In [512…
```
!wget --no-check-certificate 'https://drive.google.com/uc?export=download&id=1Yya6lSsiImeKuiCNMUyZ5P80x46SNRKh' -O Loa
```

```
--2024-09-25 18:46:41--  https://drive.google.com/uc?export=download&id=1Yya6lSsiImeKuiCNMUyZ5P80x46SNRKh
Resolving drive.google.com (drive.google.com)... 74.125.203.100, 74.125.203.101, 74.125.203.102, ...
Connecting to drive.google.com (drive.google.com)|74.125.203.100|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://drive.usercontent.google.com/download?id=1Yya6lSsiImeKuiCNMUyZ5P80x46SNRKh&export=download [followi
ng]
--2024-09-25 18:46:41--  https://drive.usercontent.google.com/download?id=1Yya6lSsiImeKuiCNMUyZ5P80x46SNRKh&export=do
wnload
Resolving drive.usercontent.google.com (drive.usercontent.google.com)... 173.194.174.132, 2404:6800:4008:c1b::84
Connecting to drive.usercontent.google.com (drive.usercontent.google.com)|173.194.174.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 100353395 (96M) [application/octet-stream]
Saving to: ‘LoanTap’

LoanTap             100%[===================>]  95.70M   181MB/s    in 0.5s

2024-09-25 18:46:47 (181 MB/s) - ‘LoanTap’ saved [100353395/100353395]
```

Importing the necessary libraries

In [513…
```
#Data processing
import pandas as pd
import numpy as np

#Data Visualisation
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
%matplotlib inline

#Stats & model building
from scipy import stats
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

```
from sklearn.metrics import (accuracy_score, confusion_matrix,
                             roc_curve, auc, ConfusionMatrixDisplay,
                             f1_score, recall_score,
                             precision_score, precision_recall_curve,
                             average_precision_score, classification_report)
from imblearn.over_sampling import SMOTE

#Hide warnings
import warnings
warnings.filterwarnings("ignore")
```

Loading the Dataset

In [514…
```
df = pd.read_csv("LoanTap")
```

In [515…
```
df.head()
```

Out[515]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verification_status	iss
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	117000.0	Not Verified	
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	Not Verified	
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RENT	43057.0	Source Verified	
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	RENT	54000.0	Not Verified	
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000.0	Verified	

About the case-study

- LoanTap is an online platform committed to delivering customized loan products to millennials. They innovate in an otherwise dull loan segment, to deliver instant, flexible loans on consumer friendly terms to salaried professionals and businessmen.
- The data science team at LoanTap is building an underwriting layer to determine the creditworthiness of MSMEs as well as individuals.

LoanTap deploys formal credit to salaried individuals and businesses 4 main financial instruments:

- Personal Loan
- EMI Free Loan
- Personal Overdraft
- Advance Salary Loan

Problem Statement:

Given a set of attributes for an Individual, determine if a credit line should be extended to them. If so, what should the repayment terms be in business recommendations?

In [516…
```
df.columns
```

Out[516]:
```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grade',
       'emp_title', 'emp_length', 'home_ownership', 'annual_inc',
       'verification_status', 'issue_d', 'loan_status', 'purpose', 'title',
       'dti', 'earliest_cr_line', 'open_acc', 'pub_rec', 'revol_bal',
       'revol_util', 'total_acc', 'initial_list_status', 'application_type',
       'mort_acc', 'pub_rec_bankruptcies', 'address'],
      dtype='object')
```

Column Description:

- loan_amnt : The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.
- term : The number of payments on the loan. Values are in months and can be either 36 or 60.
- int_rate : Interest Rate on the loan installment : The monthly payment owed by the borrower if the loan originates.

- grade : LoanTap assigned loan grade
- sub_grade : LoanTap assigned loan subgrade
- emp_title :The job title supplied by the Borrower when applying for the loan.*
- emp_length : Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.
- home_ownership : The home ownership status provided by the borrower during registration or obtained from the credit report.
- annual_inc : The self-reported annual income provided by the borrower during registration.
- verification_status : Indicates if income was verified by LoanTap, not verified, or if the income source was verified
- issue_d : The month which the loan was funded
- loan_status : Current status of the loan - Target Variable
- purpose : A category provided by the borrower for the loan request.
- title : The loan title provided by the borrower
- dti : A ratio calculated using the borrower’s total monthly debt payments on the total debt obligations, excluding mortgage and the requested LoanTap loan, divided by the borrower’s self-reported monthly income.
- earliest_cr_line :The month the borrower's earliest reported credit line was opened
- open_acc : The number of open credit lines in the borrower's credit file.
- pub_rec : Number of derogatory public records
- revol_bal : Total credit revolving balance
- revol_util : Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.
- total_acc : The total number of credit lines currently in the borrower's credit file
- initial_list_status : The initial listing status of the loan. Possible values are – W, F
- application_type : Indicates whether the loan is an individual application or a joint application with two co-borrowers
- mort_acc : Number of mortgage accounts.
- pub_rec_bankruptcies : Number of public record bankruptcies
- Address: Address of the individual

Exploratory Data analysis

In [517…
```
df.shape
```

Out[517]:
```
(396030, 27)
```

In [518…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  object 
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  object 
 5   sub_grade             396030 non-null  object 
 6   emp_title             373103 non-null  object 
 7   emp_length            377729 non-null  object 
 8   home_ownership        396030 non-null  object 
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  object 
 11  issue_d               396030 non-null  object 
 12  loan_status           396030 non-null  object 
 13  purpose               396030 non-null  object 
 14  title                 394274 non-null  object 
 15  dti                   396030 non-null  float64
 16  earliest_cr_line      396030 non-null  object 
 17  open_acc              396030 non-null  float64
 18  pub_rec               396030 non-null  float64
 19  revol_bal             396030 non-null  float64
 20  revol_util            395754 non-null  float64
 21  total_acc             396030 non-null  float64
 22  initial_list_status   396030 non-null  object 
 23  application_type      396030 non-null  object 
 24  mort_acc              358235 non-null  float64
 25  pub_rec_bankruptcies  395495 non-null  float64
 26  address               396030 non-null  object 
dtypes: float64(12), object(15)
memory usage: 81.6+ MB
```

In [519…
```
# since these two columns contain dates in them ,convert the columns datatype from object to datetime
df[['issue_d','earliest_cr_line']].head()
df['issue_d'] = pd.to_datetime(df['issue_d'])
df['earliest_cr_line'] = pd.to_datetime(df['earliest_cr_line'])
```

In [520…
```
# converting the emp_length column from object to int
d = {'10+ years':10, '4 years':4, '< 1 year':0,
     '6 years':6, '9 years':9,'2 years':2, '3 years':3,
     '8 years':8, '7 years':7, '5 years':5, '1 year':1}
df['emp_length'] = df['emp_length'].replace(d)
```

In [453…
```
# observing the address column
print(df['address'].sample(10))
# Deriving zip code and state from address
df[['state', 'zip_code']] = df['address'].apply(lambda x: pd.Series([x[-8:-6], x[-5:]]))
```

```
215284    7984 Boyle Junction Apt. 248\r\nSouth Alexandr...
288027           034 Robert Spring\r\nPort Bonnie, IA 05113
357536           07572 Tony Viaduct\r\nAshleyfort, TN 00813
72434         08098 Sullivan Drives\r\nDanielmouth, OR 00813
256388           77367 Butler Knoll\r\nMeganburgh, MA 48052
275043    767 Deleon Vista Suite 139\r\nNew Samuelhaven,...
295475       66990 Merritt Mission\r\nSouth David, NM 22690
183435             3850 Rivera Spring\r\nNew Ryan, MN 05113
45458     1168 Browning Via Suite 675\r\nNew Patrickbury...
128577       5133 Norman Station\r\nLozanochester, AR 11650
Name: address, dtype: object
```

In [454…
```
df.duplicated().any()
```

Out[454]:
```
False
```

In [455…
```
df.dtypes
```

Out[455]:

	0
loan_amnt	float64
term	object
int_rate	float64
installment	float64
grade	object
sub_grade	object
emp_title	object
emp_length	float64
home_ownership	object
annual_inc	float64
verification_status	object
issue_d	datetime64[ns]
loan_status	object
purpose	object
title	object
dti	float64
earliest_cr_line	datetime64[ns]
open_acc	float64
pub_rec	float64
revol_bal	float64
revol_util	float64
total_acc	float64
initial_list_status	object
application_type	object
mort_acc	float64

	0
pub_rec_bankruptcies	float64
address	object
state	object
zip_code	object

dtype: object

In [456…
```
cat_cols = df.columns[df.dtypes=='O']
cat_cols
```

Out[456]:
```
Index(['term', 'grade', 'sub_grade', 'emp_title', 'home_ownership',
       'verification_status', 'loan_status', 'purpose', 'title',
       'initial_list_status', 'application_type', 'address', 'state',
       'zip_code'],
      dtype='object')
```

In [457…
```
print("No of Unique values")
for i in cat_cols:
  print(i,"-->",df[i].nunique())
  print("-"*100)
```

```
No of Unique values
term --> 2
--------------------------------------------------------------------------------------------------------------
grade --> 7
--------------------------------------------------------------------------------------------------------------
sub_grade --> 35
--------------------------------------------------------------------------------------------------------------
emp_title --> 173105
--------------------------------------------------------------------------------------------------------------
home_ownership --> 6
--------------------------------------------------------------------------------------------------------------
verification_status --> 3
--------------------------------------------------------------------------------------------------------------
loan_status --> 2
--------------------------------------------------------------------------------------------------------------
purpose --> 14
--------------------------------------------------------------------------------------------------------------
title --> 48816
--------------------------------------------------------------------------------------------------------------
initial_list_status --> 2
--------------------------------------------------------------------------------------------------------------
application_type --> 3
--------------------------------------------------------------------------------------------------------------
address --> 393700
--------------------------------------------------------------------------------------------------------------
state --> 54
--------------------------------------------------------------------------------------------------------------
zip_code --> 10
--------------------------------------------------------------------------------------------------------------
```

Understanding if there is any relation between employee's job title to loan status

In [458…

```python
# Get the value counts as percentages, sort by descending order, and select the top 2
top_2_loan_affordability_by_job_titles = df['emp_title'].value_counts(normalize=True) * 100
# Display the result
print(top_2_loan_affordability_by_job_titles.head(2))
```

```
emp_title
Teacher    1.176351
Manager    1.139096
Name: proportion, dtype: float64
```

Observation:

- Top 2 affordable job titles are Teacher and Manager

In [459…
```python
# Get the value counts as percentages, sort by descending order, and select the top 2
top_2_reason_for_taking_loans = df['title'].value_counts(normalize=True) * 100
# Display the result
print(top_2_reason_for_taking_loans.head(2))
```

```
title
Debt consolidation         38.671584
Credit card refinancing    13.058685
Name: proportion, dtype: float64
```

Observation:

- Top 2 reasons for taking up loan are debt consolidation and credit card refinancing

In [460…
```python
# finding the frequencies of categorical columns with less number of sub-classes within
# and converting those columns from object to categorical columns
for i in cat_cols:
  if df[i].nunique()<=60:
    df[i] = df[i].astype('category')
    print(df[i].value_counts())
    print(100*"-")
```

```
term
 36 months    302005
 60 months     94025
Name: count, dtype: int64
```

```
grade
B    116018
C    105987
A     64187
D     63524
E     31488
F     11772
G      3054
Name: count, dtype: int64
```

```
sub_grade
B3    26655
B4    25601
C1    23662
C2    22580
B2    22495
B5    22085
C3    21221
C4    20280
B1    19182
A5    18526
C5    18244
D1    15993
A4    15789
D2    13951
D3    12223
D4    11657
A3    10576
A1     9729
D5     9700
A2     9567
E1     7917
E2     7431
E3     6207
E4     5361
E5     4572
F1     3536
F2     2766
F3     2286
```

```
F4     1787
F5     1397
G1     1058
G2      754
G3      552
G4      374
G5      316
Name: count, dtype: int64
--------------------------------------------------------------------------------------------------------
home_ownership
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
ANY              3
Name: count, dtype: int64
--------------------------------------------------------------------------------------------------------
verification_status
Verified           139563
Source Verified    131385
Not Verified       125082
Name: count, dtype: int64
--------------------------------------------------------------------------------------------------------
loan_status
Fully Paid     318357
Charged Off     77673
Name: count, dtype: int64
--------------------------------------------------------------------------------------------------------
purpose
debt_consolidation    234507
credit_card            83019
home_improvement       24030
other                  21185
major_purchase          8790
small_business          5701
car                     4697
medical                 4196
moving                  2854
vacation                2452
house                   2201
wedding                 1812
renewable_energy         329
educational              257
```

```
Name: count, dtype: int64
-----------------------------------------------------------------------------------------------------
initial_list_status
f    238066
w    157964
Name: count, dtype: int64
-----------------------------------------------------------------------------------------------------
application_type
INDIVIDUAL    395319
JOINT            425
DIRECT_PAY       286
Name: count, dtype: int64
-----------------------------------------------------------------------------------------------------
state
AP    14308
AE    14157
AA    13919
NJ     7091
WI     7081
LA     7068
NV     7038
AK     7034
VA     7022
MA     7022
VT     7005
NY     7004
MS     7003
TX     7000
SC     6973
ME     6972
OH     6969
AR     6969
GA     6967
IN     6958
ID     6958
KS     6945
WV     6944
RI     6940
MO     6939
IL     6934
WY     6933
HI     6927
NE     6927
IA     6926
```

```
FL     6921
AZ     6918
CO     6914
OK     6911
MN     6904
CT     6904
NC     6901
CA     6898
OR     6898
AL     6898
MD     6896
WA     6895
SD     6887
UT     6887
MT     6883
DE     6874
TN     6869
ND     6858
MI     6854
NM     6842
DC     6842
PA     6825
NH     6818
KY     6800
Name: count, dtype: int64
--------------------------------------------------------------------------------------------------------
zip_code
70466    56985
30723    56546
22690    56527
48052    55917
00813    45824
29597    45471
05113    45402
11650    11226
93700    11151
86630    10981
Name: count, dtype: int64
--------------------------------------------------------------------------------------------------------
```

Checking for missing values

In [461…

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

Out[461]:

	0
loan_amnt	0
term	0
int_rate	0
installment	0
grade	0
sub_grade	0
emp_title	22927
emp_length	18301
home_ownership	0
annual_inc	0
verification_status	0
issue_d	0
loan_status	0
purpose	0
title	1756
dti	0
earliest_cr_line	0
open_acc	0
pub_rec	0
revol_bal	0
revol_util	276
total_acc	0
initial_list_status	0
application_type	0
mort_acc	37795

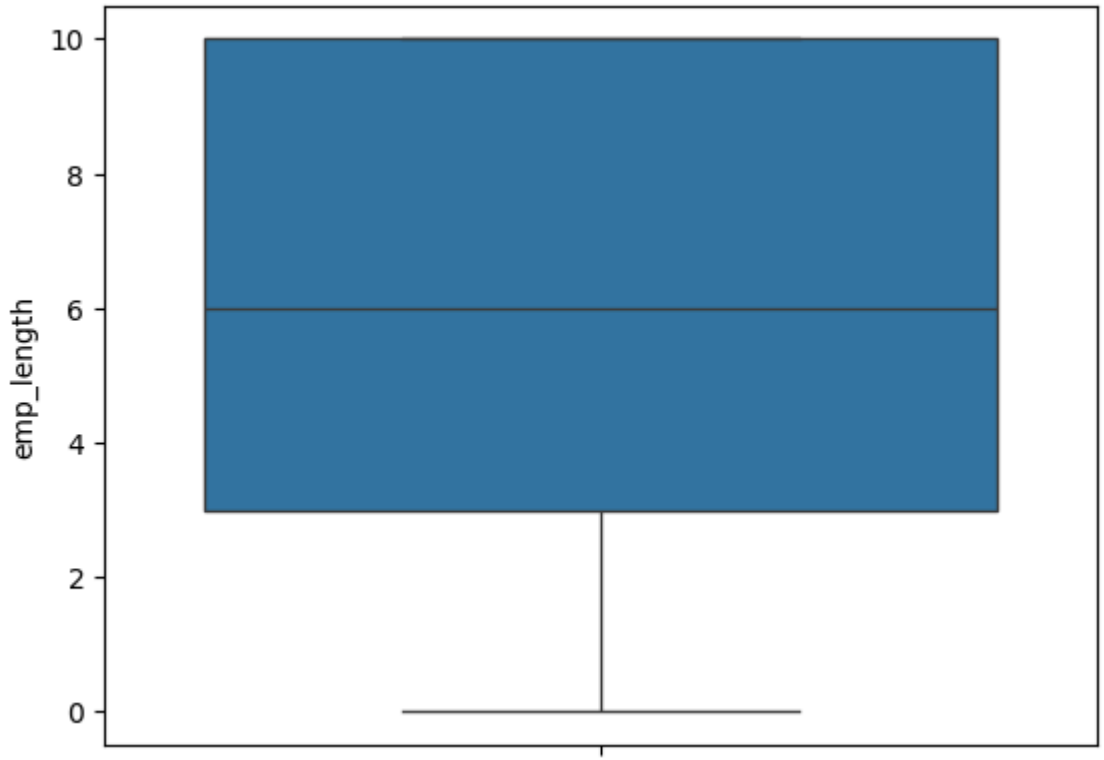
	0
pub_rec_bankruptcies	535
address	0
state	0
zip_code	0

dtype: int64

Handling missing values

In [462…
```
# filling up the null values in emp_title and title column with 'unknown'
df['emp_title']  = df['emp_title'].fillna('unknown')
df['title']  = df['title'].fillna('unknown')
```

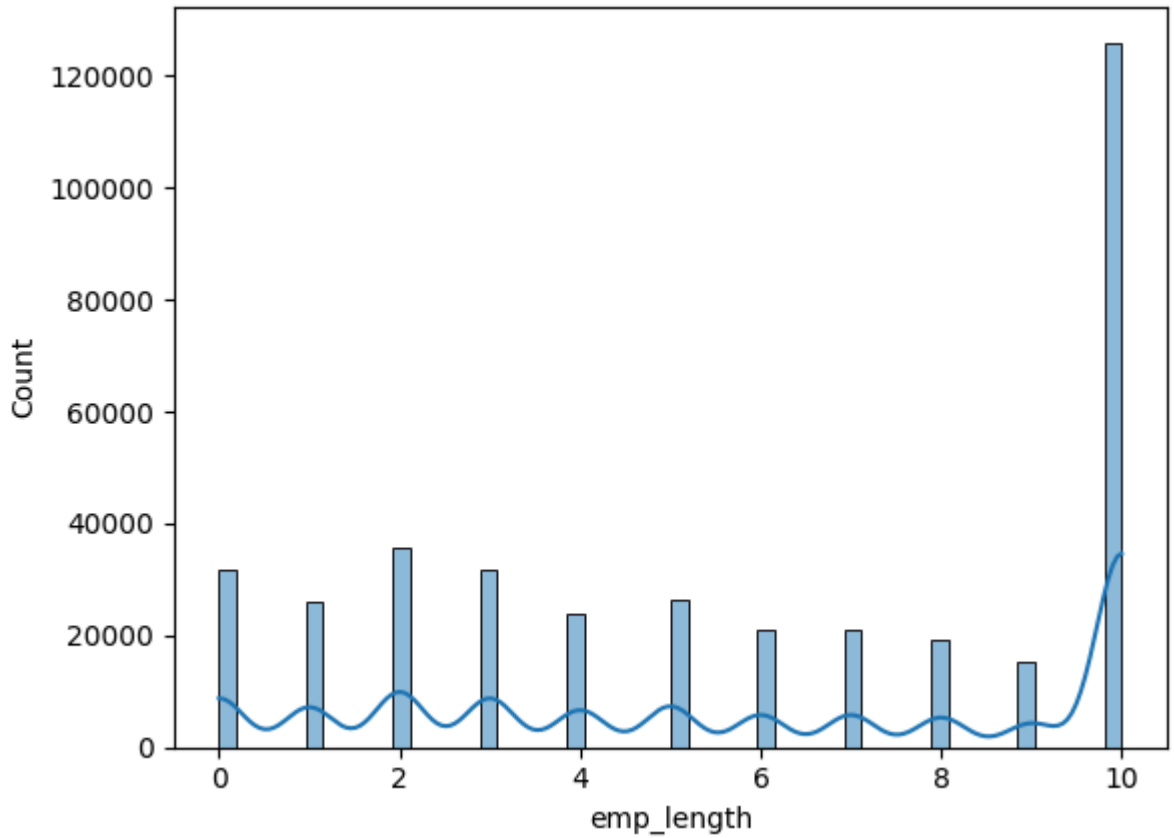
In [463…
```
# observing the median of emp_length feature with boxplot
sns.boxplot(data = df,y = 'emp_length')
plt.show()
# finding the mean
print(f"Mean of emp_length--> {df['emp_length'].mean()}")
print(f"Median of emp_length--> {df['emp_length'].median()}")
```




```
Mean of emp_length--> 5.938577657526957
Median of emp_length--> 6.0
```

In [464…

```
# observing the distribution of emp_length feature
sns.histplot(df['emp_length'],kde = True)
plt.show()
```



Observation:

- Since the distribution of emp_length is right skewed we can choose median to impute the missing values , as mean is sensitive to skeweness.

