Introduction: This case study focuses on LendingClub, a leading peer-to-peer lending platform based in San Francisco. LendingClub was the first company in the peer-to-peer lending industry to register its loan offerings as securities with the SEC and establish a secondary market for loan trading.

In this scenario, you are working for LendingClub, a company that provides loans to urban customers. When a loan application is received, the company needs to decide whether to approve the loan based on the applicant's profile.

The decision involves two types of risks:

Risk of loss if a creditworthy applicant is denied a loan, resulting in a missed business opportunity. Risk of financial loss if an applicant is likely to default on the loan but is approved. The provided dataset contains information about past loan applicants and whether they defaulted or not. The objective is to identify patterns in the data that indicate the likelihood of a person defaulting on a loan. These patterns can be used to make decisions such as denying the loan, reducing the loan amount, or charging a higher interest rate for riskier applicants.

Understanding the problem:

When a person applies for a loan, there are two possible outcomes:

Loan accepted: If the loan is approved, there are three scenarios: a. Fully paid: The applicant has successfully repaid the loan. b. Current: The applicant is in the process of repaying the loan and has not defaulted. c. Charged-off: The applicant has defaulted on the loan by not making timely repayments.

Loan rejected: The company rejects the loan application for various reasons, and as a result, there is no transactional history available for these applicants in the dataset.

Business objective:

LendingClub, as a prominent online loan marketplace, aims to provide easy access to loans at lower interest rates through their platform. However, like other lending companies, one of their biggest challenges is managing credit losses caused by borrowers who default on their loans.

Credit loss refers to the financial loss experienced by the lender when borrowers fail to repay their loans or abscond with the borrowed funds. In this case study, the focus is on identifying risky loan applicants who are more likely to default, as they are the primary source of credit losses for the company.

The objective is to use exploratory data analysis (EDA) and machine learning techniques to identify these high-risk applicants. By identifying the key factors or variables that strongly indicate loan default, the company can reduce the number of such loans and minimize credit losses. This knowledge can be utilized for portfolio management and risk assessment.

To gain a better understanding of the domain, it is recommended to conduct independent research on risk analytics, including exploring the types of variables that are significant in predicting loan default. This research will enhance the comprehension of risk assessment and aid in the analysis of the provided dataset.

#Hvplot is a Python library that provides a high-level interface to quickly and easily create interactive visualizations for your data. It is built on top of the popular plotting library, Bokeh, and integrates seamlessly with the Pandas and xarray libraries.

In [1034]:

```
#Importing all libraries
import pandas as pd
import numpy as np
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt
import hvplot.pandas

from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import MinMaxScaler

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier, AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB


pd.set_option('display.float', '{:.2f}'.format)#Pandas display option to format floating-point numbers w
pd.set_option('display.max_columns', 50)
pd.set_option('display.max_rows', 100)

```

In [1035]:

```
!pip install --upgrade scikit-learn


```

```
Requirement already satisfied: scikit-learn in ./anaconda3/lib/python3.10/site-packages (1.3.
0)
Requirement already satisfied: threadpoolctl>=2.0.0 in ./anaconda3/lib/python3.10/site-package
s (from scikit-learn) (2.2.0)
Requirement already satisfied: numpy>=1.17.3 in ./anaconda3/lib/python3.10/site-packages (from
scikit-learn) (1.23.5)
Requirement already satisfied: joblib>=1.1.1 in ./anaconda3/lib/python3.10/site-packages (from
scikit-learn) (1.1.1)
Requirement already satisfied: scipy>=1.5.0 in ./anaconda3/lib/python3.10/site-packages (from
scikit-learn) (1.10.0)
```

In [1036]:

```
from sklearn.metrics import (
    accuracy_score, confusion_matrix, classification_report,
    roc_auc_score, roc_curve, auc
)
from sklearn.metrics import ConfusionMatrixDisplay, RocCurveDisplay
```

In [1037]:

```
sns.set(style="darkgrid")
import warnings
warnings.filterwarnings('ignore')
```

In [1038]:

```
import os
current_directory =os.getcwd()
current_directory
```

Out[1038]:

```
'/Users/myyntiimac'
```

In [1039]:

```
df=pd.read_csv("/Users/myyntiimac/Desktop/lc_loan.csv")
df.head()
```

Out[1039]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	emp_title
0	1077501	1296599	5000.00	5000.00	4975.00	36 months	10.65	162.87	B	B2	NaN
1	1077430	1314167	2500.00	2500.00	2500.00	60 months	15.27	59.83	C	C4	Ryder
2	1077175	1313524	2400.00	2400.00	2400.00	36 months	15.96	84.33	C	C5	NaN
3	1076863	1277178	10000.00	10000.00	10000.00	36 months	13.49	339.31	C	C1	AIR RESOURCES BOARD
4	1075358	1311748	3000.00	3000.00	3000.00	60 months	12.69	67.79	B	B5	University Medical Group

5 rows × 74 columns

In [1040]:

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 887379 entries, 0 to 887378
Data columns (total 74 columns):
 #   Column                        Non-Null Count   Dtype  
---  ------                        --------------   -----  
 0   id                            887379 non-null  int64  
 1   member_id                     887379 non-null  int64  
 2   loan_amnt                     887379 non-null  float64
 3   funded_amnt                   887379 non-null  float64
 4   funded_amnt_inv               887379 non-null  float64
 5   term                          887379 non-null  object 
 6   int_rate                      887379 non-null  float64
 7   installment                   887379 non-null  float64
 8   grade                         887379 non-null  object 
 9   sub_grade                     887379 non-null  object 
 10  emp_title                     835917 non-null  object 
 11  emp_length                    842554 non-null  object 
 12  home_ownership                887379 non-null  object 
 13  annual_inc                    887375 non-null  float64
 14  verification_status           887379 non-null  object 
 15  issue_d                       887379 non-null  object 
 16  loan_status                   887379 non-null  object 
 17  pymnt_plan                    887379 non-null  object 
 18  url                           887379 non-null  object 
 19  desc                          126028 non-null  object 
 20  purpose                       887379 non-null  object 
 21  title                         887227 non-null  object 
 22  zip_code                      887379 non-null  object 
 23  addr_state                    887379 non-null  object 
 24  dti                           887379 non-null  float64
 25  delinq_2yrs                   887350 non-null  float64
 26  earliest_cr_line              887350 non-null  object 
 27  inq_last_6mths                887350 non-null  float64
 28  mths_since_last_delinq        433067 non-null  float64
 29  mths_since_last_record        137053 non-null  float64
 30  open_acc                      887350 non-null  float64
 31  pub_rec                       887350 non-null  float64
 32  revol_bal                     887379 non-null  float64
 33  revol_util                    886877 non-null  float64
 34  total_acc                     887350 non-null  float64
 35  initial_list_status           887379 non-null  object 
 36  out_prncp                     887379 non-null  float64
 37  out_prncp_inv                 887379 non-null  float64
 38  total_pymnt                   887379 non-null  float64
 39  total_pymnt_inv               887379 non-null  float64
 40  total_rec_prncp               887379 non-null  float64
 41  total_rec_int                 887379 non-null  float64
 42  total_rec_late_fee            887379 non-null  float64
 43  recoveries                    887379 non-null  float64
 44  collection_recovery_fee       887379 non-null  float64
 45  last_pymnt_d                  869720 non-null  object 
 46  last_pymnt_amnt               887379 non-null  float64
 47  next_pymnt_d                  634408 non-null  object 
 48  last_credit_pull_d            887326 non-null  object 
 49  collections_12_mths_ex_med    887234 non-null  float64
 50  mths_since_last_major_derog   221703 non-null  float64
 51  policy_code                   887379 non-null  float64
 52  application_type              887379 non-null  object 
 53  annual_inc_joint              511 non-null     float64
 54  dti_joint                     509 non-null     float64
 55  verification_status_joint     511 non-null     object 
 56  acc_now_delinq                887350 non-null  float64
 57  tot_coll_amt                  817103 non-null  float64
 58  tot_cur_bal                   817103 non-null  float64
 59  open_acc_6m                   21372 non-null   float64
 60  open_il_6m                    21372 non-null   float64
 61  open_il_12m                   21372 non-null   float64
 62  open_il_24m                   21372 non-null   float64
 63  mths_since_rcnt_il            20810 non-null   float64
 64  total_bal_il                  21372 non-null   float64
 65  il_util                       18617 non-null   float64
 66  open_rv_12m                   21372 non-null   float64
 67  open_rv_24m                   21372 non-null   float64
 68  max_bal_bc                    21372 non-null   float64
 69  all_util                      21372 non-null   float64
 70  total_rev_hi_lim              817103 non-null  float64
 71  inq_fi                        21372 non-null   float64
 72  total_cu_tl                   21372 non-null   float64
```

```
 73  inq_last_12m                21372 non-null   float64
dtypes: float64(49), int64(2), object(23)
memory usage: 501.0+ MB
```

```
# Checking and Removing the variable whic have more than 40%
```

In [1041]:

```
1 df.isnull().any()
```

Out[1041]:

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

In [1042]:

```
1 df.isnull().sum().sum()
```

Out[1042]:

```
17998490
```

In [1043]:

```
1 df.shape
```

Out[1043]:

```
(887379, 74)
```

In [1044]:

```
#Data cleaning and manipulation
null_percentages = df.isnull().mean() * 100
null_percentages
```

Out[1044]:

```
id                             0.00
member_id                      0.00
loan_amnt                      0.00
funded_amnt                    0.00
funded_amnt_inv                0.00
term                           0.00
int_rate                       0.00
installment                    0.00
grade                          0.00
sub_grade                      0.00
emp_title                      5.80
emp_length                     5.05
home_ownership                 0.00
annual_inc                     0.00
verification_status            0.00
issue_d                        0.00
loan_status                    0.00
pymnt_plan                     0.00
url                            0.00
desc                          85.80
purpose                        0.00
title                          0.02
zip_code                       0.00
addr_state                     0.00
dti                            0.00
delinq_2yrs                    0.00
earliest_cr_line               0.00
inq_last_6mths                 0.00
mths_since_last_delinq        51.20
mths_since_last_record        84.56
open_acc                       0.00
pub_rec                        0.00
revol_bal                      0.00
revol_util                     0.06
total_acc                      0.00
initial_list_status            0.00
out_prncp                      0.00
out_prncp_inv                  0.00
total_pymnt                    0.00
total_pymnt_inv                0.00
total_rec_prncp                0.00
total_rec_int                  0.00
total_rec_late_fee             0.00
recoveries                     0.00
collection_recovery_fee        0.00
last_pymnt_d                   1.99
last_pymnt_amnt                0.00
next_pymnt_d                  28.51
last_credit_pull_d             0.01
collections_12_mths_ex_med     0.02
mths_since_last_major_derog   75.02
policy_code                    0.00
application_type               0.00
annual_inc_joint              99.94
dti_joint                     99.94
verification_status_joint     99.94
acc_now_delinq                 0.00
tot_coll_amt                   7.92
tot_cur_bal                    7.92
open_acc_6m                   97.59
open_il_6m                    97.59
open_il_12m                   97.59
open_il_24m                   97.59
mths_since_rcnt_il            97.65
total_bal_il                  97.59
il_util                       97.90
open_rv_12m                   97.59
open_rv_24m                   97.59
max_bal_bc                    97.59
all_util                      97.59
total_rev_hi_lim               7.92
inq_fi                        97.59
total_cu_tl                   97.59
inq_last_12m                  97.59
dtype: float64
```

In [1045]:

```
#Let's plot the columns vs missing value % with 40% being the cut-off marks
# before that adds your null_percentages as column to df
null_percentages = (df.isnull().mean() * 100).reset_index()
null_percentages
```

3	funded_amnt	0.00
4	funded_amnt_inv	0.00
5	term	0.00
6	int_rate	0.00
7	installment	0.00
8	grade	0.00
9	sub_grade	0.00
10	emp_title	5.80
11	emp_length	5.05
12	home_ownership	0.00
13	annual_inc	0.00
14	verification_status	0.00

In [1046]:

```
#creates a DataFrame named null_percentageDF to store the null values percentage of each column in the 
null_percentageDF = pd.DataFrame((df.isnull().sum())*100/df.shape[0]).reset_index()
null_percentageDF
```

61	open_il_12m	97.59
62	open_il_24m	97.59
63	mths_since_rcnt_il	97.65
64	total_bal_il	97.59
65	il_util	97.90
66	open_rv_12m	97.59
67	open_rv_24m	97.59
68	max_bal_bc	97.59
69	all_util	97.59
70	total_rev_hi_lim	7.92
71	inq_fi	97.59
72	total_cu_tl	97.59

In [1047]:

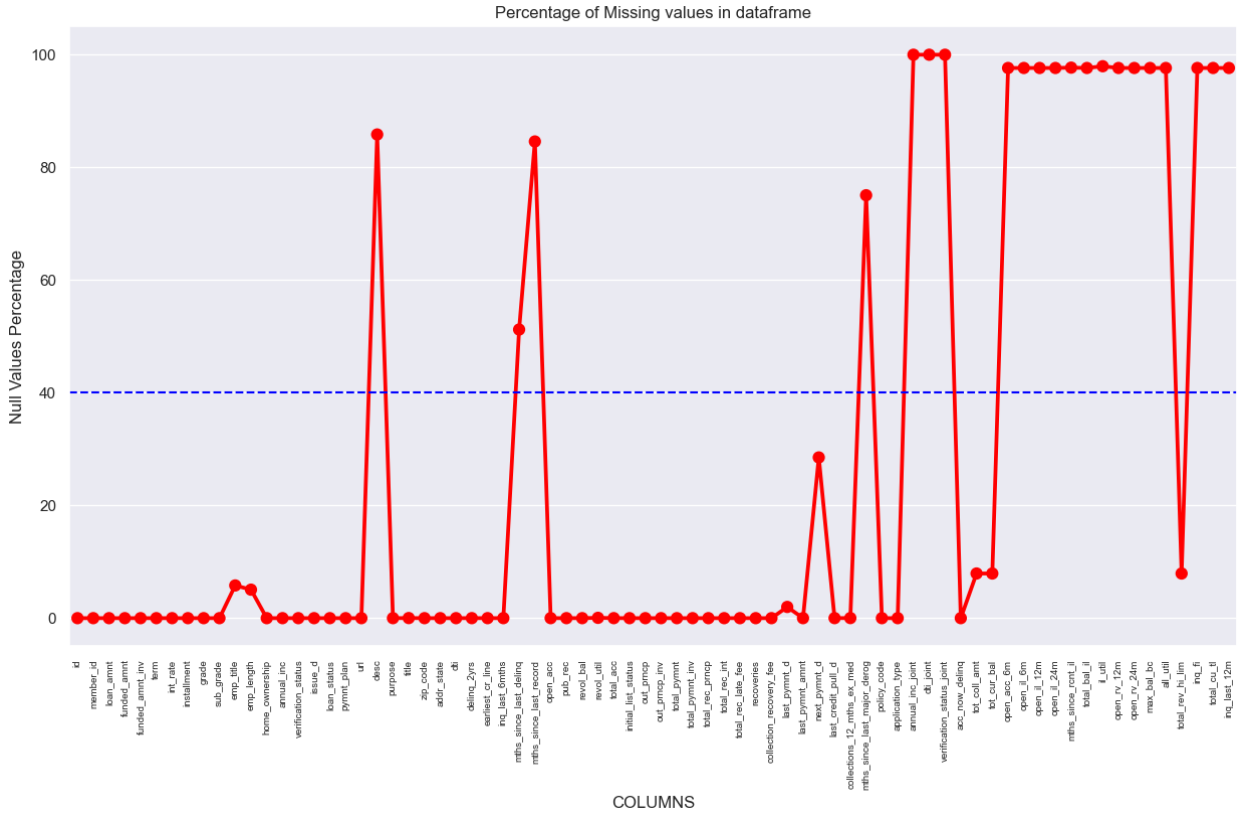
```
# asign the new column name to derived calculation of null in newly generated dataframe
null_percentageDF.columns = ['Column Name', 'Null Values Percentage']
null_percentageDF.head()
```

Out[1047]:

	Column Name	Null Values Percentage
0	id	0.00
1	member_id	0.00
2	loan_amnt	0.00
3	funded_amnt	0.00
4	funded_amnt_inv	0.00

In [1048]:

```
#no visualize the null percentage value against column name
fig = plt.figure(figsize=(15,8))#define figure so that you can manipulate figure
ax = sns.pointplot(x="Column Name", y="Null Values Percentage", data=null_percentageDF, color='red')
plt.xticks(rotation=90, fontsize=7)#for better visibility in x-axies
ax.axhline(40, ls='--', color='blue')#ccuttoff line, parameter ls like -- and color blue
plt.title("Percentage of Missing values in dataframe")
plt.ylabel("Null Values Percentage")
plt.xlabel("COLUMNS")
plt.show()
```



In [1049]:

```
# Filter and assign to varaible nullcol_40_df
nullcol_40_df = null_percentageDF[null_percentageDF["Null Values Percentage"]>=40]
nullcol_40_df
```

Out[1049]:

	Column Name	Null Values Percentage
19	desc	85.80
28	mths_since_last_delinq	51.20
29	mths_since_last_record	84.56
50	mths_since_last_major_derog	75.02
53	annual_inc_joint	99.94
54	dti_joint	99.94
55	verification_status_joint	99.94
59	open_acc_6m	97.59
60	open_il_6m	97.59
61	open_il_12m	97.59
62	open_il_24m	97.59
63	mths_since_rcnt_il	97.65
64	total_bal_il	97.59
65	il_util	97.90
66	open_rv_12m	97.59
67	open_rv_24m	97.59
68	max_bal_bc	97.59
69	all_util	97.59
71	inq_fi	97.59
72	total_cu_tl	97.59
73	inq_last_12m	97.59

In [1050]:

```
len(nullcol_40_df)
```

Out[1050]:

```
21
```

In [1051]:

```
nullcol_not40_df = null_percentageDF[null_percentageDF["Null Values Percentage"]<40]
nullcol_not40_df
```

Out[1051]:

	Column Name	Null Values Percentage
0	id	0.00
1	member_id	0.00
2	loan_amnt	0.00
3	funded_amnt	0.00
4	funded_amnt_inv	0.00
5	term	0.00
6	int_rate	0.00
7	installment	0.00
8	grade	0.00
9	sub_grade	0.00
10	emp_title	5.80
11	emp_length	5.05
12	home_ownership	0.00
13	annual_inc	0.00
14	verification_status	0.00
15	issue_d	0.00
16	loan_status	0.00
17	pymnt_plan	0.00
18	url	0.00
20	purpose	0.00
21	title	0.02
22	zip_code	0.00
23	addr_state	0.00
24	dti	0.00
25	delinq_2yrs	0.00
26	earliest_cr_line	0.00
27	inq_last_6mths	0.00
30	open_acc	0.00
31	pub_rec	0.00
32	revol_bal	0.00
33	revol_util	0.06
34	total_acc	0.00
35	initial_list_status	0.00
36	out_prncp	0.00
37	out_prncp_inv	0.00
38	total_pymnt	0.00
39	total_pymnt_inv	0.00
40	total_rec_prncp	0.00
41	total_rec_int	0.00
42	total_rec_late_fee	0.00
43	recoveries	0.00
44	collection_recovery_fee	0.00
45	last_pymnt_d	1.99
46	last_pymnt_amnt	0.00
47	next_pymnt_d	28.51
48	last_credit_pull_d	0.01

	Column Name	Null Values Percentage
49	collections_12_mths_ex_med	0.02
51	policy_code	0.00
52	application_type	0.00
56	acc_now_delinq	0.00
57	tot_coll_amt	7.92
58	tot_cur_bal	7.92
70	total_rev_hi_lim	7.92

In [1052]:
```
nullcol_not40_df["Column Name"].to_list()
```

Out[1052]:

```
['id',
 'member_id',
 'loan_amnt',
 'funded_amnt',
 'funded_amnt_inv',
 'term',
 'int_rate',
 'installment',
 'grade',
 'sub_grade',
 'emp_title',
 'emp_length',
 'home_ownership',
 'annual_inc',
 'verification_status',
 'issue_d',
 'loan_status',
 'pymnt_plan',
 'url',
 'purpose',
 'title',
 'zip_code',
 'addr_state',
 'dti',
 'delinq_2yrs',
 'earliest_cr_line',
 'inq_last_6mths',
 'open_acc',
 'pub_rec',
 'revol_bal',
 'revol_util',
 'total_acc',
 'initial_list_status',
 'out_prncp',
 'out_prncp_inv',
 'total_pymnt',
 'total_pymnt_inv',
 'total_rec_prncp',
 'total_rec_int',
 'total_rec_late_fee',
 'recoveries',
 'collection_recovery_fee',
 'last_pymnt_d',
 'last_pymnt_amnt',
 'next_pymnt_d',
 'last_credit_pull_d',
 'collections_12_mths_ex_med',
 'policy_code',
 'application_type',
 'acc_now_delinq',
 'tot_coll_amt',
 'tot_cur_bal',
 'total_rev_hi_lim']
```

In [ ]:
```
# create a datframe which have less than 40% null values
```

In [1053]:

```
df1=df[['id',
 'member_id',
 'loan_amnt',
 'funded_amnt',
 'funded_amnt_inv',
 'term',
 'int_rate',
 'installment',
 'grade',
 'sub_grade',
 'emp_title',
 'emp_length',
 'home_ownership',
 'annual_inc',
 'verification_status',
 'issue_d',
 'loan_status',
 'pymnt_plan',
 'url',
 'purpose',
 'title',
 'zip_code',
 'addr_state',
 'dti',
 'delinq_2yrs',
 'earliest_cr_line',
 'inq_last_6mths',
 'open_acc',
 'pub_rec',
 'revol_bal',
 'revol_util',
 'total_acc',
 'initial_list_status',
 'out_prncp',
 'out_prncp_inv',
 'total_pymnt',
 'total_pymnt_inv',
 'total_rec_prncp',
 'total_rec_int',
 'total_rec_late_fee',
 'recoveries',
 'collection_recovery_fee',
 'last_pymnt_d',
 'last_pymnt_amnt',
 'next_pymnt_d',
 'last_credit_pull_d',
 'collections_12_mths_ex_med',
 'policy_code',
 'application_type',
 'acc_now_delinq',
 'tot_coll_amt',
 'tot_cur_bal',
 'total_rev_hi_lim']]
```

In [1054]:

```
df1.head()
```

Out[1054]:

ership	annual_inc	verification_status	issue_d	loan_status	pymnt_plan	url	purpose
RENT	24000.00	Verified	Dec-2011	Fully Paid	n	https://www.lendingclub.com/browse/loanDetail....	credit_card
RENT	30000.00	Source Verified	Dec-2011	Charged Off	n	https://www.lendingclub.com/browse/loanDetail....	car
RENT	12252.00	Not Verified	Dec-2011	Fully Paid	n	https://www.lendingclub.com/browse/loanDetail....	small_business
RENT	49200.00	Source Verified	Dec-2011	Fully Paid	n	https://www.lendingclub.com/browse/loanDetail....	other
RENT	80000.00	Source Verified	Dec-2011	Current	n	https://www.lendingclub.com/browse/loanDetail....	other

In [1055]:

```
# Define the target variable values to filter Where loan status only contain charge off and fully paid
target_values = ['Charged Off', 'Fully Paid']

