application as default or repayment based on this target variable.

In [34]:
```
df1 = mod_df[['home_ownership','emp_length','loan_amnt', 'grade', 'annual_inc','loan_status','int_rate','term','fico_range_high']]
df1.head()
```

Out[34]:

	home_ownership	emp_length	loan_amnt	grade	annual_inc	loan_status	int_rate	term	fico_range_high
0	MORTGAGE	10+ years	3600.0	C	55000.0	Fully Paid	13.99	36 months	679.0
1	MORTGAGE	10+ years	24700.0	C	65000.0	Fully Paid	11.99	36 months	719.0
2	MORTGAGE	10+ years	20000.0	B	63000.0	Fully Paid	10.78	60 months	699.0
3	MORTGAGE	10+ years	35000.0	C	110000.0	Current	14.85	60 months	789.0
4	MORTGAGE	3 years	10400.0	F	104433.0	Fully Paid	22.45	60 months	699.0

In [35]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2260701 entries, 0 to 2260700
Data columns (total 9 columns):
 #   Column           Dtype
---  ------           -----
 0   home_ownership   object
 1   emp_length       object
 2   loan_amnt        float64
 3   grade            object
 4   annual_inc       float64
 5   loan_status      object
 6   int_rate         float64
 7   term             object
 8   fico_range_high  float64
dtypes: float64(4), object(5)
memory usage: 155.2+ MB
```

In [36]:
```
pd.value_counts(df1.loan_status)
```

Out[36]:
```
Fully Paid                                             1076751
Current                                                 878317
Charged Off                                             268559
Late (31-120 days)                                       21467
In Grace Period                                           8436
Late (16-30 days)                                         4349
Does not meet the credit policy. Status:Fully Paid        1988
Does not meet the credit policy. Status:Charged Off        761
Default                                                     40
Name: loan_status, dtype: int64
```

For analysis purpose, we will look into fully paid and charged off clients.

In [37]:
```
df1 = df1.loc[df1['loan_status'].isin(['Fully Paid','Charged Off'])]
pd.value_counts(df1.loan_status)
```

Out[37]:
```
Fully Paid     1076751
Charged Off     268559
Name: loan_status, dtype: int64
```

Classifying variables into categorical and numerical for further processes

In [38]:
```
cat_cols= ['grade','loan_status','home_ownership','emp_length','term']
num_cols=['loan_amnt', 'annual_inc','int_rate','fico_range_high']
```

- **Encoding categorical Values**
  - Working with various datasets, we discovered that some of the features are categorical; if we give the feature directly to our model, our model is unable to comprehend those feature variables. The incomprehensibility of categorical data to machines is well known. All independent and dependent variables, or input and output features, must be numerical for machines to function. To fit our data to the model, we must first encode any categorical variables in our data that are present.

- Models can only be used with numbers. In order for the machine to learn from the data and provide the appropriate model, it is important to convert the categorical values of the feature values into numerical ones. This process of transforming category data into numerical data is called Encoding.

7.1 Encoding Categorical Variables

In [39]:
```
le= LabelEncoder()

for col in cat_cols:
    df1[col] = le.fit_transform(df1[col].astype(str))

df1.head()
```

Out[39]:

	home_ownership	emp_length	loan_amnt	grade	annual_inc	loan_status	int_rate	term	fico_range_high
0	1	1	3600.0	2	55000.0	1	13.99	0	679.0
1	1	1	24700.0	2	65000.0	1	11.99	0	719.0
2	1	1	20000.0	1	63000.0	1	10.78	1	699.0
4	1	3	10400.0	5	104433.0	1	22.45	1	699.0
5	5	4	11950.0	2	34000.0	1	13.44	0	694.0

- **Imputing Missing Values**
  - Imputation maintains every instance by substituting missing data with an estimated value based on other available information. Following the imputed values for all missing items, the data set can be analyzed using methods that are typical for complete data.
  - Missing values can be imputed with a provided constant value, or using the statistics (mean, median or most frequent) of each column in which the missing values are located.

7.2 Imputing missing values

In [40]:
```
lr= LinearRegression()

imputer = IterativeImputer(random_state=42, estimator=lr, max_iter=10, n_nearest_features=2, imputation_order = 'roman')
df1_final = imputer.fit_transform(df1)