In [465…
```
# imputing the missing values in emp_length column by it's median
df['emp_length'] = df['emp_length'].fillna(df['emp_length'].median())
df['emp_length'].isnull().sum()
```

Out[465]:
```
0
```

In [466…
```
# Mean aggregation of mort_acc by total_acc to fill missing values
avg_mort = df.groupby('total_acc')['mort_acc'].mean()
def fill_mort(total_acc,mort_acc):
  if np.isnan(mort_acc):
    return avg_mort[total_acc].round()
  else:
    return mort_acc
```

In [467…
```
df['mort_acc'] = df.apply(lambda x: fill_mort(x['total_acc'],x['mort_acc']), axis=1)
```

In [468…
```
# removing the minority of missing values from revol_util and pub_rec_bankruptcies
df.dropna(inplace = True)
```

In [469…
```
df.isnull().sum()
```

Out[469]:

	0
loan_amnt	0
term	0
int_rate	0
installment	0
grade	0
sub_grade	0
emp_title	0
emp_length	0
home_ownership	0
annual_inc	0
verification_status	0
issue_d	0
loan_status	0
purpose	0
title	0
dti	0
earliest_cr_line	0
open_acc	0
pub_rec	0
revol_bal	0
revol_util	0
total_acc	0
initial_list_status	0
application_type	0
mort_acc	0

	0
pub_rec_bankruptcies	0
address	0
state	0
zip_code	0

dtype: int64

In [470…
```
df.shape
```

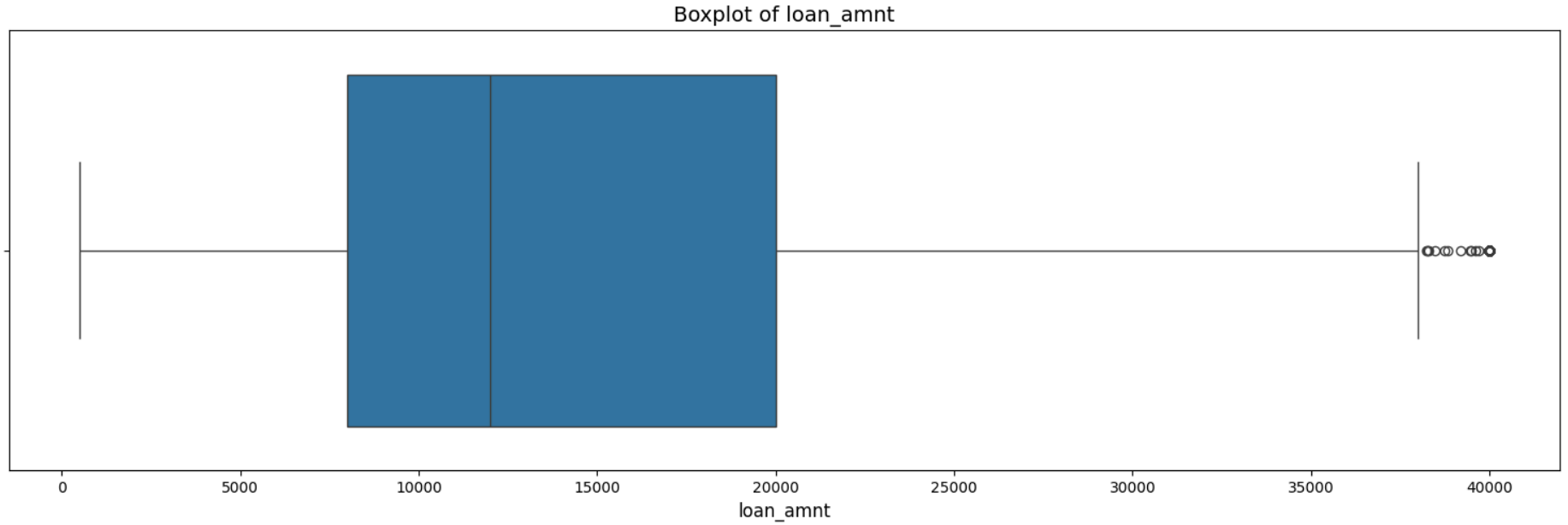
Out[470]:
```
(395219, 29)
```

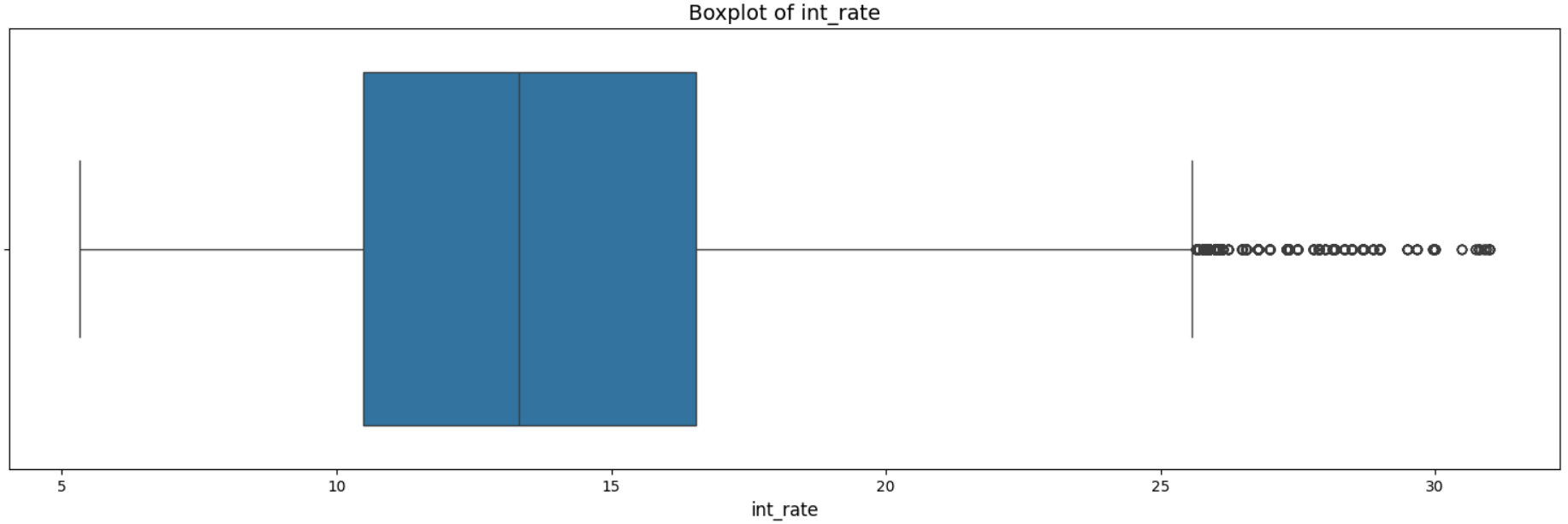
Univariate and Bivariate Analysis

In [471…
```
num_cols = df.columns[df.dtypes=='float64']
num_cols
```

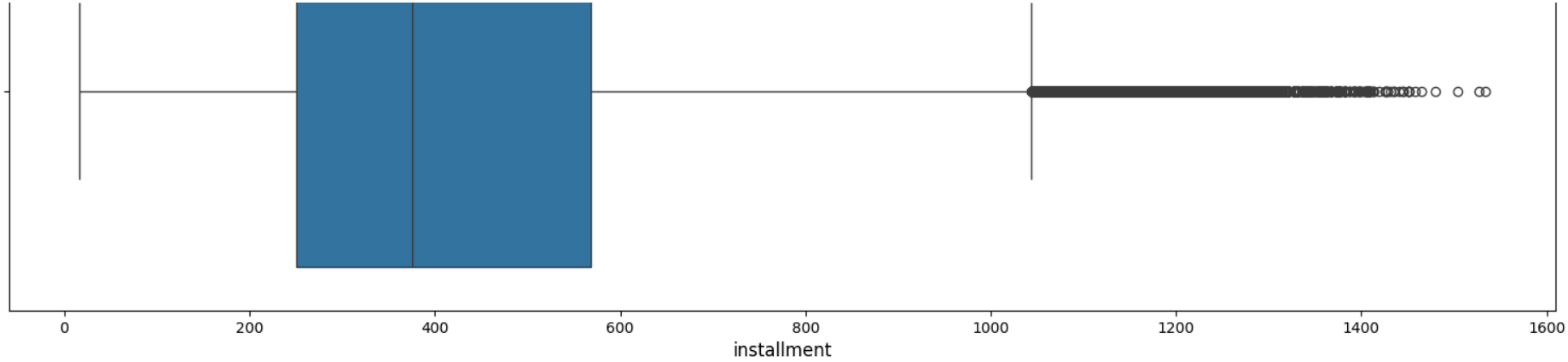
Out[471]:
```
Index(['loan_amnt', 'int_rate', 'installment', 'emp_length', 'annual_inc',
       'dti', 'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc',
       'mort_acc', 'pub_rec_bankruptcies'],
      dtype='object')
```

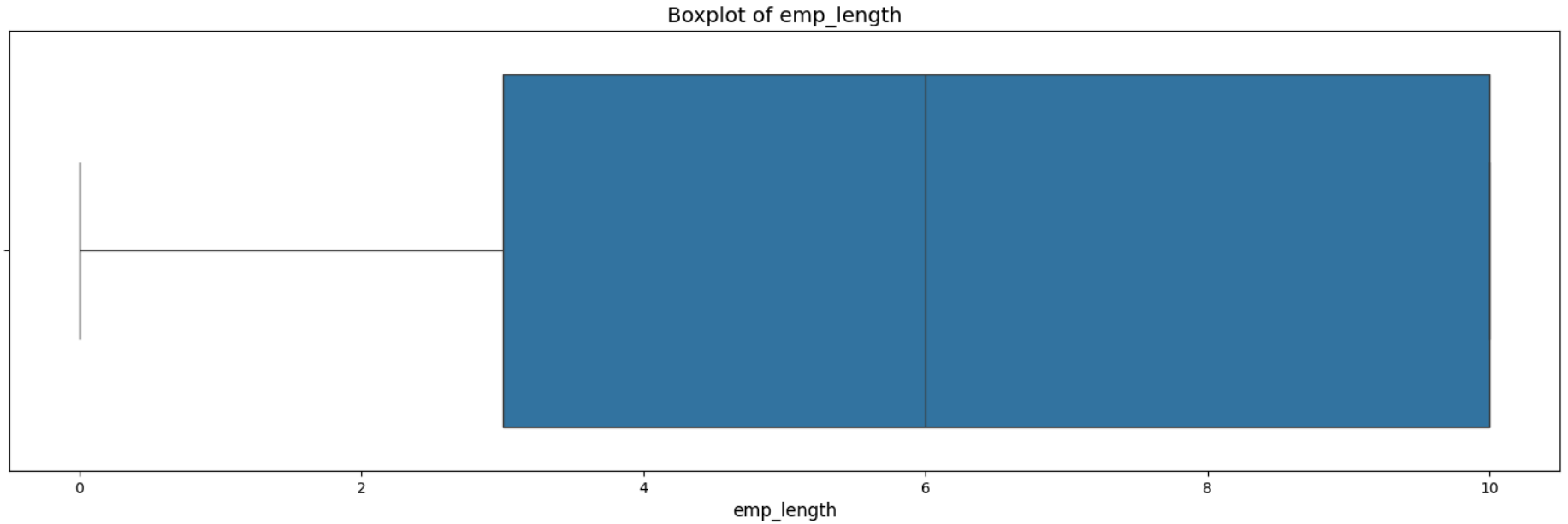
In [472…
```
plt.figure(figsize=(15, len(num_cols) * 5))
for i, col in enumerate(num_cols, 1):
    plt.subplot(len(num_cols), 1, i)  # Create a subplot for each column
    sns.boxplot(data=df, x=col)
    plt.title(f'Boxplot of {col}', fontsize=14)
    plt.xlabel(col, fontsize=12)

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```

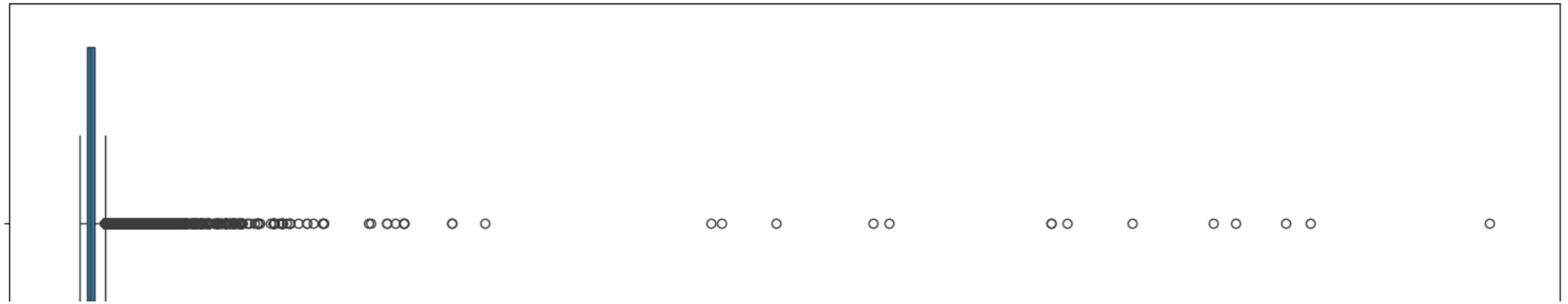

Boxplot of loan_amnt
0
5000
10000
15000
20000
25000
30000
35000
40000
loan_amnt

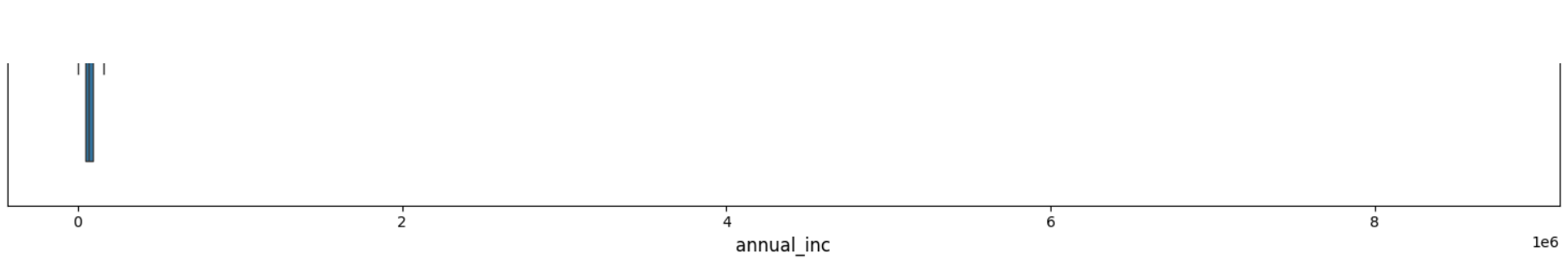

Boxplot of int_rate
5
10
15
20
25
30
int_rate

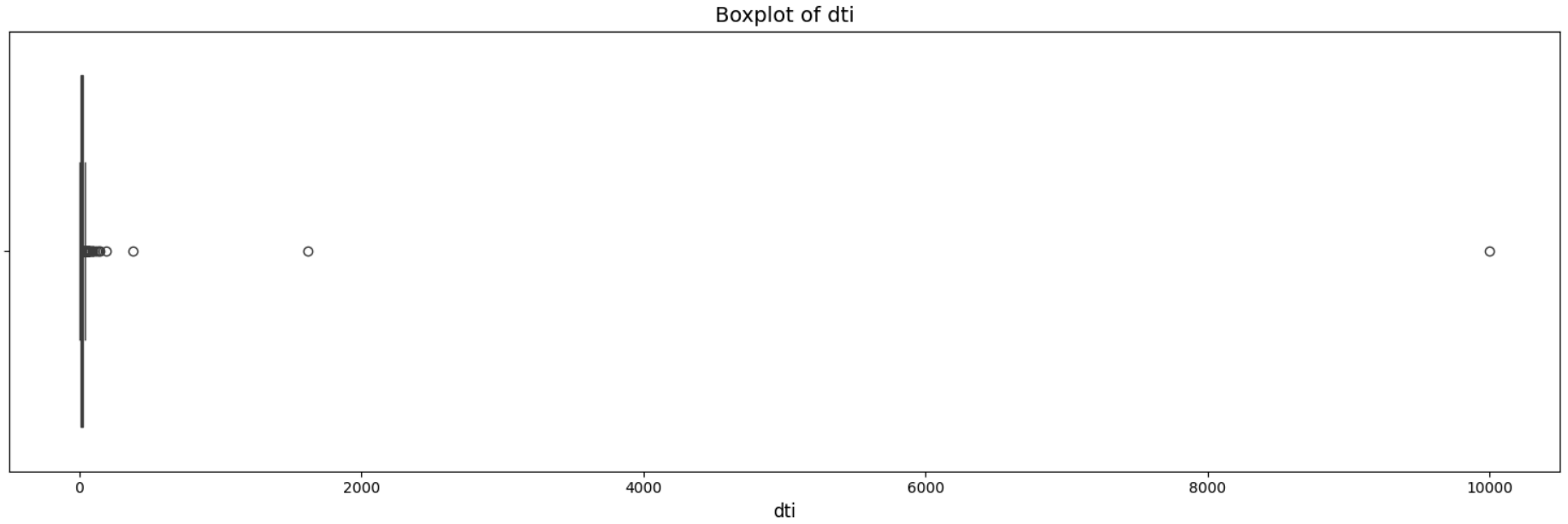

Boxplot of installment

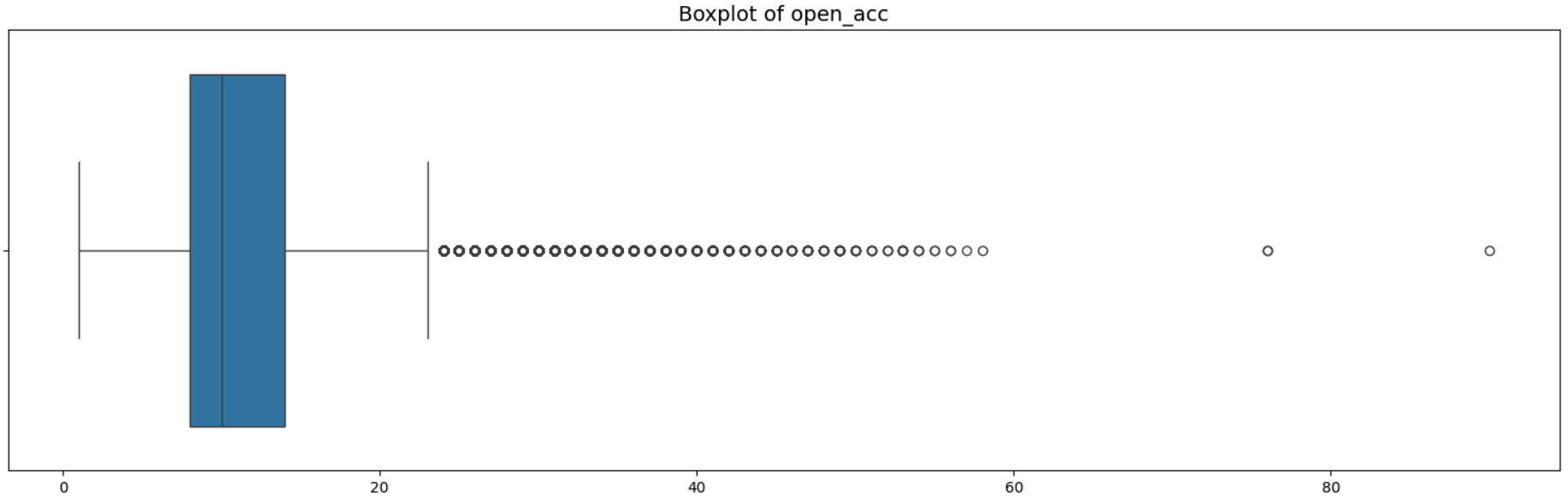

0
200
400
600
800
1000
1200
1400
1600
installment


Boxplot of emp_length
0
2
4
6
8
10
emp_length

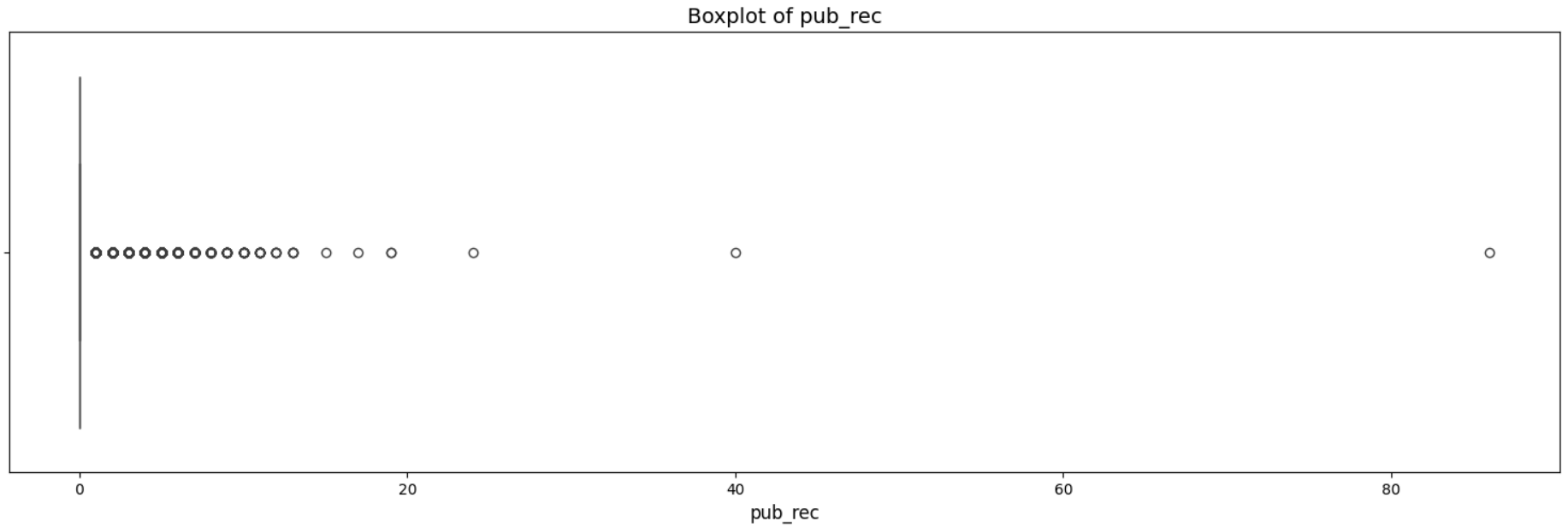

Boxplot of annual_inc

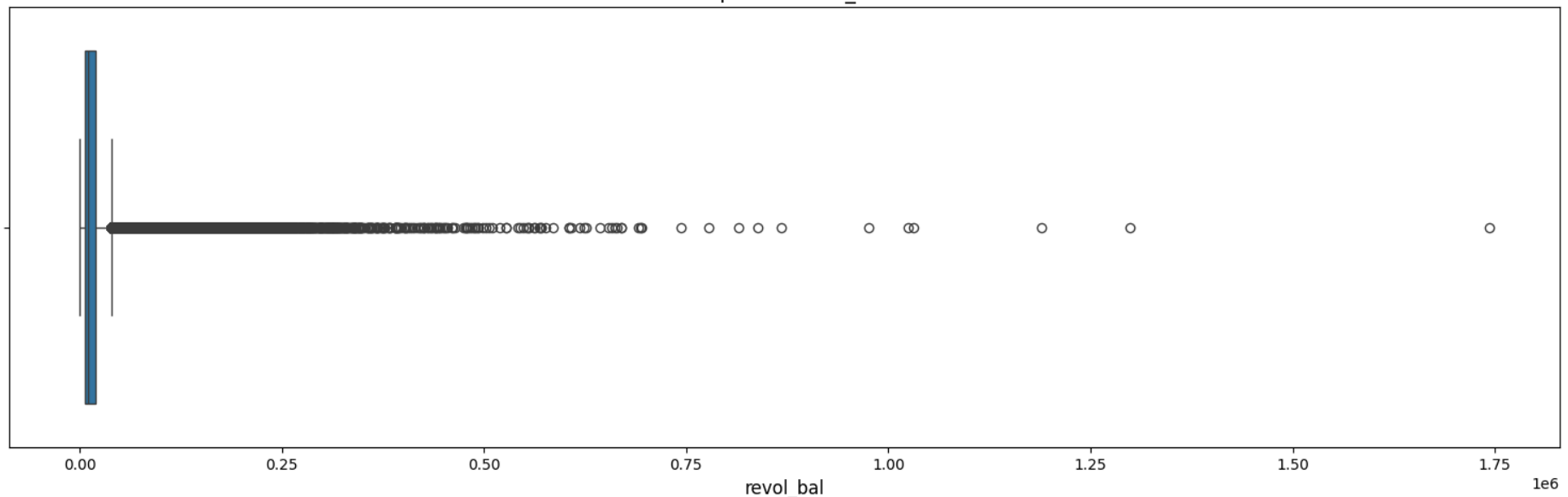

0
2
4
6
8
annual_inc
1e6


Boxplot of dti
0
2000
4000
6000
8000
10000
dti

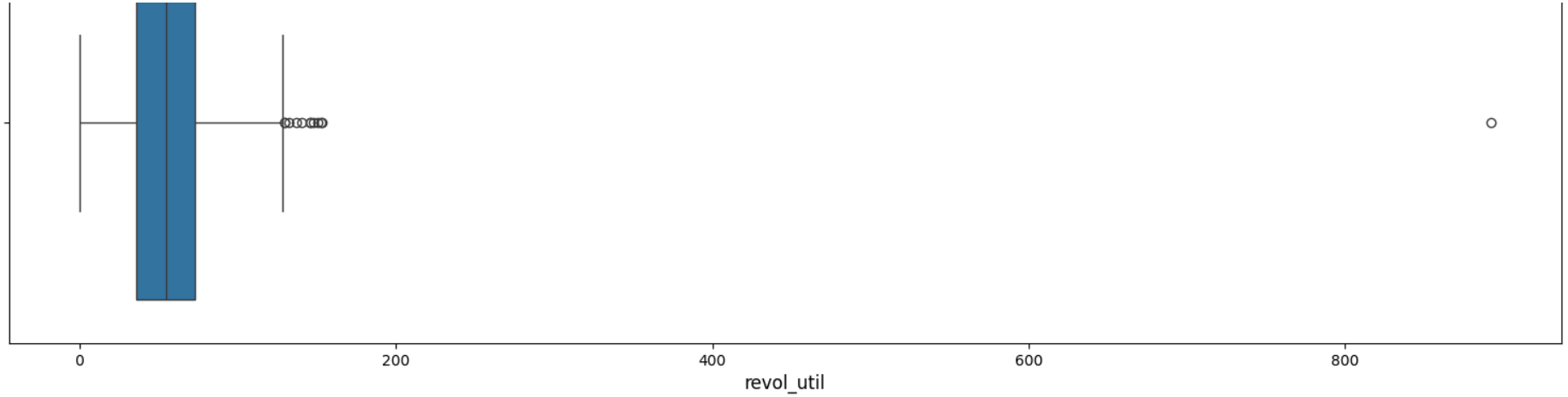

Boxplot of open_acc
0
20
40
60
80

open_acc

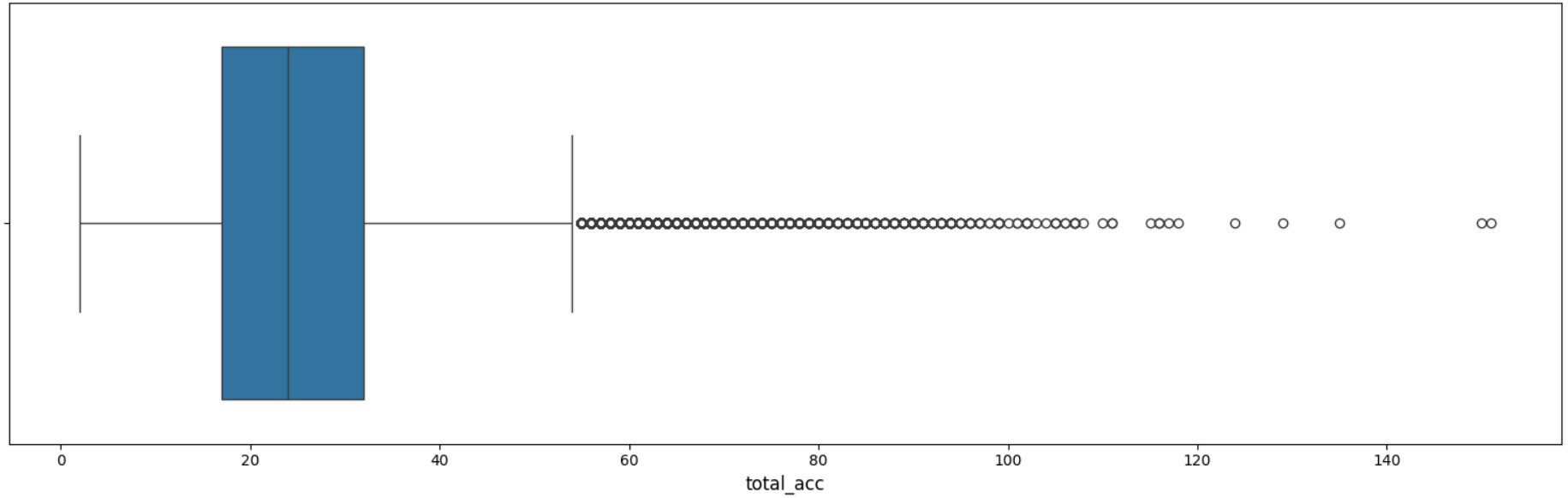

Boxplot of pub_rec
0
20
40
60
80
pub_rec


Boxplot of revol_bal
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1e6
revol_bal

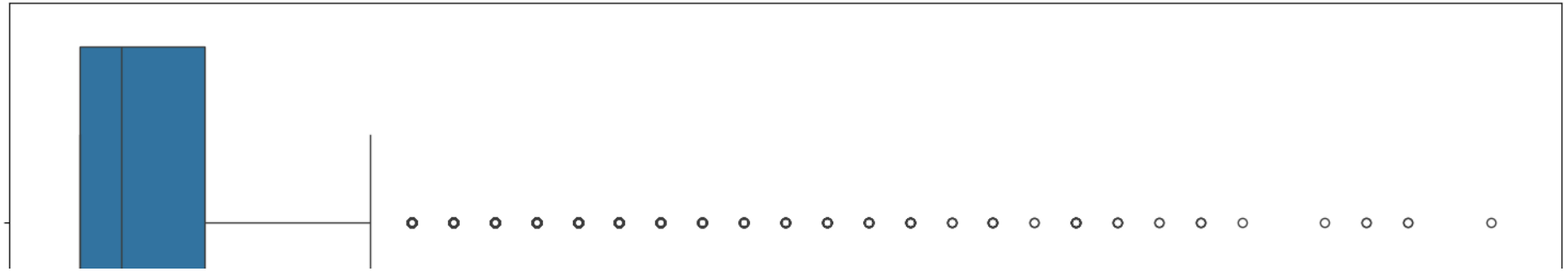

Boxplot of revol_util

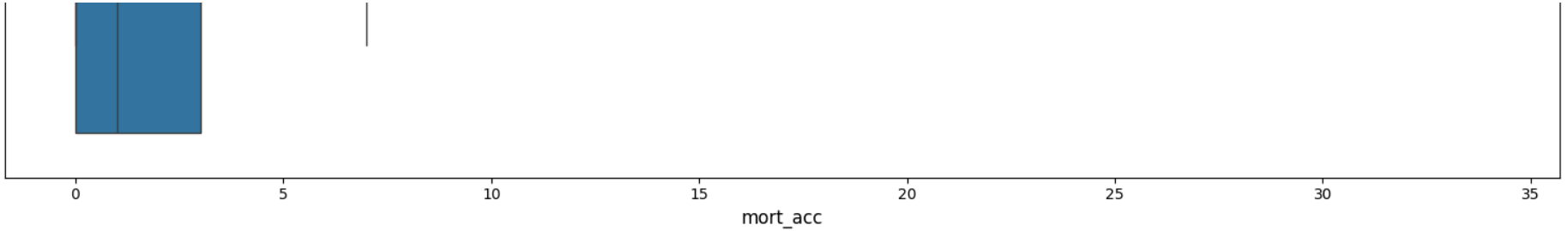


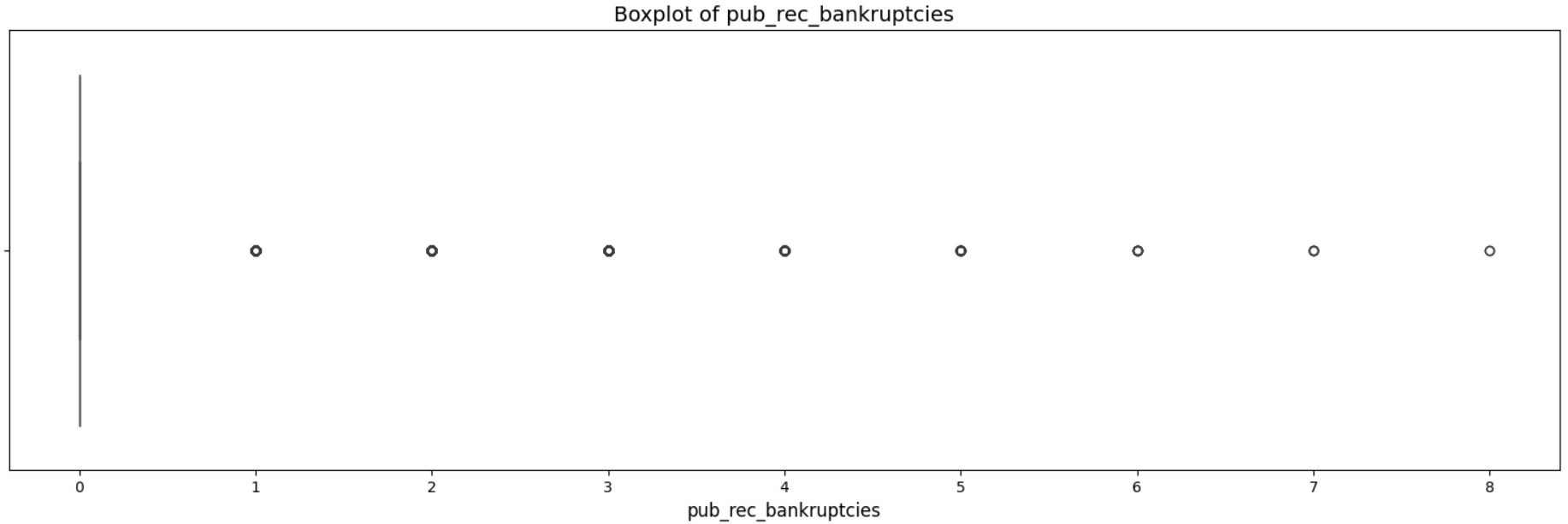
Boxplot of total_acc



Boxplot of mort_acc







In [473…