# Filter the DataFrame based on the target variable values
df1 = df1[df1['loan_status'].isin(target_values)]

# Print the filtered DataFrame
print(df1)
```

```
887351  36743377   39486112    4200.00      4200.00          4200.00
887364  36231718   38943165   10775.00     10775.00         10775.00
887366  36241316   38952731    6225.00      6225.00          6225.00
887369  36421485   39142898    4000.00      4000.00          4000.00
887371  36260758   38972123   10850.00     10850.00         10850.00

              term  int_rate  installment grade sub_grade  \
0        36 months     10.65       162.87     B        B2
1        60 months     15.27        59.83     C        C4
2        36 months     15.96        84.33     C        C5
3        36 months     13.49       339.31     C        C1
5        36 months      7.90       156.46     A        A4
...            ...       ...          ...   ...       ...
887351   36 months     15.99       147.64     D        D2
887364   36 months      6.03       327.95     A        A1
887366   36 months     16.49       220.37     D        D3
887369   36 months      8.67       126.59     B        B1
887371   36 months     19.24       399.04     E        E2

                    emp_title emp_length home_ownership  annual_inc  \
```

In [1056]:

```
# Transfer loan_status variable to numerical
# Create an instance of LabelEncoder
label_encoder = LabelEncoder()

# Fit the label encoder on the target variable
encoded_target = label_encoder.fit_transform(df1['loan_status'])

# Replace the original target variable with the encoded values
df1['loan_status'] = encoded_target

# Print the updated DataFrame
print(df1)
```

```
             id  member_id  loan_amnt  funded_amnt  funded_amnt_inv  \
0       1077501    1296599    5000.00      5000.00          4975.00
1       1077430    1314167    2500.00      2500.00          2500.00
2       1077175    1313524    2400.00      2400.00          2400.00
3       1076863    1277178   10000.00     10000.00         10000.00
5       1075269    1311441    5000.00      5000.00          5000.00
...         ...        ...        ...          ...              ...
887351  36743377   39486112    4200.00      4200.00          4200.00
887364  36231718   38943165   10775.00     10775.00         10775.00
887366  36241316   38952731    6225.00      6225.00          6225.00
887369  36421485   39142898    4000.00      4000.00          4000.00
887371  36260758   38972123   10850.00     10850.00         10850.00

             term  int_rate  installment grade sub_grade  \
0       36 months     10.65       162.87     B        B2
1       60 months     15.27        59.83     C        C4
2       36 months     15.96        84.33     C        C5
3       36 months     13.49       339.31     C        C1
5       36 months      7.90       156.46     A        A4
```

In [1057]:

```
from scipy.stats import f_oneway

```

In [1058]:

```
# Create an empty list to store the results
anova_results = []

# Iterate over each numerical variable in the DataFrame
for column in df1.select_dtypes(include=['float', 'int']):
    numerical_var = df1[column]
    f_stat, p_value = f_oneway(numerical_var, df1['loan_status'])
    anova_results.append((column, f_stat, p_value))

# Create a DataFrame to display the ANOVA results
anova_df = pd.DataFrame(anova_results, columns=['Numerical Variable', 'F-statistic', 'p-value'])

# Sort the DataFrame by p-value in ascending order
anova_df.sort_values('p-value', inplace=True)

# Print the ANOVA results
print(anova_df)
```

	Numerical Variable	F-statistic	p-value
0	id	174880.03	0.00
26	last_pymnt_amnt	194738.19	0.00
25	collection_recovery_fee	4935.31	0.00
24	recoveries	11357.76	0.00
22	total_rec_int	224516.93	0.00
21	total_rec_prncp	481269.11	0.00
20	total_pymnt_inv	513388.64	0.00
19	total_pymnt	520882.33	0.00
18	out_prncp_inv	1161326.19	0.00
17	out_prncp	1161326.19	0.00
16	total_acc	1073756.62	0.00
28	policy_code	55104.09	0.00
14	revol_bal	165653.51	0.00
12	open_acc	1076083.65	0.00
10	delinq_2yrs	118704.11	0.00
9	dti	1028395.74	0.00
7	annual_inc	384827.42	0.00
6	installment	734265.41	0.00
5	int_rate	2170744.59	0.00
4	funded_amnt_inv	690530.32	0.00
3	funded_amnt	703323.38	0.00
2	loan_amnt	703637.57	0.00
1	member_id	188858.30	0.00
13	pub_rec	344442.54	0.00
29	acc_now_delinq	1124460.36	0.00
11	inq_last_6mths	193.49	0.00
23	total_rec_late_fee	95.02	0.00
8	loan_status	-0.00	NaN
15	revol_util	NaN	NaN
27	collections_12_mths_ex_med	NaN	NaN
30	tot_coll_amt	NaN	NaN
31	tot_cur_bal	NaN	NaN
32	total_rev_hi_lim	NaN	NaN

In [1059]:

```
anova_df["Numerical Variable"].to_list()
```

Out[1059]:

```
['id',
 'last_pymnt_amnt',
 'collection_recovery_fee',
 'recoveries',
 'total_rec_int',
 'total_rec_prncp',
 'total_pymnt_inv',
 'total_pymnt',
 'out_prncp_inv',
 'out_prncp',
 'total_acc',
 'policy_code',
 'revol_bal',
 'open_acc',
 'delinq_2yrs',
 'dti',
 'annual_inc',
 'installment',
 'int_rate',
 'funded_amnt_inv',
 'funded_amnt',
 'loan_amnt',
 'member_id',
 'pub_rec',
 'acc_now_delinq',
 'inq_last_6mths',
 'total_rec_late_fee',
 'loan_status',
 'revol_util',
 'collections_12_mths_ex_med',
 'tot_coll_amt',
 'tot_cur_bal',
 'total_rev_hi_lim']
```

Insight:revol_util indicates how much of the borrower's available credit they are currently using., it can be a good parameter, so we will not delete, others varaible wich p value nun , duplicat column like member id,Founded amounnt invoice, total payment invoice, out pricipal invoice we will delete it

In [1060]:

```
df1.columns
```

Out[1060]:

```
Index(['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv',
       'term', 'int_rate', 'installment', 'grade', 'sub_grade', 'emp_title',
       'emp_length', 'home_ownership', 'annual_inc', 'verification_status',
       'issue_d', 'loan_status', 'pymnt_plan', 'url', 'purpose', 'title',
       'zip_code', 'addr_state', 'dti', 'delinq_2yrs', 'earliest_cr_line',
       'inq_last_6mths', 'open_acc', 'pub_rec', 'revol_bal', 'revol_util',
       'total_acc', 'initial_list_status', 'out_prncp', 'out_prncp_inv',
       'total_pymnt', 'total_pymnt_inv', 'total_rec_prncp', 'total_rec_int',
       'total_rec_late_fee', 'recoveries', 'collection_recovery_fee',
       'last_pymnt_d', 'last_pymnt_amnt', 'next_pymnt_d', 'last_credit_pull_d',
       'collections_12_mths_ex_med', 'policy_code', 'application_type',
       'acc_now_delinq', 'tot_coll_amt', 'tot_cur_bal', 'total_rev_hi_lim'],
      dtype='object')
```

In [1061]:

```
# Assuming df1 is a list of dictionaries
df1 = pd.DataFrame(df1)

selected_columns = ['id', 'last_pymnt_amnt', 'collection_recovery_fee', 'recoveries', 'total_rec_int',
                    'total_rec_prncp', 'total_pymnt', 'out_prncp', 'total_acc', 'policy_code', 'revol_ba
                    'open_acc', 'delinq_2yrs', 'dti', 'annual_inc', 'installment', 'int_rate', 'funded_a
                    'loan_amnt', 'pub_rec', 'acc_now_delinq', 'inq_last_6mths', 'total_rec_late_fee',
                    'revol_util', 'term', 'grade', 'sub_grade', 'emp_title', 'emp_length', 'home_ownersh
                    'verification_status', 'issue_d', 'pymnt_plan', 'url', 'purpose', 'title', 'zip_code
                    'addr_state', 'earliest_cr_line', 'initial_list_status', 'last_pymnt_d',
                    'last_credit_pull_d', 'application_type', 'loan_status']

df1 = df1[selected_columns]
```

In [1062]:

```
df1.head()
```

Out[1062]:

	id	last_pymnt_amnt	collection_recovery_fee	recoveries	total_rec_int	total_rec_prncp	total_pymnt	out_prncp	total_acc
0	1077501	171.62	0.00	0.00	861.07	5000.00	5861.07	0.00	9.00
1	1077430	119.66	1.11	117.08	435.17	456.46	1008.71	0.00	4.00
2	1077175	649.91	0.00	0.00	603.65	2400.00	3003.65	0.00	10.00
3	1076863	357.48	0.00	0.00	2209.33	10000.00	12226.30	0.00	37.00
5	1075269	161.03	0.00	0.00	631.38	5000.00	5631.38	0.00	12.00

In [1063]:

```
df1.shape
```

Out[1063]:

```
(252971, 44)
```

In [1064]:

```


# Select categorical columns
categorical_columns = df1.select_dtypes(include=['object', 'category']).columns
categorical_columns


```

Out[1064]:

```
Index(['term', 'grade', 'sub_grade', 'emp_title', 'emp_length',
       'home_ownership', 'verification_status', 'issue_d', 'pymnt_plan', 'url',
       'purpose', 'title', 'zip_code', 'addr_state', 'earliest_cr_line',
       'initial_list_status', 'last_pymnt_d', 'last_credit_pull_d',
       'application_type'],
      dtype='object')
```

In [1065]:

```
len(categorical_columns)
```

Out[1065]:

```
19
```

In [1066]:

```
df1.isnull().any()
```

Out[1066]:

```
id                          False
last_pymnt_amnt             False
collection_recovery_fee     False
recoveries                  False
total_rec_int               False
total_rec_prncp             False
total_pymnt                 False
out_prncp                   False
total_acc                   False
policy_code                 False
revol_bal                   False
open_acc                    False
delinq_2yrs                 False
dti                         False
annual_inc                  False
installment                 False
int_rate                    False
funded_amnt                 False
loan_amnt                   False
pub_rec                     False
acc_now_delinq              False
inq_last_6mths              False
total_rec_late_fee          False
revol_util                   True
term                        False
grade                       False
sub_grade                   False
emp_title                    True
emp_length                   True
home_ownership              False
verification_status         False
issue_d                     False
pymnt_plan                  False
url                         False
purpose                     False
title                        True
zip_code                    False
addr_state                  False
earliest_cr_line            False
initial_list_status         False
last_pymnt_d                 True
last_credit_pull_d           True
application_type            False
loan_status                 False
dtype: bool
```

In [1067]:

```
from scipy.stats import chi2_contingency, f_oneway
```

In [1068]:

```
categorical_vars = df1.select_dtypes(include='object').columns
categorical_vars
```

Out[1068]:

```
Index(['term', 'grade', 'sub_grade', 'emp_title', 'emp_length',
       'home_ownership', 'verification_status', 'issue_d', 'pymnt_plan', 'url',
       'purpose', 'title', 'zip_code', 'addr_state', 'earliest_cr_line',
       'initial_list_status', 'last_pymnt_d', 'last_credit_pull_d',
       'application_type'],
      dtype='object')
```

AN approach is to calculate the information value (IV) for each categorical variable. IV measures the predictive power of a variable with respect to the target variable

In [1069]:

```
def calculate_iv(df1, categorical_vars, target_var):
    iv_scores = {}
    for column in categorical_vars:
        grouped = df1.groupby(column)[target_var].agg(['count', 'sum'])
        grouped.columns = ['n', 'n_class']
        total = grouped['n_class'].sum()
        grouped['n_total'] = total
        grouped['p_class'] = grouped['n_class'] / grouped['n']
        grouped['p_total'] = grouped['n_total'] / total
        grouped['WOE'] = np.log(grouped['p_class'] / grouped['p_total'])
        grouped['IV'] = (grouped['p_class'] - grouped['p_total']) * grouped['WOE']
        iv_scores[column] = grouped['IV'].sum()
    return iv_scores

iv_scores = calculate_iv(df1, categorical_vars, 'loan_status')
print("Information Values (IV) for categorical variables:")
for column, iv in iv_scores.items():
    print(column, ":", iv)
```

```
Information Values (IV) for categorical variables:
term : 0.12079328022100824
grade : 0.6683210357508107
sub_grade : 3.446761248720657
emp_title : inf
emp_length : 0.38483001115312565
home_ownership : 0.17942750858221335
verification_status : 0.1082092488970875
issue_d : 2.915364520094225
pymnt_plan : 0.03524967077824518
url : inf
purpose : 0.5290375183168444
title : inf
zip_code : inf
addr_state : 1.681218117440674
earliest_cr_line : inf
initial_list_status : 0.06923417705628099
last_pymnt_d : 8.681412614272453
last_credit_pull_d : 4.243794749010719
application_type : 0.035249377413888096
```

Insight:from that information value we priortize Based on the IV scores you provided, you can prioritize the variables as follows:

High IV:

sub_grade last_pymnt_d last_credit_pull_d Moderate IV:

grade addr_state purpose and from lower value qe will consider term emp_length home_ownership(throgh business undrstanding) The variable like Variables with infinite IV scores (emp_title, url, title, zip_code, earliest_cr_line) may need further investigation,but we only keep Zipcode and emp titlebecause others not going to effect our target variable We also delete issue date as because we dont know futre issue date , and its not agffcts our model We can also dellete id

Data leakage occurs when information that would not be available at the time of prediction is used as a predictor variable. In the case of loan prediction, the issue date is typically not known beforehand and including it as a feature would provide direct information about the loan outcome, leading to biased and inaccurate predictions.

#lets build a new df

In [1070]:

```
# after this catagorical value and target variable  We will make a new data frame
df2=df1[['term','last_pymnt_amnt', 'collection_recovery_fee', 'recoveries', 'total_rec_int',
                    'total_rec_prncp', 'total_pymnt', 'out_prncp', 'total_acc', 'policy_code', 'revol_ba
                    'open_acc', 'delinq_2yrs', 'dti', 'annual_inc', 'installment', 'int_rate', 'funded_a
                    'loan_amnt', 'pub_rec', 'acc_now_delinq', 'inq_last_6mths', 'total_rec_late_fee',
                    'revol_util', 'grade', 'sub_grade', 'emp_title','emp_length', 'home_ownership',
                    'purpose', 'zip_code','addr_state', 'earliest_cr_line', 'initial_list_status', 'last
                    'last_credit_pull_d', 'loan_status']]
```

In [1071]:

```
df2.head()
```

Out[1071]:

	term	last_pymnt_amnt	collection_recovery_fee	recoveries	total_rec_int	total_rec_prncp	total_pymnt	out_prncp	total_acc
0	36 months	171.62	0.00	0.00	861.07	5000.00	5861.07	0.00	9.00
1	60 months	119.66	1.11	117.08	435.17	456.46	1008.71	0.00	4.00
2	36 months	649.91	0.00	0.00	603.65	2400.00	3003.65	0.00	10.00
3	36 months	357.48	0.00	0.00	2209.33	10000.00	12226.30	0.00	37.00
5	36 months	161.03	0.00	0.00	631.38	5000.00	5631.38	0.00	12.00

In [1072]:

```
df2.shape
```

Out[1072]:

```
(252971, 37)
```

In [1073]:

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 252971 entries, 0 to 887371
Data columns (total 37 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   term                      252971 non-null  object 
 1   last_pymnt_amnt           252971 non-null  float64
 2   collection_recovery_fee   252971 non-null  float64
 3   recoveries                252971 non-null  float64
 4   total_rec_int             252971 non-null  float64
 5   total_rec_prncp           252971 non-null  float64
 6   total_pymnt               252971 non-null  float64
 7   out_prncp                 252971 non-null  float64
 8   total_acc                 252971 non-null  float64
 9   policy_code               252971 non-null  float64
 10  revol_bal                 252971 non-null  float64
 11  open_acc                  252971 non-null  float64
 12  delinq_2yrs               252971 non-null  float64
 13  dti                       252971 non-null  float64
 14  annual_inc                252971 non-null  float64
 15  installment               252971 non-null  float64
 16  int_rate                  252971 non-null  float64
 17  funded_amnt               252971 non-null  float64
 18  loan_amnt                 252971 non-null  float64
 19  pub_rec                   252971 non-null  float64
 20  acc_now_delinq            252971 non-null  float64
 21  inq_last_6mths            252971 non-null  float64
 22  total_rec_late_fee        252971 non-null  float64
 23  revol_util                252772 non-null  float64
 24  grade                     252971 non-null  object 
 25  sub_grade                 252971 non-null  object 
 26  emp_title                 239040 non-null  object 
 27  emp_length                243074 non-null  object 
 28  home_ownership            252971 non-null  object 
 29  purpose                   252971 non-null  object 
 30  zip_code                  252971 non-null  object 
 31  addr_state                252971 non-null  object 
 32  earliest_cr_line          252971 non-null  object 
 33  initial_list_status       252971 non-null  object 
 34  last_pymnt_d              252444 non-null  object 
 35  last_credit_pull_d        252950 non-null  object 
 36  loan_status               252971 non-null  int64  
dtypes: float64(23), int64(1), object(13)
memory usage: 73.3+ MB
```

In [1074]:

```
# Find the null values in each column
null_columns = df2.isnull().sum()

# Filter the columns with null values
columns_with_null = null_columns[null_columns > 0]

# Print the columns with null values
print("Columns with Null Values:")
print(columns_with_null)
```

```
Columns with Null Values:
revol_util              199
emp_title             13931
emp_length             9897
last_pymnt_d            527
last_credit_pull_d       21
dtype: int64
```

In [1075]:

```
# the missing value exist in catagorical column, so we impute it with mode strategy
#before that check all nuul variable
df2.emp_title.nunique()
```

Out[1075]:

```
132761
```

#emp_title" column has 299,271 unique categorical values, it can be challenging to directly impute or use those values for analysis. In this case, it may be more practical to discard the column from the DataFrame

In [1076]:

```
# Drop the 'emp_title' column
df2.drop('emp_title', axis=1, inplace=True)
```

In [1077]:

```
df2.emp_length.unique()
```

Out[1077]:

```
array(['10+ years', '< 1 year', '3 years', '9 years', '4 years',
       '5 years', '1 year', '6 years', '2 years', '7 years', '8 years',
       nan], dtype=object)
```

In [1078]:

```
for year in df2.emp_length.unique():
    print(f"{year} years in this position:")
    print(f"{df2[df2.emp_length == year].loan_status.value_counts(normalize=True)}")
    print('========================================')
```

```
10+ years years in this position:
1   0.83
0   0.17
Name: loan_status, dtype: float64
========================================
< 1 year years in this position:
1   0.82
0   0.18
Name: loan_status, dtype: float64
========================================
3 years years in this position:
1   0.83
0   0.17
Name: loan_status, dtype: float64
========================================
9 years years in this position:
1   0.81
0   0.19
Name: loan_status, dtype: float64
========================================
4 years years in this position:
1   0.83
0   0.17
Name: loan_status, dtype: float64
========================================
5 years years in this position:
1   0.82
0   0.18
Name: loan_status, dtype: float64
========================================
1 year years in this position:
1   0.82
0   0.18
Name: loan_status, dtype: float64
========================================
6 years years in this position:
1   0.82
0   0.18
Name: loan_status, dtype: float64
========================================
2 years years in this position:
1   0.83
0   0.17
Name: loan_status, dtype: float64
========================================
7 years years in this position:
1   0.82
0   0.18
Name: loan_status, dtype: float64
========================================
8 years years in this position:
1   0.82
0   0.18
Name: loan_status, dtype: float64
========================================
nan years in this position:
Series([], Name: loan_status, dtype: float64)
========================================
```

#The code aims to analyze the loan status distribution based on the number of years an employee has been in a particular position. It prints the proportion of different loan status categories for each unique value in the "emp_length" column, providing insights into how the loan status varies based on employment length.