df1_final = pd.DataFrame(df1_final, columns = df1.columns)
```

In [41]:
```
df1_final.isnull().sum()
```

Out[41]:
```
home_ownership    0
emp_length        0
loan_amnt         0
grade             0
annual_inc        0
loan_status       0
int_rate          0
term              0
fico_range_high   0
dtype: int64
```

In [42]:
```
df1_final.sample(5)
```

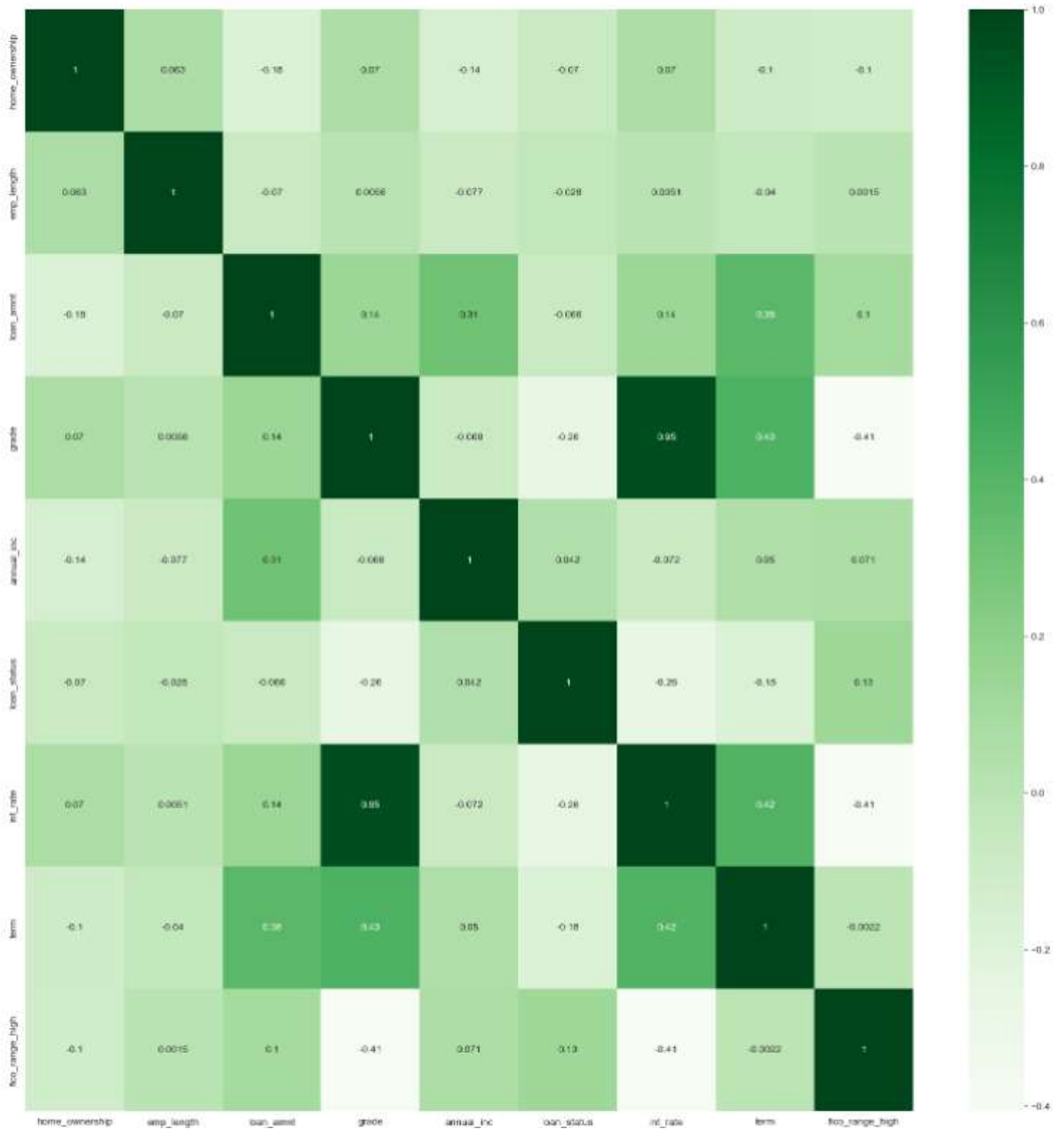
Out[42]:

	home_ownership	emp_length	loan_amnt	grade	annual_inc	loan_status	int_rate	term	fico_range_high
603391	1.0	1.0	32000.0	1.0	82000.0	1.0	10.75	0.0	679.0
1139328	1.0	0.0	6000.0	3.0	48360.0	1.0	18.75	0.0	694.0
424207	4.0	1.0	24000.0	4.0	66843.0	1.0	25.82	0.0	689.0
524380	5.0	2.0	20000.0	1.0	65000.0	1.0	10.91	0.0	689.0
1120503	1.0	9.0	10000.0	0.0	51000.0	1.0	7.90	0.0	754.0

- **Correlation Matrix**

```
In [43]: plt.figure(figsize=(20,20))
         sns.heatmap(df1_final.corr(), annot=True, cmap="Greens")
```

```
Out[43]: <AxesSubplot:>
```



- **Splitting into Train & Test**
  - When machine learning algorithms are used to make predictions on data that was not used to train the model, their performance is estimated using the train-test split technique.
  - It is a quick and simple process to carry out, and the outcomes let you compare the effectiveness of machine learning algorithms for your particular predictive modeling issue. While being straightforward to use and understand, there are some circumstances in which the method shouldn't be applied, such as when the dataset is tiny and further configuration is needed, as when it is used for classification and the dataset is unbalanced.
  - The test size we have used is 0.2 which indicates 80% of the data will be trained and 20% will be used on testing models to get the accuracy.

7.4 Spliting the data into test and train

In [45]:
```
X= df1_final[['loan_amnt', 'annual_inc','int_rate','fico_range_high',
              'grade','home_ownership','emp_length','term']]

y= df1_final['loan_status']
X_train, X_test,y_train,y_test= train_test_split(X,y, test_size=0.2,random_state=42)
```

- **Standardizing Numerical Data**
  - The process of transforming data into a standardized format so that users may process and evaluate it is known as data standardization.
  - For several reasons, data standardization is important. At the beginning, it aids in the establishment of distinct, clearly defined elements and attributes and offers a thorough catalog of your data. Understanding your data well is an essential first step, regardless of the insights or issues you're seeking to solve.

7.5 Standardizing numerical data

In [46]:
```
ind_num_cols= ['loan_amnt',  'annual_inc','int_rate','fico_range_high']
ss= StandardScaler()
X_train[ind_num_cols]= ss.fit_transform(X_train[ind_num_cols])
X_test[ind_num_cols]= ss.transform(X_test[ind_num_cols])
X_train
```

Out[46]:

	loan_amnt	annual_inc	int_rate	fico_range_high	grade	home_ownership	emp_length	term
690509	-0.736162	0.196593	-1.415759	0.747491	0.0	5.0	0.0	0.0
907486	1.214316	0.053675	2.433070	-0.665236	4.0	5.0	6.0	0.0
87973	1.240131	0.768265	-0.115337	0.433552	2.0	5.0	1.0	0.0
317614	-0.506694	-0.660915	0.018900	-0.822206	2.0	5.0	4.0	0.0
1210762	1.558518	2.268903	-0.471904	-0.351297	1.0	1.0	5.0	1.0
...	...	...	...	...	...	...	...	...
110268	0.755380	-0.232161	-0.471904	-0.979176	1.0	5.0	2.0	0.0
259178	-0.277226	0.339511	-1.122115	-0.351297	0.0	4.0	5.0	0.0
131932	0.411178	-0.446538	-0.681650	0.433552	1.0	4.0	8.0	0.0
671155	-1.011523	-0.546580	0.530679	-0.508267	3.0	5.0	10.0	0.0
121958	-0.277226	-0.046367	-1.661161	2.003250	0.0	1.0	9.0	0.0

1076248 rows × 8 columns

- **Training Data (All models)**
  - We took following models to train the data:
    - LogisticRegression
    - DecisionTreeClassifier
    - DecisionTreeRegressor
    - GaussianNB
    - LGBMClassifier
    - KNeighborsClassifier
  - Also, we planned to use RandomForestClassifier, SVC, CatBoostClassifier & XGBClassifier for training but these models were taking a lot of time to process, some models were getting crashed too. Hence, we omitted these models.

7.6 Training the data

```
In [47]: model_start = time.time()
```

```
In [48]: key= ['LogisticRegression', 'DecisionTreeClassifier', 'DecisionTreeRegressor', 'GaussianNB', 'LGBMClassifier', 'KNeighborsClassif

         value= [LogisticRegression(), DecisionTreeClassifier(), DecisionTreeRegressor(), GaussianNB(), LGBMClassifier(), KNeighborsClassi

         models= dict(zip(key, value))