```
# observing the number of unique values in pub_rec and pub_rec_bankruptcies columns
print(df['pub_rec'].nunique())
print(df['pub_rec'].unique())
print(df['pub_rec_bankruptcies'].nunique())
print(df['pub_rec_bankruptcies'].unique())
```

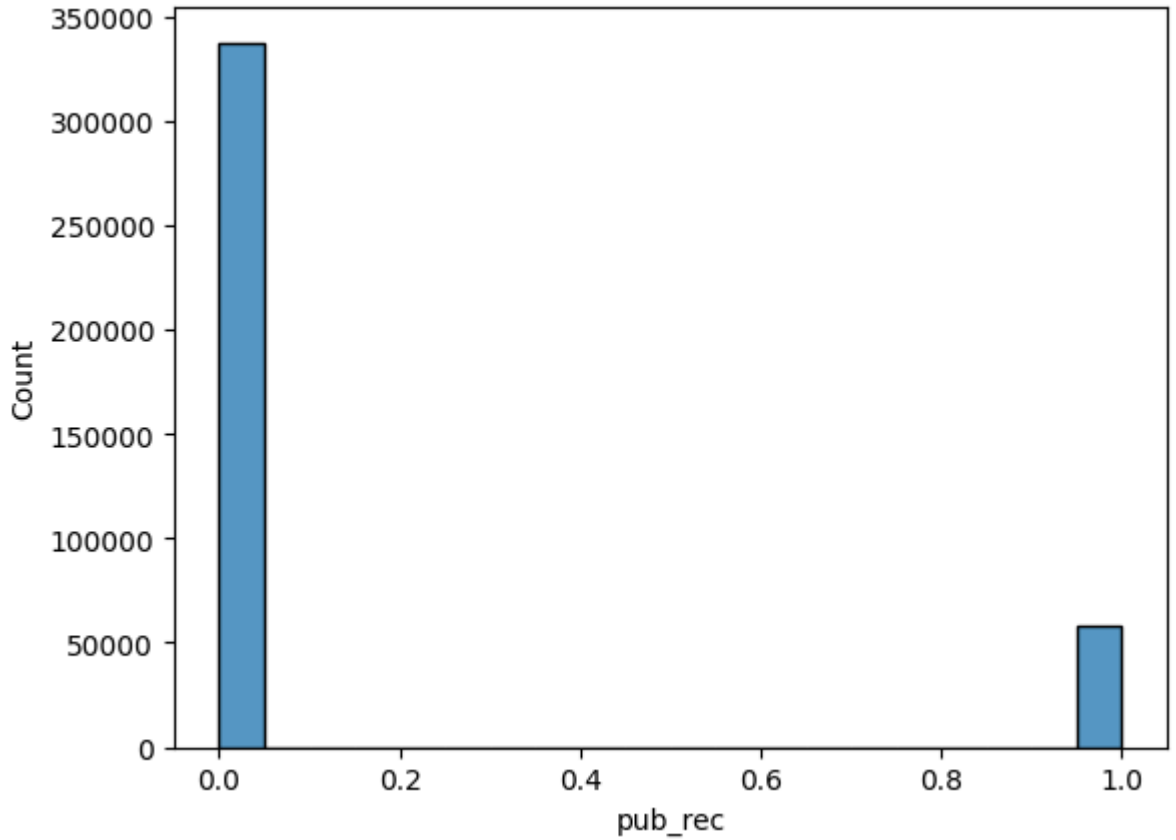
```
20
[ 0.  1.  2.  3.  4.  6.  5.  8.  9. 10. 11.  7. 19. 13. 40. 17. 86. 12.
 24. 15.]
9
[0. 1. 2. 3. 4. 5. 6. 7. 8.]
```

Observation:

- Since these 2 columns (pub_rec,pub_rec_bankruptcies) depict the record of having derogatory public records we can modify these columns into 0 or 1 indicating whether they had public records bankruptcies or not.

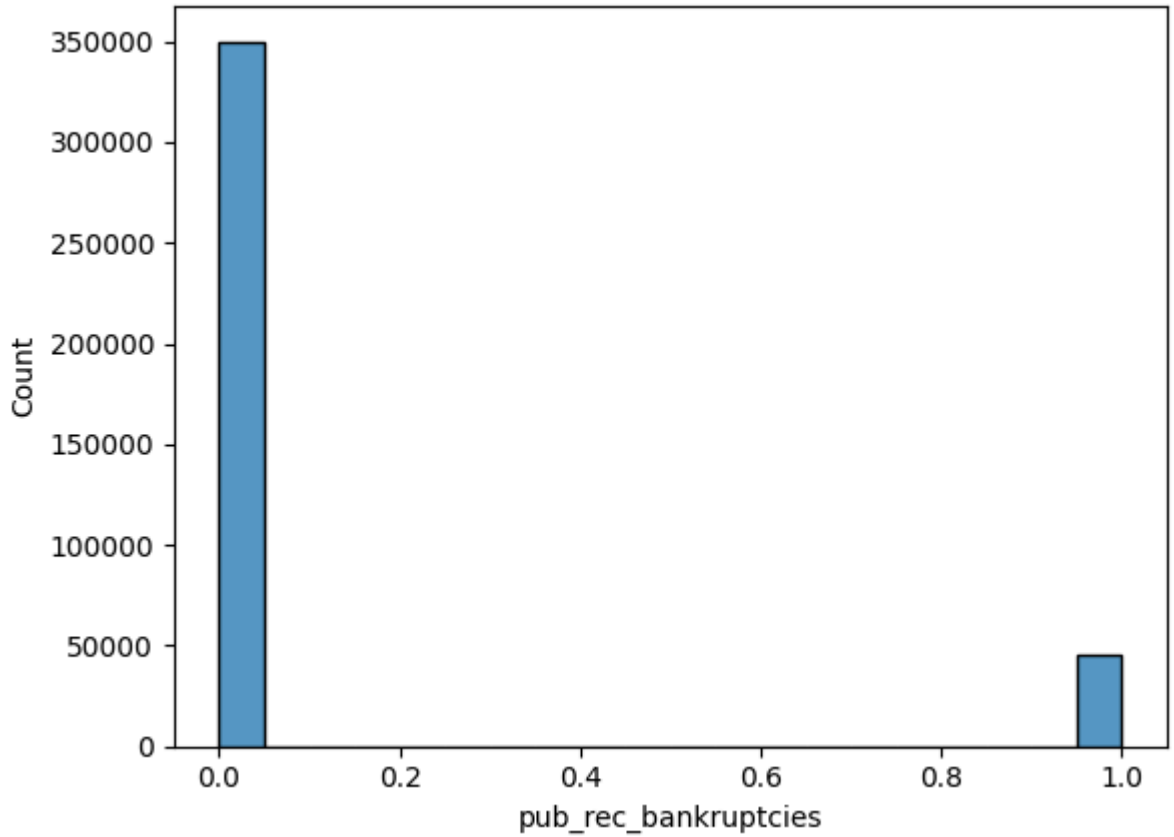
In [474…
```
# Create a new column with 1 if value > 0, otherwise 0
df['pub_rec'] = df['pub_rec'].apply(lambda x: 1 if x > 0 else 0)
print(df['pub_rec'].unique())
sns.histplot(df['pub_rec'])
plt.show()
```

```
[0 1]
```



In [475…
```
df['pub_rec_bankruptcies'] = df['pub_rec_bankruptcies'].apply(lambda x: 1 if x > 0 else 0)
print(df['pub_rec_bankruptcies'].unique())
sns.histplot(df['pub_rec_bankruptcies'])
plt.show()
```