Insight: we see length of job is not afffecting not affecting the charge off the loan , all the lengh have equal value in relation of charge off, so we can delete the row

In [1079]:

```
df2.drop('emp_length', axis=1, inplace=True)
```

In [1080]:

```
df2.last_credit_pull_d.unique()
```

Out[1080]:

```
array(['Jan-2016', 'Sep-2013', 'Jan-2015', 'Sep-2015', 'Dec-2014',
       'Aug-2012', 'Mar-2013', 'Dec-2015', 'Aug-2013', 'Nov-2012',
       'Mar-2014', 'Apr-2015', 'May-2014', 'Jul-2015', 'Jul-2012',
       'Sep-2012', 'May-2013', 'Oct-2015', 'Jun-2012', 'Mar-2015',
       'Dec-2012', 'Jul-2014', 'Sep-2014', 'Feb-2014', 'Jun-2015',
       'Oct-2013', 'Apr-2014', 'Oct-2014', 'Feb-2013', 'Nov-2015',
       'Oct-2012', 'Nov-2013', 'Nov-2014', 'Feb-2012', 'Apr-2012',
       'Aug-2015', 'Jun-2014', 'Jan-2012', 'Aug-2014', 'Jun-2013',
       'Dec-2013', 'May-2012', 'Jan-2014', 'Jul-2013', 'Apr-2013',
       'May-2015', 'Feb-2015', 'Mar-2012', 'Nov-2011', 'Dec-2011',
       'Jan-2013', 'Oct-2011', 'Sep-2011', 'Aug-2011', 'Jul-2011',
       'Jun-2011', 'May-2011', 'Apr-2011', 'Mar-2011', 'Feb-2011',
       'Jan-2011', 'Dec-2010', 'Nov-2010', 'Oct-2010', nan, 'Sep-2010',
       'Aug-2010', 'Jul-2010', 'Jun-2010', 'May-2010', 'Apr-2010',
       'Feb-2010', 'Mar-2010', 'Aug-2007', 'Jan-2010', 'Dec-2009',
       'Nov-2009', 'Oct-2009', 'Sep-2009', 'Jul-2009', 'Aug-2009',
       'Jun-2009', 'May-2009', 'Apr-2009', 'Mar-2009', 'Feb-2009',
       'Jan-2009', 'Dec-2008', 'Jun-2008', 'Sep-2008', 'May-2008',
       'Aug-2008', 'Mar-2008', 'Oct-2008', 'Feb-2008', 'Jan-2008',
       'Dec-2007', 'Jul-2008', 'Oct-2007', 'Sep-2007', 'Jun-2007',
       'May-2007', 'Jul-2007'], dtype=object)
```

In [1081]:

```
#we can convert it into year
df2['last_credit_pull_d'] = pd.to_datetime(df2['last_credit_pull_d'])
df2['last_credit_pull_year'] = df2['last_credit_pull_d'].dt.year
```

In [1082]:

```
df2.last_credit_pull_year.nunique()
```

Out[1082]:

```
10
```

In [1083]:

```
df2.last_credit_pull_year.value_counts()
```

Out[1083]:

```
2016.00    102053
2015.00     99088
2014.00     31135
2013.00     12169
2012.00      5049
2011.00      2276
2010.00       868
2009.00       235
2008.00        41
2007.00        36
Name: last_credit_pull_year, dtype: int64
```

In [1084]:

```
#As its datae  we can impute with mode
df2['last_credit_pull_year'].fillna(df2['last_credit_pull_year'].mode()[0], inplace=True)

```

In [1085]:

```
df2.last_pymnt_d.unique()
```

Out[1085]:

```
array(['Jan-2015', 'Apr-2013', 'Jun-2014', 'Apr-2012', 'Nov-2012',
       'Jun-2013', 'Sep-2013', 'Jul-2012', 'Oct-2013', 'May-2013',
       'Feb-2015', 'Aug-2015', 'Oct-2012', 'Sep-2012', nan, 'Dec-2012',
       'Dec-2014', 'Aug-2013', 'Nov-2013', 'Jan-2014', 'Apr-2014',
       'Aug-2014', 'Oct-2014', 'Aug-2012', 'Jul-2014', 'Jul-2013',
       'Jan-2016', 'Apr-2015', 'Feb-2014', 'Sep-2014', 'Jun-2012',
       'Feb-2013', 'Mar-2013', 'May-2014', 'Mar-2015', 'Jan-2013',
       'Dec-2013', 'Feb-2012', 'Mar-2014', 'Sep-2015', 'Nov-2015',
       'Jan-2012', 'Oct-2015', 'Nov-2014', 'Mar-2012', 'May-2012',
       'Dec-2015', 'Jun-2015', 'May-2015', 'Jul-2015', 'Dec-2011',
       'Nov-2011', 'Oct-2011', 'Sep-2011', 'Aug-2011', 'Jul-2011',
       'Jun-2011', 'May-2011', 'Apr-2011', 'Mar-2011', 'Feb-2011',
       'Jan-2011', 'Dec-2010', 'Nov-2010', 'Oct-2010', 'Sep-2010',
       'Aug-2010', 'Jul-2010', 'Jun-2010', 'May-2010', 'Apr-2010',
       'Mar-2010', 'Feb-2010', 'Jan-2010', 'Dec-2009', 'Nov-2009',
       'Oct-2009', 'Sep-2009', 'Aug-2009', 'Jul-2009', 'Jun-2009',
       'May-2009', 'Apr-2009', 'Mar-2009', 'Feb-2009', 'Jan-2009',
       'Dec-2008', 'Oct-2008', 'Aug-2008', 'Jul-2008', 'Sep-2008',
       'Jun-2008', 'May-2008', 'Nov-2008', 'Apr-2008', 'Mar-2008',
       'Feb-2008', 'Jan-2008'], dtype=object)
```

In [1086]:

```
#we can convert it into year
df2['last_pymnt_d'] = pd.to_datetime(df2['last_pymnt_d'])
df2['last_pymnt_d_year'] = df2['last_pymnt_d'].dt.year
```

In [1087]:

```
df2.last_pymnt_d_year.value_counts()
```

Out[1087]:

```
2015.00    135480
2014.00     66542
2013.00     26841
2012.00     11366
2011.00      4996
2016.00      4675
2010.00      1848
2009.00       559
2008.00       137
Name: last_pymnt_d_year, dtype: int64
```

In [1088]:

```
#As its datae  we can impute with mode
df2['last_pymnt_d_year'].fillna(df2['last_pymnt_d_year'].mode()[0], inplace=True)

```

In [1089]:

```
df2.revol_util .value_counts()
```

Out[1089]:

```
0.00      1804
63.00      452
62.00      444
58.00      442
57.00      440
          ...
18.82        1
5.79         1
27.81        1
10.17        1
114.50       1
Name: revol_util, Length: 1199, dtype: int64
```

In [1090]:

```
# Mean imputation
mean_revol_util = df2['revol_util'].mean()
df2['revol_util'].fillna(mean_revol_util, inplace=True)
```

In [1091]:

```
df2.head()
```

Out[1091]:

	term	last_pymnt_amnt	collection_recovery_fee	recoveries	total_rec_int	total_rec_prncp	total_pymnt	out_prncp	total_acc
0	36 months	171.62	0.00	0.00	861.07	5000.00	5861.07	0.00	9.00
1	60 months	119.66	1.11	117.08	435.17	456.46	1008.71	0.00	4.00
2	36 months	649.91	0.00	0.00	603.65	2400.00	3003.65	0.00	10.00
3	36 months	357.48	0.00	0.00	2209.33	10000.00	12226.30	0.00	37.00
5	36 months	161.03	0.00	0.00	631.38	5000.00	5631.38	0.00	12.00

In [1092]:

```
df2.shape
```

Out[1092]:

```
(252971, 37)
```

In [1093]:

```
df2.isnull().any()
```

Out[1093]:

```
term                        False
last_pymnt_amnt             False
collection_recovery_fee     False
recoveries                  False
total_rec_int               False
total_rec_prncp             False
total_pymnt                 False
out_prncp                   False
total_acc                   False
policy_code                 False
revol_bal                   False
open_acc                    False
delinq_2yrs                 False
dti                         False
annual_inc                  False
installment                 False
int_rate                    False
funded_amnt                 False
loan_amnt                   False
pub_rec                     False
acc_now_delinq              False
inq_last_6mths              False
total_rec_late_fee          False
revol_util                  False
grade                       False
sub_grade                   False
home_ownership              False
purpose                     False
zip_code                    False
addr_state                  False
earliest_cr_line            False
initial_list_status         False
last_pymnt_d                 True
last_credit_pull_d           True
loan_status                 False
last_credit_pull_year       False
last_pymnt_d_year           False
dtype: bool
```

In [1094]:

```
df2.shape
```

Out[1094]:

```
(252971, 37)
```

In [1095]:

```
df2.isnull().sum().sum()
```

Out[1095]:

```
548
```

```
Insight: the null value showing from last_pymnt_d, last_credit_pull_d what we transfered by other
converted column , Later ,we will delete repeated column
```

In [1096]:

```
df2.head()
```

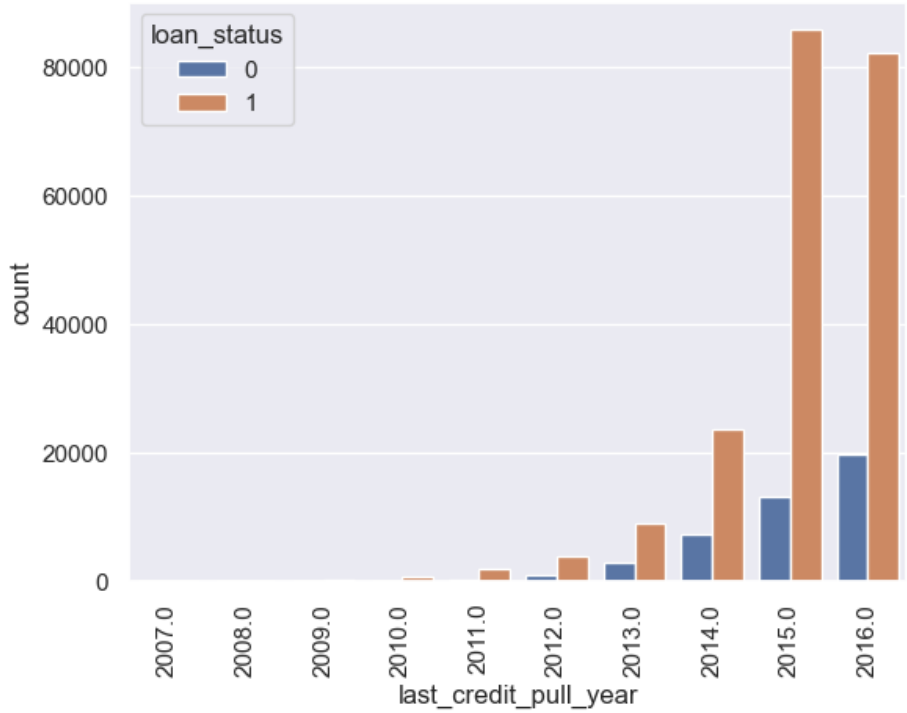
Out[1096]:

	term	last_pymnt_amnt	collection_recovery_fee	recoveries	total_rec_int	total_rec_prncp	total_pymnt	out_prncp	total_acc
0	36 months	171.62	0.00	0.00	861.07	5000.00	5861.07	0.00	9.00
1	60 months	119.66	1.11	117.08	435.17	456.46	1008.71	0.00	4.00
2	36 months	649.91	0.00	0.00	603.65	2400.00	3003.65	0.00	10.00
3	36 months	357.48	0.00	0.00	2209.33	10000.00	12226.30	0.00	37.00
5	36 months	161.03	0.00	0.00	631.38	5000.00	5631.38	0.00	12.00

Visualization to confirm aboout the date and loan status In here 0=Charge off or defaulter

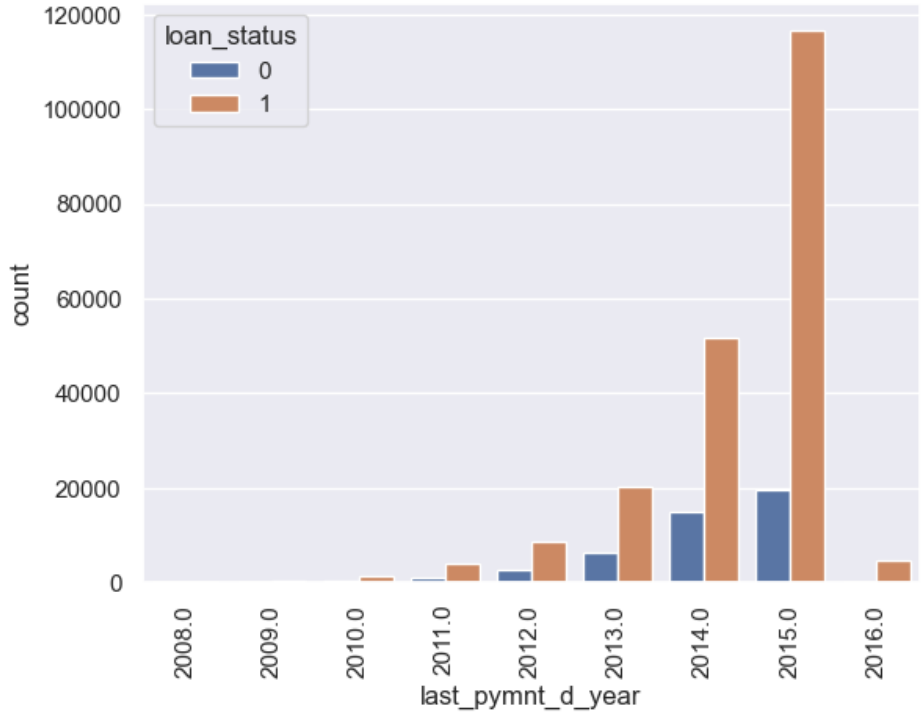
In [1097]:

```
sns.countplot(x='last_credit_pull_year', hue='loan_status', data=df2)
plt.xticks(rotation=90)
plt.show()
```



In [1098]:

```
sns.countplot(x='last_pymnt_d_year', hue='loan_status', data=df2)
plt.xticks(rotation=90)
plt.show()
```




```
In both plots Loan  charge_off increasing from 2010 to 2015, where 2015 is hifher
This observation can be further analyzed to understand the underlying reasons for this pattern. It could be due to various factors such as economic conditions, changes in lending practices, or specific events that occurred during those years. By investigating the causes behind this trend, we  gain insights into potential risk factors or areas that need attention to reduce default rates in future loan issuances.


By including the year or other related variables that capture temporal information, you enable the model to learn from past trends and potentially predict future default rates more accurately.
```

In [1099]:

```
df2.describe()
```

Out[1099]:

	last_pymnt_amnt	collection_recovery_fee	recoveries	total_rec_int	total_rec_prncp	total_pymnt	out_prncp	total_acc	poli
count	252971.00	252971.00	252971.00	252971.00	252971.00	252971.00	252971.00	252971.00	2
mean	6461.74	16.75	159.34	1946.69	11502.86	13609.60	0.00	25.04	
std	7363.83	114.06	748.09	2065.50	8339.04	9483.85	0.00	11.75	
min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	
25%	489.45	0.00	0.00	629.54	5000.00	6400.06	0.00	16.00	
50%	3943.09	0.00	0.00	1310.22	10000.00	11453.60	0.00	23.00	
75%	10037.92	0.00	0.00	2486.14	16000.00	18619.10	0.00	32.00	
max	36475.59	7002.19	33520.27	22777.58	35000.03	57777.58	0.00	150.00	

```
insight:Most numrical variable the standard deviation is large compared to the mean, it indicates a wider spread or dispersion of values. it does not directly tell about outliar , we need to investigate further , but first we go for visualization of each variable in relation with loann status

We will final select the important variable by Visualllization
```

```
6 First catagorical variable and second Numerical variable
```

In [1100]:

```
1 # Data imblance check
```

In [1101]:

```
1 #First check the data balance
2 # Calculate the imbalance ratio
3 # and  see in visualization
4 imbalance = df2["loan_status"].value_counts().reset_index()
5 imbalance
```

Out[1101]:

	index	loan_status
0	1	207723
1	0	45248

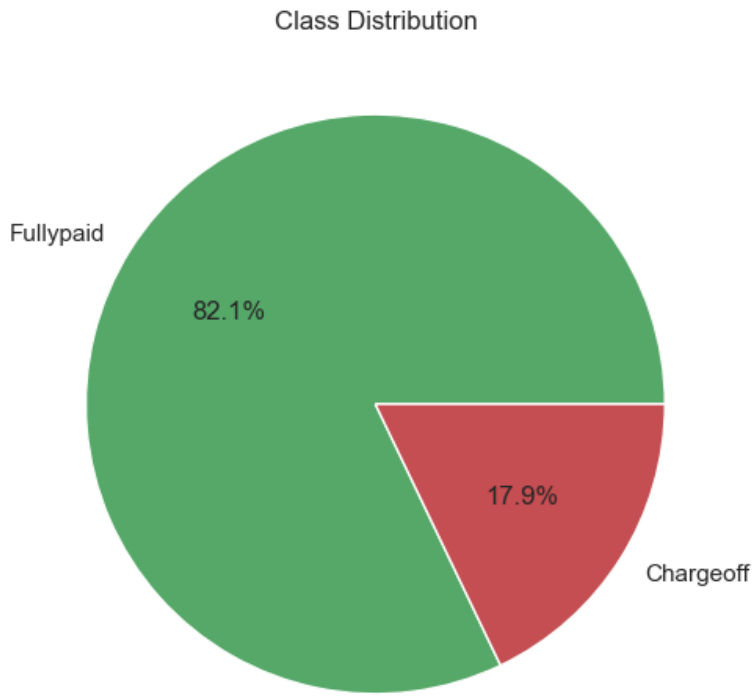
In [1102]:

```
1 # See this result by visualization
2 # Calculate the imbalance ratio
3 imbalance_ratio = imbalance.iloc[1, 1] / imbalance.iloc[0, 1]
4 print("Imbalance Ratio:", imbalance_ratio)
5
```

```
Imbalance Ratio: 0.2178285505216081
```

In [1103]:

```
1 # Plot the class distribution using a pie chart
2 plt.figure(figsize=(6, 6))
3 x= ['Fullypaid','Chargeoff']
4 plt.pie(imbalance["loan_status"], labels=x, autopct='%1.1f%%', colors=['g', 'r'])
5 plt.title("Class Distribution")
6 plt.show()
```



In [1104]:

```
#Lets  do univariate analysis  by defining function
def univariate_categorical(variable, target):
    plt.figure(figsize=(10, 6))
    sns.countplot(data=df2, x=variable, hue=target)
    plt.title(f"{variable} Distribution by {target}")
    plt.xlabel(variable)
    plt.ylabel("Count")
    plt.xticks(rotation=45)
    plt.show()
```

In [1105]:

```
print(df2.columns)
```

```
Index(['term', 'last_pymnt_amnt', 'collection_recovery_fee', 'recoveries',
       'total_rec_int', 'total_rec_prncp', 'total_pymnt', 'out_prncp',
       'total_acc', 'policy_code', 'revol_bal', 'open_acc', 'delinq_2yrs',
       'dti', 'annual_inc', 'installment', 'int_rate', 'funded_amnt',
       'loan_amnt', 'pub_rec', 'acc_now_delinq', 'inq_last_6mths',
       'total_rec_late_fee', 'revol_util', 'grade', 'sub_grade',
       'home_ownership', 'purpose', 'zip_code', 'addr_state',
       'earliest_cr_line', 'initial_list_status', 'last_pymnt_d',
       'last_credit_pull_d', 'loan_status', 'last_credit_pull_year',
       'last_pymnt_d_year'],
      dtype='object')
```

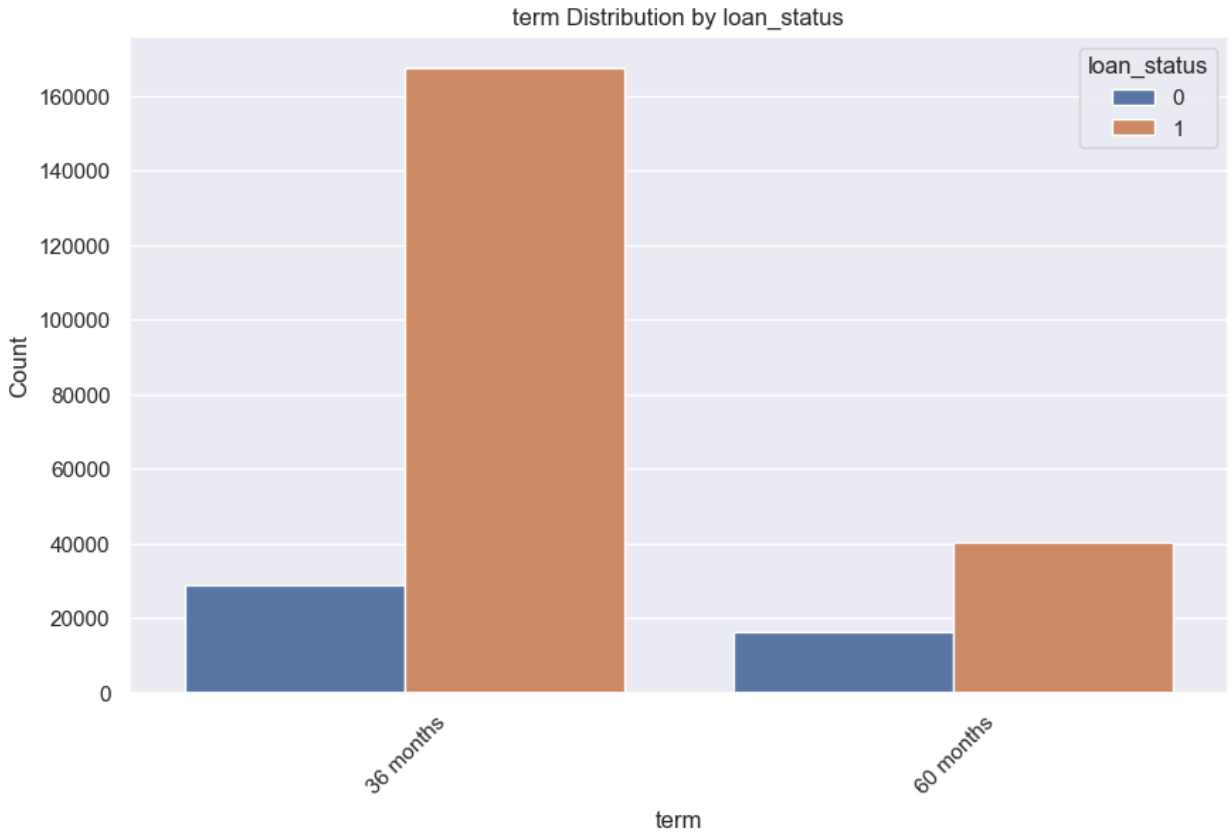
In [1106]:

```
categorical_vars = df2.select_dtypes(include=['object']).columns
print(categorical_vars)
```

```
Index(['term', 'grade', 'sub_grade', 'home_ownership', 'purpose', 'zip_code',
       'addr_state', 'earliest_cr_line', 'initial_list_status'],
      dtype='object')
```

In [1107]:

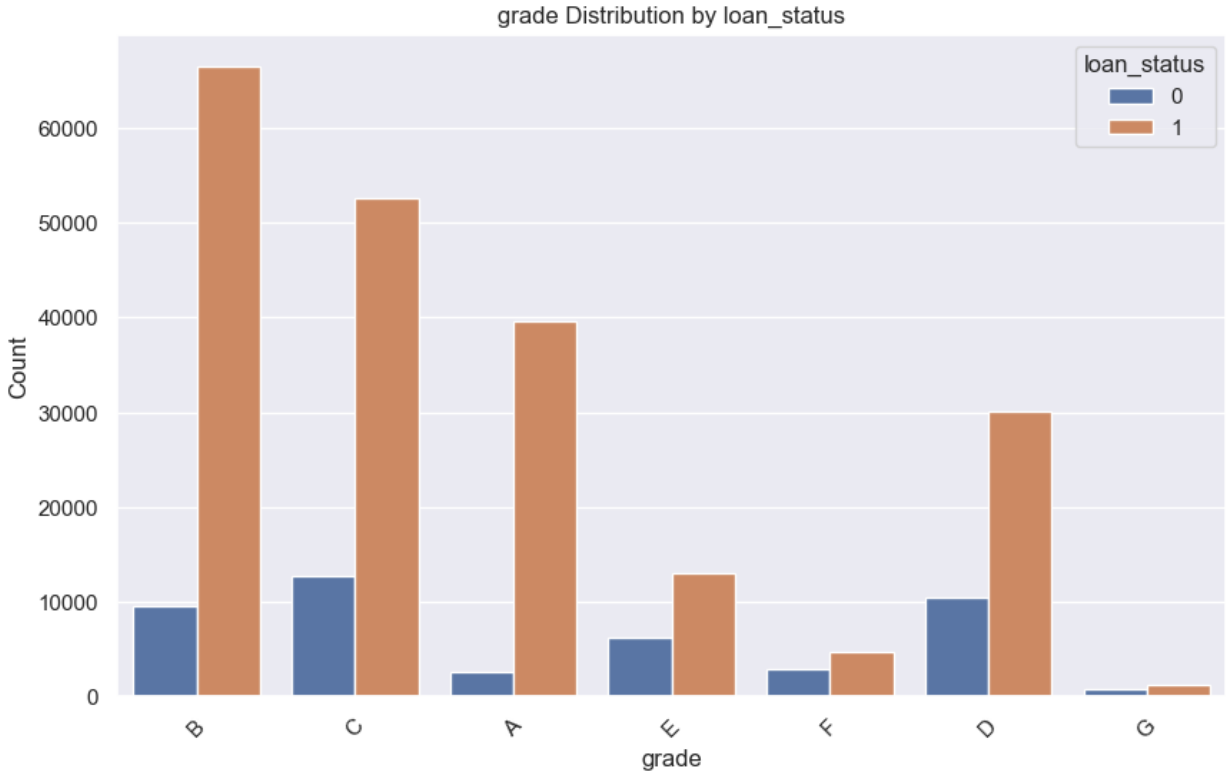
```
#Blue is defaulter  and 1 is fully paid
univariate_categorical('term', 'loan_status')
```



insight: people who take loan for 36 month are slightly more risk in chargeoff

In [1108]:

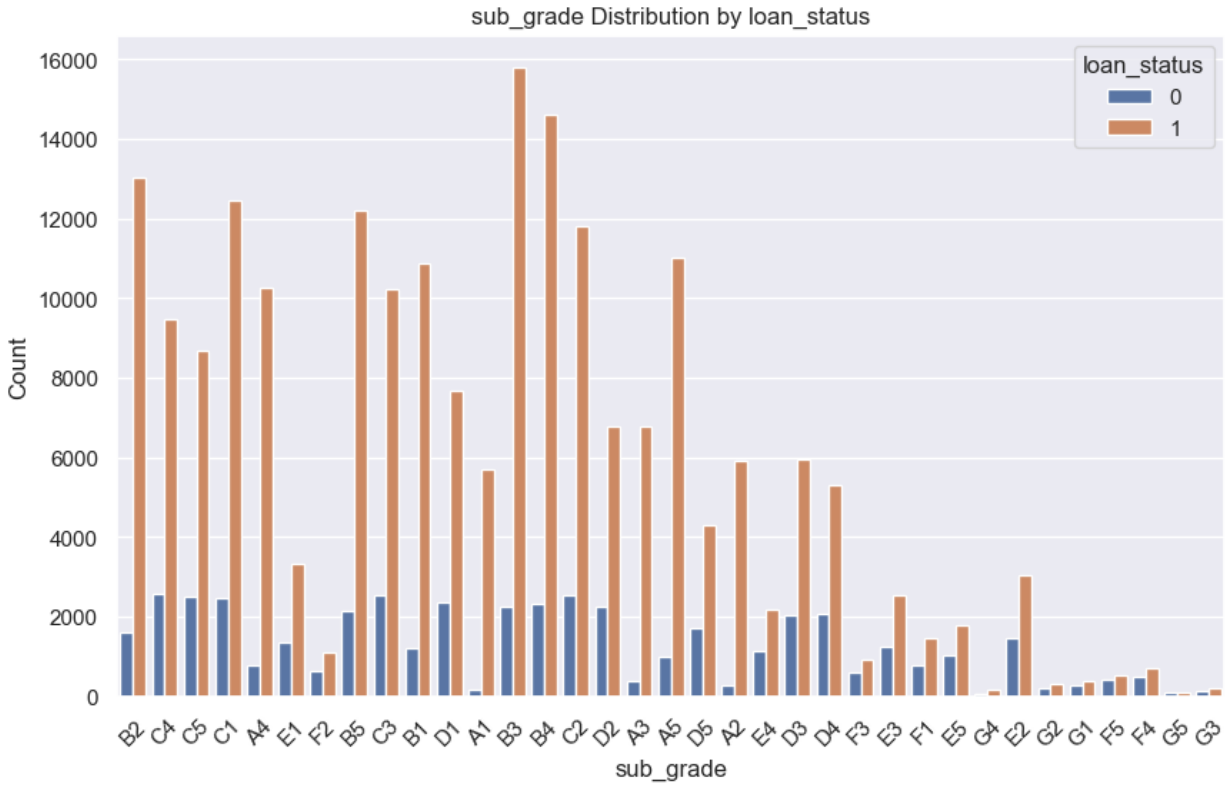
```
#Blue is defaulter  and 1 is fully paid
univariate_categorical('grade', 'loan_status')
```



Insight: in charge off , Grade C and D have higher risk than others but grade B is highest repay and followedby C,A nad D

In [1109]:

```
univariate_categorical('sub_grade', 'loan_status')
```



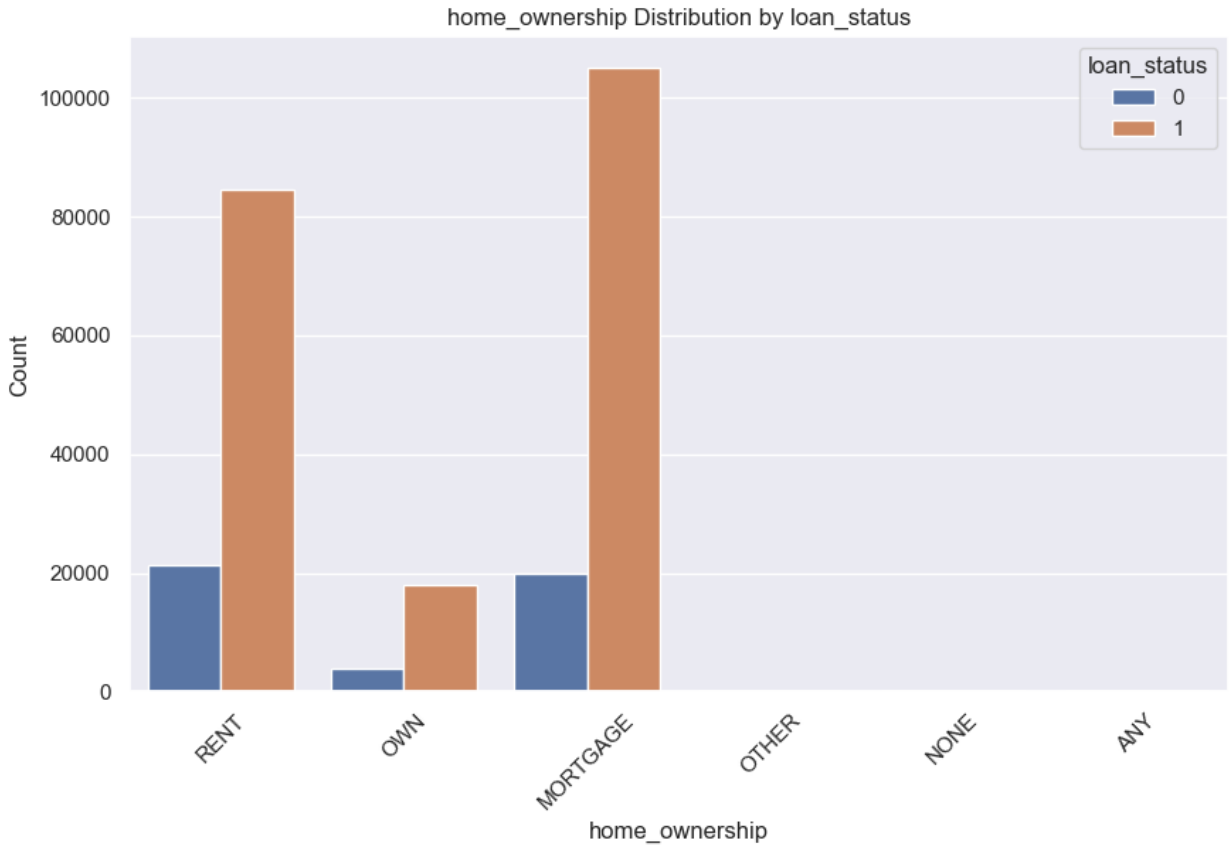
Insight: in charge off , Sugrade F and G have higher risk of charge off where B is highest repay and followedby C,A nad D, so its similler to grade we can delete this variable

In [1110]:

```
df2.drop('sub_grade', axis=1, inplace=True)
```

In [1111]:

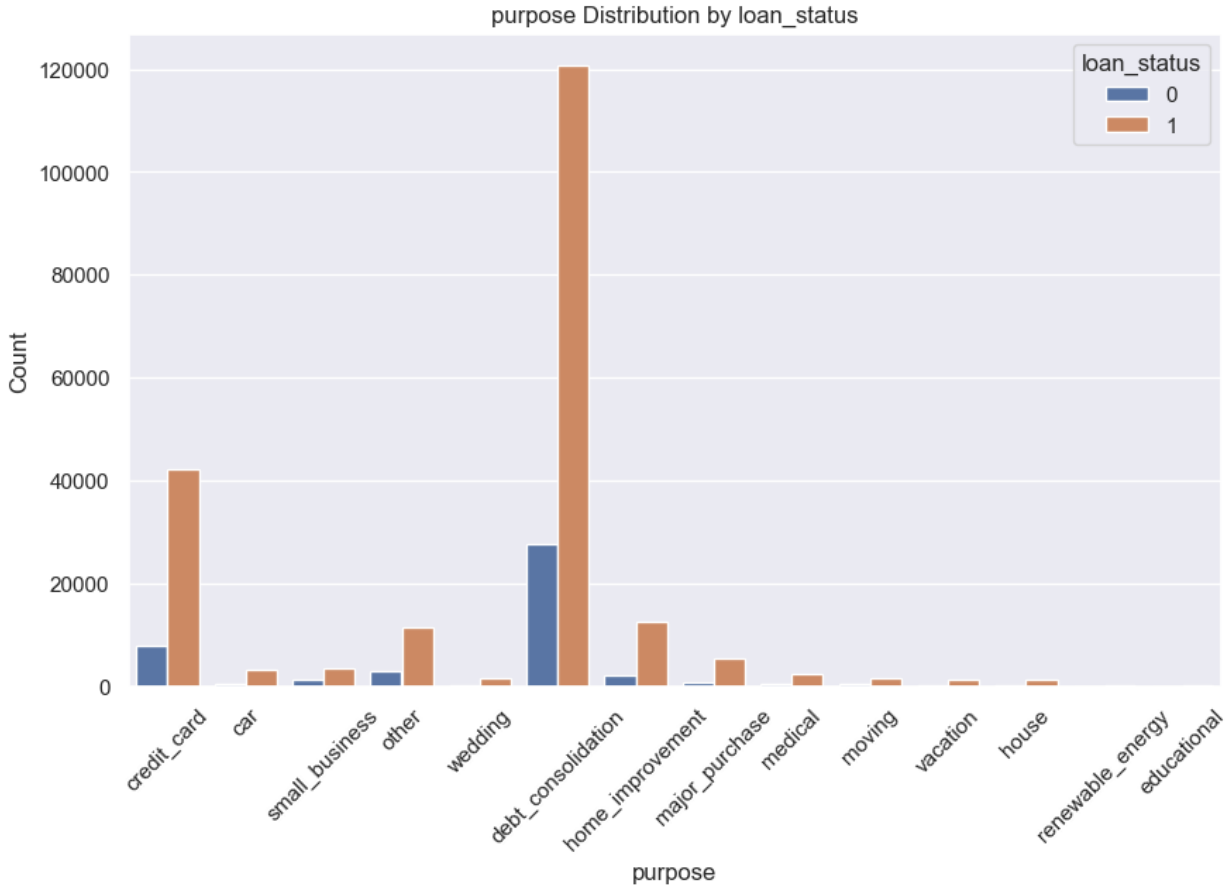
```
univariate_categorical('home_ownership', 'loan_status')
```



Insight:people who are on mortgage and rent have a risk in chargeoff than own home

In [1112]:

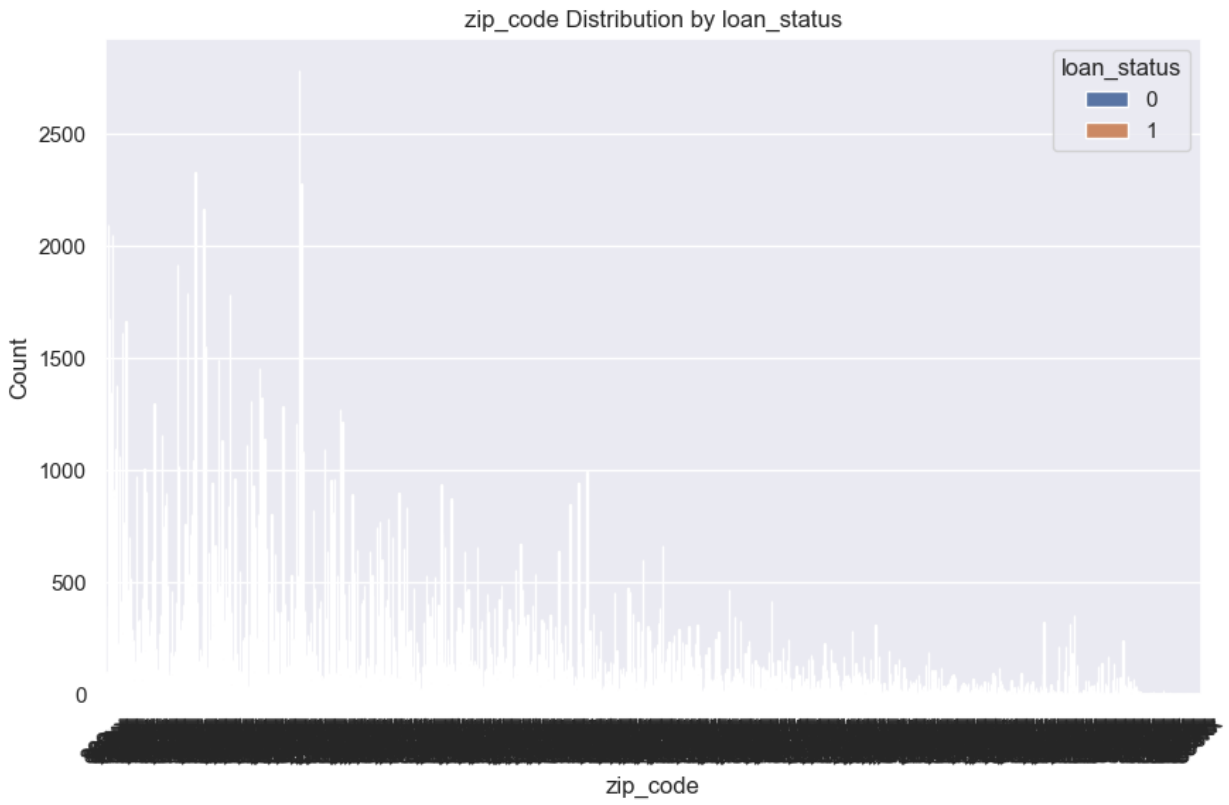
```
univariate_categorical('purpose', 'loan_status')
```



Insight:People who take loan for weeding highr isk in charge off followed credit card, other, and deb consolidation

In [1113]:

```
univariate_categorical('zip_code', 'loan_status')
```



In [1114]:

```
df5.zip_code.value_counts()
```

Out[1114]:

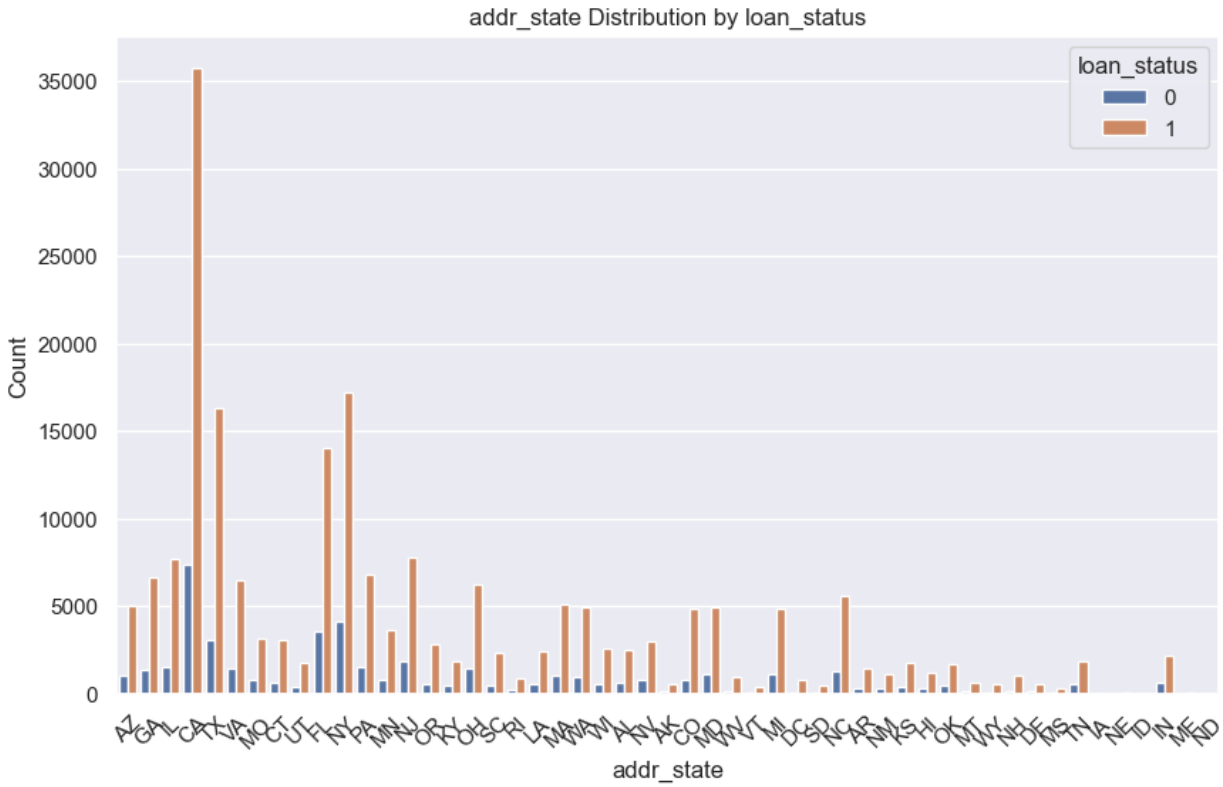
```
945xx    3286
112xx    2832
750xx    2767
100xx    2522
900xx    2474
         ...
929xx       1
938xx       1
524xx       1
510xx       1
580xx       1
Name: zip_code, Length: 886, dtype: int64
```

In [1115]:

```
# its need to further analysis
```

In [1116]:

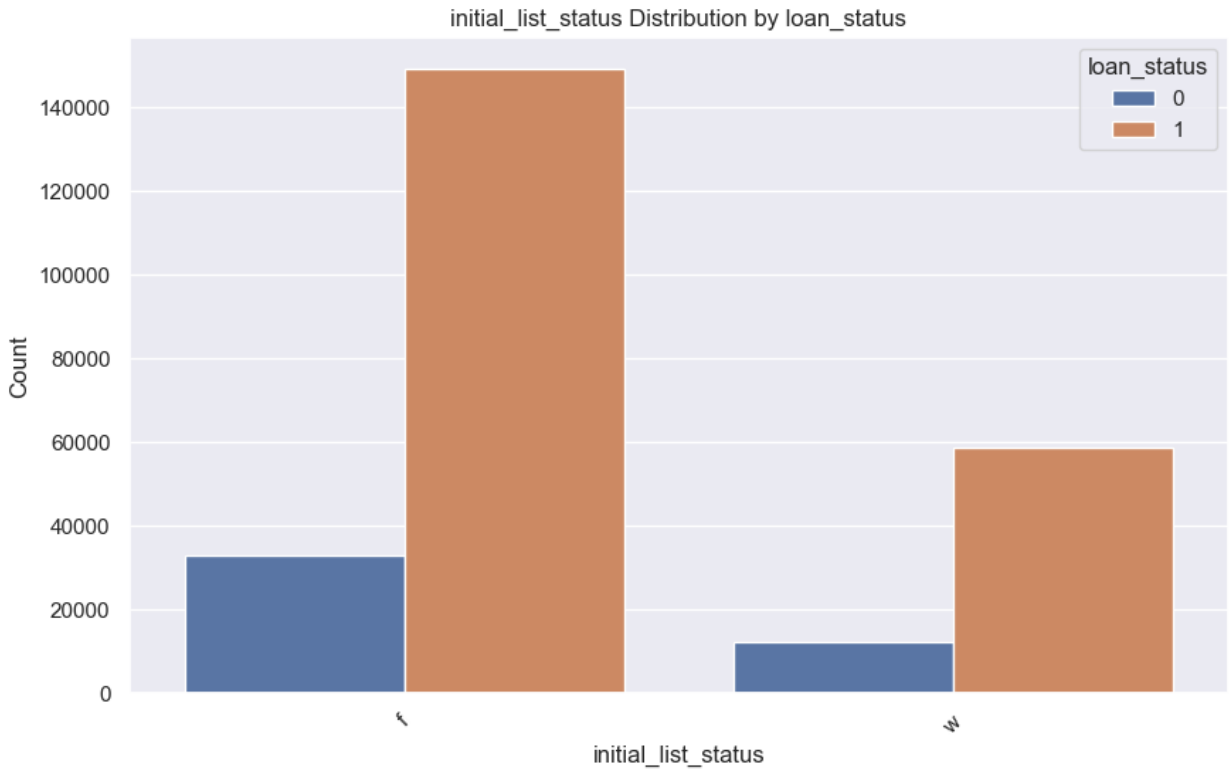
```
univariate_categorical('addr_state', 'loan_status')
```



Insight:need further analysis

In [1117]:

```
univariate_categorical('initial_list_status', 'loan_status')
```



Insight:fractionla loan have higher risk than full loan

In [1118]:

```
df2.earliest_cr_line.unique()
```

```
'Nov-1987', 'May-1995', 'Mar-1993', 'Jun-2008', 'Jul-1980',
'Dec-1982', 'Mar-1975', 'Oct-1984', 'Mar-1988', 'Feb-1980',
'Nov-1988', 'Apr-1988', 'Sep-1985', 'Sep-1971', 'Mar-1978',
'Feb-2008', 'Aug-1978', 'Nov-1970', 'Jun-1979', 'Jun-1980',
'Apr-1989', 'Sep-1983', 'Feb-1989', 'Oct-1982', 'Aug-1986',
'May-1988', 'Dec-1985', 'Jan-1982', 'Sep-1977', 'Dec-1976',
'Apr-1982', 'May-1984', 'Apr-2008', 'Feb-1979', 'Jan-2008',
'Jul-1987', 'Jan-1978', 'May-1989', 'Oct-1977', 'Dec-1975',
'Jan-1984', 'Oct-2008', 'Feb-1985', 'Nov-1982', 'May-1975',
'May-1985', 'Feb-1971', 'Jun-1977', 'Apr-1981', 'May-1979',
'Jan-1972', 'Jun-1986', 'Sep-1967', 'Apr-1978', 'Feb-1965',
'Nov-1975', 'Jun-1967', 'Feb-1991', 'Dec-1979', 'Aug-1967',
'Apr-1971', 'Sep-1984', 'Aug-1982', 'May-1981', 'Dec-1970',
'Oct-1973', 'Jan-1971', 'Dec-1963', 'Apr-1974', 'Jan-1980',
'Apr-1975', 'Jul-1977', 'Mar-1977', 'Nov-1969', 'Jan-1976',
'Nov-1983', 'Mar-1982', 'Apr-1987', 'Dec-1969', 'May-1974',
'Aug-1974', 'Jun-1991', 'Jun-1972', 'Mar-1963', 'Aug-1969',
'Oct-1980', 'Jul-1972', 'Aug-1975', 'Sep-1982', 'Sep-1974',
'Aug-1981', 'Nov-1976', 'May-1973', 'Dec-1973', 'Sep-1973',
'Mar-1973', 'Dec-1977', 'Oct-1976', 'Jan-1974', 'Jan-1970',
```

In [1119]:

```
#we can convert it into year
df2['earliest_cr_line'] = pd.to_datetime(df2['earliest_cr_line'])
df2['earliest_crln_year'] = df2['earliest_cr_line'].dt.year
```

In [1121]:

```
df2.columns
```

Out[1121]:

```
Index(['term', 'last_pymnt_amnt', 'collection_recovery_fee', 'recoveries',
       'total_rec_int', 'total_rec_prncp', 'total_pymnt', 'out_prncp',
       'total_acc', 'policy_code', 'revol_bal', 'open_acc', 'delinq_2yrs',
       'dti', 'annual_inc', 'installment', 'int_rate', 'funded_amnt',
       'loan_amnt', 'pub_rec', 'acc_now_delinq', 'inq_last_6mths',
       'total_rec_late_fee', 'revol_util', 'grade', 'home_ownership',
       'purpose', 'zip_code', 'addr_state', 'earliest_cr_line',
       'initial_list_status', 'last_pymnt_d', 'last_credit_pull_d',
       'loan_status', 'last_credit_pull_year', 'last_pymnt_d_year',
       'earliest_crln_year'],
      dtype='object')
```