         #key = ['RandomForestClassifier', 'SVC', 'CatBoostClassifier', 'XGBClassifier']
         #values = [RandomForestClassifier(), SVC(), CatBoostClassifier(verbose=False), XGBClassifier()]
         #models taking time: RandomForestClassifier, CatBoostClassifier, SVC
```

```
In [49]: models

Out[49]: {'LogisticRegression': LogisticRegression(),
          'DecisionTreeClassifier': DecisionTreeClassifier(),
          'DecisionTreeRegressor': DecisionTreeRegressor(),
          'GaussianNB': GaussianNB(),
          'LGBMClassifier': LGBMClassifier(),
          'KNeighborsClassifier': KNeighborsClassifier()}
```

- **Generating Scores and Accuracy**

```
In [50]: scores=[]
         for key,value in models.items():
             score= -1*cross_val_score(value, X,y, cv=5, scoring='neg_mean_absolute_error')
             scores.append(score)
             print(key, score.mean())
```

```
LogisticRegression 0.19968780429789415
DecisionTreeClassifier 0.31458028261144266
DecisionTreeRegressor 0.3068951341007968
GaussianNB 0.2099902624673867
LGBMClassifier 0.19942689788970572
KNeighborsClassifier 0.2306509280388907
```

```
In [51]: accuracy_scores=[]
         for key,value in models.items():
             value.fit(X_train,y_train)
             y_pred= value.predict(X_test)
             accuracy= value.score(X_test,y_test)

             accuracy_scores.append(accuracy)
             print(key, accuracy)
```

```
LogisticRegression 0.8015215823862158
DecisionTreeClassifier 0.6963673800090685
DecisionTreeRegressor -0.897112946423325
GaussianNB 0.7616274315956917
LGBMClassifier 0.8035917372204175
KNeighborsClassifier 0.7730151414915521
```

- **As it can be observed, LogisticRegression and LGBMClassifier gives accuracy of 0.8015215 and 0.8035915 respectively. Hence we used these two models for further processes.**

MODEL DIAGNOSTICS

7.7.1 Logistic Regression Model

In [54]:
```
logis= LogisticRegression(random_state=42)
logis.fit(X_train,y_train)
y_pred= logis.predict(X_test)

conf_m = confusion_matrix(y_test,y_pred)
print('The confusion Matrix: \n', conf_m)

clas_report = classification_report(y_test,y_pred)
print('\n\n\n Classification report: \n', clas_report)

p_pred = logis.predict_proba(X)
print('\n\n\n The predicted probability: \n',p_pred)

y_pred = logis.predict(X)
print('\n\n\n Model Prediction: \n',y_pred)

score_ = logis.score(X, y)
print('\n\n\n Model Score :\n',score)
```

```
The confusion Matrix:
 [[  3777  49677]
 [  3726 211882]]


 Classification report:
               precision    recall  f1-score   support

          0.0       0.50      0.07      0.12     53454
          1.0       0.81      0.98      0.89    215608

    accuracy                           0.80    269062
   macro avg       0.66      0.53      0.51    269062
weighted avg       0.75      0.80      0.74    269062



 The predicted probability:
 [[0. 1.]
 [0. 1.]
 [0. 1.]
 ...
 [0. 1.]
 [0. 1.]
 [0. 1.]]


 Model Prediction:
 [1. 1. 1. ... 1. 1. 1.]


 Model Score :
 [0.2262787  0.22998788 0.23114375 0.23147825 0.23436606]
```

Tuning the hyperparameters

- **Tuning Hyperparameter**
  - The control of a machine learning model's behavior requires hyperparameter adjustment. Our predicted model parameters will yield less-than-ideal outcomes if our hyperparameters aren't properly tuned to minimize the loss function. This indicates that our model has more flaws. The accuracy or confusion matrix will really be worse in practice.
  - As hyperparameters are unique to the algorithm, we are unable to determine their values from the data. To determine the model parameters, we employ hyperparameters. For a given data collection, various hyperparameter values result in various model parameter values.

Tuning the hyperparameters

In [55]:
```
# define dataset
X, y = make_blobs(n_samples=1000, centers=2, n_features=100, cluster_std=20)
# define models and parameters
model = LogisticRegression(verbose = False)
solvers = ['newton-cg', 'lbfgs', 'liblinear']
penalty = ['l2']
c_values = [100, 10, 1.0, 0.1, 0.01]
# define grid search
grid = dict(solver=solvers,penalty=penalty,C=c_values)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy',error_score=0)
logis_clf = grid_search.fit(X, y)
```

In [56]:
```
logis_clf.best_estimator_
```

Out[56]:
```
LogisticRegression(C=0.01, solver='liblinear', verbose=False)
```

In [57]:
```
logis_clf.best_score_
```

Out[57]:
```
0.9520000000000001
```

After tuning the hyperparameters, the model has been fitted to get the probability of the test data.

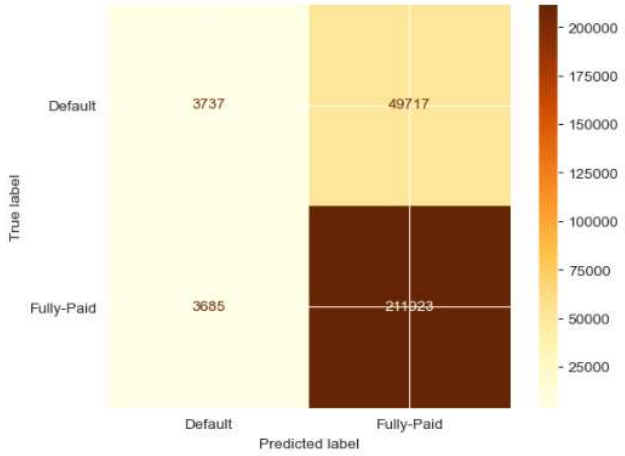
In [58]:
```
logis_clf.best_estimator_.fit(X_train,y_train)
y_pred= logis_clf.best_estimator_.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[  3737  49717]
 [  3685 211923]]
              precision    recall  f1-score   support