```
[0 1]
```



In [476…

```
# checking the datatypes of these modified columns
df[['pub_rec','pub_rec_bankruptcies']].dtypes
```

Out[476]:

	0
pub_rec	int64
pub_rec_bankruptcies	int64

dtype: object

- Since we have changed these 2 columns into categorical information, we can exclude these columns from rest of the numerical columns

In [477…
```
# List of columns to exclude
exclude_cols = ['pub_rec', 'pub_rec_bankruptcies']
# Filter out the columns from num_cols
num_cols = [col for col in num_cols if col not in exclude_cols]
# Now select those columns from the dataframe
num_cols = df[num_cols]
num_cols
```

Out[477]:

	loan_amnt	int_rate	installment	emp_length	annual_inc	dti	open_acc	revol_bal	revol_util	total_acc	mort_acc
0	10000.0	11.44	329.48	10.0	117000.0	26.24	16.0	36369.0	41.8	25.0	0.0
1	8000.0	11.99	265.68	4.0	65000.0	22.05	17.0	20131.0	53.3	27.0	3.0
2	15600.0	10.49	506.97	0.0	43057.0	12.79	13.0	11987.0	92.2	26.0	0.0
3	7200.0	6.49	220.65	6.0	54000.0	2.60	6.0	5472.0	21.5	13.0	0.0
4	24375.0	17.27	609.33	9.0	55000.0	33.95	13.0	24584.0	69.8	43.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...
396025	10000.0	10.99	217.38	2.0	40000.0	15.63	6.0	1990.0	34.3	23.0	0.0
396026	21000.0	12.29	700.42	5.0	110000.0	21.45	6.0	43263.0	95.7	8.0	1.0
396027	5000.0	9.99	161.32	10.0	56500.0	17.56	15.0	32704.0	66.9	23.0	0.0
396028	21000.0	15.31	503.02	10.0	64000.0	15.88	9.0	15704.0	53.8	20.0	5.0
396029	2000.0	13.61	67.98	10.0	42996.0	8.32	3.0	4292.0	91.3	19.0	1.0

395219 rows × 11 columns

In [478…
```
num_cols.columns
```

Out[478]:
```
Index(['loan_amnt', 'int_rate', 'installment', 'emp_length', 'annual_inc',
       'dti', 'open_acc', 'revol_bal', 'revol_util', 'total_acc', 'mort_acc'],
      dtype='object')
```

Observation:

- There are lot of oultiers in almost all the other numerical columns like in loan_amnt, int_rate, installment, annual_inc, dti, open_acc, revol_bal, revol_util, total_acc, mort_acc so treating outliers using standard deviation method

Treating Outliers

In [479…
```
#Removing outliers using standard deviation
for col in num_cols:
  mean=df[col].mean()
  std=df[col].std()
  upper = mean + (3*std)
  df = df[~(df[col]>upper)]
```

Statistical Summary

In [480…
```
num_cols.describe()
```

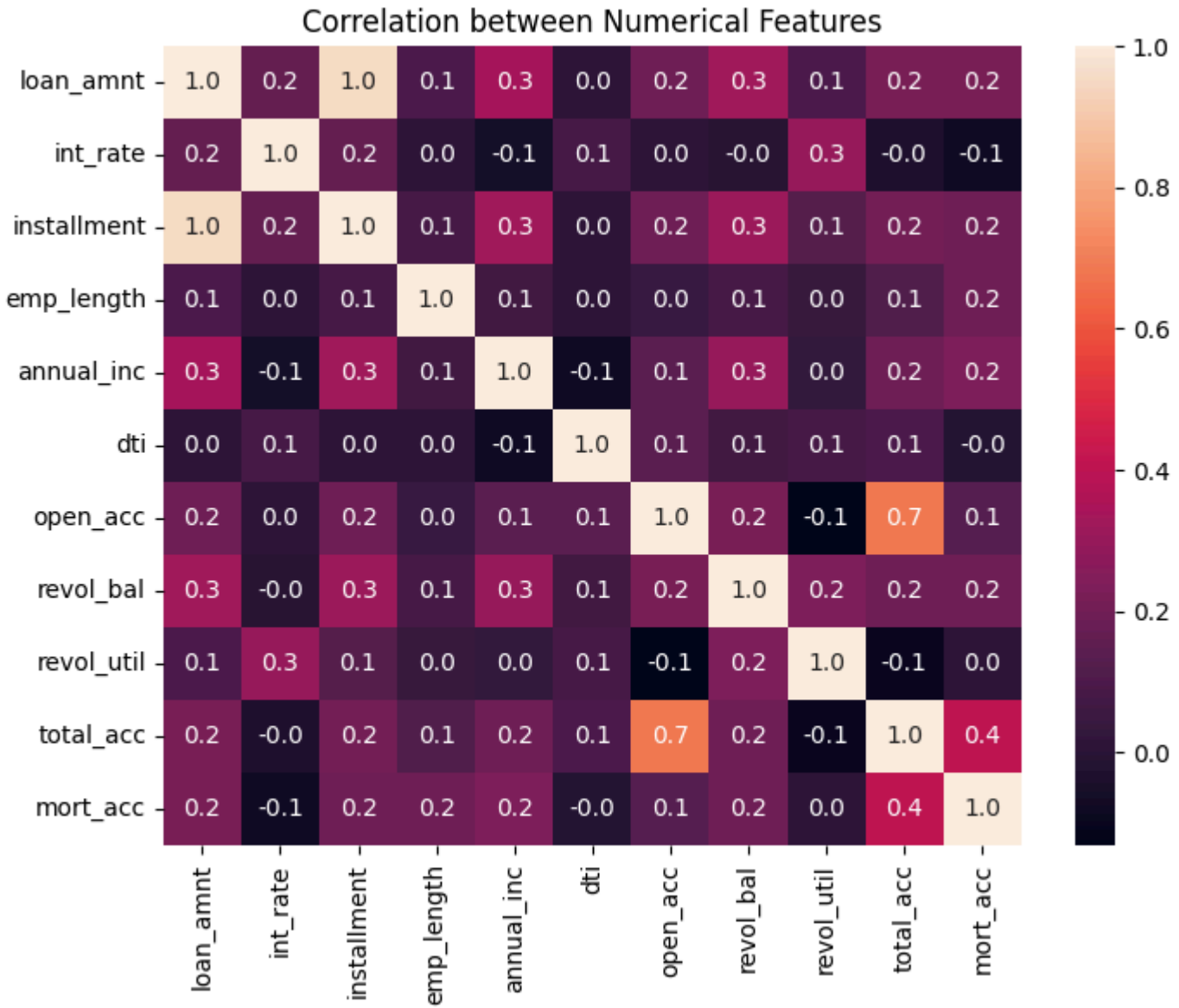
Out[480]:

	loan_amnt	int_rate	installment	emp_length	annual_inc	dti	open_acc	revol_bal	rev
count	395219.000000	395219.000000	395219.000000	395219.000000	3.952190e+05	395219.000000	395219.000000	3.952190e+05	395219.0
mean	14122.061250	13.642094	432.069020	5.944565	7.419937e+04	17.390758	11.318494	1.585165e+04	53.8
std	8357.054944	4.472944	250.700153	3.559331	6.155725e+04	18.032696	5.134901	2.058427e+04	24.4
min	500.000000	5.320000	16.080000	0.000000	0.000000e+00	0.000000	1.000000	0.000000e+00	0.0
25%	8000.000000	10.490000	250.330000	3.000000	4.500000e+04	11.300000	8.000000	6.038000e+03	35.9
50%	12000.000000	13.330000	375.490000	6.000000	6.400000e+04	16.920000	10.000000	1.119000e+04	54.8
75%	20000.000000	16.550000	567.790000	10.000000	9.000000e+04	22.990000	14.000000	1.962600e+04	72.9
max	40000.000000	30.990000	1533.810000	10.000000	8.706582e+06	9999.000000	90.000000	1.743266e+06	892.3

In [481…
```
df.shape
```

Out[481]:
```
(368079, 29)
```

In [482…
```
#Correlation between numerical features
plt.figure(figsize=(8,6))
sns.heatmap(num_cols.corr(), annot=True, fmt=".1f")
plt.title('Correlation between Numerical Features')
plt.show()
```



Observation:

- open_acc and total_acc are highly correlated (0.7).

- mort_acc and total_acc have a moderate positive correlation (0.4).
- loan_amnt and installment show perfect positive correlation (1.0).
- loan_amnt is moderately correlated with annual_inc (0.3).
- loan_amnt is weakly correlated with open_acc (0.2) and revol_bal (0.3).
- revol_bal and revol_util have a weak positive correlation (0.3).
- dti shows weak correlations with most other features.
- No strong negative correlations are observed between any variables.
- Many feature relationships, like int_rate and annual_inc, are weak or negligible.

Note - VIF

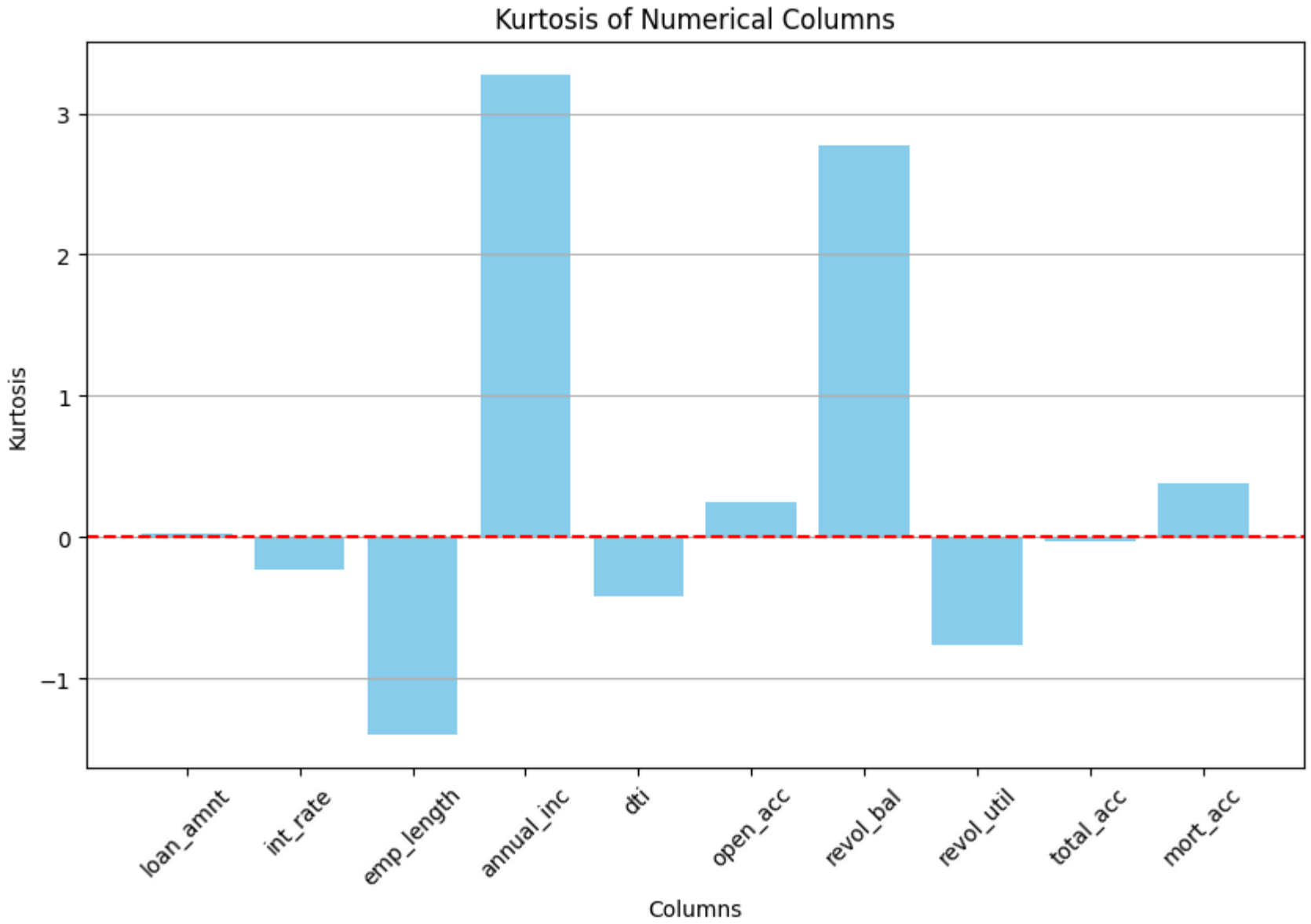
- since installments and loan amount are strongly positively correlated drop installment column to avoid multicolinearity
- since we have extracted the state and zipcode from address column we can drop that column as well

In [483…
```
df.drop(columns=['installment', 'address'], inplace=True)
```

Observing the kurtosis

In [484…
```
from scipy.stats import kurtosis

# Sample data (replace this with your actual DataFrame)
# df = pd.read_csv('your_data.csv')

# Get the numerical columns
num_cols = df.select_dtypes(include=['float64']).columns

# Calculate kurtosis for each numerical column
kurtosis_values = [kurtosis(df[col].dropna()) for col in num_cols]

# Create a DataFrame to hold the kurtosis values
kurtosis_df = pd.DataFrame({'Column': num_cols, 'Kurtosis': kurtosis_values})

# Plotting
plt.figure(figsize=(10, 6))
plt.bar(kurtosis_df['Column'], kurtosis_df['Kurtosis'], color='skyblue')
```

```
plt.axhline(0, color='red', linestyle='--')  # Reference line at 0
plt.title('Kurtosis of Numerical Columns')
plt.xlabel('Columns')
plt.ylabel('Kurtosis')
plt.xticks(rotation=45)  # Rotate x labels for better readability
plt.grid(axis='y')  # Add grid lines for better visualization
plt.show()
```



Observation:

- Annual income (annual_inc) shows the highest positive kurtosis, significantly above all other features. This indicates a distribution with heavy tails and a sharp peak, suggesting the presence of extreme values or outliers in borrowers' income levels.
- Employment length (emp_length) displays the most negative kurtosis, indicating a very flat or potentially bimodal distribution. This suggests a diverse range of employment durations among borrowers, which could impact risk assessment.
- Revolving balance (revol_bal) exhibits the second-highest positive kurtosis, implying a distribution with heavier tails than most other features. This could indicate some borrowers have significantly higher revolving credit balances than others, potentially affecting their credit risk profile.

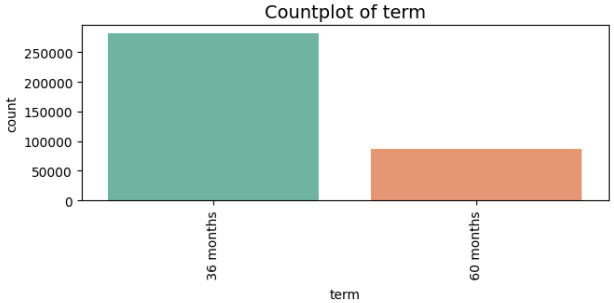
Observing the frequencies of all the categorical columns

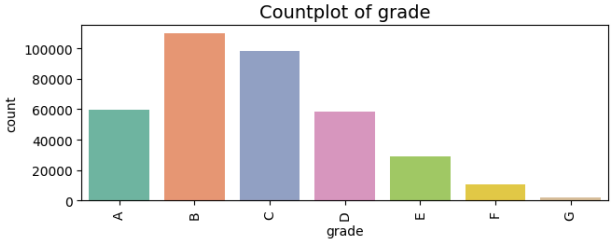
In [485…
```
cat_cols = df.columns[df.dtypes=='category']
cat_cols
```

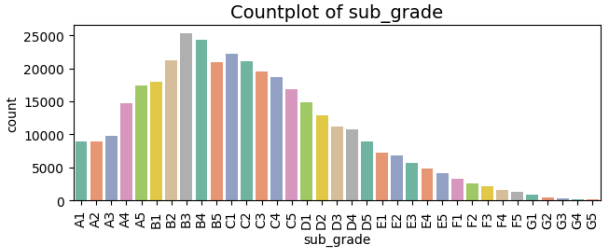
Out[485]:
```
Index(['term', 'grade', 'sub_grade', 'home_ownership', 'verification_status',
       'loan_status', 'purpose', 'initial_list_status', 'application_type',
       'state', 'zip_code'],
      dtype='object')
```

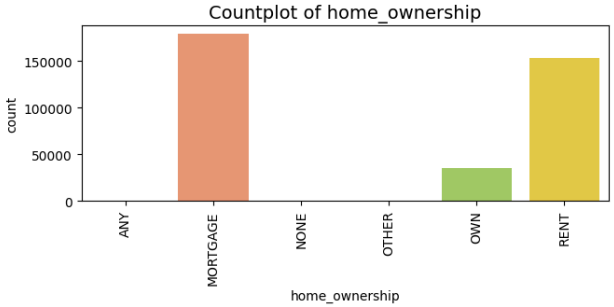
In [486…
```
# Number of categorical columns
cat_cols = ['term', 'grade', 'sub_grade', 'home_ownership', 'verification_status',
            'loan_status', 'purpose', 'initial_list_status', 'application_type',
            'state', 'zip_code']

# Calculate the number of rows and columns for subplots
n_cols = 3
n_rows = (len(cat_cols) + n_cols - 1) // n_cols  # This ensures we have enough rows

# Create subplots
fig, axes = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(20, 15))

# Flatten axes to iterate
axes = axes.flatten()

# Define a list of color palettes to use
palettes = sns.color_palette('Set2', len(cat_cols))  # Using 'Set2' palette, but you can mix others

# Loop over each column and plot the countplot with different palettes
for i, col in enumerate(cat_cols):
    sns.countplot(data=df, x=col, ax=axes[i], palette=palettes)
```

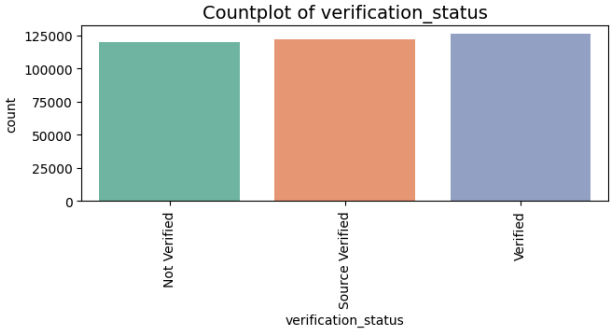
```
    axes[i].set_title(f'Countplot of {col}', fontsize=14)
    axes[i].tick_params(axis='x', rotation=90)  # Rotate x-axis labels for visibility

# Hide any extra subplots (if there are fewer columns than subplots)
for j in range(i+1, len(axes)):
    fig.delaxes(axes[j])

# Adjust layout to avoid overlap
plt.tight_layout()

# Show the plot
plt.show()
```

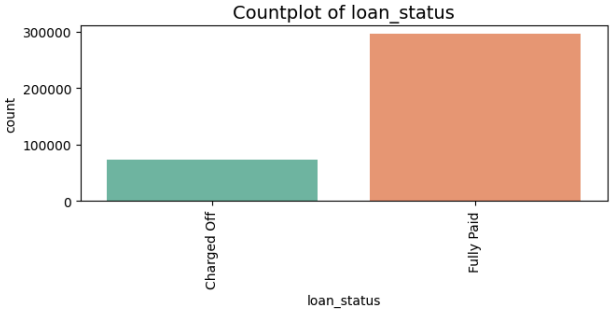

Countplot of term
count
term
36 months
60 months

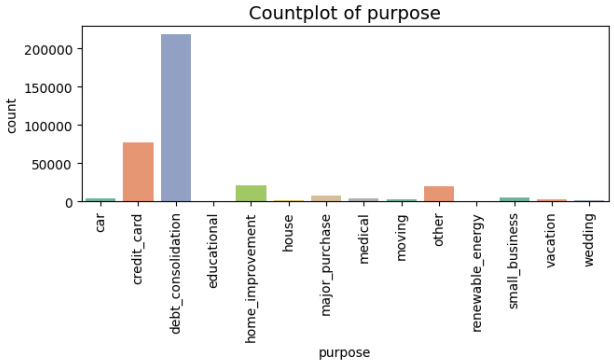

Countplot of grade
count
grade
A
B
C
D
E
F
G

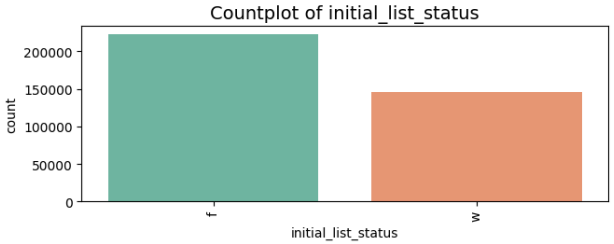

Countplot of sub_grade
count
sub_grade

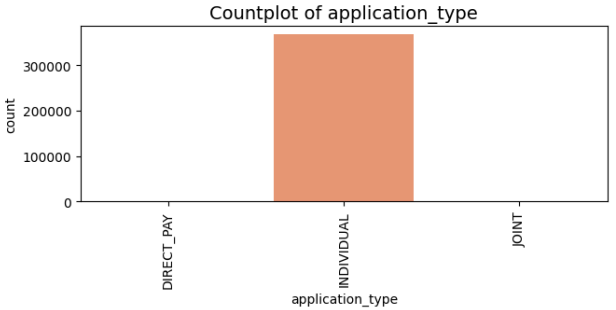

Countplot of home_ownership
count
home_ownership
ANY
MORTGAGE
NONE
OTHER
OWN
RENT

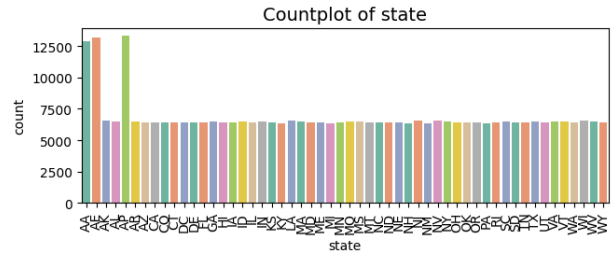

Countplot of verification_status
count
verification_status
Not Verified
Source Verified
Verified

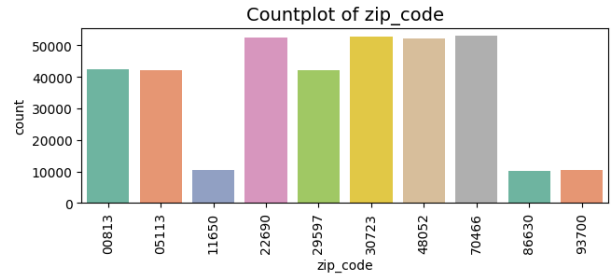

Countplot of loan_status
count
loan_status
Charged Off
Fully Paid


Countplot of purpose
count
purpose
car
credit_card
debt_consolidation
educational
home_improvement
house
major_purchase
medical
moving
other
renewable_energy
small_business
vacation
wedding


Countplot of initial_list_status
count
initial_list_status
f
w


Countplot of application_type
count
application_type
DIRECT_PAY
INDIVIDUAL
JOINT


Countplot of state
count
state


Countplot of zip_code
count
zip_code
00813
05113
11650
22690
29597
30723
48052
70466
86630
93700

Observation:

- Loan Terms: 36-month loans (265,000) vastly outnumber 60-month loans (80,000), indicating a 3:1 preference for shorter terms.
- Credit Grades: B (110,000) and C (100,000) grades dominate, comprising over 50% of all loans.
- Verification: Even distribution across statuses - Not Verified, Source Verified, and Verified (each ~120,000), suggesting flexible verification policies.
- Loan Purpose: Debt consolidation (220,000) is the primary reason, followed by credit card refinancing (80,000).
- Home Ownership: Renters (170,000) and mortgage holders (150,000) are the main borrower types.
- Top ZIP Codes: 22690, 30723, 70466, and 48052 show highest loan concentrations (each ~50,000).
- Application Type: Individual applications (350,000) strongly preferred over joint applications.
- Loan Status: Fully paid loans (300,000) outnumber charged-off loans (80,000) by nearly 4:1.
- State Distribution: CA, NY, and TX show slightly higher loan counts, but generally even across states.
- Initial List Status: Whole loans (220,000) preferred over fractional loans (150,000).

understanding the impact of categorical columns on loan status column

- here we need to remove the loan status column from the cat_cols , before that let's see the percentage of customers who has fully paid there loans

In [487…

```
# Get the value counts as percentages
loan_status_percentage = df['loan_status'].value_counts(normalize=True) * 100

# Display the result
print(loan_status_percentage)
```

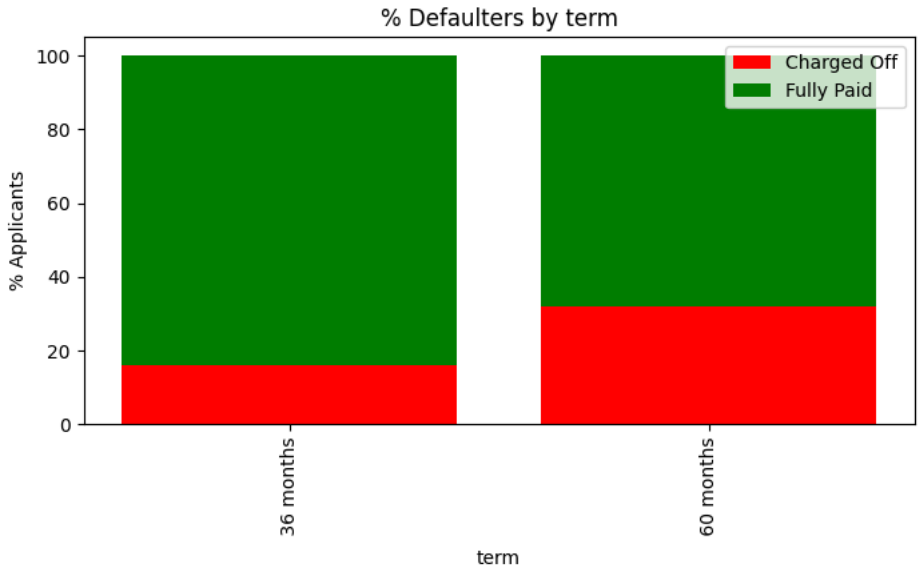
```
loan_status
Fully Paid     80.287112
Charged Off    19.712888
Name: proportion, dtype: float64
```

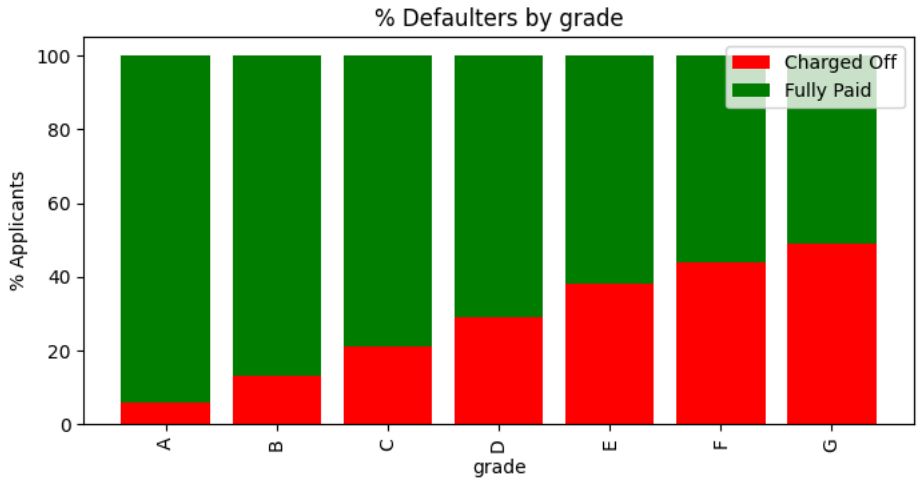
In [488…

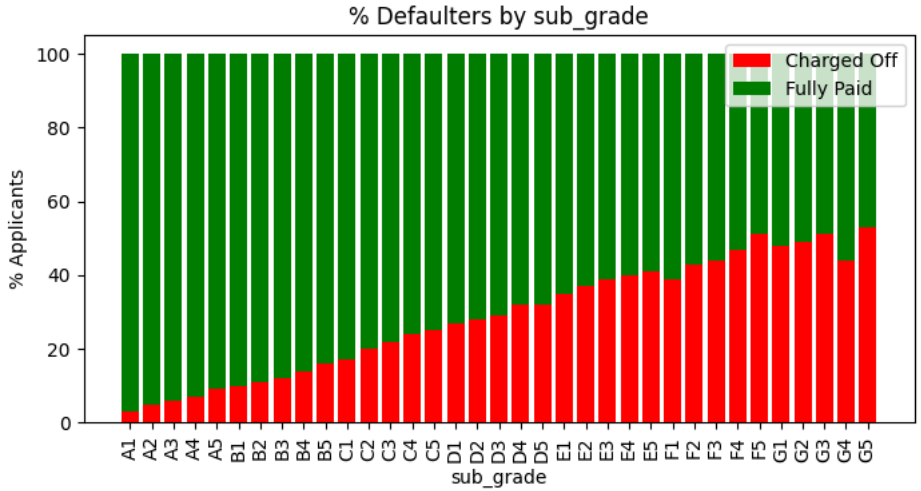
```
cat_cols = ['term','grade','sub_grade',
            'home_ownership','verification_status','purpose',
            'initial_list_status','application_type','state','zip_code']
```

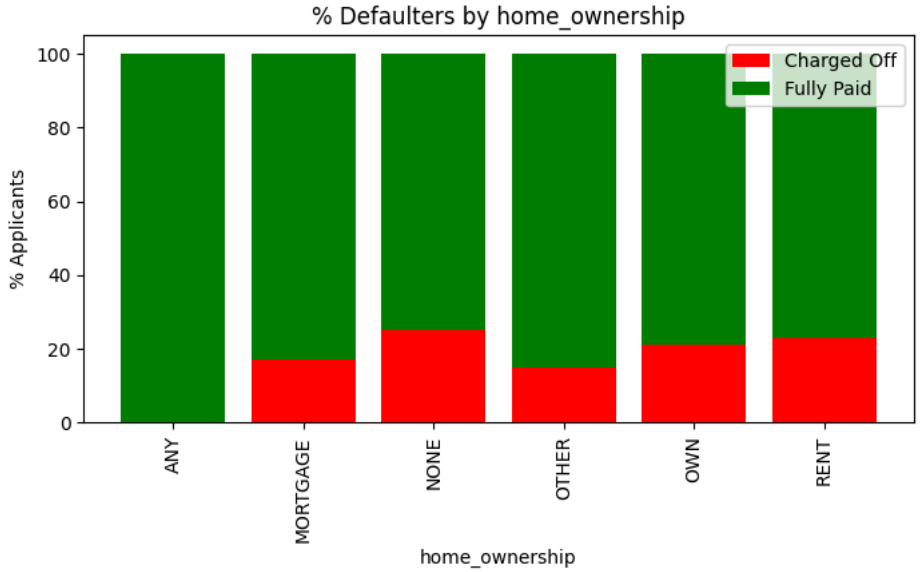
In [489…

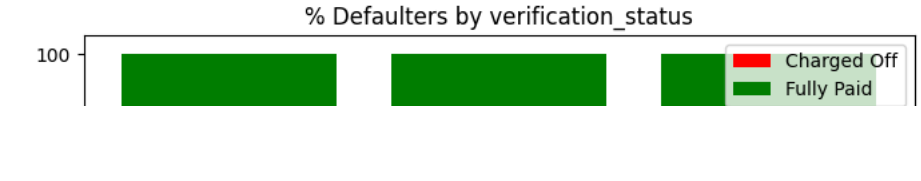
```
import matplotlib.pyplot as plt

plt.figure(figsize=(14, 24))
num_plots = len(cat_cols)
rows = (num_plots // 2) + (num_plots % 2)  # Calculate the number of rows needed
i = 1

for col in cat_cols:
    ax = plt.subplot(rows, 2, i)