In [1122]:

```
#Lets delete the duplicated column during code
df2.drop(['earliest_cr_line'], axis=1, inplace=True)
```

In [1123]:

```
df2.shape
```

Out[1123]:

```
(252971, 36)
```

In [1124]:

```
numerical_columns = df2.select_dtypes(include=['int64', 'float64']).columns
numerical_columns
```

Out[1124]:

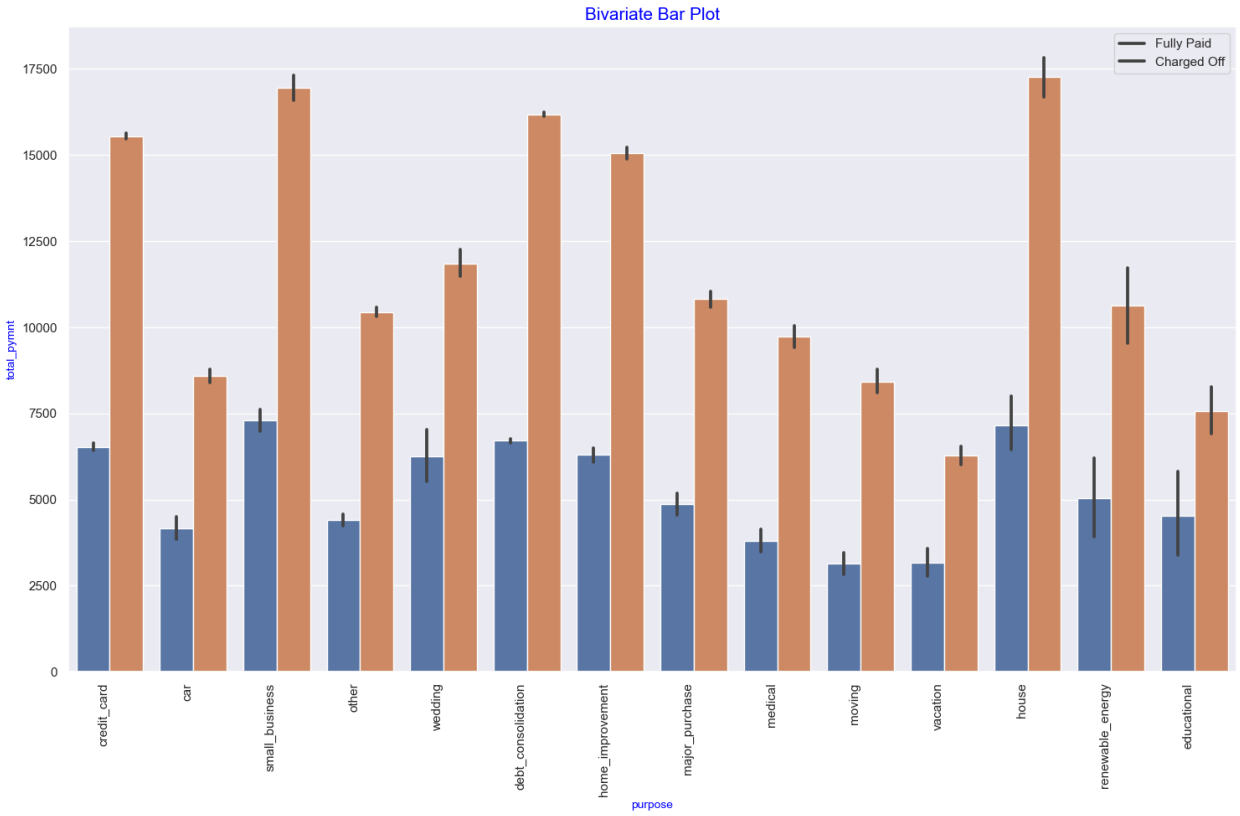
```
Index(['last_pymnt_amnt', 'collection_recovery_fee', 'recoveries',
       'total_rec_int', 'total_rec_prncp', 'total_pymnt', 'out_prncp',
       'total_acc', 'policy_code', 'revol_bal', 'open_acc', 'delinq_2yrs',
       'dti', 'annual_inc', 'installment', 'int_rate', 'funded_amnt',
       'loan_amnt', 'pub_rec', 'acc_now_delinq', 'inq_last_6mths',
       'total_rec_late_fee', 'revol_util', 'loan_status',
       'last_credit_pull_year', 'last_pymnt_d_year', 'earliest_crln_year'],
      dtype='object')
```

In [1125]:

```
#Bivariate anlysis
#Define Bi-variate function
# function for plotting repetitive countplots in bivariate categorical analysis

def bivariate_bar(x, y, df, hue, figsize, title):
    plt.figure(figsize=figsize)
    sns.barplot(x=x, y=y, data=df, hue=hue)
    plt.xlabel(x, fontdict={'fontsize' : 10, 'fontweight' : 3, 'color' : 'Blue'})
    plt.ylabel(y, fontdict={'fontsize' : 10, 'fontweight' : 3, 'color' : 'Blue'})
    plt.title(title, fontdict={'fontsize' : 15, 'fontweight' : 5, 'color' : 'Blue'})
    plt.xticks(rotation=90, ha='right')
    plt.legend(labels=['Fully Paid', 'Charged Off'])
    plt.show()

```

In [1126]:

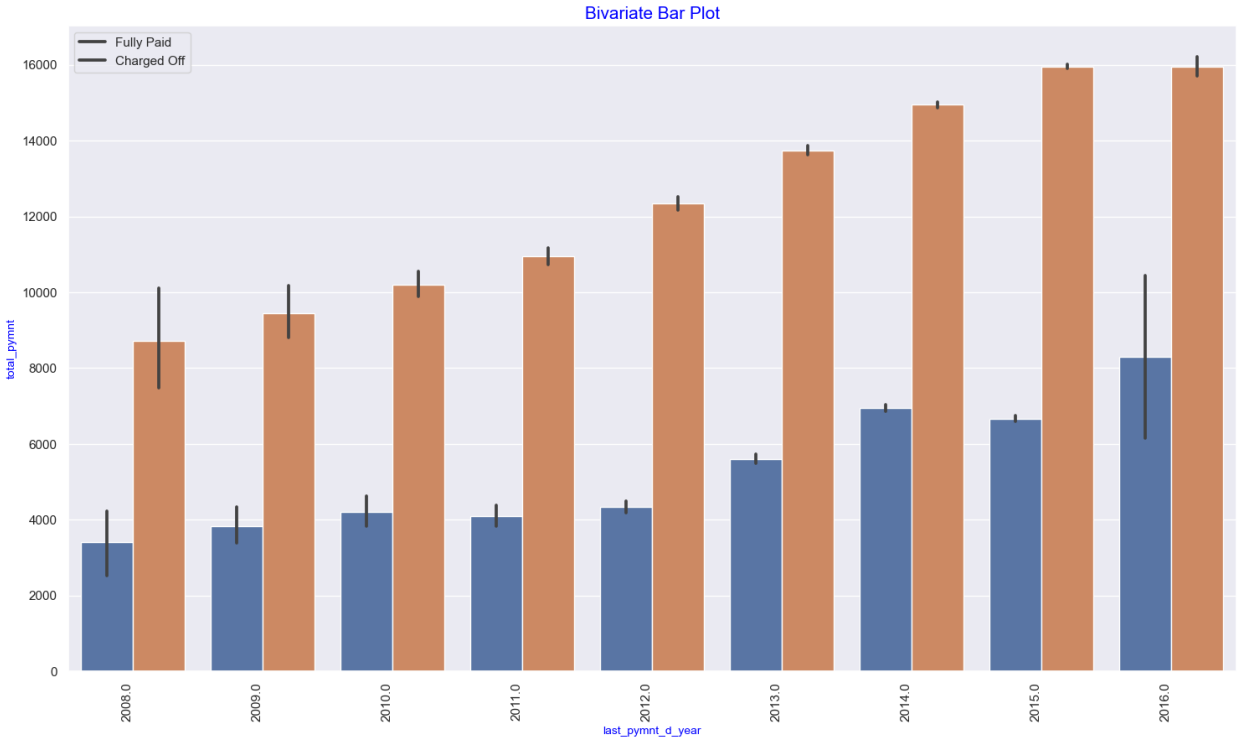
```
bivariate_bar("purpose", "total_pymnt", df2, "loan_status", (18, 10), "Bivariate Bar Plot")

```



Insight:people who take the loan for housing there total payment is high and they also higher risk in defaulting followed small businesss, an d weeding

In [1127]:

```
bivariate_bar('last_pymnt_d_year', 'total_pymnt', df2, "loan_status", (18, 10), "Bivariate Bar Plot")
```



```
Insight: there is a positive relation with loan status with last payment d year, so we count this variable for model
```

In [1128]:

```
df2.columns

```

Out[1128]:

```
Index(['term', 'last_pymnt_amnt', 'collection_recovery_fee', 'recoveries',
       'total_rec_int', 'total_rec_prncp', 'total_pymnt', 'out_prncp',
       'total_acc', 'policy_code', 'revol_bal', 'open_acc', 'delinq_2yrs',
       'dti', 'annual_inc', 'installment', 'int_rate', 'funded_amnt',
       'loan_amnt', 'pub_rec', 'acc_now_delinq', 'inq_last_6mths',
       'total_rec_late_fee', 'revol_util', 'grade', 'home_ownership',
       'purpose', 'zip_code', 'addr_state', 'initial_list_status',
       'last_pymnt_d', 'last_credit_pull_d', 'loan_status',
       'last_credit_pull_year', 'last_pymnt_d_year', 'earliest_crln_year'],
      dtype='object')
```

In [1129]:

```
df2.total_rec_int.unique()
```

Out[1129]:

```
array([861.07, 435.17, 603.65, ..., 231.68, 296.87, 990.28])
```

In [1130]:

```
# Create DataFrame for 'Charged Off' loans
charged_off_df = df2[df2['loan_status'] == 0]

# Create DataFrame for 'Fully Paid' loans
fully_paid_df = df2[df2['loan_status'] == 1]
```

In [1131]:

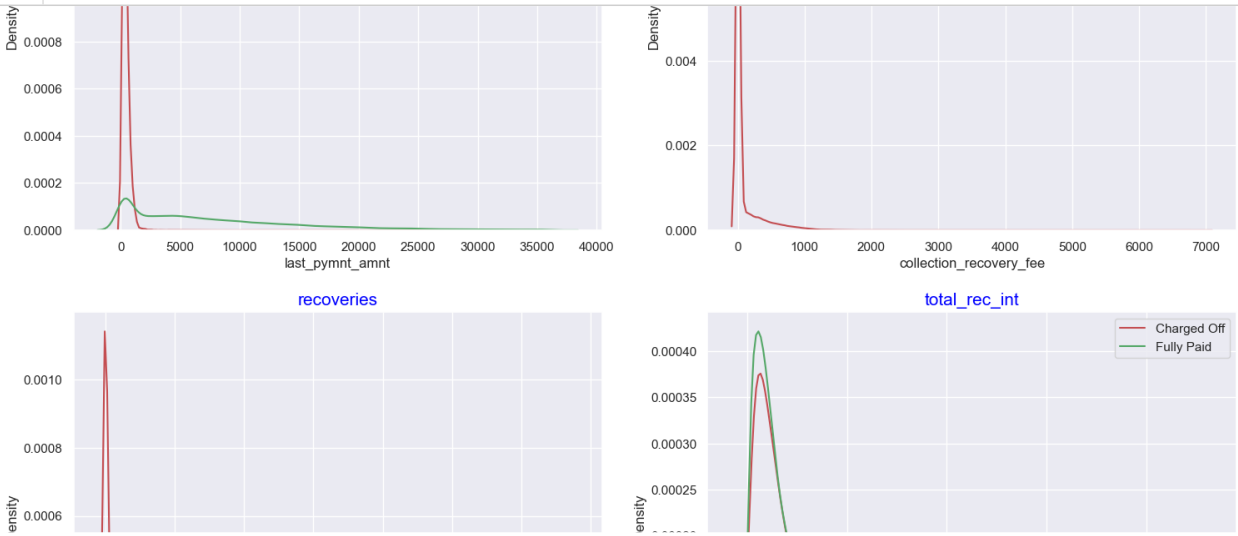
```
charged_off_df.head()
```

Out[1131]:

	term	last_pymnt_amnt	collection_recovery_fee	recoveries	total_rec_int	total_rec_prncp	total_pymnt	out_prncp	total_acc
1	60 months	119.66	1.11	117.08	435.17	456.46	1008.71	0.00	4.00
8	60 months	152.39	2.09	189.06	294.94	162.02	646.02	0.00	13.00
9	60 months	121.45	2.52	269.29	533.42	673.48	1476.19	0.00	3.00
12	36 months	305.38	4.16	444.30	570.26	1256.14	2270.70	0.00	9.00
14	36 months	325.74	6.31	645.10	1393.42	5433.47	7471.99	0.00	29.00

In [1132]:

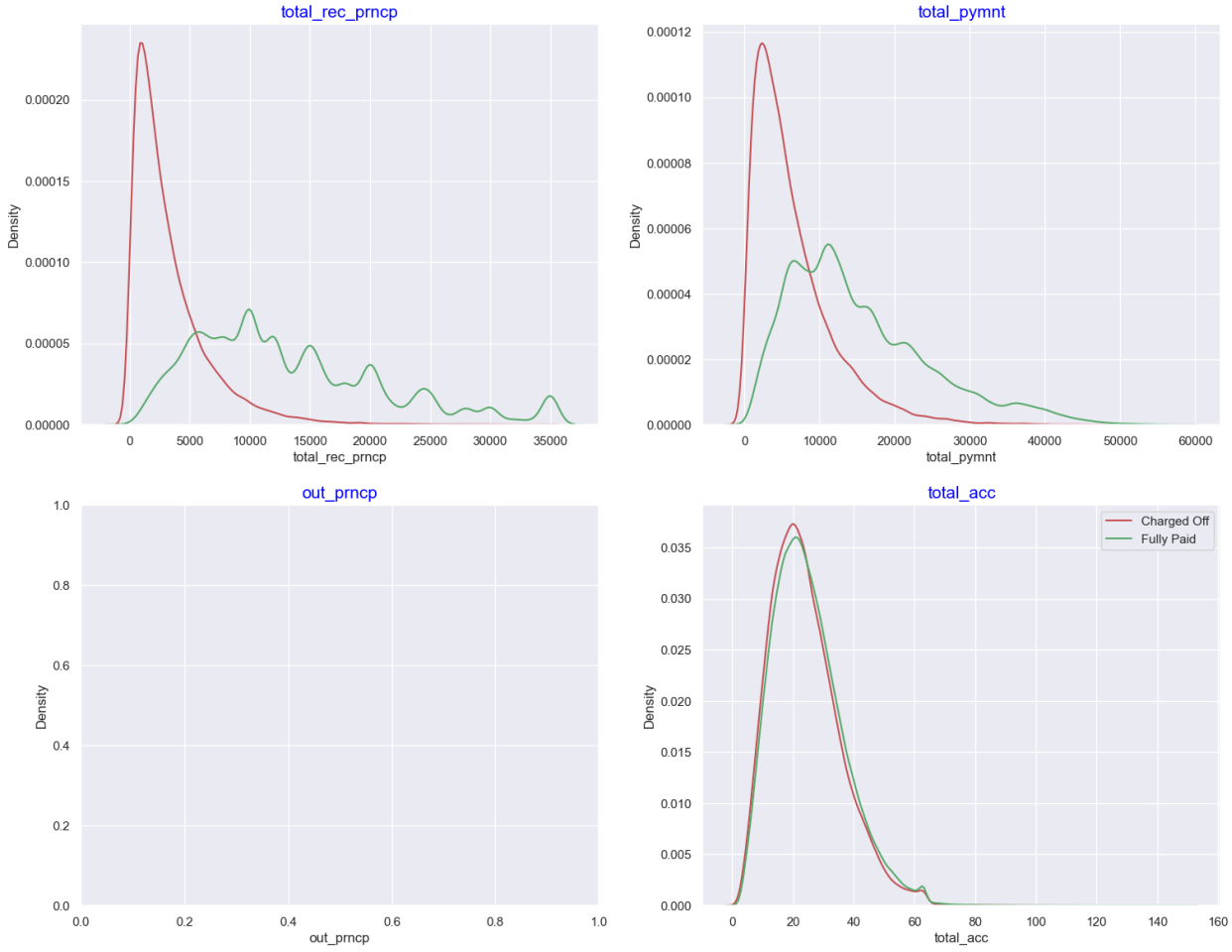
```
#In this code , we are going to  visualize in grapph of the attribute which have higher corelation and i
#identify distribution, comparing distribution,and detect outliar
amount = df2[['last_pymnt_amnt', 'collection_recovery_fee', 'recoveries','total_rec_int']]

fig = plt.figure(figsize=(18,14))

for i in enumerate(amount):
    plt.subplot(2,2,i[0]+1)
    sns.distplot(charged_off_df[i[1]], hist=False, color='r',label ="Charged Off")
    sns.distplot(fully_paid_df[i[1]], hist=False, color='g', label ="Fully Paid")
    plt.title(i[1], fontdict={'fontsize' : 15, 'fontweight' : 5, 'color' : 'Blue'})

plt.legend()

plt.show()
```




```
:recovery and colection recovery only have relation with charge off , in funded amount overlap each
other . not distinguishable. The other feature Based on this observation, it can be inferred that
'recoveries' and 'collection_recovery_fee' represent amounts recovered for loans that have defaulted
or charged off. These features might not be available or relevant for loans that have been fully paid.
```

In [1133]:

```
amount = df2[['total_rec_prncp', 'total_pymnt', 'out_prncp',
       'total_acc']]

fig = plt.figure(figsize=(18,14))

for i in enumerate(amount):
    plt.subplot(2,2,i[0]+1)
    sns.distplot(charged_off_df[i[1]], hist=False, color='r',label ="Charged Off")
    sns.distplot(fully_paid_df[i[1]], hist=False, color='g', label ="Fully Paid")
    plt.title(i[1], fontdict={'fontsize' : 15, 'fontweight' : 5, 'color' : 'Blue'})

plt.legend()

plt.show()
```



In [1134]:

```
#lets check the variable out_prncp
df2.out_prncp.value_counts()
```

Out[1134]:

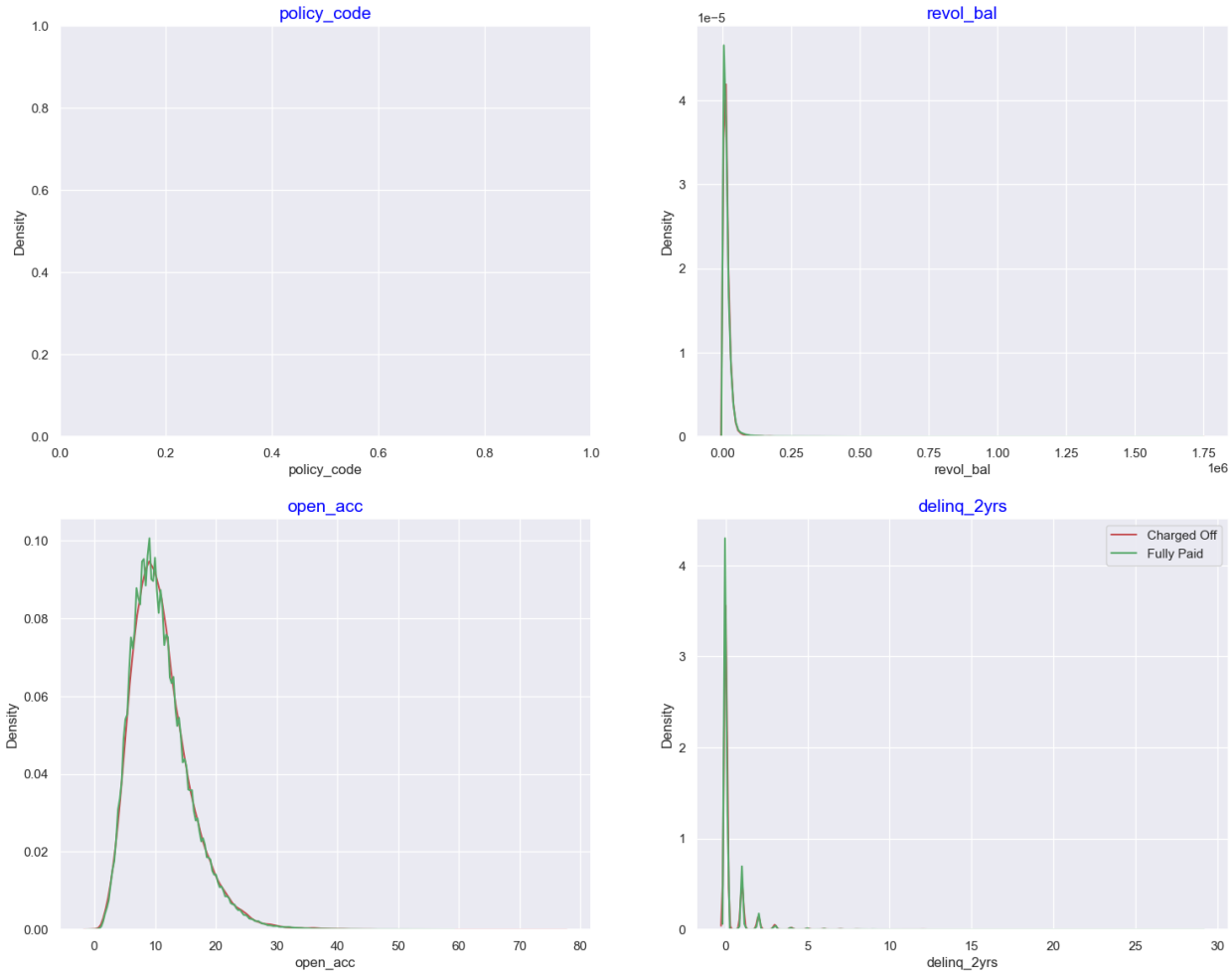
```
0.00    252971
Name: out_prncp, dtype: int64
```

In [1135]:

```
#we can delete this column
df2.drop('out_prncp', axis=1, inplace=True)
```

In [1136]:

```
amount = df2[['policy_code', 'revol_bal', 'open_acc', 'delinq_2yrs',]]

fig = plt.figure(figsize=(18,14))

for i in enumerate(amount):
    plt.subplot(2,2,i[0]+1)
    sns.distplot(charged_off_df[i[1]], hist=False, color='r',label ="Charged Off")
    sns.distplot(fully_paid_df[i[1]], hist=False, color='g', label ="Fully Paid")
    plt.title(i[1], fontdict={'fontsize' : 15, 'fontweight' : 5, 'color' : 'Blue'})

plt.legend()

plt.show()
```



In [1137]:

```
df2.policy_code.value_counts()
```

Out[1137]:

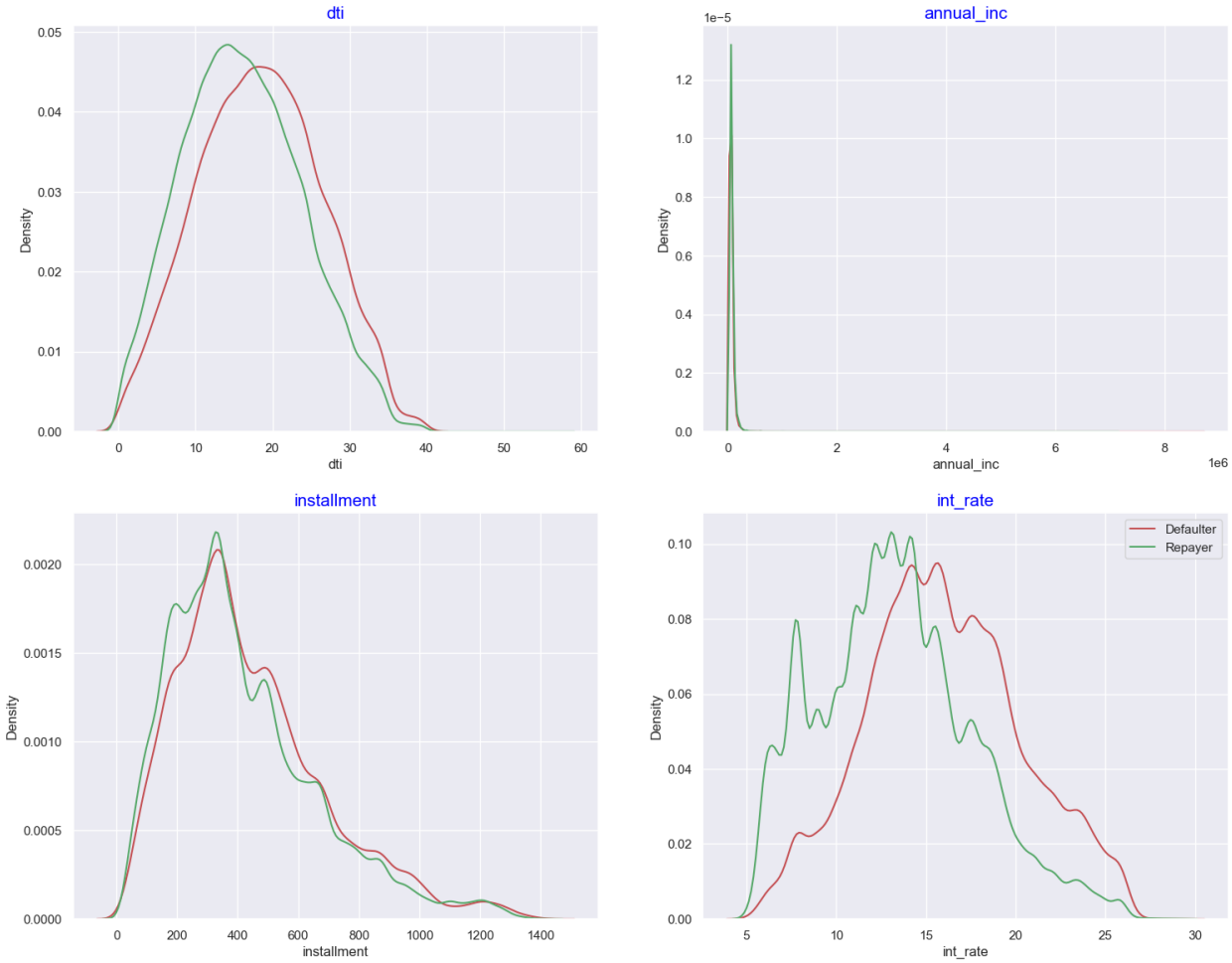
```
1.00    252971
Name: policy_code, dtype: int64
```

In [1138]:

```
#we can delete this column
df2.drop('policy_code', axis=1, inplace=True)
```

In [1139]:

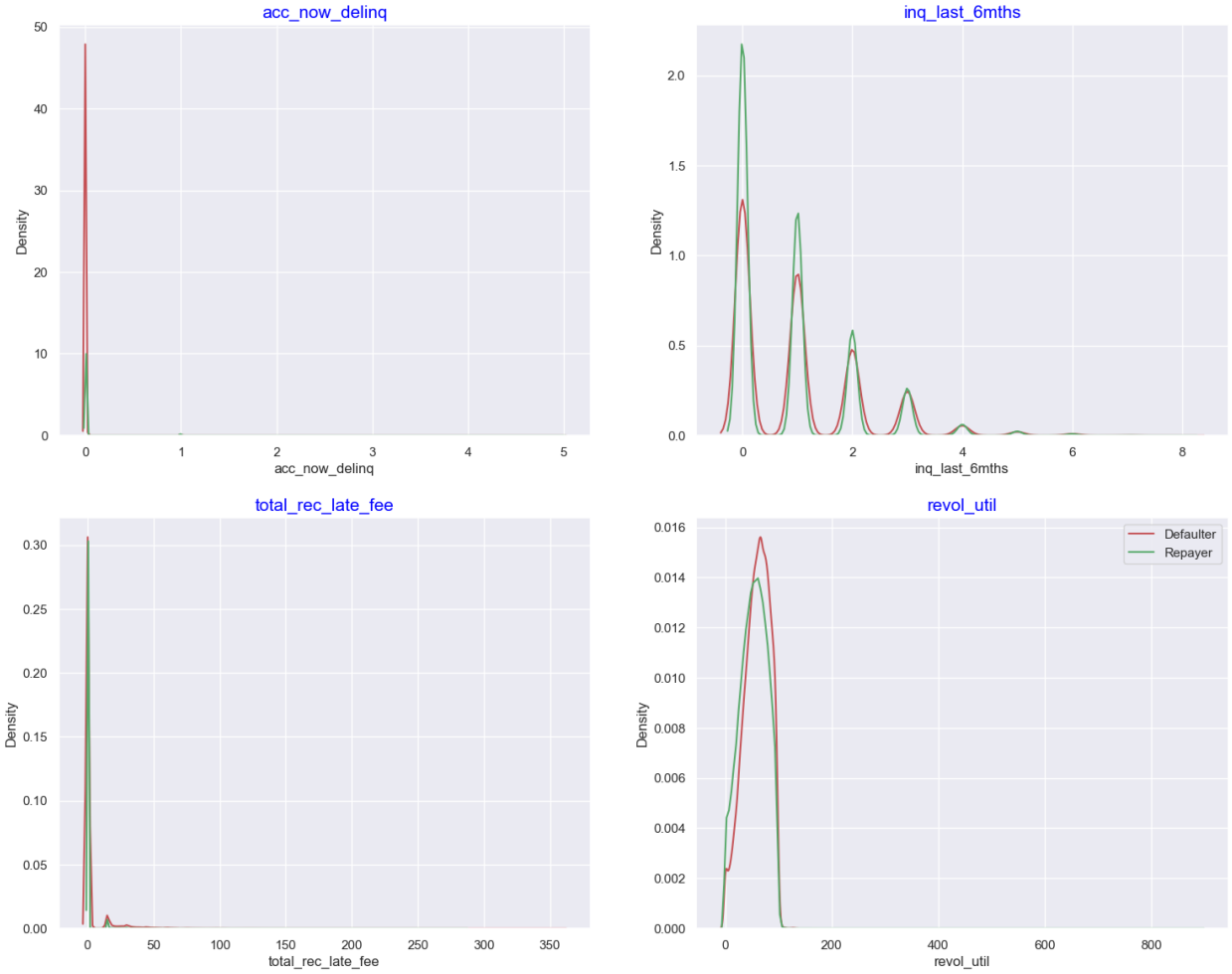
```
amount = df2[['dti', 'annual_inc', 'installment', 'int_rate']]

fig = plt.figure(figsize=(18,14))

for i in enumerate(amount):
    plt.subplot(2,2,i[0]+1)
    sns.distplot(charged_off_df[i[1]], hist=False, color='r',label ="Defaulter")
    sns.distplot(fully_paid_df[i[1]], hist=False, color='g', label ="Repayer")
    plt.title(i[1], fontdict={'fontsize' : 15, 'fontweight' : 5, 'color' : 'Blue'})

plt.legend()

plt.show()
```




```
insight:Though anual income only poor  associated with Loanstatus  but it will be a good variable to
detect charge off , we will keep it do further analysis
```

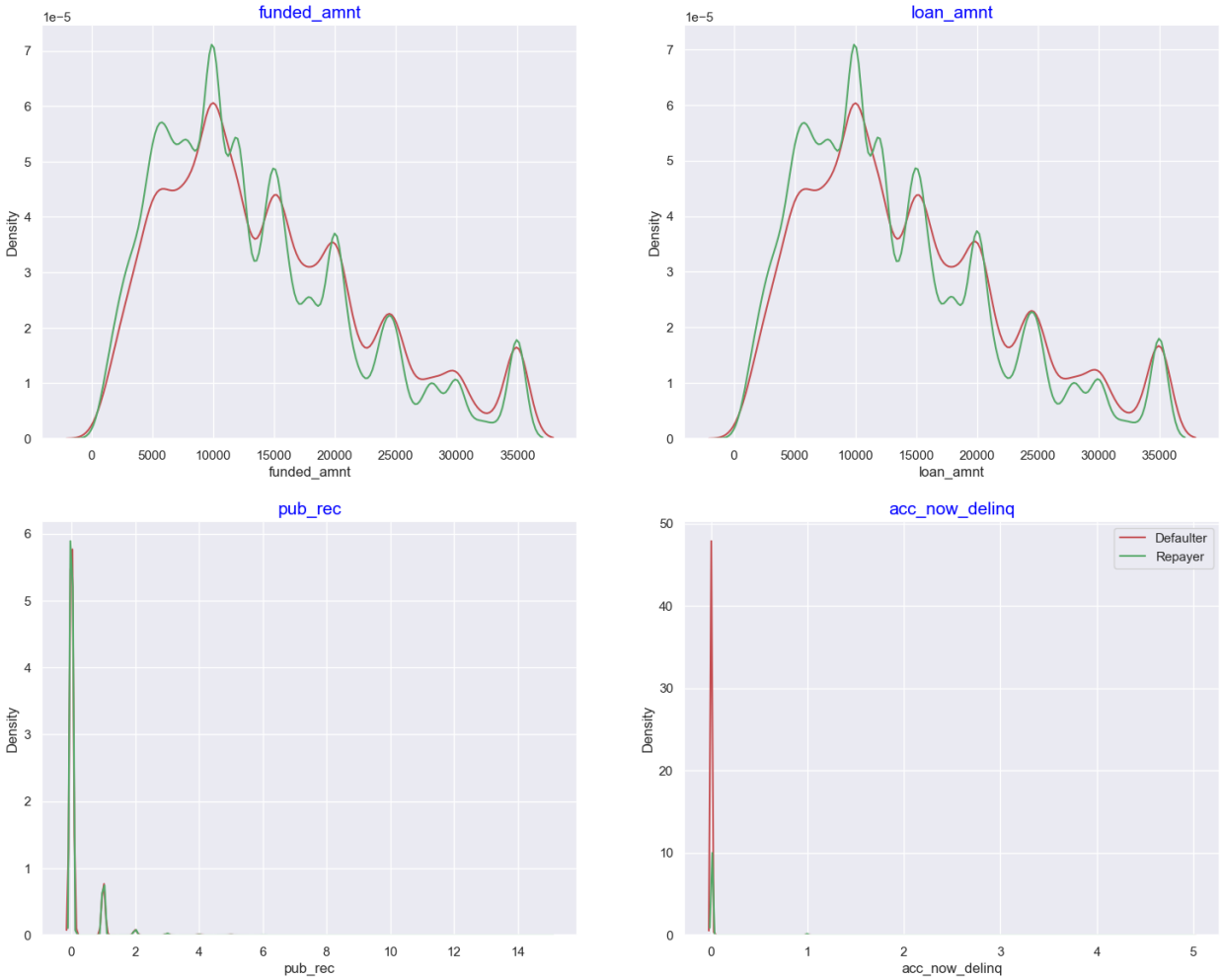
In [1140]:

```
amount = df2[['acc_now_delinq', 'inq_last_6mths',
       'total_rec_late_fee', 'revol_util']]

fig = plt.figure(figsize=(18,14))

for i in enumerate(amount):
    plt.subplot(2,2,i[0]+1)
    sns.distplot(charged_off_df[i[1]], hist=False, color='r',label ="Defaulter")
    sns.distplot(fully_paid_df[i[1]], hist=False, color='g', label ="Repayer")
    plt.title(i[1], fontdict={'fontsize' : 15, 'fontweight' : 5, 'color' : 'Blue'})

plt.legend()

plt.show()
```



In [1141]:

```
amount = df2[['funded_amnt',
       'loan_amnt', 'pub_rec', 'acc_now_delinq']]

fig = plt.figure(figsize=(18,14))

for i in enumerate(amount):
    plt.subplot(2,2,i[0]+1)
    sns.distplot(charged_off_df[i[1]], hist=False, color='r',label ="Defaulter")
    sns.distplot(fully_paid_df[i[1]], hist=False, color='g', label ="Repayer")
    plt.title(i[1], fontdict={'fontsize' : 15, 'fontweight' : 5, 'color' : 'Blue'})

plt.legend()

plt.show()
```

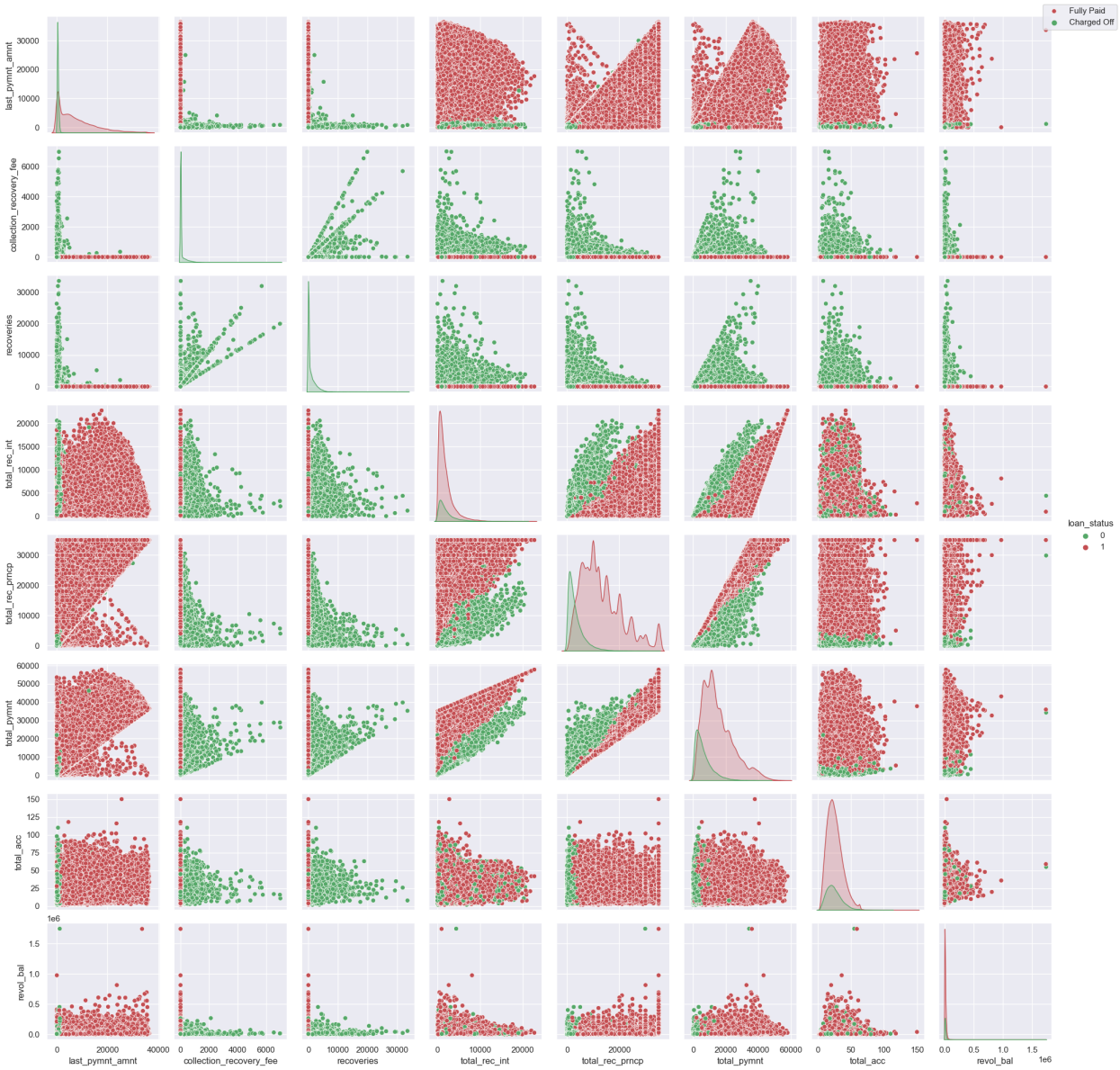


In [ ]:

```
Insight:the variable which is ambigious to detect importance , we will do further analysis
```

In [565]:

```
# Plotting pairplot between amount variable to draw reference against loan repayment status
amount = df2[['last_pymnt_amnt', 'collection_recovery_fee', 'recoveries',
       'total_rec_int', 'total_rec_prncp', 'total_pymnt','total_acc','revol_bal','loan_status']]
ax = sns.pairplot(amount, hue='loan_status', palette=["g", "r"])
ax.fig.legend(labels=['Fully Paid', 'Charged Off'])
plt.show()
```

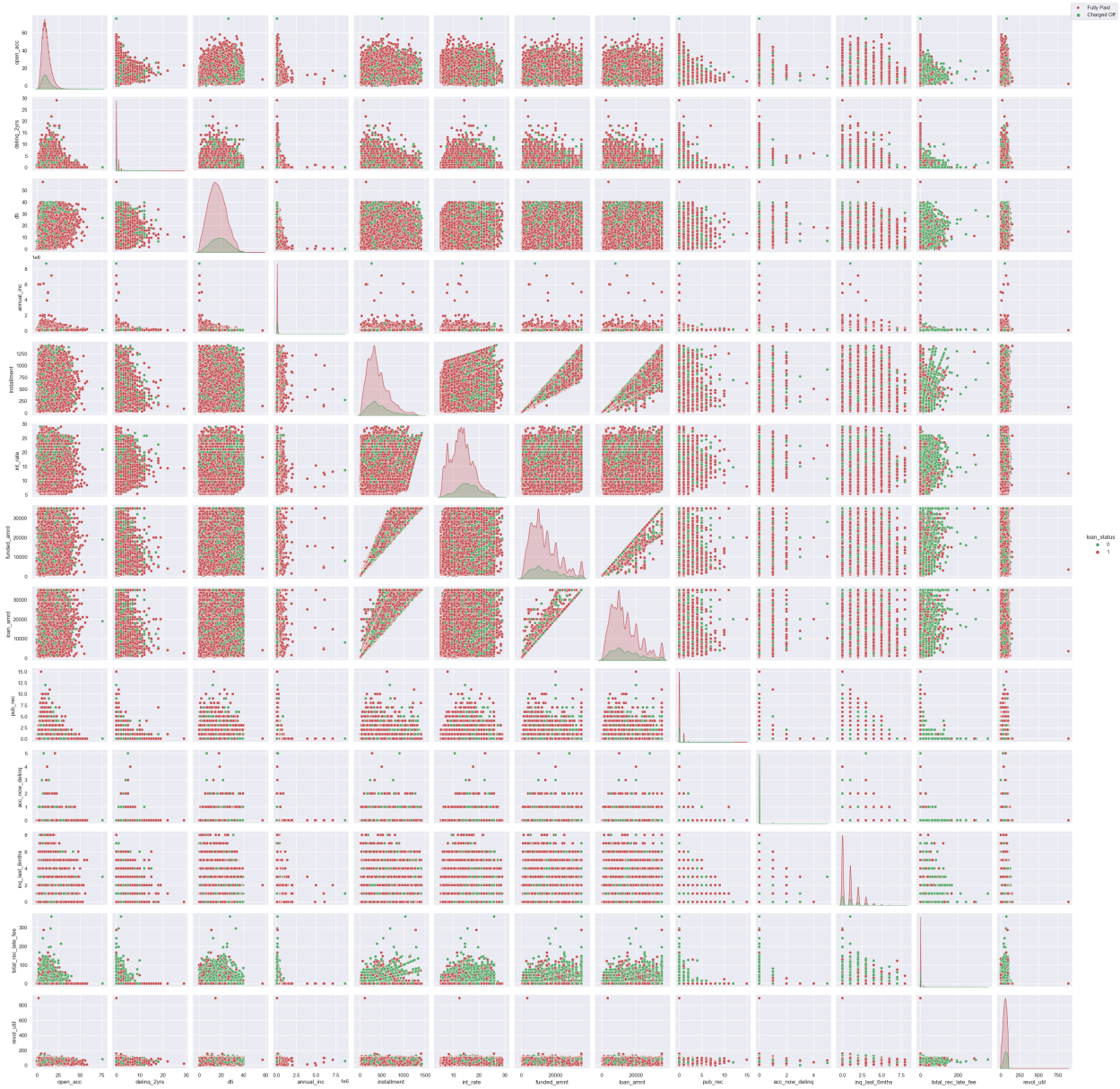


In [569]:

```
amount = df2[['open_acc', 'delinq_2yrs', 'dti', 'annual_inc', 'installment', 'int_rate', 'funded_amnt',
              'loan_amnt', 'pub_rec', 'acc_now_delinq', 'inq_last_6mths', 'total_rec_late_fee',
              'revol_util', 'loan_status']]

ax = sns.pairplot(amount, hue='loan_status', palette=["g", "r"])
ax.fig.legend(labels=['Fully Paid', 'Charged Off'])
```

Out[569]:

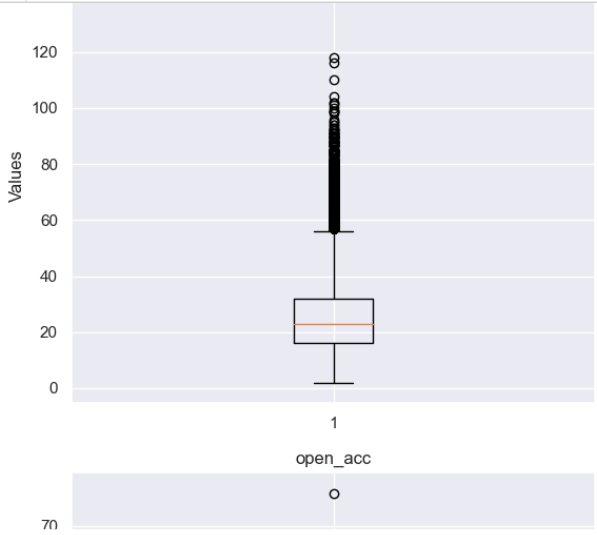
```
<matplotlib.legend.Legend at 0x7f83079ce260>
```