         0.0       0.50      0.07      0.12     53454
         1.0       0.81      0.98      0.89    215608

    accuracy                           0.80    269062
   macro avg       0.66      0.53      0.51    269062
weighted avg       0.75      0.80      0.74    269062
```

- **Plotting Confusion Matrix**

Plotting the confusion matrix for logistic regression model

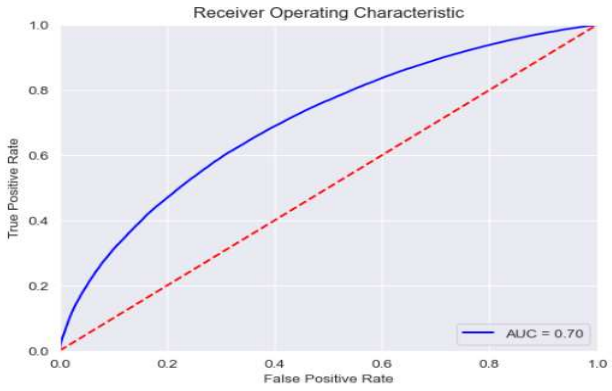
```
In [59]: disp = plot_confusion_matrix(
             logis_clf, X_test, y_test,
             cmap='YlOrBr', values_format='d',
             display_labels=['Default', 'Fully-Paid']
         )
```



- **Plotting ROC Curve**
  - An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:
    - True Positive Rate (TPR = TP / [TP + FN])
    - False Positive Rate (FPR = FP / [FP + TN])

Plotting the ROC curve for logistic regression model

```
In [60]: probs = logis_clf.predict_proba(X_test)
         preds = probs[:,1]
         fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
         roc_auc = metrics.auc(fpr, tpr)

         plt.title('Receiver Operating Characteristic')
         plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
         plt.legend(loc = 'lower right')
         plt.plot([0, 1], [0, 1],'r--')
         plt.xlim([0, 1])
         plt.ylim([0, 1])
         plt.ylabel('True Positive Rate')
         plt.xlabel('False Positive Rate')
         plt.show()
```



NOTE : Similar Process is followed for LGBMClassifier

- **Model Diagnostics for LGBM Classifier**

7.7.2 LightGBM Classification Model

In [61]:
```
lgbm= LGBMClassifier(random_state=42)
lgbm.fit(X_train,y_train)
y_pred= lgbm.predict(X_test)

print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[  3079  50375]
 [  2498 213110]]
              precision    recall  f1-score   support

         0.0       0.55      0.06      0.10     53454
         1.0       0.81      0.99      0.89    215608

    accuracy                           0.80    269062
   macro avg       0.68      0.52      0.50    269062
weighted avg       0.76      0.80      0.73    269062
```

- **Tuning Hyperparameter**

Tuning the hyperparameters

In [62]:
```
lgbm_params = {
 'colsample_bytree': [0.6,0.8,1.0],
 'learning_rate': [0.033,0.057,0.1],
 'max_depth': [3,4,5,6],
 'min_child_weight': [1,5,10,12],
 'n_estimators': [150,300,450,600],
 'num_leaves': [10,20,30],
 'subsample': [0.6,0.8,1.0]
 }

lgbm= LGBMClassifier(**lgbm_params)

lgbm
```

Out[62]:
```
LGBMClassifier(colsample_bytree=[0.6, 0.8, 1.0],
               learning_rate=[0.033, 0.057, 0.1], max_depth=[3, 4, 5, 6],
               min_child_weight=[1, 5, 10, 12],
               n_estimators=[150, 300, 450, 600], num_leaves=[10, 20, 30],
               subsample=[0.6, 0.8, 1.0])
```

In [63]:
```
lgbm_clf= RandomizedSearchCV(lgbm, param_distributions=lgbm_params, cv = 5, n_iter=5, verbose=1)

lgbm_clf.fit(X,y)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
```

Out[63]:
```
RandomizedSearchCV(cv=5,
                   estimator=LGBMClassifier(colsample_bytree=[0.6, 0.8, 1.0],
                                            learning_rate=[0.033, 0.057, 0.1],
                                            max_depth=[3, 4, 5, 6],
                                            min_child_weight=[1, 5, 10, 12],
                                            n_estimators=[150, 300, 450, 600],
                                            num_leaves=[10, 20, 30],
                                            subsample=[0.6, 0.8, 1.0]),
                   n_iter=5,
                   param_distributions={'colsample_bytree': [0.6, 0.8, 1.0],
                                        'learning_rate': [0.033, 0.057, 0.1],
                                        'max_depth': [3, 4, 5, 6],
                                        'min_child_weight': [1, 5, 10, 12],
                                        'n_estimators': [150, 300, 450, 600],
                                        'num_leaves': [10, 20, 30],
                                        'subsample': [0.6, 0.8, 1.0]},
                   verbose=1)
```

In [64]:
```
lgbm_clf.best_estimator_
```

Out[64]:
```
LGBMClassifier(colsample_bytree=0.8, learning_rate=0.033, max_depth=4,
               min_child_weight=1, n_estimators=450, num_leaves=20,
               subsample=0.6)
```

In [65]:
```
lgbm_clf.best_score_
```

Out[65]:
```
0.9349999999999999
```

After tuning the hyperparameters, the model has been fitted to get the probability of the test data.

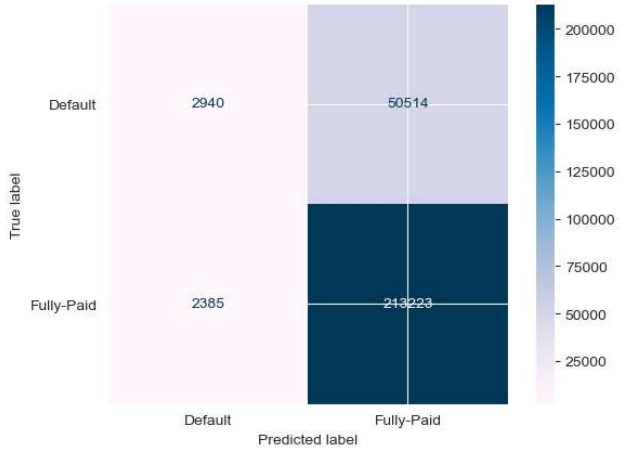
In [66]:
```
lgbm_clf.best_estimator_.fit(X_train,y_train)
y_pred= lgbm_clf.best_estimator_.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[  2940  50514]
 [  2385 213223]]
              precision    recall  f1-score   support