    # Create a pivot table to calculate the percentage
    data = df.pivot_table(index=col, columns='loan_status', aggfunc='size', fill_value=0)
    data = data.div(data.sum(axis=1), axis=0).multiply(100).round()
    data.reset_index(inplace=True)

    # Plotting
    plt.bar(data[col], data['Charged Off'], color='red', label='Charged Off')
    plt.bar(data[col], data['Fully Paid'], color='green', bottom=data['Charged Off'], label='Fully Paid')

    plt.xlabel(f'{col}')
    plt.ylabel('% Applicants')
    plt.title(f'% Defaulters by {col}')
    plt.xticks(rotation=90)

    # Add a legend to the subplot
    plt.legend(loc='upper right')
    i += 1

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```

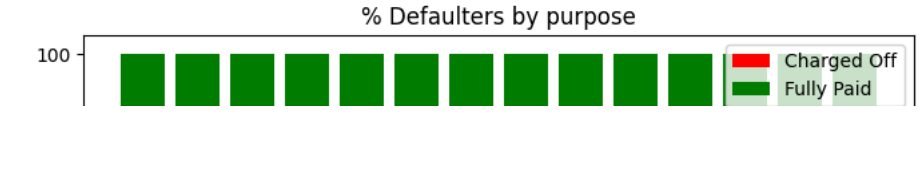

% Defaulters by term
Charged Off
Fully Paid
% Applicants
0
20
40
60
80
100
36 months
60 months
term

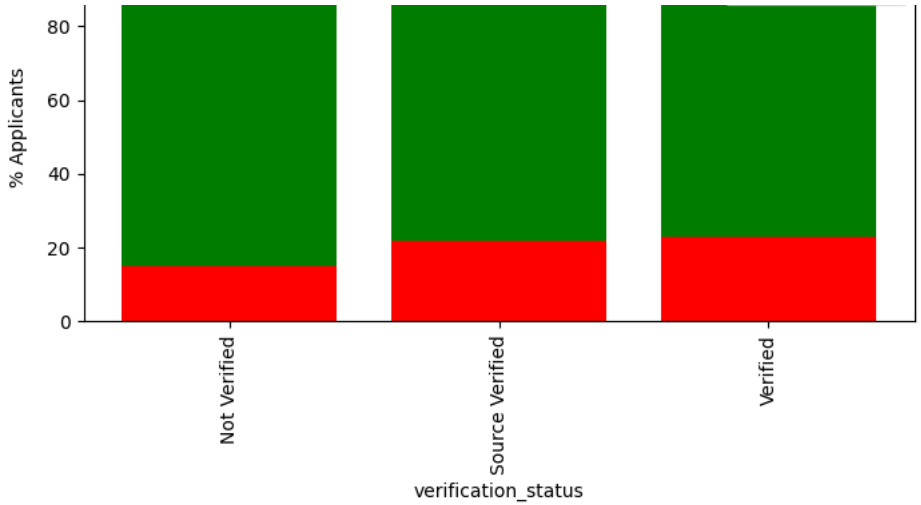

% Defaulters by grade
Charged Off
Fully Paid
% Applicants
0
20
40
60
80
100
A
B
C
D
E
F
G
grade

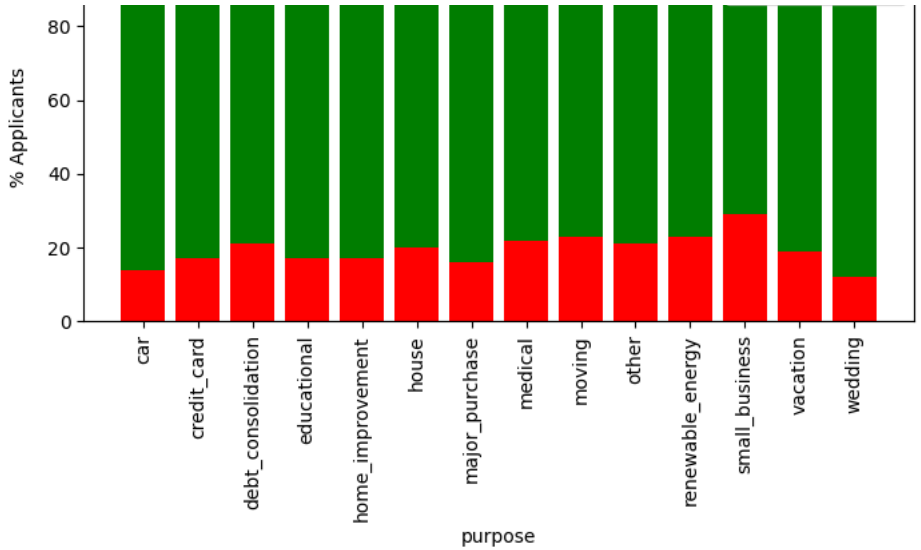

% Defaulters by sub_grade
Charged Off
Fully Paid
% Applicants
0
20
40
60
80
100
A1
A2
A3
A4
A5
B1
B2
B3
B4
B5
C1
C2
C3
C4
C5
D1
D2
D3
D4
D5
E1
E2
E3
E4
E5
F1
F2
F3
F4
F5
G1
G2
G3
G4
G5
sub_grade

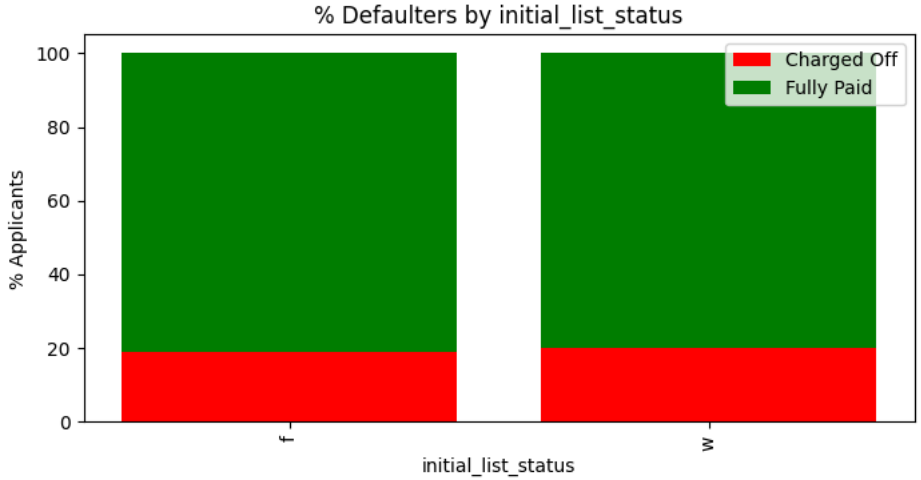

% Defaulters by home_ownership
Charged Off
Fully Paid
% Applicants
0
20
40
60
80
100
ANY
MORTGAGE
NONE
OTHER
OWN
RENT
home_ownership

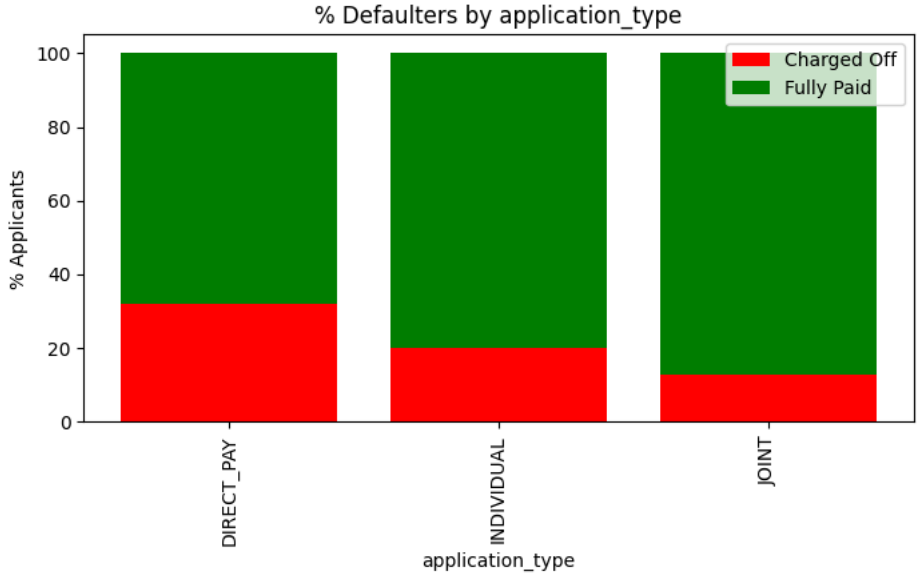

% Defaulters by verification_status
Charged Off
Fully Paid
100

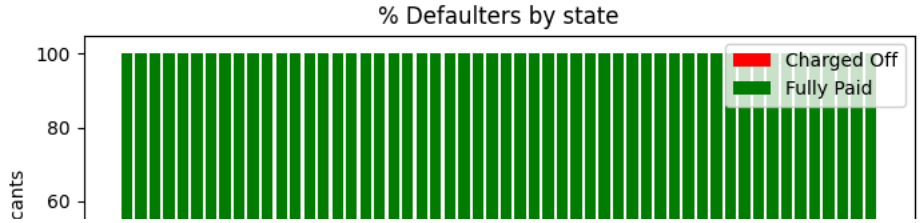

% Defaulters by purpose
Charged Off
Fully Paid
100

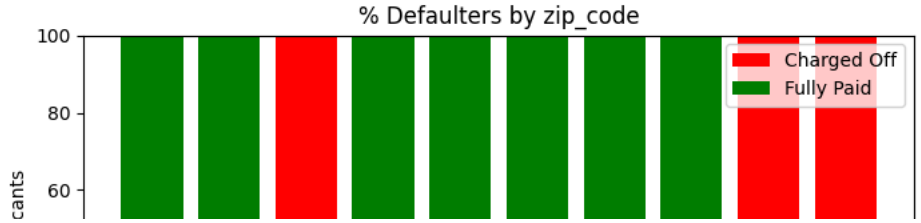

% Applicants
0
20
40
60
80
Not Verified
Source Verified
Verified
verification_status

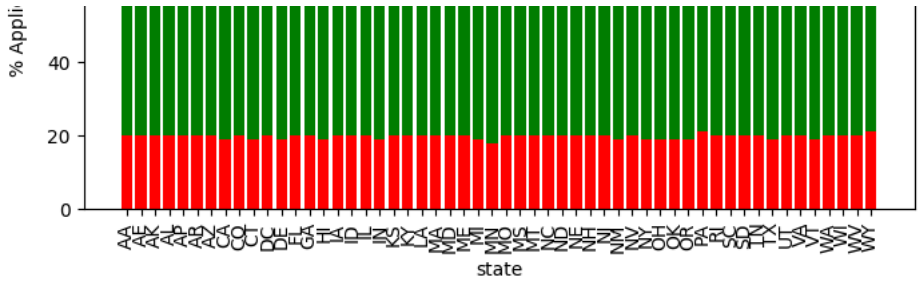

% Applicants
0
20
40
60
80
car
credit_card
debt_consolidation
educational
home_improvement
house
major_purchase
medical
moving
other
renewable_energy
small_business
vacation
wedding
purpose

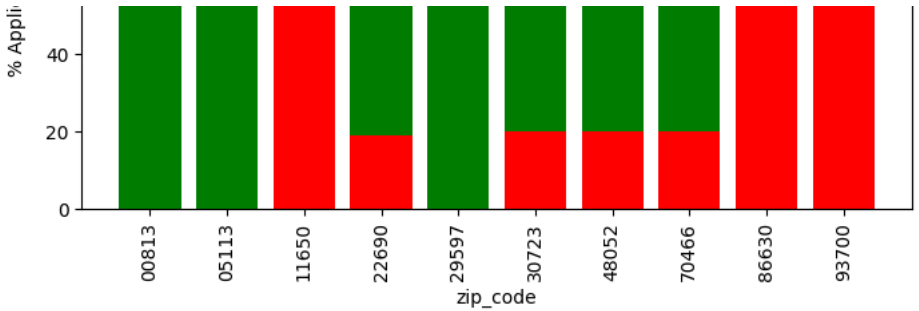

% Defaulters by initial_list_status
Charged Off
Fully Paid
% Applicants
0
20
40
60
80
100
f
w
initial_list_status


% Defaulters by application_type
Charged Off
Fully Paid
% Applicants
0
20
40
60
80
100
DIRECT_PAY
INDIVIDUAL
JOINT
application_type


% Defaulters by state
Charged Off
Fully Paid
cants
60
80
100


% Defaulters by zip_code
Charged Off
Fully Paid
cants
60
80
100



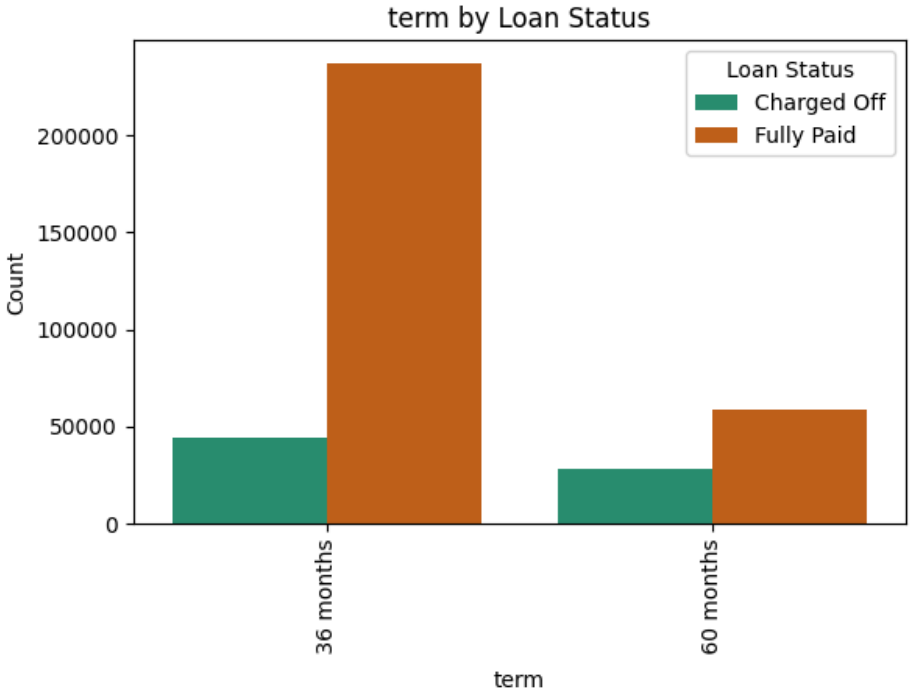


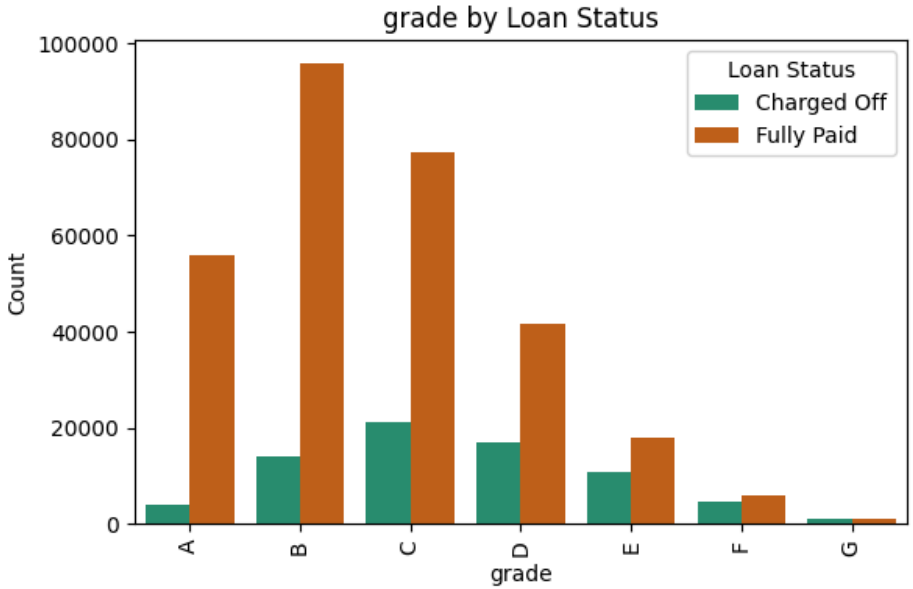
Observations:

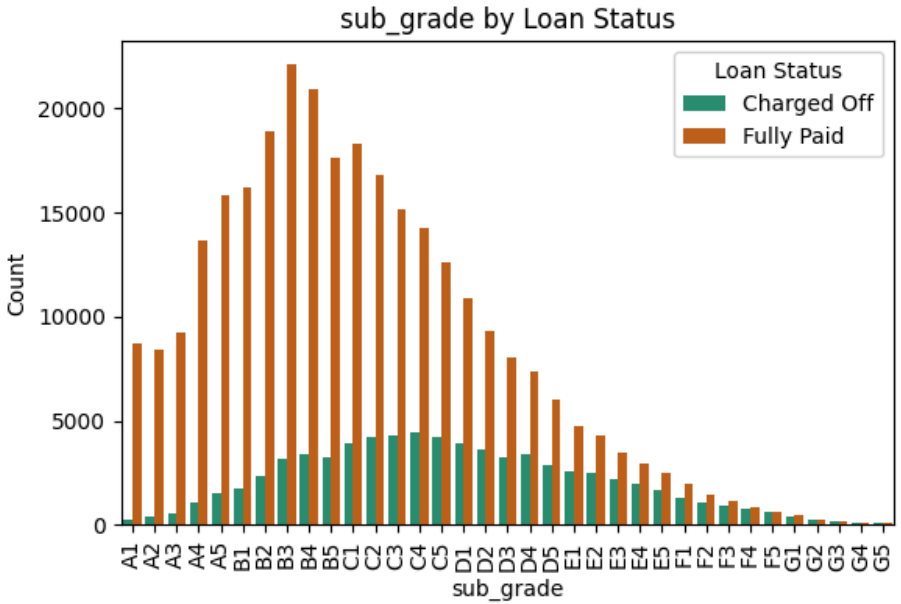
- % Defaulters by term: The 60-month term loans have a significantly higher percentage of "Charged off" loans (39%) compared to 36-month term loans (18%)
- % Defaulters by grade: Lower grades (E, F, G) have progressively higher percentages of "Charged off" loans, with Grade G having the highest percentage (51.2%) compared to higher grades (A, B, C), with Grade A having the lowest percentage (14.5%).
- % Defaulters by home_ownership: Renters ("rent" category) appear to have a higher percentage of "Charged off" loans (38.5%) compared to homeowners ("own" category) (24.1%) and those with mortgages (29.4%).
- Verification status impact: The "Verified" category has the highest percentage of charged-off loans at approximately 24%.
- Loan purpose variation: The "small_business" purpose category has the highest proportion of charged-off loans at about 32%.
- Application type difference: "Joint App" has the lowest percentage of charged-off loans at roughly 14%, significantly lower than "Individual" applications at about 21%.
- State-level consistency: Most states show a similar pattern with roughly 80% fully paid (green) and 20% charged off (red) loans.
- Zip code variation: The zip code chart shows more variation between zip codes, with some having very high percentages of fully paid loans (green) and others having higher percentages of charged off loans (red).
- Extreme cases: Some zip codes (like 00813 and 05113) show nearly 100% full payment rates, while others (like 11650 and 93700) show very high charge-off rates, approaching or reaching 100%.