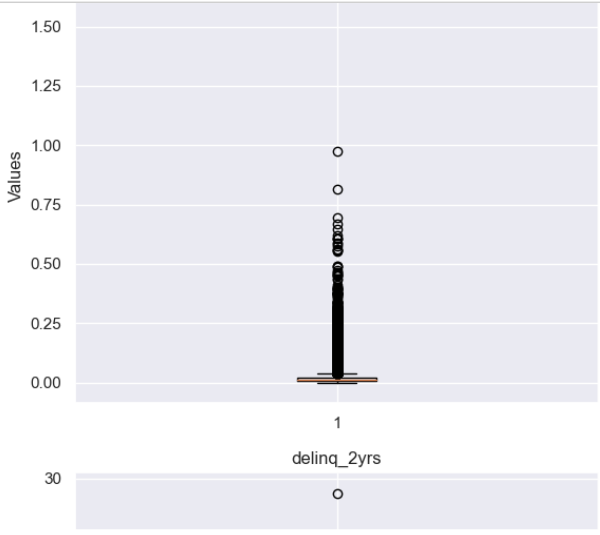



```
Insight: From the pair plot we see some  of the variable negativly associate and some variable
positively and some variable very poor , so we nedd for further analysis
```

In [570]:

```
import math
# Select numerical variables
numerical_vars = df2.select_dtypes(include=['int', 'float']).columns

# Calculate the number of rows and columns for the grid
num_plots = len(numerical_vars)
num_cols = 2
num_rows = math.ceil(num_plots / num_cols)

# Create the subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(12, num_rows*5))

# Flatten the axes array to simplify indexing
axes = axes.flatten()

# Create a box plot for each numerical variable
for i, var in enumerate(numerical_vars):
    axes[i].boxplot(df2[var].dropna().values)
    axes[i].set_title(var)
    axes[i].set_ylabel('Values')

# Hide empty subplots if there are any
for j in range(i+1, num_rows*num_cols):
    fig.delaxes(axes[j])

# Adjust spacing between subplots
fig.tight_layout()

# Display the box plots
plt.show()
```






```
IQR and Upperbound check for removing variable which IQR=0 and UB  select for  select outliar and removing
If the interquartile range (IQR) of a column is 0, it means that all the values in that column are the same. In such cases, the column does not provide any useful information for analysis or modeling because it lacks variability

```

In [605]:

```
# Access the column and convert it to a numpy array
data = df2["revol_util"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[605]:

```
130.3
```

In [604]:

```
# Access the column and convert it to a numpy array
data = df2["total_rec_late_fee"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[604]:

```
0.0
```

In [603]:

```
# Access the column and convert it to a numpy array
data = df2["inq_last_6mths"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[603]:

```
2.5
```

In [602]:

```
# Access the column and convert it to a numpy array
data = df2["acc_now_delinq"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[602]:

```
0.0
```

In [601]:

```
# Access the column and convert it to a numpy array
data = df2["pub_rec"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[601]:

```
0.0
```

In [600]:

```
# Access the column and convert it to a numpy array
data = df2["loan_amnt"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[600]:

```
34825.0
```

In [598]:

```
# Access the column and convert it to a numpy array
data = df2["funded_amnt"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[598]:

```
34387.5
```

In [597]:

```
# Access the column and convert it to a numpy array
data = df2["int_rate"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[597]:

```
25.265
```

In [596]:

```
# Access the column and convert it to a numpy array
data = df2["installment"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[596]:

```
1009.5499999999998
```

In [595]:

```
# Access the column and convert it to a numpy array
data = df2["annual_inc"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[595]:

```
150000.0
```

In [594]:

```
# Access the column and convert it to a numpy array
data = df2["dti"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[594]:

```
38.849999999999994
```

In [593]:

```
# Access the column and convert it to a numpy array
data = df2["delinq_2yrs"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[593]:

```
0.0
```

In [592]:

```
# Access the column and convert it to a numpy array
data = df2["open_acc"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[592]:

```
24.5
```

In [585]:

```


# Access the column and convert it to a numpy array
data = df2["collection_recovery_fee"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[585]:

```
0.0
```

In [587]:

```
# Access the column and convert it to a numpy array
data = df2["last_pymnt_amnt"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[587]:

```
24360.612500000003
```

In [584]:

```
'total_rec_int'
# Access the column and convert it to a numpy array
data = df2["total_rec_int"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[584]:

```
5271.045
```

In [588]:

```
# Access the column and convert it to a numpy array
data = df2["total_pymnt"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[588]:

```
36947.66681713249
```

In [247]:

```

# Access the column and convert it to a numpy array
data = df2["installment"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[247]:

```
1009.5499999999998
```

In [248]:

```
# Access the column and convert it to a numpy array
data = df2["total_rec_prncp"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[248]:

```
32500.0
```

In [583]:

```
# Access the column and convert it to a numpy array
data = df2["last_pymnt_amnt"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[583]:

```
24360.612500000003
```

In [250]:

```
# Access the column and convert it to a numpy array
data = df2["recoveries"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[250]:

```
0.0
```

In [252]:

```

# Access the column and convert it to a numpy array
data = df2["funded_amnt"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[252]:

```
34387.5
```

In [590]:

```
# Access the column and convert it to a numpy array
data = df2["revol_bal"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[590]:

```
38873.25
```

In [591]:

```

# Access the column and convert it to a numpy array
data = df2["total_acc"].values

# Calculate the IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
IQR
upper_bound = Q3 + 1.5 * IQR
upper_bound
```

Out[591]:

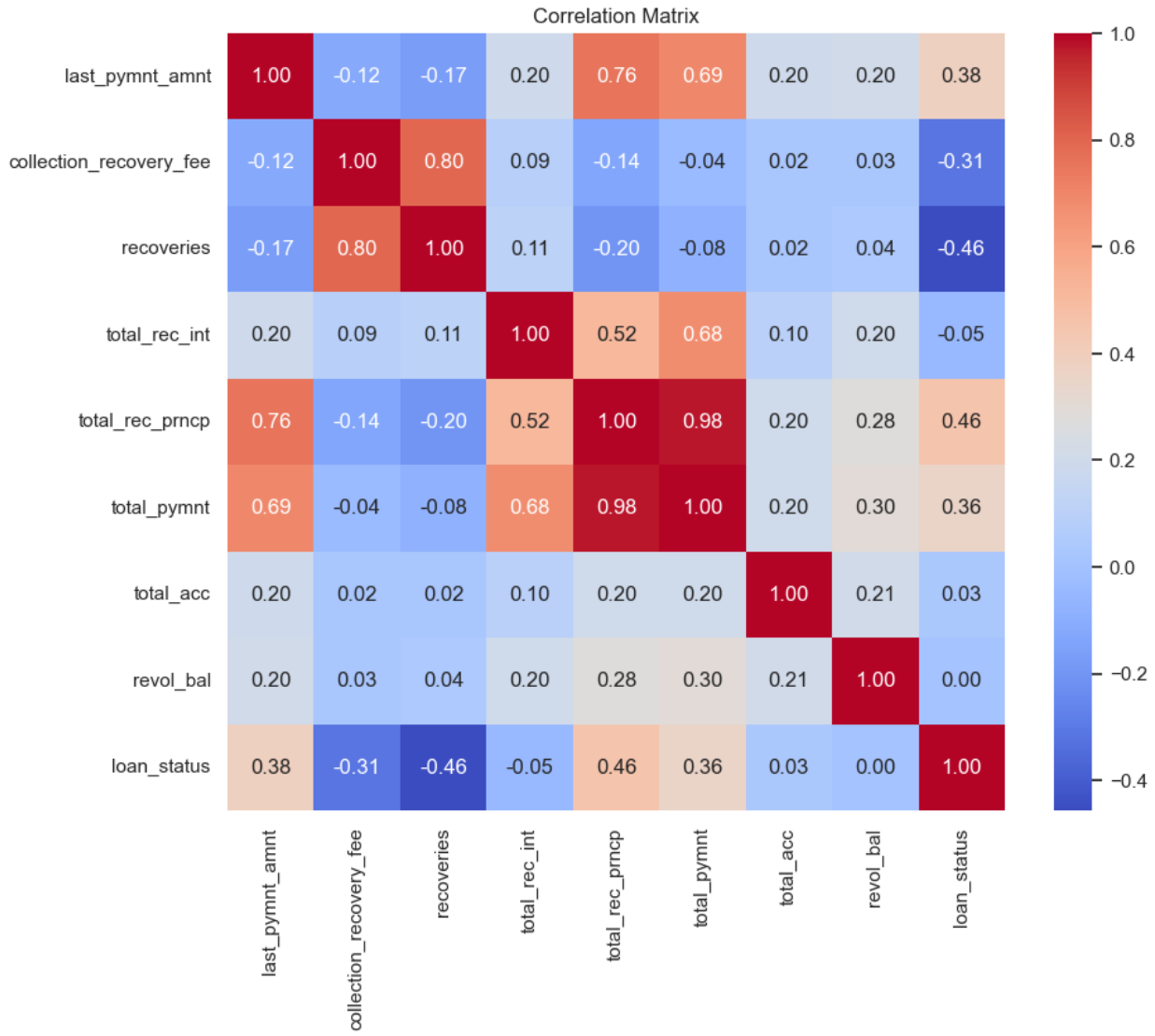
```
56.0
```

```
Insight:If the interquartile range (IQR) of a variable is 0 and the upper bound is also 0, it means
that all the values in that variable are the same. In this case, the variable does not provide any
useful information for analysis or modeling because it lacks variability.

Deleting a variable with an IQR of 0 and an upper bound of 0 is generally a reasonable decision since
it does not contribute any meaningful information to your analysis.
```

```
4
```

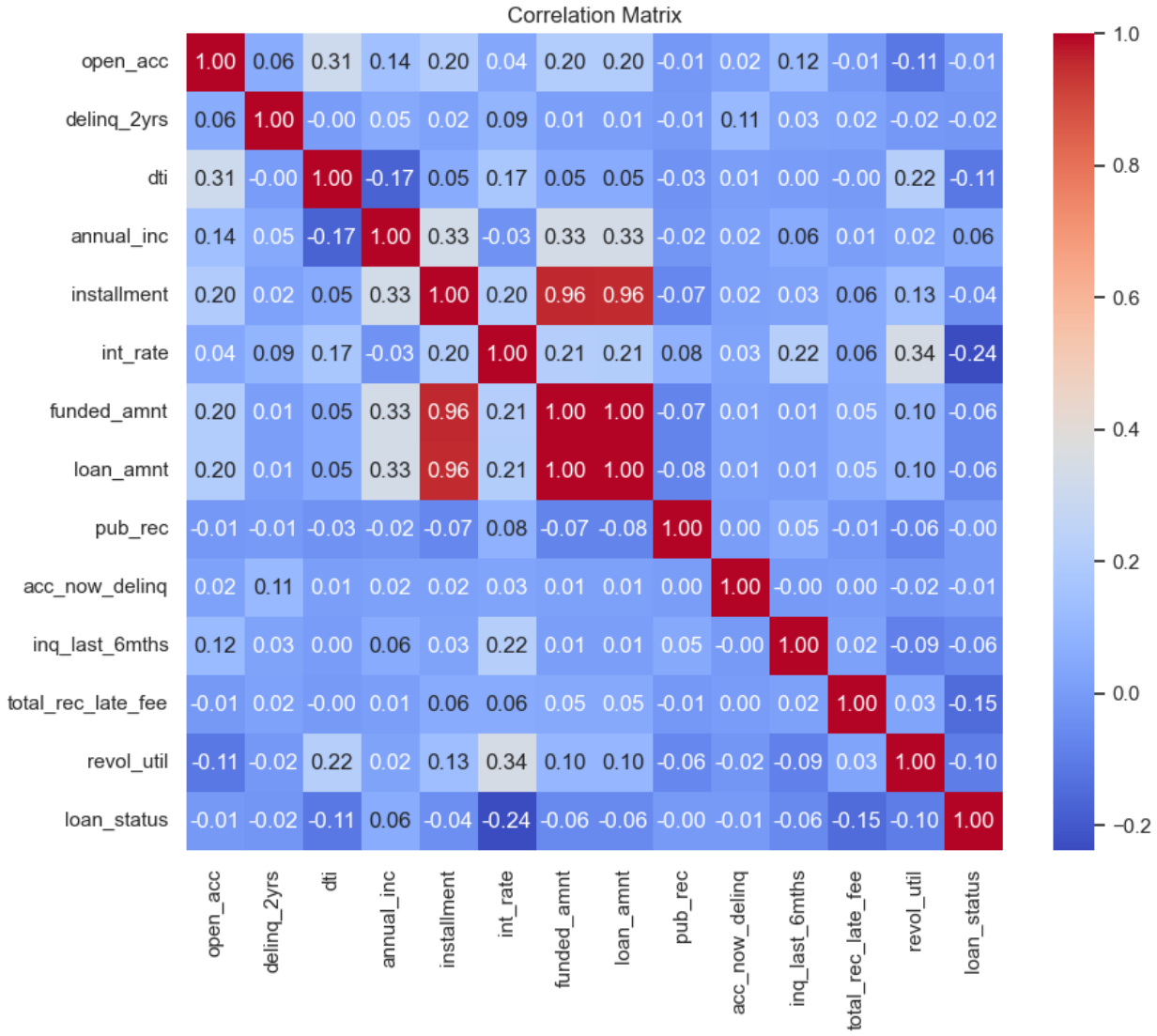
In [577]:

```
#Select numerical variables from the DataFrame
numerical_vars = df2[['last_pymnt_amnt', 'collection_recovery_fee', 'recoveries',
       'total_rec_int', 'total_rec_prncp', 'total_pymnt','total_acc','revol_bal','loan_status']]  # Rep

# Calculate the correlation matrix
corr_matrix = numerical_vars.corr()

# Create the correlation matrix plot
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", square=True)
plt.title('Correlation Matrix')
plt.show()
```



In [578]:

```
#Select numerical variables from the DataFrame
numerical_vars = df2[['open_acc', 'delinq_2yrs', 'dti', 'annual_inc', 'installment', 'int_rate', 'funded
              'loan_amnt', 'pub_rec', 'acc_now_delinq', 'inq_last_6mths', 'total_rec_late_fee',
              'revol_util', 'loan_status']]  # Replace 'var1', 'var2', 'var3', ... with your variable na

# Calculate the correlation matrix
corr_matrix = numerical_vars.corr()

# Create the correlation matrix plot
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", square=True)
plt.title('Correlation Matrix')
plt.show()
```




```
Insight:From the heat map anlysis we can see some of the variable have very poor relationship with our
target variable loanstatus, so  will delete those variable from df2 And variable which iQR=0  we will
delete and create new df
```

In [1142]:

```
df2.columns
```

Out[1142]:

```
Index(['term', 'last_pymnt_amnt', 'collection_recovery_fee', 'recoveries',
       'total_rec_int', 'total_rec_prncp', 'total_pymnt', 'total_acc',
       'revol_bal', 'open_acc', 'delinq_2yrs', 'dti', 'annual_inc',
       'installment', 'int_rate', 'funded_amnt', 'loan_amnt', 'pub_rec',
       'acc_now_delinq', 'inq_last_6mths', 'total_rec_late_fee', 'revol_util',
       'grade', 'home_ownership', 'purpose', 'zip_code', 'addr_state',
       'initial_list_status', 'last_pymnt_d', 'last_credit_pull_d',
       'loan_status', 'last_credit_pull_year', 'last_pymnt_d_year',
       'earliest_crln_year'],
      dtype='object')
```

In [ ]:

```
# final numerical variable selection on the basis of distribution, pairplot trends , boxplot and heatma
```

In [1143]:

```
columns_to_delete = ['collection_recovery_fee', 'recoveries','revol_bal', 'open_acc', 'delinq_2yrs','pu
       'acc_now_delinq', 'inq_last_6mths', 'total_rec_late_fee','last_pymnt_d', 'last_credit_pull_d',]
df3 = df2.drop(columns=columns_to_delete)
```

In [1144]:

```
df3.shape
```

Out[1144]:

```
(252971, 23)
```

In [1145]:

```
df3.isnull().sum().sum()
```

Out[1145]:

```
0
```

In [1146]:

```
df3.isnull().any()
```

Out[1146]:

```
term                    False
last_pymnt_amnt         False
total_rec_int           False
total_rec_prncp         False
total_pymnt             False
total_acc               False
dti                     False
annual_inc              False
installment             False
int_rate                False
funded_amnt             False
loan_amnt               False
revol_util              False
grade                   False
home_ownership          False
purpose                 False
zip_code                False
addr_state              False
initial_list_status     False
loan_status             False
last_credit_pull_year   False
last_pymnt_d_year       False
earliest_crln_year      False
dtype: bool
```

In [1147]:

```
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 252971 entries, 0 to 887371
Data columns (total 23 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   term                   252971 non-null  object 
 1   last_pymnt_amnt        252971 non-null  float64
 2   total_rec_int          252971 non-null  float64
 3   total_rec_prncp        252971 non-null  float64
 4   total_pymnt            252971 non-null  float64
 5   total_acc              252971 non-null  float64
 6   dti                    252971 non-null  float64
 7   annual_inc             252971 non-null  float64
 8   installment            252971 non-null  float64
 9   int_rate               252971 non-null  float64
 10  funded_amnt            252971 non-null  float64
 11  loan_amnt              252971 non-null  float64
 12  revol_util             252971 non-null  float64
 13  grade                  252971 non-null  object 
 14  home_ownership         252971 non-null  object 
 15  purpose                252971 non-null  object 
 16  zip_code               252971 non-null  object 
 17  addr_state             252971 non-null  object 
 18  initial_list_status    252971 non-null  object 
 19  loan_status            252971 non-null  int64  
 20  last_credit_pull_year  252971 non-null  float64
 21  last_pymnt_d_year      252971 non-null  float64
 22  earliest_crln_year     252971 non-null  int64  
dtypes: float64(14), int64(2), object(7)
memory usage: 46.3+ MB
```

In [1148]:

```
#Lebel encoding catagorical variable
print([column for column in df3.columns if df3[column].dtype == object])
```

```
['term', 'grade', 'home_ownership', 'purpose', 'zip_code', 'addr_state', 'initial_list_statu
s']
```

In [1149]:

```
df3.term.unique()
```

Out[1149]:

```
array([' 36 months', ' 60 months'], dtype=object)
```

In [1150]:

```
term_values = {' 36 months': 36, ' 60 months': 60}
df3['term'] = df3.term.map(term_values)
```

In [1151]:

```
df3.head()
```

Out[1151]:

	term	last_pymnt_amnt	total_rec_int	total_rec_prncp	total_pymnt	total_acc	dti	annual_inc	installment	int_rate	funded_ar
0	36	171.62	861.07	5000.00	5861.07	9.00	27.65	24000.00	162.87	10.65	5000
1	60	119.66	435.17	456.46	1008.71	4.00	1.00	30000.00	59.83	15.27	2500
2	36	649.91	603.65	2400.00	3003.65	10.00	8.72	12252.00	84.33	15.96	2400
3	36	357.48	2209.33	10000.00	12226.30	37.00	20.00	49200.00	339.31	13.49	10000
5	36	161.03	631.38	5000.00	5631.38	12.00	11.20	36000.00	156.46	7.90	5000

In [1152]:

```
dummies = ['grade', 'home_ownership','purpose','initial_list_status']
df3 = pd.get_dummies(df3, columns=dummies, drop_first=True)
```

we see our zipcode column contain mixed word, lets check if we find Zipcode in address

In [1153]:

```
df3.zip_code.unique()
```

Out[1153]:

```
array(['860xx', '309xx', '606xx', '917xx', '852xx', '900xx', '958xx',
       '774xx', '853xx', '913xx', '245xx', '951xx', '641xx', '921xx',
       '067xx', '890xx', '770xx', '335xx', '799xx', '605xx', '103xx',
       '150xx', '326xx', '564xx', '141xx', '080xx', '974xx', '934xx',
       '405xx', '946xx', '445xx', '850xx', '292xx', '088xx', '180xx',
       '029xx', '700xx', '010xx', '441xx', '104xx', '061xx', '616xx',
       '947xx', '914xx', '765xx', '980xx', '017xx', '972xx', '752xx',
       '787xx', '077xx', '540xx', '225xx', '440xx', '437xx', '559xx',
       '912xx', '325xx', '300xx', '923xx', '352xx', '013xx', '146xx',
       '074xx', '786xx', '937xx', '331xx', '115xx', '191xx', '114xx',
       '908xx', '902xx', '750xx', '950xx', '329xx', '226xx', '992xx',
       '614xx', '083xx', '100xx', '926xx', '931xx', '712xx', '060xx',
       '707xx', '342xx', '604xx', '895xx', '430xx', '919xx', '996xx',
       '891xx', '935xx', '801xx', '928xx', '233xx', '927xx', '970xx',
       '211xx', '303xx', '070xx', '194xx', '263xx', '403xx', '301xx',
       '553xx', '993xx', '312xx', '432xx', '602xx', '216xx', '151xx',
       '971xx', '305xx', '334xx', '050xx', '129xx', '925xx', '483xx',
       '760xx', '961xx', '200xx', '085xx', '981xx', '330xx', '601xx',
```

In [1154]:

```
df3['zip_code'] = df3['zip_code'].str.extract('(\d+)').astype(int)

```

In [1155]:

```
df3.addr_state.unique()
```

Out[1155]:

```
array(['AZ', 'GA', 'IL', 'CA', 'TX', 'VA', 'MO', 'CT', 'UT', 'FL', 'NY',
       'PA', 'MN', 'NJ', 'OR', 'KY', 'OH', 'SC', 'RI', 'LA', 'MA', 'WA',
       'WI', 'AL', 'NV', 'AK', 'CO', 'MD', 'WV', 'VT', 'MI', 'DC', 'SD',
       'NC', 'AR', 'NM', 'KS', 'HI', 'OK', 'MT', 'WY', 'NH', 'DE', 'MS',
       'TN', 'IA', 'NE', 'ID', 'IN', 'ME', 'ND'], dtype=object)
```

In [1156]:

```
df3 = pd.get_dummies(df3, columns=['addr_state'], prefix='state')
```

In [1157]:

```
df3.head()
```

Out[1157]:

	term	last_pymnt_amnt	total_rec_int	total_rec_prncp	total_pymnt	total_acc	dti	annual_inc	installment	int_rate	funded_ar
0	36	171.62	861.07	5000.00	5861.07	9.00	27.65	24000.00	162.87	10.65	5000
1	60	119.66	435.17	456.46	1008.71	4.00	1.00	30000.00	59.83	15.27	2500
2	36	649.91	603.65	2400.00	3003.65	10.00	8.72	12252.00	84.33	15.96	2400
3	36	357.48	2209.33	10000.00	12226.30	37.00	20.00	49200.00	339.31	13.49	10000
5	36	161.03	631.38	5000.00	5631.38	12.00	11.20	36000.00	156.46	7.90	5000

5 rows × 94 columns

In [1158]:

```
1 df3.isnull().any()
```

Out[1158]:

```
term                          False
last_pymnt_amnt               False
total_rec_int                 False
total_rec_prncp               False
total_pymnt                   False
total_acc                     False
dti                           False
annual_inc                    False
installment                   False
int_rate                      False
funded_amnt                   False
loan_amnt                     False
revol_util                    False
zip_code                      False
loan_status                   False
last_credit_pull_year         False
last_pymnt_d_year             False
earliest_crln_year            False
grade_B                       False
grade_C                       False
grade_D                       False
grade_E                       False
grade_F                       False
grade_G                       False
home_ownership_MORTGAGE       False
home_ownership_NONE           False
home_ownership_OTHER          False
home_ownership_OWN            False
home_ownership_RENT           False
purpose_credit_card           False
purpose_debt_consolidation    False
purpose_educational           False
purpose_home_improvement      False
purpose_house                 False
purpose_major_purchase        False
purpose_medical               False
purpose_moving                False
purpose_other                 False
purpose_renewable_energy      False
purpose_small_business        False
purpose_vacation              False
purpose_wedding               False
initial_list_status_w         False
state_AK                      False
state_AL                      False
state_AR                      False
state_AZ                      False
state_CA                      False
state_CO                      False
state_CT                      False
state_DC                      False
state_DE                      False
state_FL                      False
state_GA                      False
state_HI                      False
state_IA                      False
state_ID                      False
state_IL                      False
state_IN                      False
state_KS                      False
state_KY                      False
state_LA                      False
state_MA                      False
state_MD                      False
state_ME                      False
state_MI                      False
state_MN                      False
state_MO                      False
state_MS                      False
state_MT                      False
state_NC                      False
state_ND                      False
state_NE                      False
state_NH                      False
state_NJ                      False
state_NM                      False
state_NV                      False
state_NY                      False
```

```
state_OH                    False
state_OK                    False
state_OR                    False
state_PA                    False
state_RI                    False
state_SC                    False
state_SD                    False
state_TN                    False
state_TX                    False
state_UT                    False
state_VA                    False
state_VT                    False
state_WA                    False
state_WI                    False
state_WV                    False
state_WY                    False
dtype: bool
```

In [1160]:

```
import statsmodels.api as sm

# X: independent variables, y: dependent variable
X = df3.drop(columns=['loan_status'])
y = df3['loan_status']

# Add a constant term to X for intercept in the model
X = sm.add_constant(X)

# Fit the model
model = sm.OLS(y, X).fit()

# Get the p-values of the features
p_values = model.pvalues

# Perform backward elimination
while p_values.max() > 0.05:
    # Find the index of the least significant feature
    feature_to_remove = p_values.idxmax()

    # Remove the feature from X
    X = X.drop(columns=[feature_to_remove])

    # Fit the model again
    model = sm.OLS(y, X).fit()

    # Update the p-values
    p_values = model.pvalues

# Final selected features
selected_features = X.columns

# Print the selected features
print("Selected features:", selected_features)

```

```
Selected features: Index(['const', 'term', 'last_pymnt_amnt', 'total_rec_int', 'total_rec_prnc
p',
       'total_pymnt', 'total_acc', 'dti', 'annual_inc', 'installment',
       'int_rate', 'funded_amnt', 'revol_util', 'zip_code',
       'last_credit_pull_year', 'last_pymnt_d_year', 'grade_B', 'grade_D',
       'grade_E', 'grade_F', 'grade_G', 'home_ownership_OWN',
       'home_ownership_RENT', 'purpose_credit_card',
       'purpose_debt_consolidation', 'purpose_home_improvement',
       'purpose_house', 'purpose_medical', 'purpose_moving', 'purpose_other',
       'purpose_renewable_energy', 'purpose_small_business',
       'purpose_vacation', 'initial_list_status_w', 'state_AK', 'state_AL',
       'state_AR', 'state_AZ', 'state_CA', 'state_CO', 'state_CT', 'state_DC',
       'state_DE', 'state_FL', 'state_GA', 'state_HI', 'state_IA', 'state_ID',
       'state_IL', 'state_IN', 'state_KS', 'state_KY', 'state_LA', 'state_MA',
       'state_MD', 'state_ME', 'state_MI', 'state_MN', 'state_MO', 'state_MS',
       'state_MT', 'state_NC', 'state_ND', 'state_NE', 'state_NH', 'state_NJ',
       'state_NM', 'state_NV', 'state_NY', 'state_OH', 'state_OK', 'state_OR',
       'state_PA', 'state_RI', 'state_SC', 'state_SD', 'state_TN', 'state_TX',
       'state_UT', 'state_VA', 'state_VT', 'state_WA', 'state_WI', 'state_WV',
       'state_WY'],
      dtype='object')
```

In [1170]:

```
train, test = train_test_split(df3, test_size=0.35, random_state=42)

print(train.shape)
print(test.shape)
```

```
(164431, 94)
(88540, 94)
```

```
'last_pymnt_amnt', 'collection_recovery_fee', 'recoveries',
       'total_rec_int', 'total_rec_prncp', 'total_pymnt','total_acc','revol_bal','loan_status']
    'open_acc', 'delinq_2yrs', 'dti', 'annual_inc', 'installment', 'int_rate', 'funded_amnt',
              'loan_amnt', 'pub_rec', 'acc_now_delinq', 'inq_last_6mths', 'total_rec_late_fee',
```

```
            'revol_util', 'loan_status'
```

In [1171]:

```
# outliar treatment of numerical variable
print(train.shape)
train = train[train['last_pymnt_amnt'] <= 24000]
train = train[train['total_rec_prncp'] <= 32500]
train = train[train['total_pymnt'] <= 37000]
train = train[train['total_acc'] <= 56]
train = train[train['dti'] <= 39]
train = train[train['annual_inc'] <= 150000]
train = train[train['installment'] <= 1000]
train = train[train['int_rate'] <= 25]
train = train[train['funded_amnt'] <= 34000]
train = train[train['loan_amnt'] <= 334825]
train = train[train['revol_util'] <= 130]



print(train.shape)
```

```
(164431, 94)
(145573, 94)
```

In [1172]:

```
#normalizing the data
X_train, y_train = train.drop('loan_status', axis=1), train.loan_status
X_test, y_test = test.drop('loan_status', axis=1), test.loan_status
```

In [1173]:

```
1 X_train.dtypes
```

Out[1173]:

```
term                                 int64
last_pymnt_amnt                    float64
total_rec_int                      float64
total_rec_prncp                    float64
total_pymnt                        float64
total_acc                          float64
dti                                float64
annual_inc                         float64
installment                        float64
int_rate                           float64
funded_amnt                        float64
loan_amnt                          float64
revol_util                         float64
zip_code                             int64
last_credit_pull_year              float64
last_pymnt_d_year                  float64
earliest_crln_year                   int64
grade_B                              uint8
grade_C                              uint8
grade_D                              uint8
grade_E                              uint8
grade_F                              uint8
grade_G                              uint8
home_ownership_MORTGAGE              uint8
home_ownership_NONE                  uint8
home_ownership_OTHER                 uint8
home_ownership_OWN                   uint8
home_ownership_RENT                  uint8
purpose_credit_card                  uint8
purpose_debt_consolidation           uint8
purpose_educational                  uint8
purpose_home_improvement             uint8
purpose_house                        uint8
purpose_major_purchase               uint8
purpose_medical                      uint8
purpose_moving                       uint8
purpose_other                        uint8
purpose_renewable_energy             uint8
purpose_small_business               uint8
purpose_vacation                     uint8
purpose_wedding                      uint8
initial_list_status_w                uint8
state_AK                             uint8
state_AL                             uint8
state_AR                             uint8
state_AZ                             uint8
state_CA                             uint8
state_CO                             uint8
state_CT                             uint8
state_DC                             uint8
state_DE                             uint8
state_FL                             uint8
state_GA                             uint8
state_HI                             uint8
state_IA                             uint8
state_ID                             uint8
state_IL                             uint8
state_IN                             uint8
state_KS                             uint8
state_KY                             uint8
state_LA                             uint8
state_MA                             uint8
state_MD                             uint8
state_ME                             uint8
state_MI                             uint8
state_MN                             uint8
state_MO                             uint8
state_MS                             uint8
state_MT                             uint8
state_NC                             uint8
state_ND                             uint8
state_NE                             uint8
state_NH                             uint8
state_NJ                             uint8
state_NM                             uint8
state_NV                             uint8
state_NY                             uint8
state_OH                             uint8
```

In [174]:

```
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [175]:

```
# Initialize a dictionary to store the accuracy scores
accuracy_scores = {}

# Define the classifier models
models = {
    'Random Forest': RandomForestClassifier(),
    'Logistic Regression': LogisticRegression(),
    'Support Vector Machine': SVC(),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Gradient_boosting' : GradientBoostingClassifier(),
    'AdaBoostClassifier': AdaBoostClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'Naive Bayes': GaussianNB()
}

# Iterate over each model
for model_name, model in models.items():
    # Fit the model on the training data
    model.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = model.predict(X_test)

    # Calculate the accuracy score
    accuracy = accuracy_score(y_test, y_pred)

    # Store the accuracy score in the dictionary
    accuracy_scores[model_name] = accuracy

# Sort the accuracy scores in descending order
sorted_scores = sorted(accuracy_scores.items(), key=lambda x: x[1], reverse=True)

# Print the ranking and accuracy scores of the models
print("Model Rankings based on Accuracy Score:")
for rank, (model_name, accuracy) in enumerate(sorted_scores, start=1):
    print(f"Rank {rank}: {model_name} - Accuracy: {accuracy:.4f}")

# Print all the accuracy scores
print("\nAll Accuracy Scores:")
for model_name, accuracy in accuracy_scores.items():
    print(f"{model_name}: {accuracy:.4f}")
```

```
Model Rankings based on Accuracy Score:
Rank 1: Naive Bayes - Accuracy: 0.7780

All Accuracy Scores:
Naive Bayes: 0.7780
```

```
state_OK                        uint8
state_OR                        uint8
state_PA                        uint8
state_RI                        uint8
state_SC                        uint8
state_SD                        uint8
state_TN                        uint8
state_TX                        uint8
state_UT                        uint8
state_VA                        uint8
state_VT                        uint8
state_WA                        uint8
state_WI                        uint8
state_WV                        uint8
state_WY                        uint8
dtype: object
```

In [1234]:

```
Regression=GaussianNB()
```

In [1235]:

```
Regression.fit(X_train, y_train)
```

Out[1235]:

GaussianNB()

In [1236]:

```
y_pred =Regression.predict(X_test)
```

In [1237]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[15343   531]
 [19128 53538]]
```

In [1238]:

```
bias =Regression.score(X_train, y_train)
bias
```

Out[1238]:

```
0.7590830717234652
```

In [1239]:

```
variance = Regression.score(X_test, y_test)
variance
```

Out[1239]:

```
0.7779647616896318
```

In [1240]:

```
#ROC AND AUC
from sklearn.metrics import roc_curve, roc_auc_score
```

In [1241]:

```
# Compute the False Positive Rate (FPR), True Positive Rate (TPR), and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
```

In [1242]:

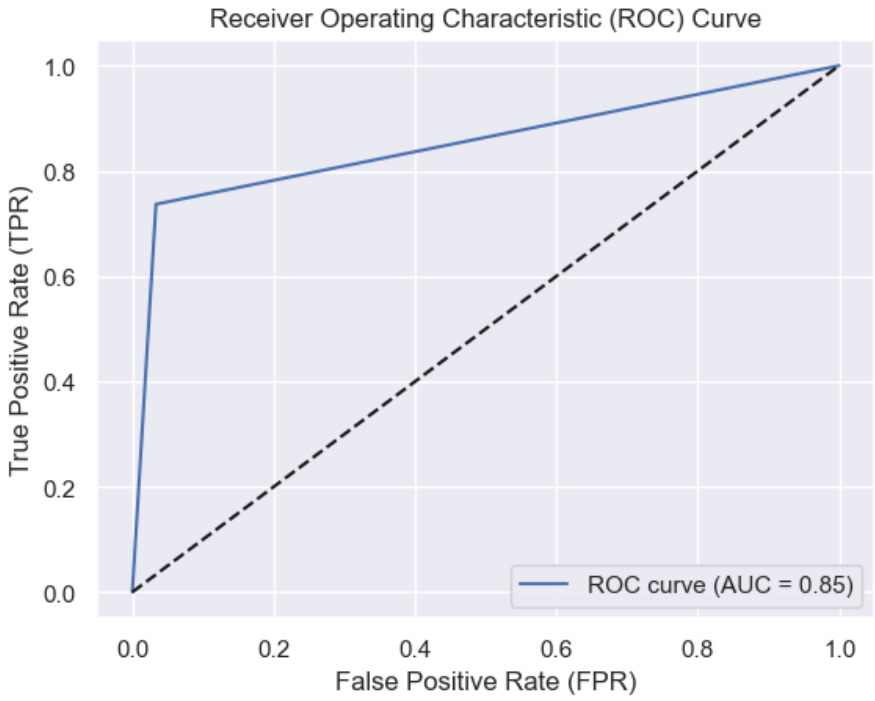
```
auc = roc_auc_score(y_test, y_pred)
auc
```

Out[1242]:

```
0.8516586505944286
```

In [1243]:

```
# Plotting the ROC curve
plt.plot(fpr, tpr, label='ROC curve (AUC = {:.2f})'.format(auc))
plt.plot([0, 1], [0, 1], 'k--')  # Random guess line
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



In [1186]:

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = Regression, X = X_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 75.07 %
Standard Deviation: 1.79 %
```

In [1187]:

```
from sklearn.model_selection import GridSearchCV
```

In [1188]:

```
param_grid = {
    'var_smoothing': [1e-9, 1e-8, 1e-7]  # Example grid with different values of var_smoothing
}
```

In [1189]:

```
# Create an object of GridSearchCV
grid_search = GridSearchCV(estimator= Regression, param_grid=param_grid, cv=10)
grid_search = grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print("Best Accuracy: {:.2f} %".format(best_accuracy*100))
print("Best Parameters:", best_parameters)
```

```
Best Accuracy: 79.60 %
Best Parameters: {'var_smoothing': 1e-07}
```

In [1191]:

```
Random_Forest=RandomForestClassifier()
```

In [1192]:

```
Random_Forest.fit(X_train, y_train)
```

Out[1192]:

```
RandomForestClassifier
RandomForestClassifier()
```

In [1194]:

```
y_pred1 =Random_Forest.predict(X_test)
```

In [1195]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred1)
print(cm)
```

```
[[15366   508]
 [   14 72652]]
```

In [1197]:

```
ac = accuracy_score(y_test, y_pred1)
ac
```

Out[1197]:

```
0.994104359611475
```

In [1199]:

```
bias =Random_Forest.score(X_train, y_train)
bias
```

Out[1199]:

```
1.0
```

In [1202]:

```
variance = Random_Forest.score(X_test, y_test)
variance
```

Out[1202]:

```
0.994104359611475
```

In [1244]:

```
#ROC AND AUC
from sklearn.metrics import roc_curve, roc_auc_score
```

In [1245]:

```
# Compute the False Positive Rate (FPR), True Positive Rate (TPR), and thresholds
fpr1, tpr1, thresholds = roc_curve(y_test, y_pred1)
```

In [1246]:

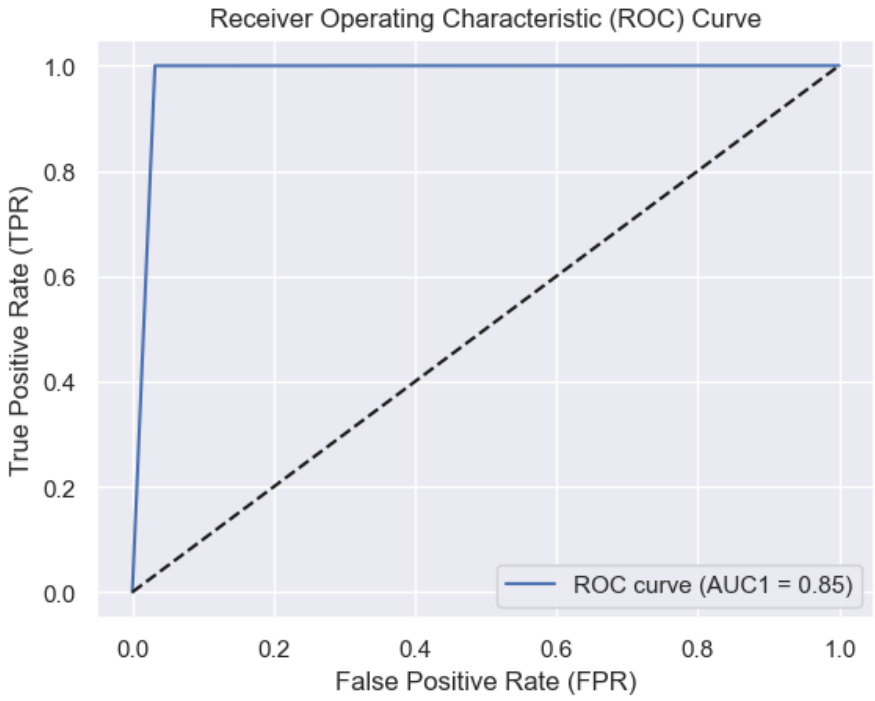
```
auc1 = roc_auc_score(y_test, y_pred1)
auc1
```

Out[1246]:

```
0.9839026609034907
```

In [1248]:

```
# Plotting the ROC curve
plt.plot(fpr1, tpr1, label='ROC curve (AUC1 = {:.2f})'.format(auc))
plt.plot([0, 1], [0, 1], 'k--')  # Random guess line
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



In [1207]:

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = Random_Forest, X = X_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 99.42 %
Standard Deviation: 0.06 %
```

In [1210]:

```
from sklearn.model_selection import GridSearchCV
```

In [1217]:

```
from sklearn.model_selection import GridSearchCV
#Define the parameter grid for grid search
param_grid1 = {
    'n_estimators': [100, 200, 300],  # Number of trees in the forest
    'max_depth': [None, 5, 10],       # Maximum depth of each tree
    'min_samples_split': [2, 5, 10],  # Minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4]     # Minimum number of samples required to be at a leaf node
}


```

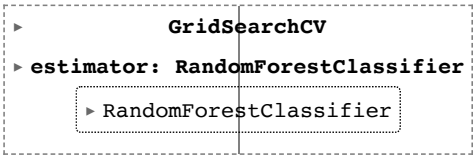
In [1218]:

```
# Create the GridSearchCV object
grid_search = GridSearchCV(estimator=Random_Forest, param_grid=param_grid1, cv=5)


```

In [1219]:

```
# Fit the grid search to the data
grid_search.fit(X = X_train, y = y_train)
```

Out[1219]:



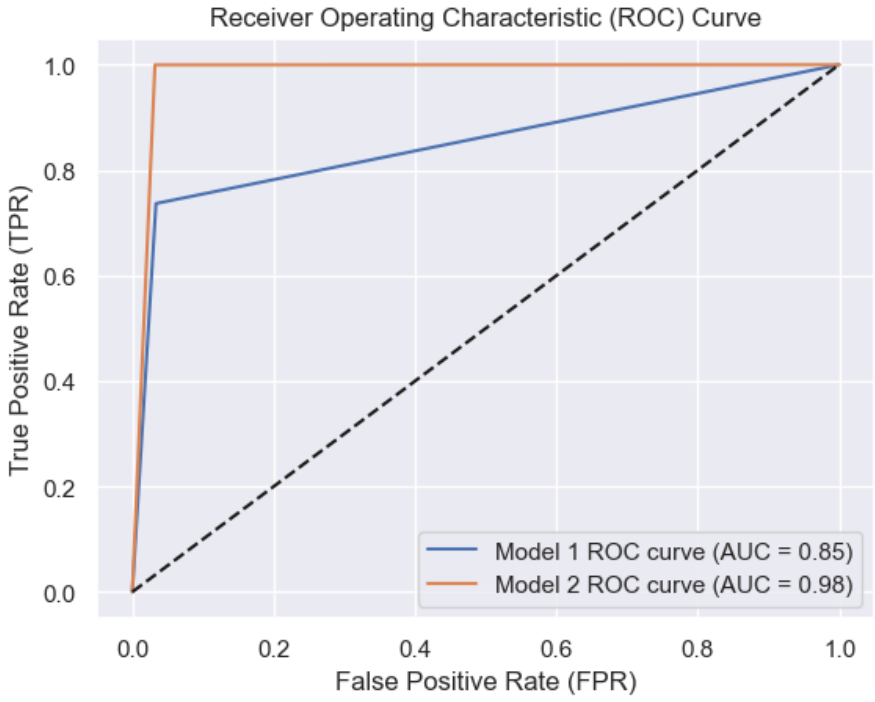
In [1220]:

```
# Get the best parameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

# Print the best parameters and best score
print("Best Parameters:", best_params)
print("Best Score:", best_score)
```

```
Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimat
ors': 200}
Best Score: 0.9940098748533173
```

In [1249]:

```
plt.plot(fpr, tpr, label='Model 1 ROC curve (AUC = {:.2f})'.format(auc))
plt.plot(fpr1, tpr1, label='Model 2 ROC curve (AUC = {:.2f})'.format(auc1))
plt.plot([0, 1], [0, 1], 'k--')  # Random guess line
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```




```
Insight:The AUC score represents the area under the ROC curve, which plots the true positive rate
(sensitivity) against the false positive rate (1-specificity). A higher AUC score suggests that Model
2 has a better ability to correctly classify positive cases and distinguish them from negative cases
compared to Model 1.
```

```
Conclusion:
     term ,last_pymnt_amnt ,total_rec_int ,total_rec_prncp ,total_pymnt, total_acc
,dti,annual_inc,installment              int_rate
```

funded_amnt,loan_amnt ,revol_util,grade ,home_ownership ,purpose ,zip_code ,addr_state ,initial_list_status , thse variable consider for model build for degfault asses because

The loan term, which refers to the length of time the borrower has to repay the loan, can influence default risk. Longer-term loans might have higher default rates,

These variables represent the total interest and principal amounts already received by the lender. If these values are low relative to the total loan amount, it could indicate that the borrower has not made significant progress in repaying the loan, potentially indicating higher default risk.

The amount of the last payment made by the borrower could be an indicator of their financial stability. Lower last payment amounts might suggest difficulties in meeting the obligations and, thus, a higher risk of default.

This is the total payment received by the lender, which includes both principal and interest. If the total payment is substantially lower than the expected amount at a particular point in the loan term

This ratio measures the borrower's debt relative to their income. A high DTI could indicate that the borrower is taking on too much debt compared to their earnings, making them more prone to default.

The borrower's annual income is a crucial factor in assessing their ability to repay the loan. Higher incomes generally indicate better financial stability and a lower risk of default.

The monthly installment amount reflects the borrower's monthly payment obligation. If this amount is too high relative to the borrower's income, it may increase the likelihood of default.
The interest rate on the loan directly impacts the borrower's repayment burden. Higher interest rates might lead to higher default rates,

The initial loan amount can influence default risk. Larger loans might lead to higher default rates if borrowers struggle to manage larger debt burdens.

he borrower's revolving credit utilization rate measures the amount of credit they are using compared to their total available credit. High utilization could suggest financial strain and may be associated with higher default risk.

while lower-grade loans might have a higher likelihood of default.

homeownership might imply more financial stability than renting.

The purpose of the loan can also influence default risk.

zip_code and addr_state: The borrower's location can play a role in assessing default risk as economic conditions and job markets can vary significantly between regions.

initial_list_status: The loan listing status can indicate whether the borrower applied for the loan as a new listing or a previously listed loan. It might provide insights into the demand for the loan and potential risk

Belows variable taken into acount because we saw there is relationship betwen those variable and loan status
last_credit_pull_year
last_pymnt_d_year
earliest_crln_year