         0.0       0.55      0.06      0.10     53454
         1.0       0.81      0.99      0.89    215608

    accuracy                           0.80    269062
   macro avg       0.68      0.52      0.49    269062
weighted avg       0.76      0.80      0.73    269062
```

- **Plotting Confusion Matrix**

Plotting the confusion matrix for LightGBM classification model

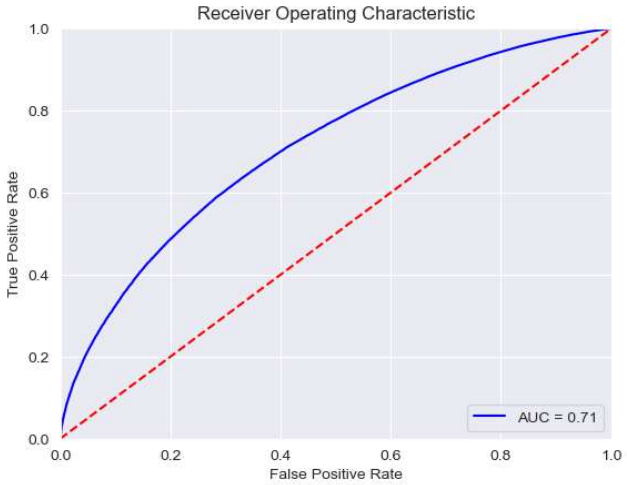
In [67]:
```
disp = plot_confusion_matrix(
    lgbm_clf, X_test, y_test,
    cmap='PuBu', values_format='d',
    display_labels=['Default', 'Fully-Paid']
)
```



- **Plotting ROC Curve**

Plotting the ROC curve for Lightgbm classification model

In [68]:
```
probs = lgbm_clf.predict_proba(X_test)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



MODELS COMPARISON

7.8 Model comparison

In [69]:
```
ml_models = {
    'Logistic Regression': logis_clf,
    'LightGBM': lgbm_clf,
}

for model in ml_models:
    print(f"{model.upper():{30}} roc_auc_score: {roc_auc_score(y_test, ml_models[model].predict(X_test)):.3f}")
```

```
LOGISTIC REGRESSION            roc_auc_score: 0.526
LIGHTGBM                       roc_auc_score: 0.522
```

From the model diagnostics and roc_auc score, it can be stated that the logistic regression model is better for predicting the default client in the lending club dataset.

BUSINESS CASE 2

Problem Statement: Defining a business case for building a linear regression model. Define Y and choose appropriate independent variables such that there should be a sound business reason to do this .Defining the use context for such a model clearly. Presenting the model and quantifying the benefits of the model.

Business Model: To predict the loan amount that can be borrowed.

To predict how much loan can be borrowed, we chose the following variables:

Independent Variables:
Term, interest rate, installment, employment length, home ownership, annual income, purpose, application type, and FICO range (low and high)

Dependent Variable:

loan_amnt: the listed amount of loan applied for by the borrower

Use Context: A customer has approached Lending Club and requested a loan. Lending Club will collect the following information from her/him: Employee history, homeownership, annual income, the reason for the loan, loan type, high and low fico range, and so on.Investors can analyze the loan amount that it can provide, the interest rate, and the monthly installments based on the data shared by a customer and classify it as either default or repay.Based on the independent variables mentioned above, the model is designed to calculate an estimated loan that can be given to a customer.

In [70]:
```
column = (mod_df.columns.values.tolist())
print(column)
```

```
['loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term', 'int_rate', 'installment', 'grade', 'sub_grade', 'emp_title', 'emp_leng
th', 'home_ownership', 'annual_inc', 'verification_status', 'issue_d', 'loan_status', 'pymnt_plan', 'url', 'purpose', 'title',
'zip_code', 'addr_state', 'dti', 'delinq_2yrs', 'earliest_cr_line', 'fico_range_low', 'fico_range_high', 'inq_last_6mths', 'ope
n_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_pym
nt_d', 'last_pymnt_amnt', 'last_credit_pull_d', 'last_fico_range_high', 'last_fico_range_low', 'collections_12_mths_ex_med', 'p
olicy_code', 'application_type', 'acc_now_delinq', 'tot_coll_amt', 'tot_cur_bal', 'open_acc_6m', 'open_act_il', 'open_il_12m',
'open_il_24m', 'open_rv_12m', 'open_rv_24m', 'all_util', 'total_rev_hi_lim', 'inq_fi', 'total_cu_tl', 'inq_last_12m', 'acc_open
_past_24mths', 'avg_cur_bal', 'bc_util', 'chargeoff_within_12_mths', 'delinq_amnt', 'mo_sin_old_il_acct', 'mo_sin_old_rev_tl_o
p', 'mo_sin_rcnt_rev_tl_op', 'mo_sin_rcnt_tl', 'mort_acc', 'mths_since_recent_bc', 'mths_since_recent_inq', 'num_accts_ever_120
_pd', 'num_actv_bc_tl', 'num_actv_rev_tl', 'num_bc_sats', 'num_bc_tl', 'num_il_tl', 'num_op_rev_tl', 'num_rev_accts', 'num_rev_
tl_bal_gt_0', 'num_sats', 'num_tl_120dpd_2m', 'num_tl_30dpd', 'num_tl_90g_dpd_24m', 'num_tl_op_past_12m', 'pct_tl_nvr_dlq', 'pe
rcent_bc_gt_75', 'pub_rec_bankruptcies', 'tax_liens', 'tot_hi_cred_lim', 'total_bal_ex_mort', 'total_bc_limit', 'total_il_high_
credit_limit', 'hardship_flag', 'disbursement_method', 'debt_settlement_flag']
```

After checking the definition of all the columns, some are being dropped as those are not related to the business model.

In [71]:
```
df2 = mod_df[['loan_amnt', 'term', 'int_rate', 'installment', 'application_type', 'emp_length', 'home_ownership', 'annual_inc', '
df2.head()
```

Out[71]:

	loan_amnt	term	int_rate	installment	application_type	emp_length	home_ownership	annual_inc	purpose	fico_range_low	fico_range_high
0	3600.0	36 months	13.99	123.03	Individual	10+ years	MORTGAGE	55000.0	debt_consolidation	675.0	679.0
1	24700.0	36 months	11.99	820.28	Individual	10+ years	MORTGAGE	65000.0	small_business	715.0	719.0
2	20000.0	60 months	10.78	432.66	Joint App	10+ years	MORTGAGE	63000.0	home_improvement	695.0	699.0
3	35000.0	60 months	14.85	829.90	Individual	10+ years	MORTGAGE	110000.0	debt_consolidation	785.0	789.0
4	10400.0	60 months	22.45	289.91	Individual	3 years	MORTGAGE	104433.0	major_purchase	695.0	699.0