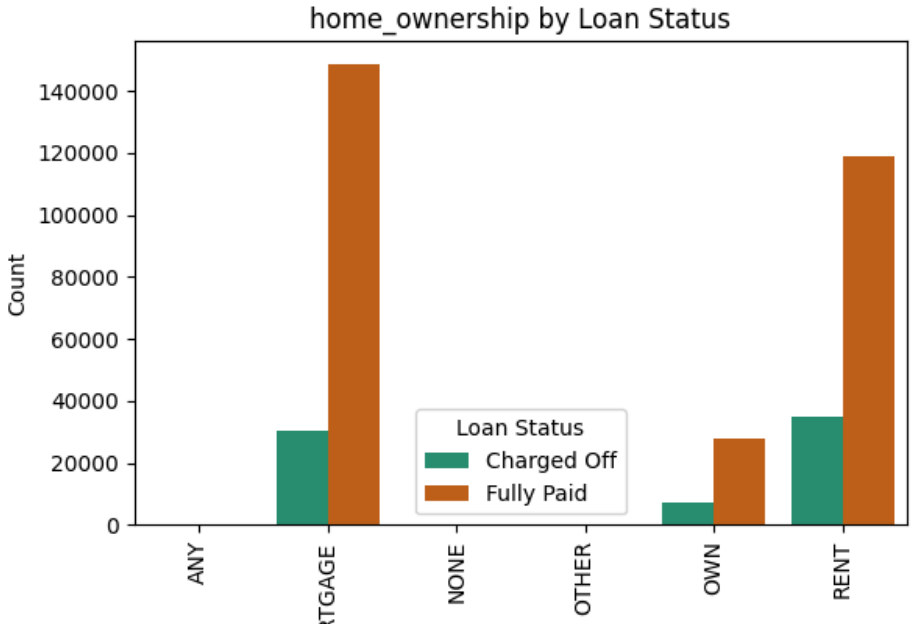
In [490…

```
num_plots = len(cat_cols)
rows = (num_plots // 2) + (num_plots % 2)  # Calculate the number of rows needed

plt.figure(figsize=(12, rows * 5))  # Adjust the height based on the number of rows

for i, col in enumerate(cat_cols, 1):
    ax = plt.subplot(rows, 2, i)  # Create a subplot for each column

    # Create count plot for the categorical column
    sns.countplot(x=col, data=df, hue='loan_status', palette='Dark2', ax=ax)

    ax.set_title(f'{col} by Loan Status')
    ax.set_xlabel(col)
    ax.set_ylabel('Count')
    ax.tick_params(axis='x', rotation=90)  # Rotate x labels if needed for better readability
    ax.legend(title='Loan Status')

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```


term by Loan Status
Loan Status
Charged Off
Fully Paid
Count
200000
150000
100000
50000
0
36 months
60 months
term

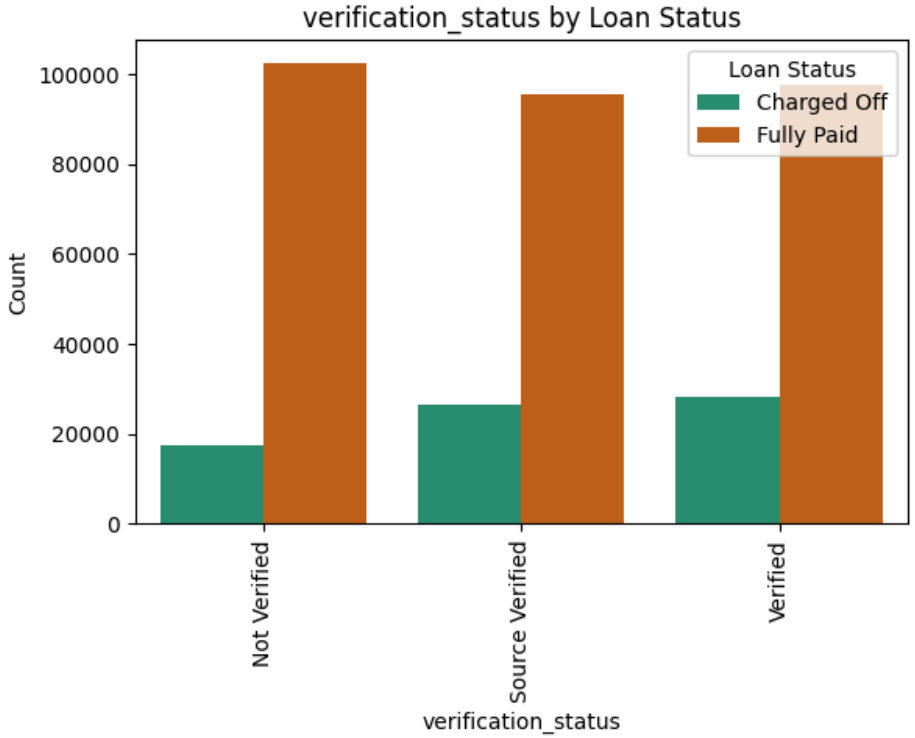

grade by Loan Status
Loan Status
Charged Off
Fully Paid
Count
100000
80000
60000
40000
20000
0
A
B
C
D
E
F
G
grade

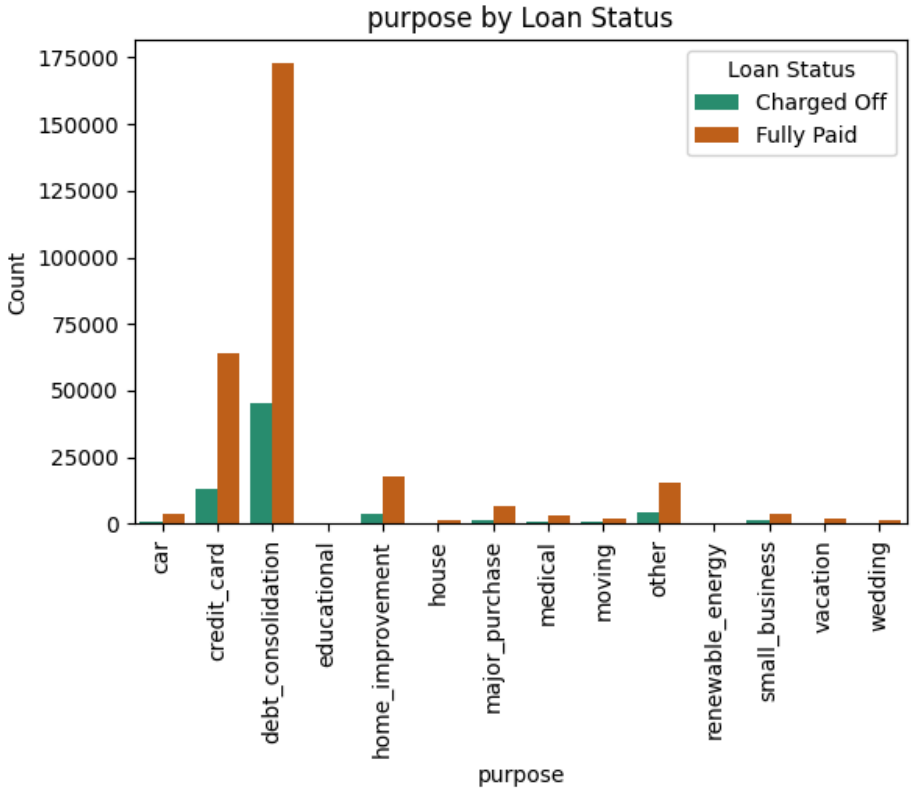

sub_grade by Loan Status
Loan Status
Charged Off
Fully Paid
Count
20000
15000
10000
5000
0
A1 A2 A3 A4 A5 B1 B2 B3 B4 B5 C1 C2 C3 C4 C5 D1 D2 D3 D4 D5 E1 E2 E3 E4 E5 F1 F2 F3 F4 F5 G1 G2 G3 G4 G5
sub_grade

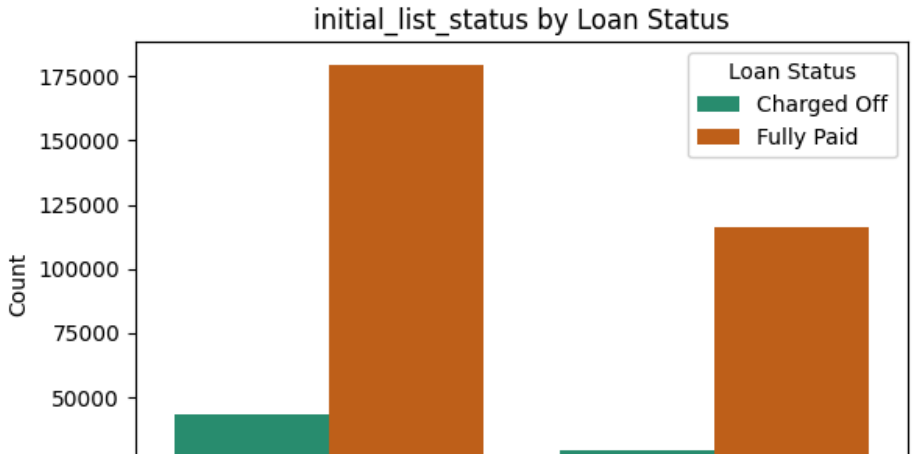

home_ownership by Loan Status
Loan Status
Charged Off
Fully Paid
Count
140000
120000
100000
80000
60000
40000
20000
0
ANY
NONE
OTHER
OWN
RENT

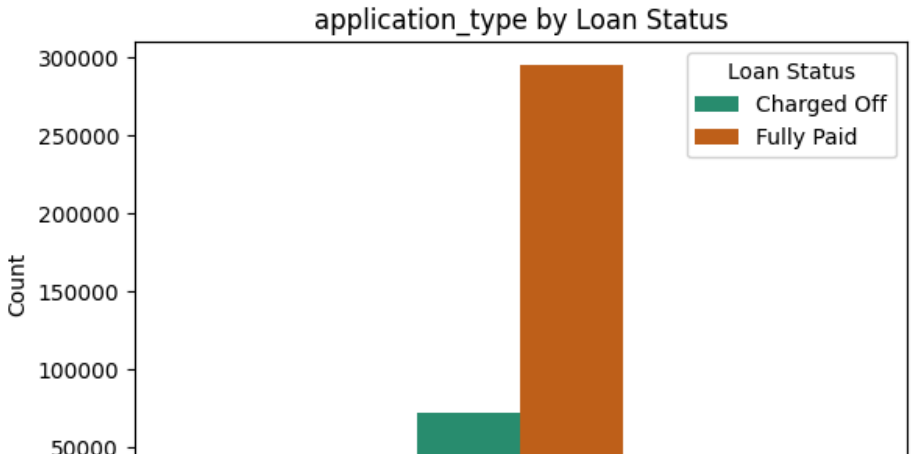
MOF

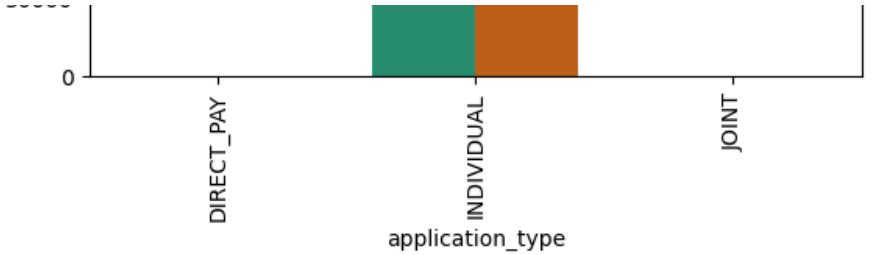
home_ownership

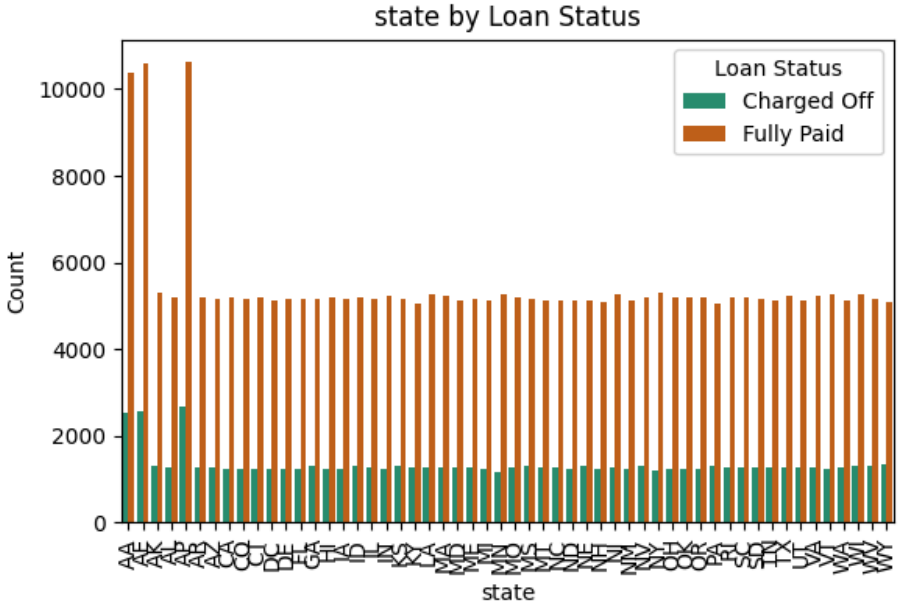


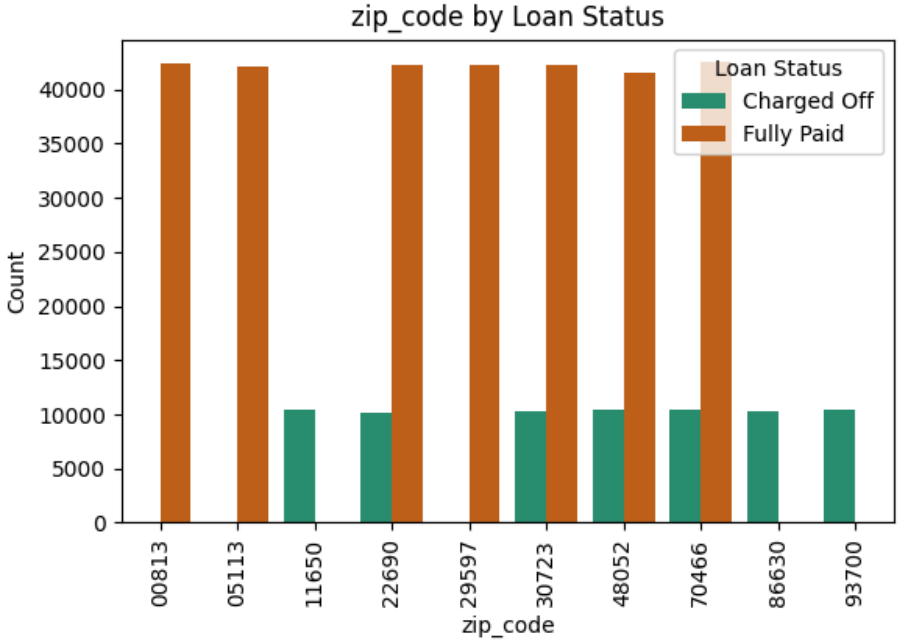






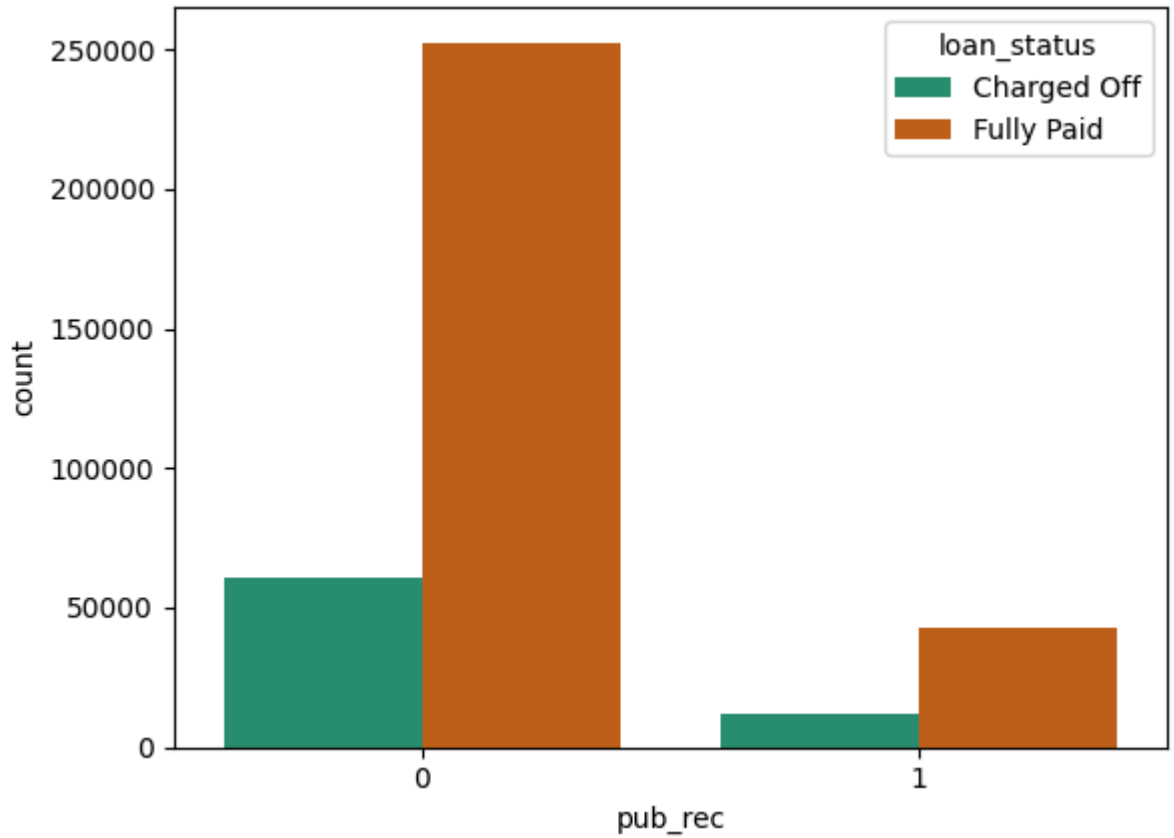






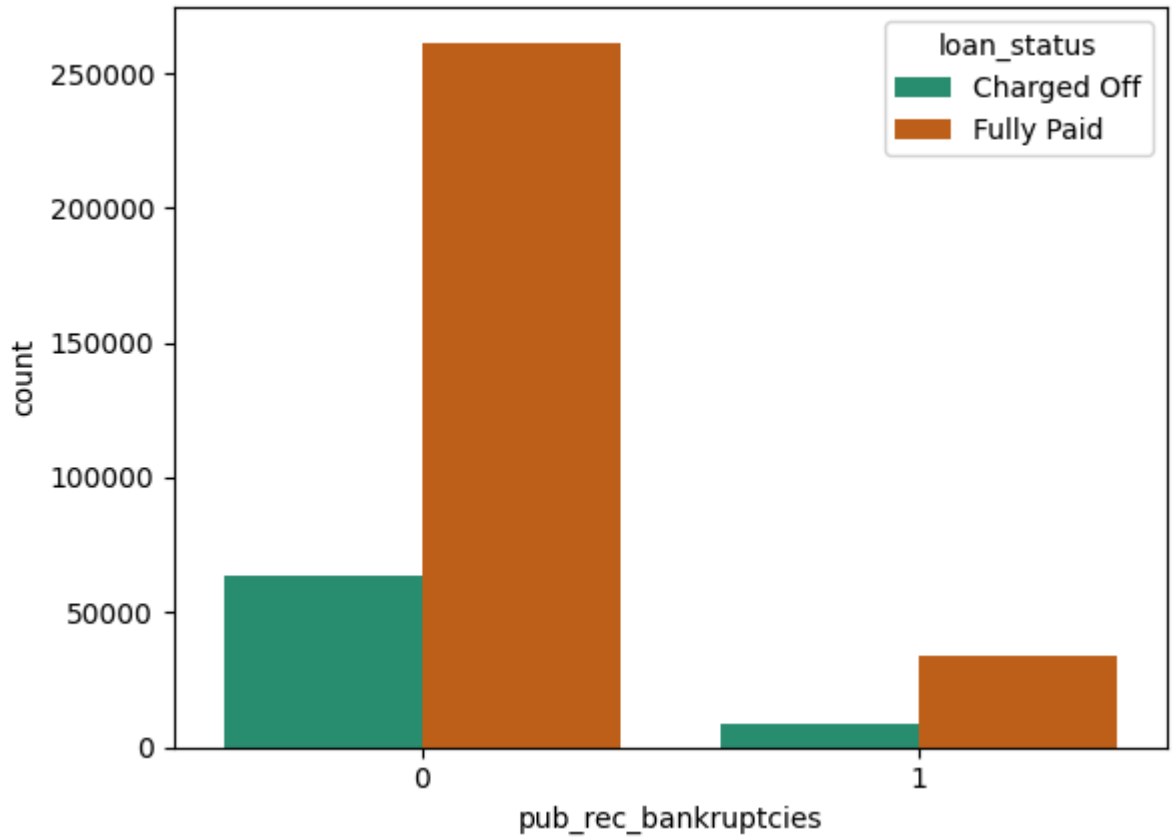
In [491…

```
sns.countplot(data = df, x = 'pub_rec',hue = 'loan_status',palette='Dark2')
plt.show()
```



In [492…