In [72]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2260701 entries, 0 to 2260700
Data columns (total 11 columns):
 #   Column            Dtype
---  ------            -----
 0   loan_amnt         float64
 1   term              object
 2   int_rate          float64
 3   installment       float64
 4   application_type  object
 5   emp_length        object
 6   home_ownership    object
 7   annual_inc        float64
 8   purpose           object
 9   fico_range_low    float64
 10  fico_range_high   float64
dtypes: float64(6), object(5)
memory usage: 189.7+ MB
```

In [73]:
```
df2.isnull().sum()
```

Out[73]:
```
loan_amnt               33
term                    33
int_rate                33
installment             33
application_type        33
emp_length          146940
home_ownership          33
annual_inc              37
purpose                 33
fico_range_low          33
fico_range_high         33
dtype: int64
```

In [74]:
```
df2.isnull().mean()
```

Out[74]:
```
loan_amnt           0.000015
term                0.000015
int_rate            0.000015
installment         0.000015
application_type    0.000015
emp_length          0.064998
home_ownership      0.000015
annual_inc          0.000016
purpose             0.000015
fico_range_low      0.000015
fico_range_high     0.000015
dtype: float64
```

The columns have been divided into numerical and categorical for future analysis

In [75]:
```
num_cols= ['loan_amnt' ,'int_rate', 'installment', 'annual_inc', 'fico_range_low', 'fico_range_high']
cat_cols= df2.drop(['loan_amnt' ,'int_rate', 'installment', 'annual_inc', 'fico_range_low', 'fico_range_high'], axis=1).columns
```

In [76]:
```
cat_cols
```

Out[76]:
```
Index(['term', 'application_type', 'emp_length', 'home_ownership', 'purpose'], dtype='object')
```

In [77]:
```
pd.value_counts(df2.term)
```

Out[77]:
```
 36 months    1609754
 60 months     650914
Name: term, dtype: int64
```

In [78]:
```
pd.value_counts(df2.application_type)
```

Out[78]:
```
Individual    2139958
Joint App      120710
Name: application_type, dtype: int64
```

In [79]:
```
pd.value_counts(df2.emp_length)
```

Out[79]:
```
10+ years    748005
2 years      203677
< 1 year     189988
3 years      180753
1 year       148403
5 years      139698
4 years      136605
6 years      102628
7 years       92695
8 years       91914
9 years       79395
Name: emp_length, dtype: int64
```

In [80]:
```
pd.value_counts(df2.home_ownership)
```

Out[80]:
```
MORTGAGE    1111450
RENT         894929
OWN          253057
ANY             996
OTHER           182
NONE             54
Name: home_ownership, dtype: int64
```

- **Encoding categorical Values**

8.1 Encoding Categorical Variables

In [81]:
```
le= LabelEncoder()

for col in cat_cols:
    df2[col] = le.fit_transform(df2[col].astype(str))

df2.head()
```

Out[81]:

	loan_amnt	term	int_rate	installment	application_type	emp_length	home_ownership	annual_inc	purpose	fico_range_low	fico_range_high
0	3600.0	0	13.99	123.03	0	1	1	55000.0	2	675.0	679.0
1	24700.0	0	11.99	820.28	0	1	1	65000.0	12	715.0	719.0
2	20000.0	1	10.78	432.66	1	1	1	63000.0	4	695.0	699.0
3	35000.0	1	14.85	829.90	0	1	1	110000.0	2	785.0	789.0
4	10400.0	1	22.45	289.91	0	3	1	104433.0	6	695.0	699.0

- **Imputing Missing Values**

8.2 Imputing missing values

In [82]:
```
lr= LinearRegression()

imputer = IterativeImputer(random_state=42, estimator=lr, max_iter=10, n_nearest_features=2, imputation_order = 'roman')
df2_final = imputer.fit_transform(df2)

df2_final = pd.DataFrame(df2_final, columns = df2.columns)
```

In [83]:
```
df2_final.isnull().sum()
```

Out[83]:
```
loan_amnt           0
term                0
int_rate            0
installment         0
application_type    0
emp_length          0
home_ownership      0
annual_inc          0
purpose             0
fico_range_low      0
fico_range_high     0
dtype: int64
```

In [84]:
```
df2_final.sample(5)
```

Out[84]:

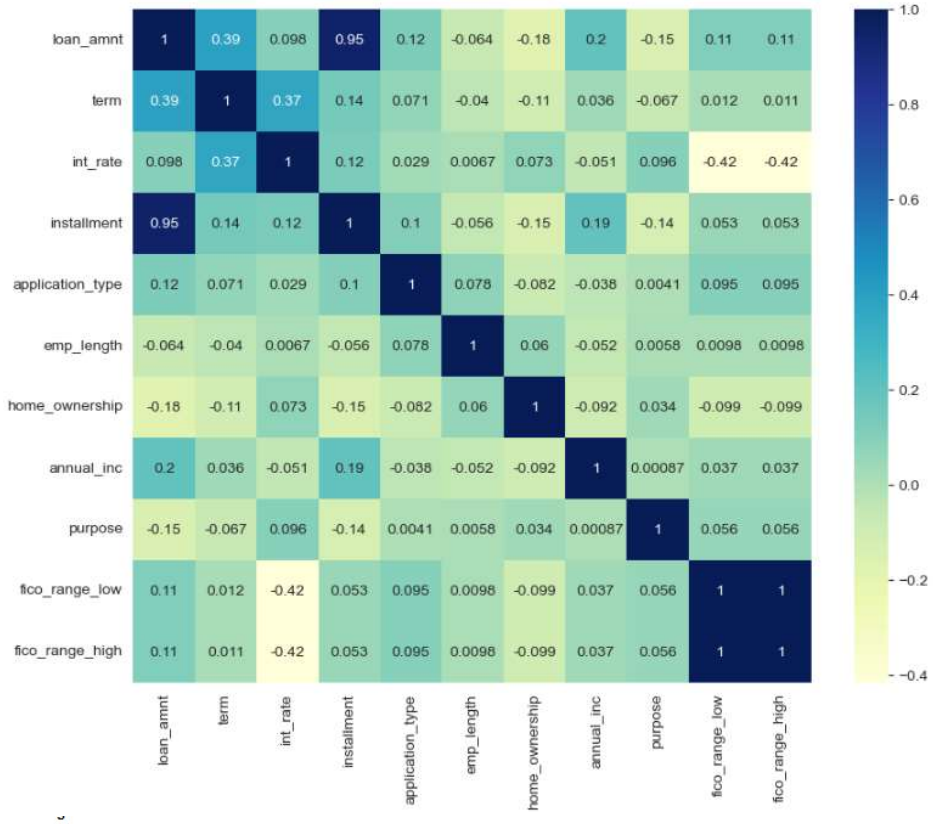
	loan_amnt	term	int_rate	installment	application_type	emp_length	home_ownership	annual_inc	purpose	fico_range_low	fico_range_high
878495	25000.0	0.0	11.99	830.24	0.0	1.0	1.0	92000.0	2.0	665.0	669.0
1908074	12000.0	0.0	8.90	381.04	0.0	3.0	1.0	71000.0	2.0	725.0	729.0
255813	11000.0	0.0	8.18	345.62	0.0	1.0	1.0	86000.0	2.0	670.0	674.0
1614405	6000.0	0.0	8.90	190.52	0.0	2.0	5.0	55000.0	2.0	715.0	719.0
1037633	16000.0	1.0	9.75	337.99	0.0	1.0	1.0	69000.0	2.0	705.0	709.0

- **Correlation Matrix**

8.3 Correlation Matrix

In [85]:
```
plt.figure(figsize=(10,8))
sns.heatmap(df2_final.corr(), annot=True, cmap="YlGnBu")
```

Out[85]: <AxesSubplot:>



In [86]:
```
cor = df2_final.corr()
cor.loc[:,:] = np.tril(cor, k=-1) # below main lower triangle of an array
cor = cor.stack()
cor[(cor > 0.1) | (cor < -0.1)]
```

Out[86]:
```
term              loan_amnt          0.393766
int_rate          term               0.372626
installment       loan_amnt          0.945625
                  term               0.139222
                  int_rate           0.123951
application_type  loan_amnt          0.118437
                  installment        0.102258
home_ownership    loan_amnt         -0.179576
                  term              -0.108580
                  installment       -0.145435
annual_inc        loan_amnt          0.197246
                  installment        0.190270
purpose           loan_amnt         -0.150363
                  installment       -0.135561
fico_range_low    loan_amnt          0.110585
                  int_rate          -0.415989
fico_range_high   loan_amnt          0.110584
                  int_rate          -0.415997
                  fico_range_low     0.999997
dtype: float64
```

- **Splitting into Train & Test**

8.4 Spliting the data into test and train

In [87]:
```
X= df2_final[['term', 'int_rate', 'installment', 'application_type', 'emp_length', 'home_ownership', 'annual_inc', 'purpose', 'fi
y= df2_final['loan_amnt']
X_train, X_test,y_train,y_test= train_test_split(X,y, test_size=0.2,random_state=42)
```

In [88]:
```
X_train
```

Out[88]:

	term	int_rate	installment	application_type	emp_length	home_ownership	annual_inc	purpose	fico_range_low	fico_range_high
1800887	0.0	14.33	446.40	0.0	5.0	1.0	83000.0	2.0	680.0	684.0
1130140	0.0	14.31	34.33	0.0	10.0	1.0	30000.0	10.0	720.0	724.0
459855	1.0	16.01	778.35	0.0	1.0	1.0	105000.0	2.0	710.0	714.0
283070	0.0	6.39	642.58	0.0	4.0	1.0	145000.0	1.0	695.0	699.0
1785392	0.0	12.99	404.27	0.0	2.0	5.0	55000.0	2.0	690.0	694.0
...	...	...	...	...	...	...	...	...	...	...
732180	1.0	12.99	568.70	0.0	1.0	5.0	67000.0	1.0	675.0	679.0
110268	0.0	7.89	1095.00	0.0	0.0	5.0	150000.0	1.0	720.0	724.0
1692743	1.0	11.49	527.71	0.0	1.0	5.0	90000.0	2.0	730.0	734.0
2229084	1.0	15.99	364.70	0.0	1.0	1.0	48000.0	2.0	715.0	719.0
2219110	0.0	11.49	553.92	0.0	9.0	5.0	112000.0	2.0	675.0	679.0

1808560 rows × 10 columns

In [89]:
```
y_train
```

Out[89]:
```
1800887    13000.0
1130140     1000.0
459855     32000.0
283070     21000.0
1785392    12000.0
             ...
732180     25000.0
110268     35000.0
1692743    24000.0
2229084    15000.0
2219110    16800.0
Name: loan_amnt, Length: 1808560, dtype: float64
```

- **Standardizing Numerical Data**

8.5 Standardizing numerical data

In [90]:
```
mod_num_cols= ['int_rate', 'installment', 'annual_inc', 'fico_range_low', 'fico_range_high']
ss= StandardScaler()
X_train[mod_num_cols]= ss.fit_transform(X_train[mod_num_cols])
X_test[mod_num_cols]= ss.transform(X_test[mod_num_cols])
X_train
```

Out[90]:

	term	int_rate	installment	application_type	emp_length	home_ownership	annual_inc	purpose	fico_range_low	fico_range_high
1800887	0.0	0.255814	0.002094	0.0	5.0	1.0	0.041501	2.0	-0.563046	-0.563027
1130140	0.0	0.251675	-1.540097	0.0	10.0	1.0	-0.399323	10.0	0.648830	0.648815
459855	1.0	0.603484	1.244432	0.0	1.0	1.0	0.224484	2.0	0.345861	0.345855
283070	0.0	-1.387344	0.736307	0.0	4.0	1.0	0.557181	1.0	-0.108593	-0.108586
1785392	0.0	-0.021495	-0.155580	0.0	2.0	5.0	-0.191387	2.0	-0.260077	-0.260066
...	...	...	...	...	...	...	...	...	...	...
732180	1.0	-0.021495	0.459807	0.0	1.0	5.0	-0.091578	1.0	-0.714531	-0.714507
110268	0.0	-1.076924	2.429510	0.0	0.0	5.0	0.598768	1.0	0.648830	0.648815
1692743	1.0	-0.331915	0.306400	0.0	1.0	5.0	0.099723	2.0	0.951800	0.951776
2229084	1.0	0.599345	-0.303672	0.0	1.0	1.0	-0.249609	2.0	0.497346	0.497335
2219110	0.0	-0.331915	0.404493	0.0	9.0	5.0	0.282706	2.0	-0.714531	-0.714507

1808560 rows × 10 columns

- **Fitting Linear Regression Model**

8.6 Linear Regression Model

Fitting The Model

In [91]:
```
lr.fit(X_train, y_train)
```

Out[91]:
```
LinearRegression()
```

In [92]:
```
print(lr.intercept_)
print(lr.coef_)
```