```
sns.countplot(data = df, x = 'pub_rec_bankruptcies',hue = 'loan_status',palette='Dark2' )
plt.show()
```



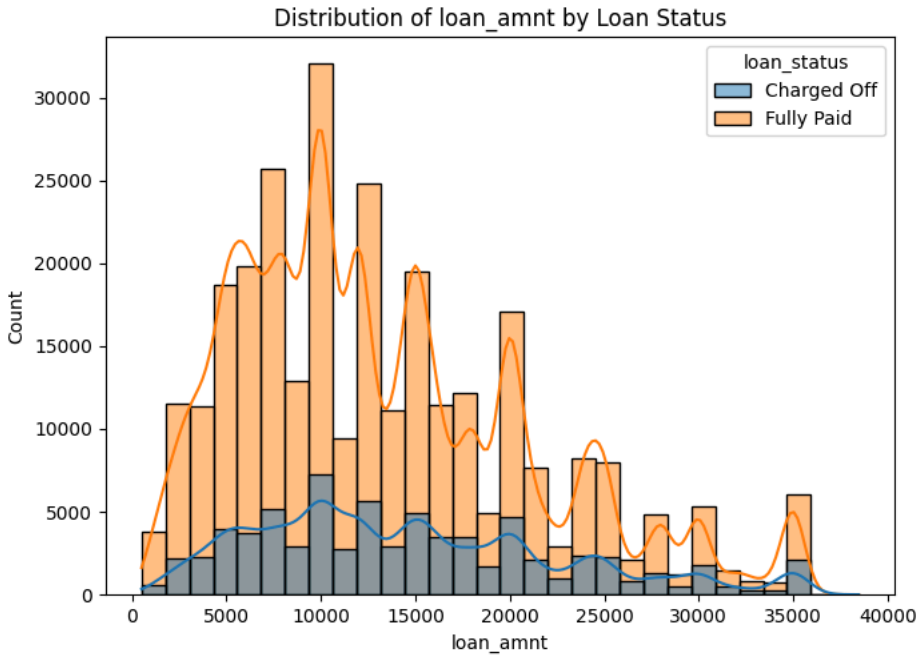
Understanding the importance of numerical features on loan status column

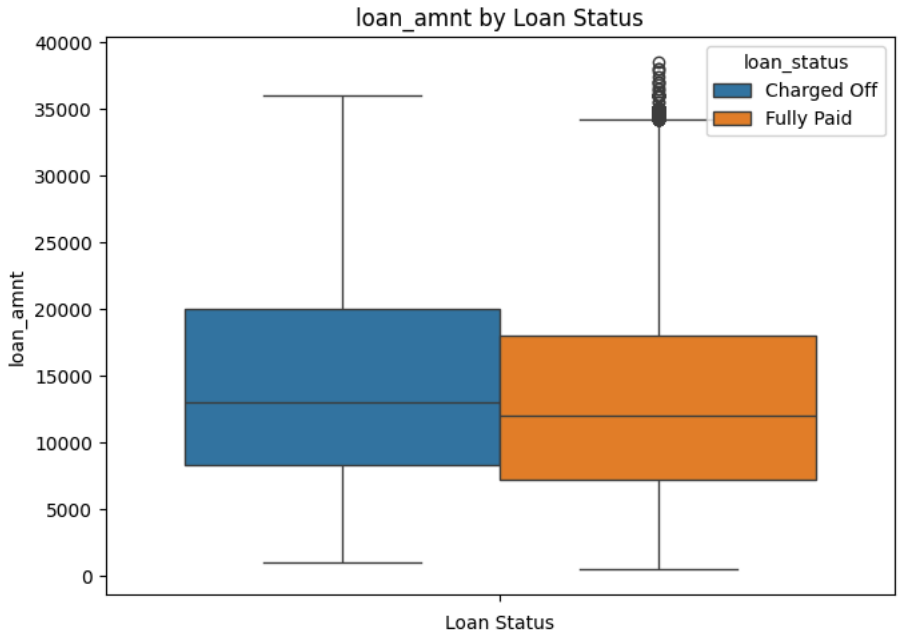
In [493…
```
num_cols = df.columns[df.dtypes == 'float64']
num_cols
```

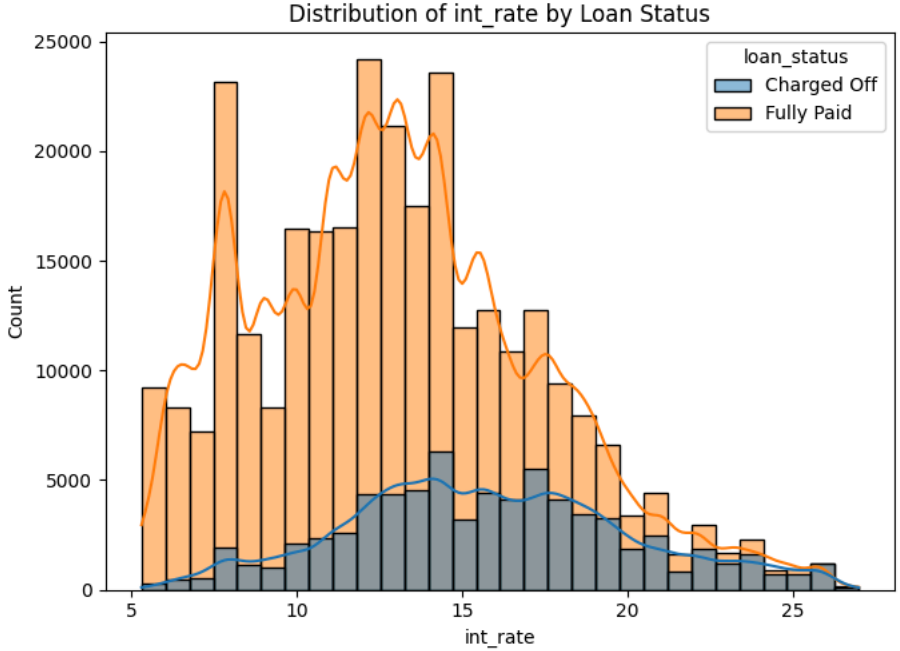
Out[493]:
```
Index(['loan_amnt', 'int_rate', 'emp_length', 'annual_inc', 'dti', 'open_acc',
       'revol_bal', 'revol_util', 'total_acc', 'mort_acc'],
      dtype='object')
```

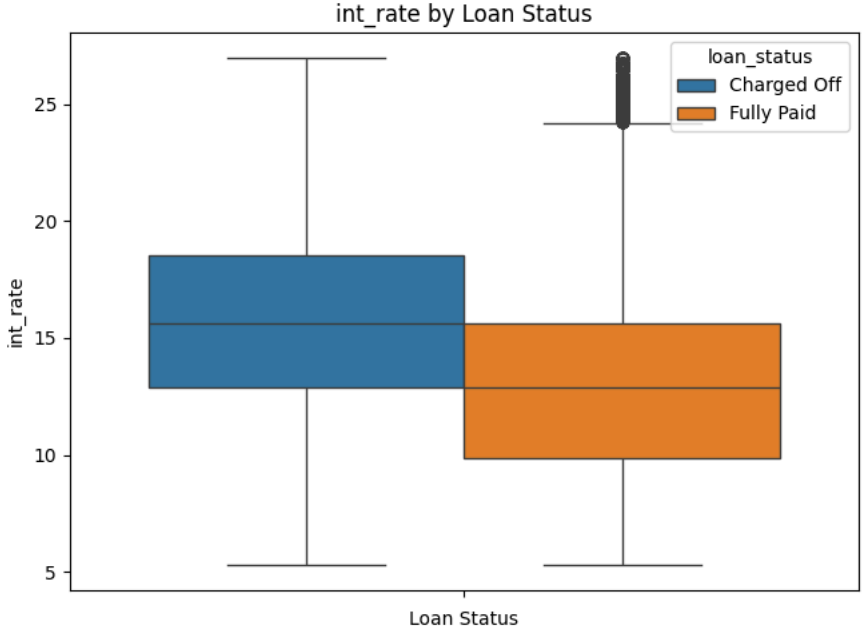
In [494…
```
len_cols = len(num_cols)

# Set the number of subplots
```

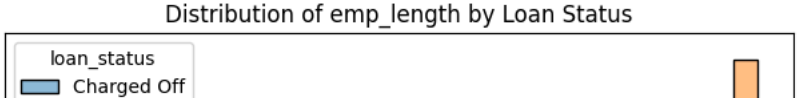
```python
fig, axes = plt.subplots(len_cols, 2, figsize=(14, 5 * len_cols))  # Create a subplot for each numerical column

for i, col in enumerate(num_cols):
    # Histogram
    sns.histplot(data=df, x=col, hue='loan_status', kde=True, ax=axes[i, 0], bins=30)
    axes[i, 0].set_title(f'Distribution of {col} by Loan Status')
    axes[i, 0].set_xlabel(col)
    axes[i, 0].set_ylabel('Count')

    # Boxplot
    sns.boxplot(data=df, y=col, hue='loan_status', ax=axes[i, 1])
    axes[i, 1].set_title(f'{col} by Loan Status')
    axes[i, 1].set_xlabel('Loan Status')
    axes[i, 1].set_ylabel(col)

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```

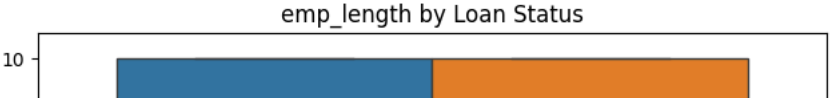

Distribution of loan_amnt by Loan Status
loan_status
Charged Off
Fully Paid
Count
loan_amnt
0
5000
10000
15000
20000
25000
30000
35000
40000

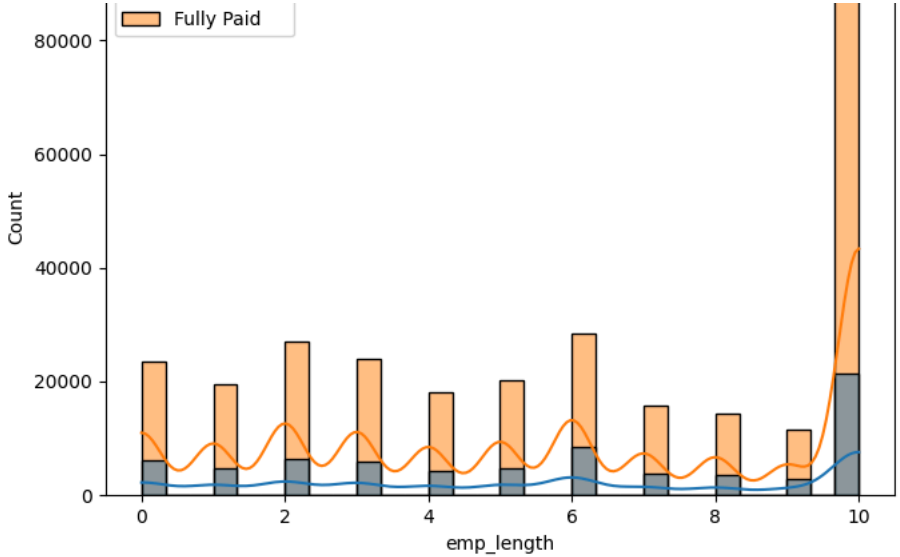

loan_amnt by Loan Status
loan_status
Charged Off
Fully Paid
loan_amnt
Loan Status

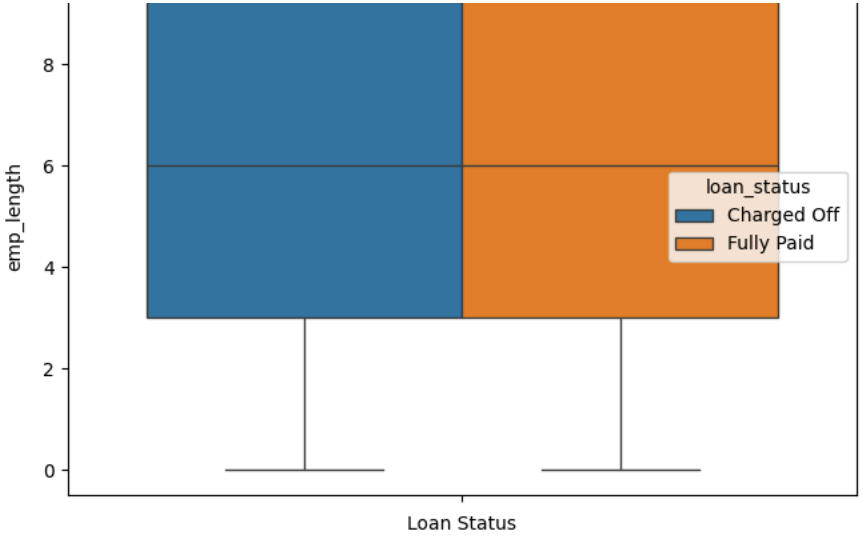

Distribution of int_rate by Loan Status
loan_status
Charged Off
Fully Paid
Count
int_rate
5
10
15
20
25

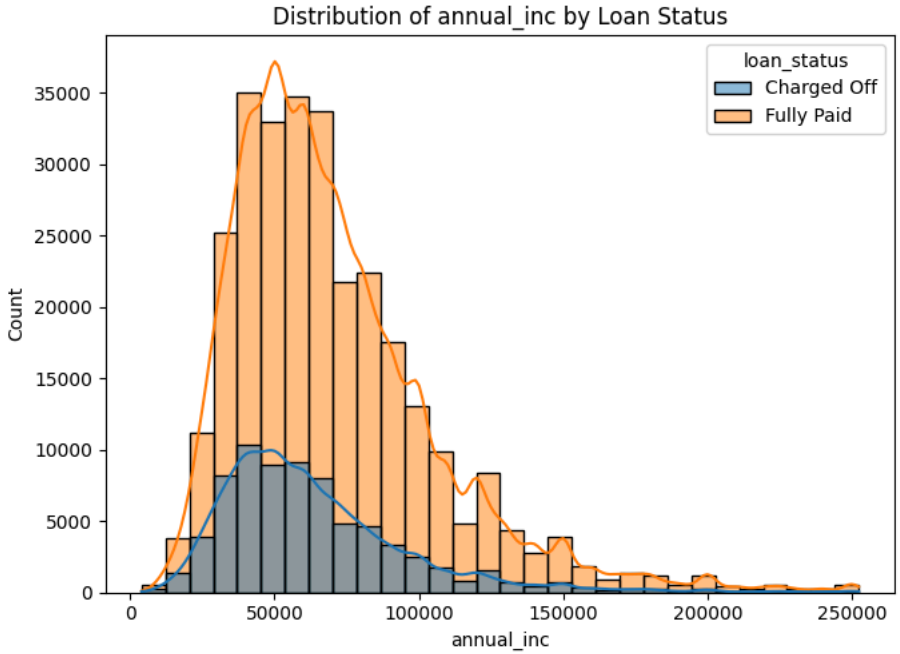

int_rate by Loan Status
loan_status
Charged Off
Fully Paid
int_rate
Loan Status

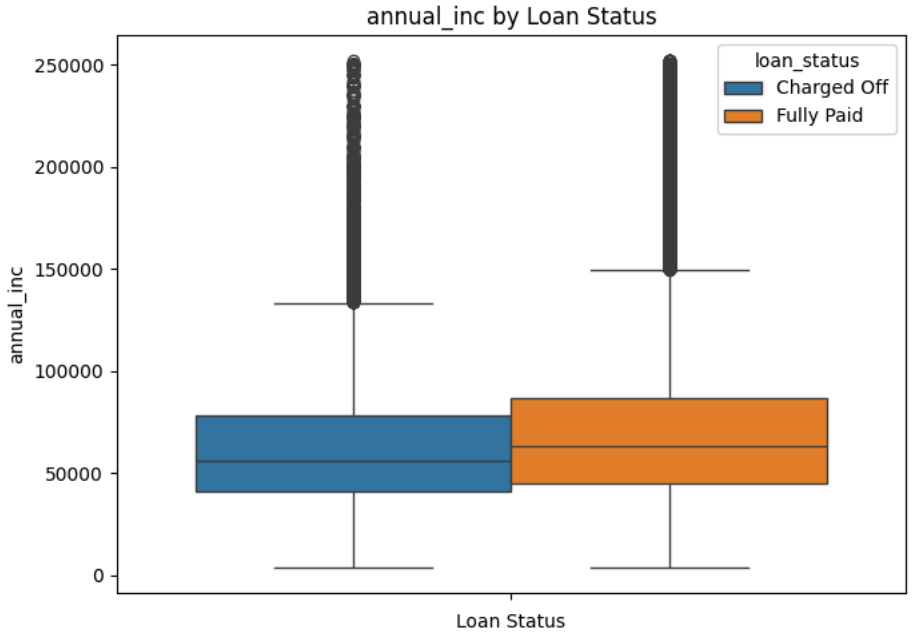

Distribution of emp_length by Loan Status
loan_status
Charged Off


emp_length by Loan Status

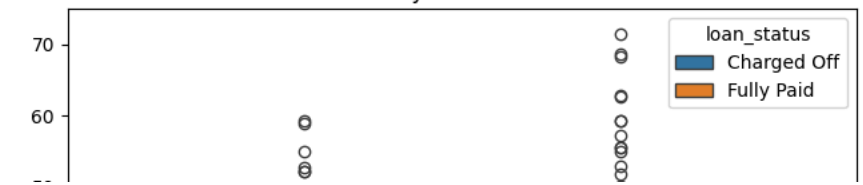

Fully Paid
Count
emp_length

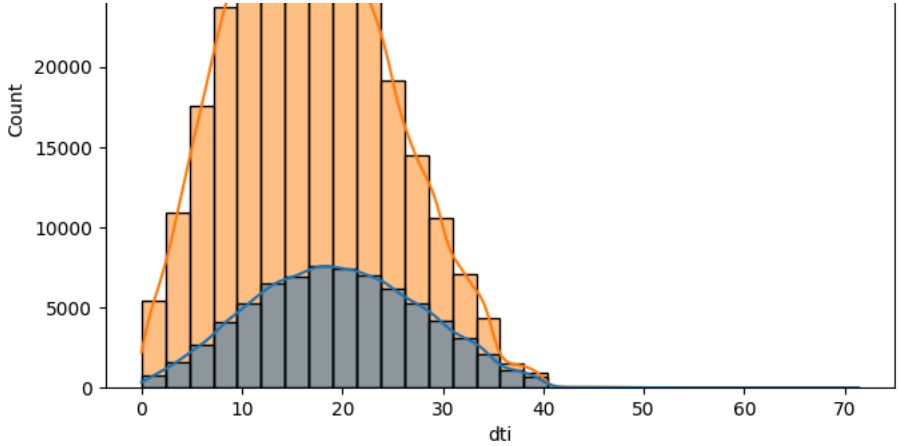

emp_length
loan_status
Charged Off
Fully Paid
Loan Status

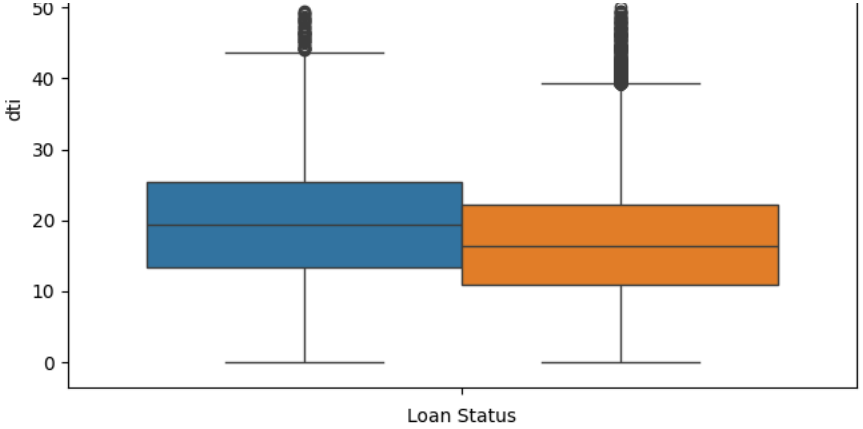

Distribution of annual_inc by Loan Status
loan_status
Charged Off
Fully Paid
Count
annual_inc

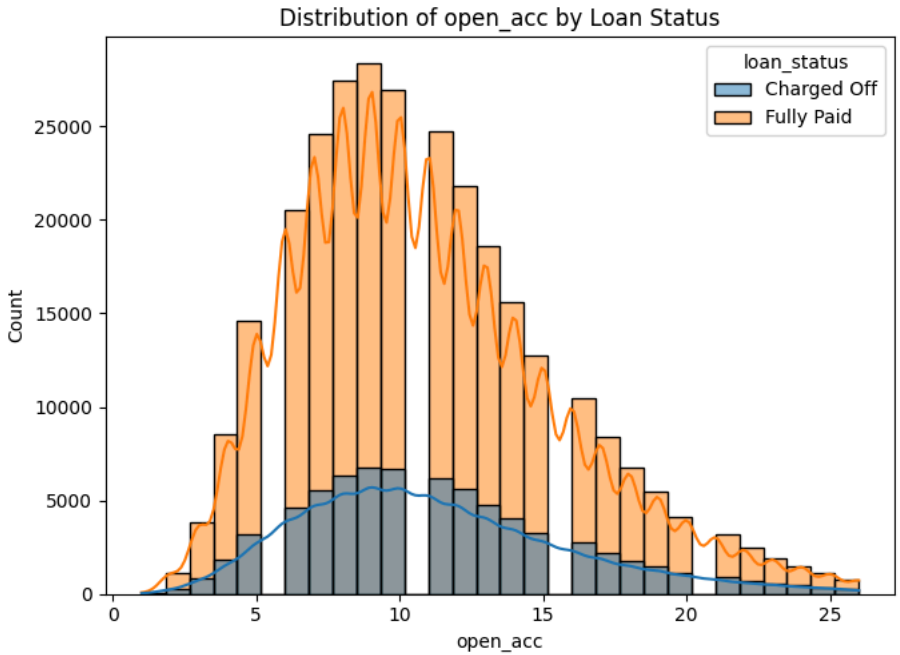

annual_inc by Loan Status
loan_status
Charged Off
Fully Paid
annual_inc
Loan Status

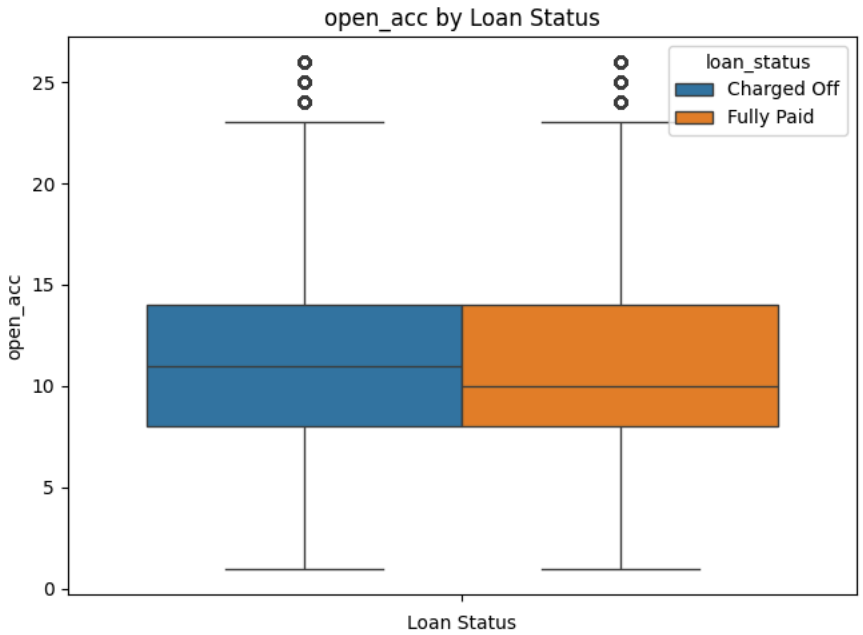

Distribution of dti by Loan Status
loan_status
Charged Off
Fully Paid

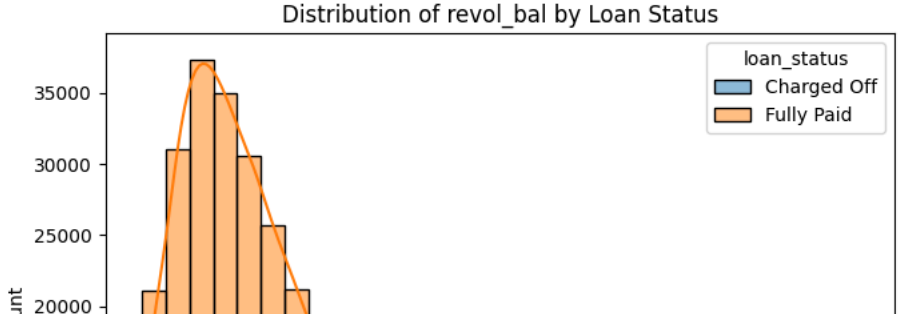

dti by Loan Status
loan_status
Charged Off
Fully Paid

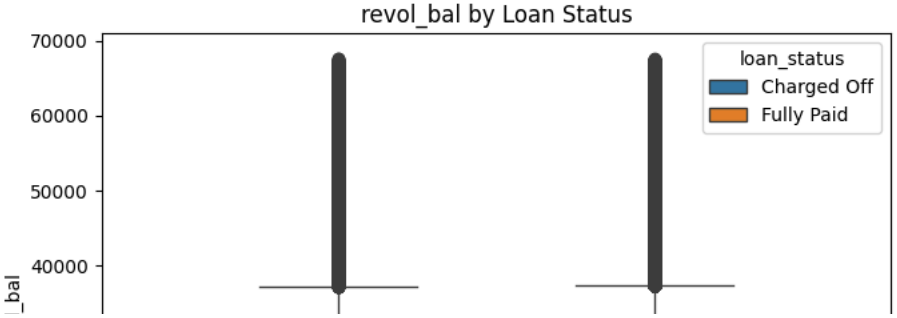

Count
dti

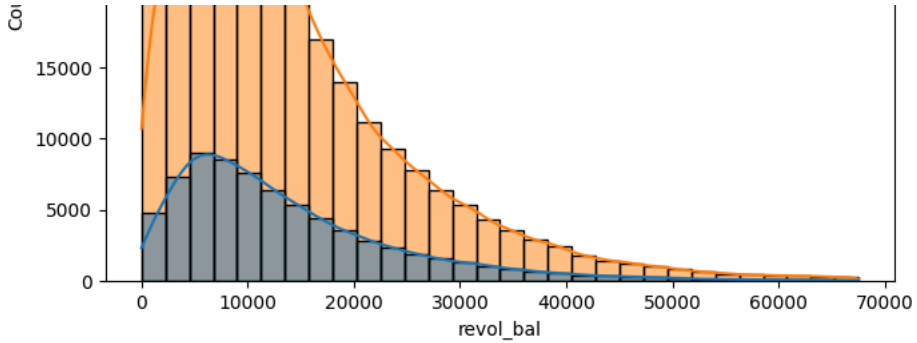

dti
Loan Status

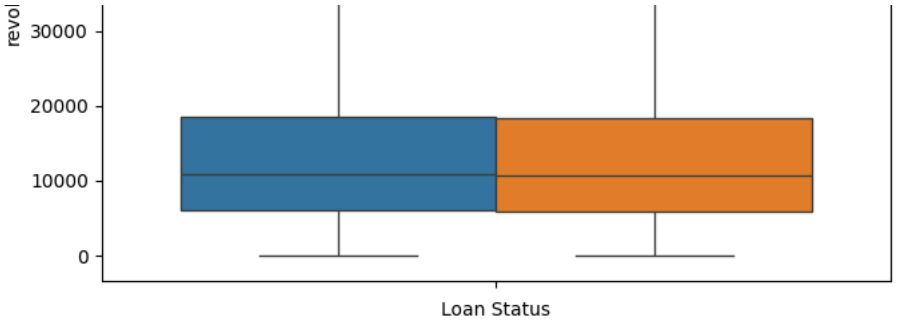

Distribution of open_acc by Loan Status
loan_status
Charged Off
Fully Paid
Count
open_acc