```
13129.437283064552
[ 6.40593985e+03 -1.22603625e+03  8.43452154e+03  2.50996009e+02
  2.93280909e-01 -6.61533700e+00  3.96623878e+01  2.82557459e+01
 -1.79658419e+04  1.79776136e+04]
```

In [93]:
```
coeff_df2 = pd.DataFrame(lr.coef_, X.columns, columns=['Coefficient'])
coeff_df2
```

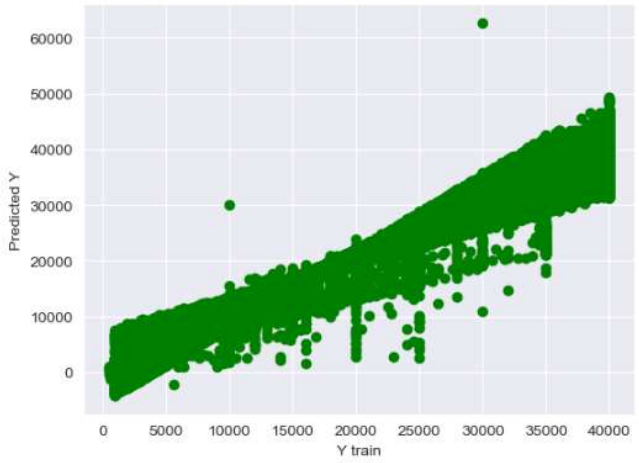
Out[93]:

	Coefficient
term	6405.939853
int_rate	-1226.036253
installment	8434.521540
application_type	250.996009
emp_length	0.293281
home_ownership	-6.615337
annual_inc	39.662388
purpose	28.255746
fico_range_low	-17965.841861
fico_range_high	17977.613616

In [94]:
```
y_pred_train = lr.predict(X_train)
```

Plotting the actual Y vs predicted Y for train dataset

In [95]:
```
plt.scatter(x = y_train, y = y_pred_train, color = "green")
plt.xlabel('Y train')
plt.ylabel('Predicted Y')
plt.show()
```



- **Model Evaluation**

Evaluating The Model

Looking at the metrics for the train data.

In [96]:
```
print('MAE:', metrics.mean_absolute_error(y_train, y_pred_train))
print('MSE:', metrics.mean_squared_error(y_train, y_pred_train))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_train, y_pred_train))),
print('R-squared: ',explained_variance_score(y_train, y_pred_train))
```

```
MAE: 847.8638429833705
MSE: 1721971.227166671
RMSE: 1312.239012972359
R-squared:  0.9796212239435175
```

Making Predictions

In [97]:
```
y_pred = lr.predict(X_test)
```

Comparing the actual output values for X_test with the predicted values.

In [98]:
```
df3 = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df3
```

Out[98]:

	Actual	Predicted
392949	32000.0	27905.665953
1273506	9600.0	9294.973589
324024	4000.0	4757.173943
2066630	6025.0	5857.825417
477199	25000.0	25911.088008
...	...	...
1205942	35000.0	33326.458249
860827	35000.0	34779.460899
267134	24000.0	24419.624979
867240	10000.0	10510.119486
1123633	25000.0	25111.512795

452141 rows × 2 columns

Evaluating The Algorithm

Looking at the metrics for the test data.

In [99]:
```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R-squared: ',explained_variance_score(y_test, y_pred))
```

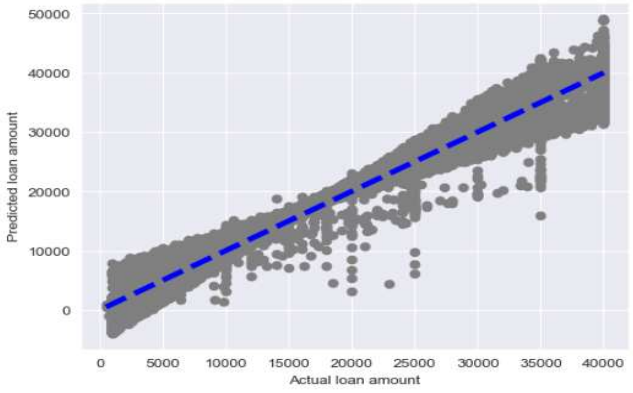
```
Mean Absolute Error: 845.591377543284
Mean Squared Error: 1711299.4271931706
Root Mean Squared Error: 1308.1664371146244
R-squared:  0.9797008030847885
```

- **Plotting Actual vs Predicted**

Plotting Actual vs Predicted Values

Plotting The Best Fit Line

In [100]:
```
fig, ax = plt.subplots()
ax.scatter(y_test, y_pred, color='grey')
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4, color='blue')
ax.set_xlabel('Actual loan amount')
ax.set_ylabel('Predicted loan amount')
plt.show()
```

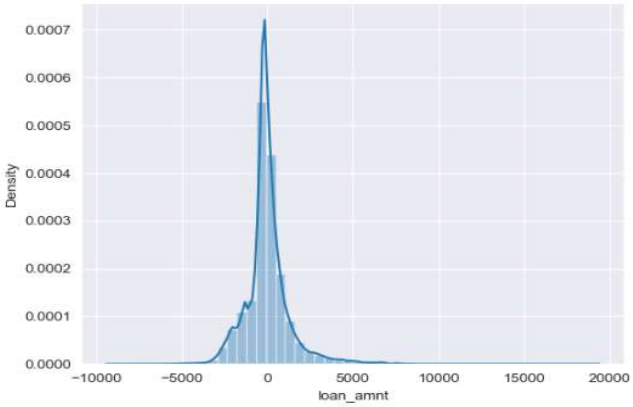


Checking the residuals

Plotting a histogram of the residuals to check if it is normally distributed.

In [101]:
```
sns.distplot((y_test-y_pred), bins = 50)
```

Out[101]:
```
<AxesSubplot:xlabel='loan_amnt', ylabel='Density'>
```



It can be seen that the histogram of the residuals is normally distributed.

In [102]:
```
prog_end = time.time()
```

In [103]:
```
print("Total MODEL EXECUTION time: ", model_total_time/60 , "Minutes")
```

```
Total MODEL EXECUTION time:  5.956804740428924 Minutes
```

In [104]:
```
total_prog_time = prog_end - prog_start

print("Total PROGRAM EXECUTION time: ", total_prog_time/60, "Minutes")
```

```
Total PROGRAM EXECUTION time:  9.879445731639862 Minutes
```

- **Limitation**
    - Few models take more time to execute, sometimes the kernel gets crashed. Hence, Higher RAM is preferred.

- **Future Scope**
  To increase the process time, the following processes can be executed on GPUs as it takes most of the time running in series:
    - Reading the dataset
    - Calculating model scores
    - Calculating Accuracy