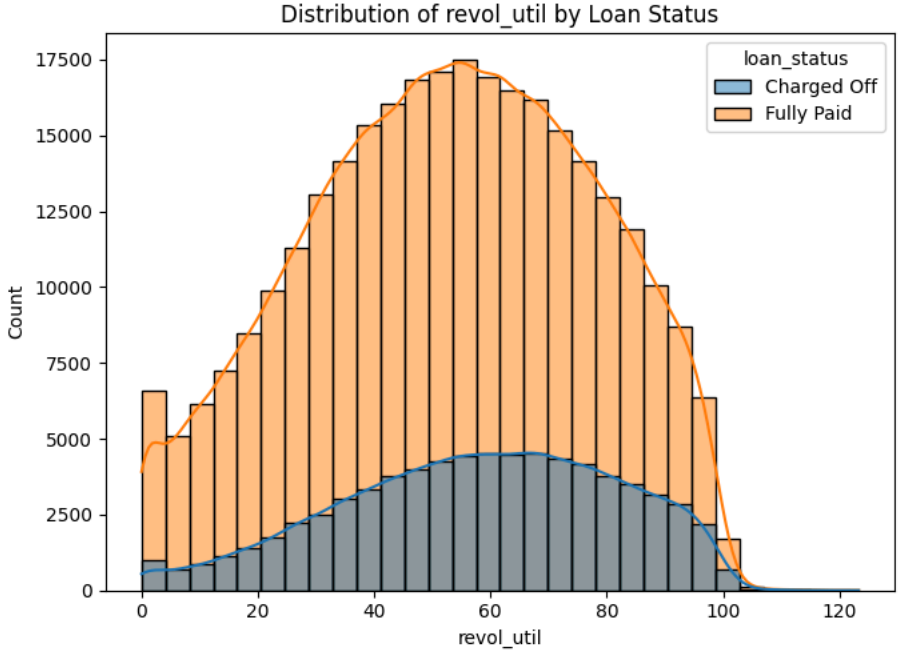

open_acc by Loan Status
loan_status
Charged Off
Fully Paid
open_acc
Loan Status

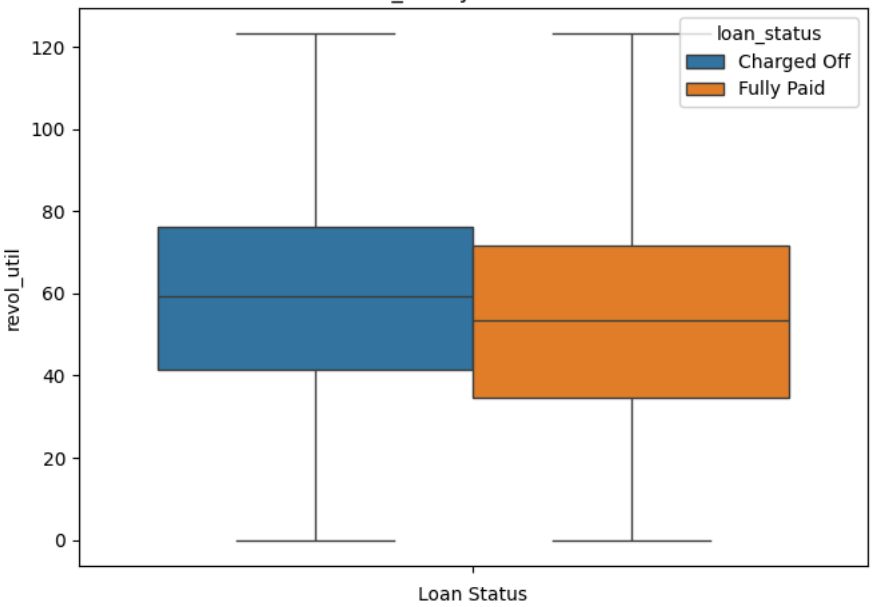

Distribution of revol_bal by Loan Status
loan_status
Charged Off
Fully Paid

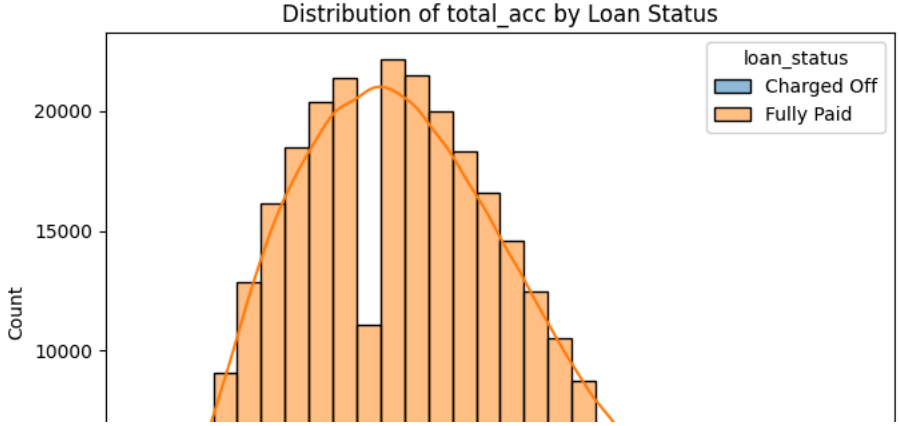

revol_bal by Loan Status
loan_status
Charged Off
Fully Paid

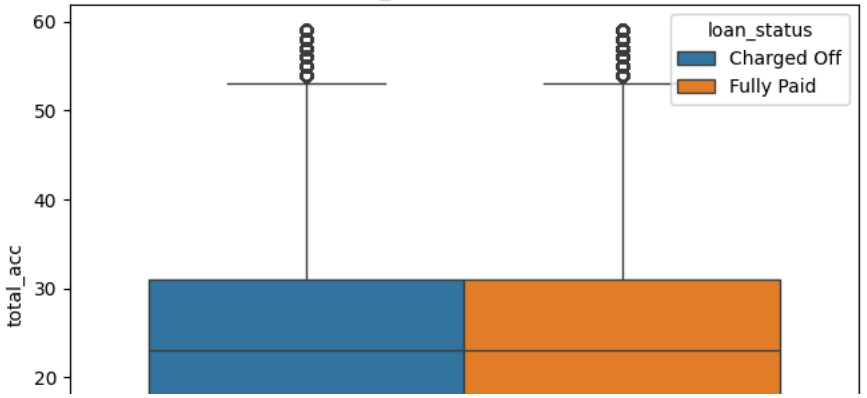

revol_bal

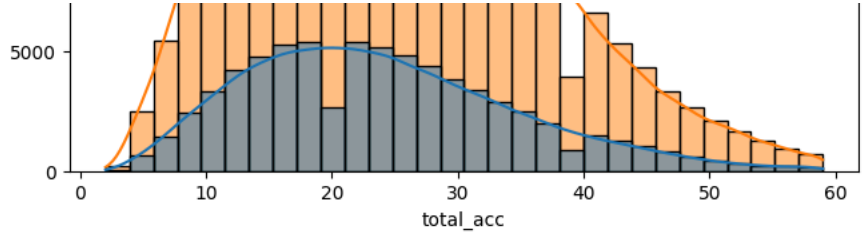

Loan Status

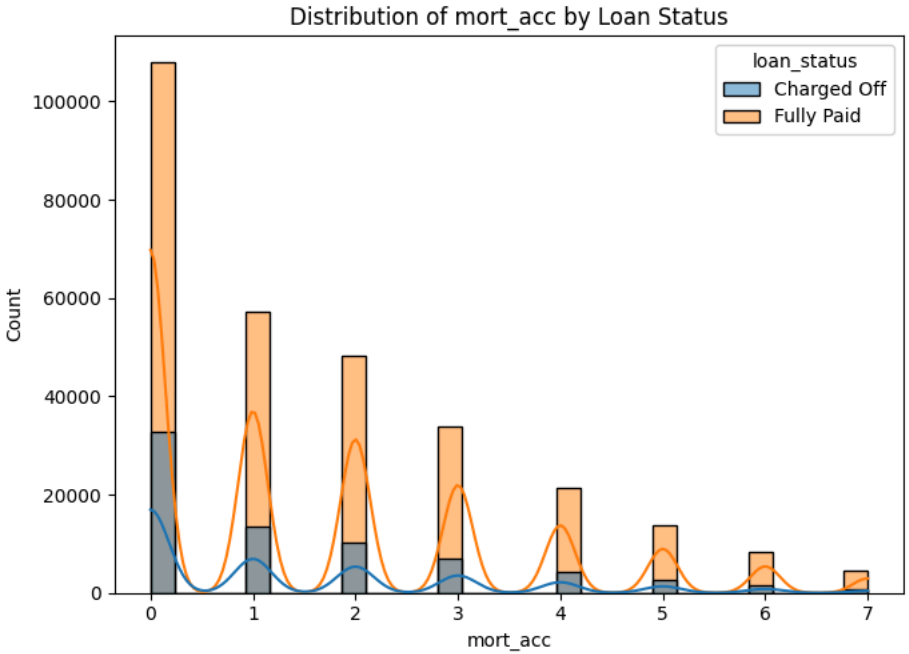

Distribution of revol_util by Loan Status
loan_status
Charged Off
Fully Paid
Count
revol_util

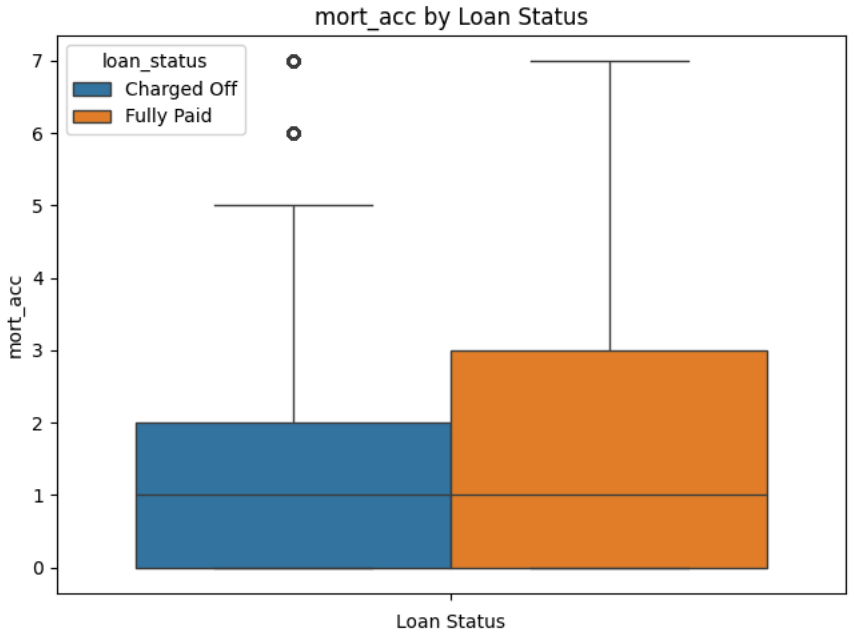

revol_util by Loan Status
loan_status
Charged Off
Fully Paid
revol_util
Loan Status


Distribution of total_acc by Loan Status
loan_status
Charged Off
Fully Paid
Count


total_acc by Loan Status
loan_status
Charged Off
Fully Paid
total_acc







Observation:

- From the boxplots, it can be observed that the mean loan_amnt, int_rate, dti, open_acc and revol_util are slightly higher for defaulters while annual income is lower

Data Preprocessing

In [495…

```
# Remove columns which do not have an impact on loan_status
df.drop(columns=['initial_list_status','state',
```

```
                'emp_title', 'title','earliest_cr_line',
                'issue_d','sub_grade'], inplace=True)

# Subgrade is removed because grade and subgrade are similar features
df['zip_code']=df['zip_code'].astype('category')
```

In [496…
```
df.columns
```

Out[496]:
```
Index(['loan_amnt', 'term', 'int_rate', 'grade', 'emp_length',
       'home_ownership', 'annual_inc', 'verification_status', 'loan_status',
       'purpose', 'dti', 'open_acc', 'pub_rec', 'revol_bal', 'revol_util',
       'total_acc', 'application_type', 'mort_acc', 'pub_rec_bankruptcies',
       'zip_code'],
      dtype='object')
```

Encoding the categorical variables

In [497…
```
# Encoding Target Variable
df['loan_status']=df['loan_status'].map({'Fully Paid': 0, 'Charged Off':1}).astype(int)
```

seperating features and target variables

In [498…
```
x = df.drop(columns=['loan_status'])
x.reset_index(inplace=True, drop=True)
y = df['loan_status']
y.reset_index(drop=True, inplace=True)
```

In [499…
```
# Encoding Binary features into numerical dtype
x['term']=x['term'].map({' 36 months': 36, ' 60 months':60}).astype(int)
```

In [500…
```
# Select categorical columns
cat_cols = x.select_dtypes('category').columns

# Initialize OneHotEncoder with the correct parameter
encoder = OneHotEncoder(sparse_output=False)  # Use sparse_output instead of sparse

# Fit and transform the categorical columns
encoded_data = encoder.fit_transform(x[cat_cols])
```

```
# Create a DataFrame for the encoded data
encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_out(cat_cols))

# Concatenate the encoded DataFrame with the original DataFrame
x = pd.concat([x, encoded_df], axis=1)

# Drop the original categorical columns
x.drop(columns=cat_cols, inplace=True)

# Display the first few rows of the updated DataFrame
x.head()
```

Out[500]:

	loan_amnt	term	int_rate	emp_length	annual_inc	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	mort_acc	pub_rec_bankrup
0	10000.0	36	11.44	10.0	117000.0	26.24	16.0	0	36369.0	41.8	25.0	0.0	
1	8000.0	36	11.99	4.0	65000.0	22.05	17.0	0	20131.0	53.3	27.0	3.0	
2	15600.0	36	10.49	0.0	43057.0	12.79	13.0	0	11987.0	92.2	26.0	0.0	
3	7200.0	36	6.49	6.0	54000.0	2.60	6.0	0	5472.0	21.5	13.0	0.0	
4	24375.0	60	17.27	9.0	55000.0	33.95	13.0	0	24584.0	69.8	43.0	1.0	

Performing the test train split

In [501…
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.20,stratify=y,random_state=42)
```

In [502…
```
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

Out[502]:
```
((294463, 56), (294463,), (73616, 56), (73616,))
```

Feature Scaling

In [503…
```
scaler = MinMaxScaler()
x_train = pd.DataFrame(scaler.fit_transform(x_train), columns=x_train.columns)
x_test = pd.DataFrame(scaler.transform(x_test), columns=x_test.columns)
```

In [504…
```
x_train.tail()
```

Out[504]:

	loan_amnt	term	int_rate	emp_length	annual_inc	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	mort_acc	pub_r
294458	0.435813	1.0	0.700046	0.2	0.175148	0.248739	0.24	0.0	0.233164	0.705596	0.263158	0.285714	
294459	0.302831	0.0	0.313798	0.8	0.185214	0.324370	0.56	0.0	0.226103	0.202758	0.368421	0.000000	
294460	0.381830	0.0	0.154592	0.2	0.159767	0.223529	0.12	0.0	0.117411	0.357664	0.070175	0.000000	
294461	0.160632	0.0	0.399631	0.7	0.120792	0.437955	0.24	0.0	0.258331	0.557178	0.140351	0.000000	
294462	0.092166	0.0	0.307799	0.5	0.237570	0.186975	1.00	1.0	0.178197	0.557989	0.807018	0.000000	

In [505…
```
y_train.value_counts()
```

Out[505]:

loan_status	count
0	236416
1	58047

dtype: int64

- Since the data is imbalanced in nature we can balance out this by using SMOTE method before model building

Over sampling using SMOTE

In [506…
```
sm = SMOTE(random_state=42)
x_train_res, y_train_res = sm.fit_resample(x_train,y_train.ravel())
print(f"Before OverSampling, count of label 1: {sum(y_train == 1)}")
print(f"Before OverSampling, count of label 0: {sum(y_train == 0)}")
print(f"After OverSampling, count of label 1: {sum(y_train_res == 1)}")
print(f"After OverSampling, count of label 0: {sum(y_train_res == 0)}")
```

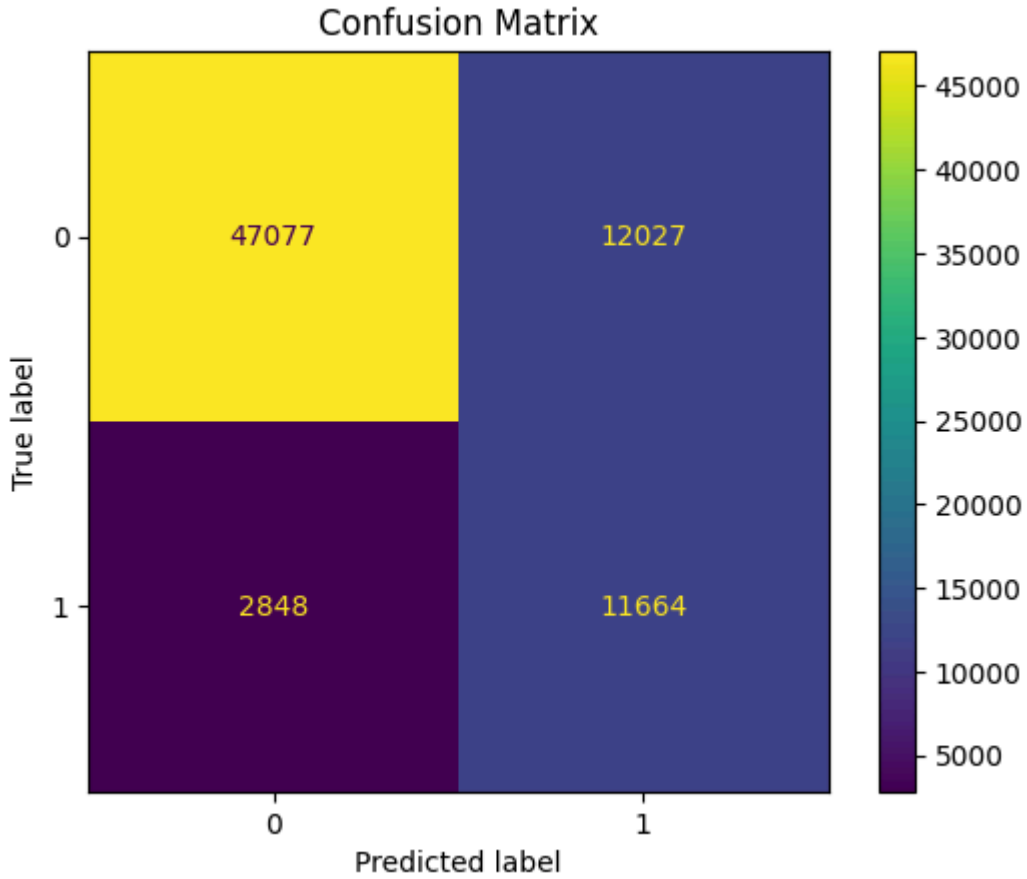
```
Before OverSampling, count of label 1: 58047
Before OverSampling, count of label 0: 236416
After OverSampling, count of label 1: 236416
After OverSampling, count of label 0: 236416
```

Model building using Logistic Regression

In [507…

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, recall_score, precision_score, confusion_matrix, ConfusionMatrixDisplay

# Train the model on resampled data
model = LogisticRegression()
model.fit(x_train_res, y_train_res)

# Predictions for the resampled training set and test set
train_preds = model.predict(x_train)
test_preds = model.predict(x_test)

# Model evaluation on the resampled training set
print('Train Accuracy :', round(model.score(x_train, y_train), 2))
print('Train F1 Score:', round(f1_score(y_train,train_preds), 2))
print('Train Recall Score:', round(recall_score(y_train,train_preds), 2))
print('Train Precision Score:', round(precision_score(y_train, train_preds), 2))

print('\nTest Accuracy :',round(model.score(x_test, y_test), 2))
print('Test F1 Score:',round(f1_score(y_test,test_preds), 2))
print('Test Recall Score:',round(recall_score(y_test,test_preds), 2))
print('Test Precision Score:',round(precision_score(y_test, test_preds), 2))


# Confusion Matrix for the test set
cm = confusion_matrix(y_test, test_preds)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title('Confusion Matrix')
plt.show()
```

```
Train Accuracy : 0.8
Train F1 Score: 0.61
Train Recall Score: 0.81
Train Precision Score: 0.5

Test Accuracy : 0.8
Test F1 Score: 0.61
Test Recall Score: 0.8
Test Precision Score: 0.49
```



Observation:

- Model Performance: The model performs well in correctly classifying true positives (47077) and true negatives (11664), as these numbers are significantly higher than the false positives (12027) and false negatives (2848).

In [508…
```
print(classification_report(y_test, test_preds))
```

```
              precision    recall  f1-score   support

           0       0.94      0.80      0.86     59104
           1       0.49      0.80      0.61     14512

    accuracy                           0.80     73616
   macro avg       0.72      0.80      0.74     73616
weighted avg       0.85      0.80      0.81     73616
```

Observations on Class Imbalance, Precision Disparity, Consistent Recall, F1 Score Differnece:

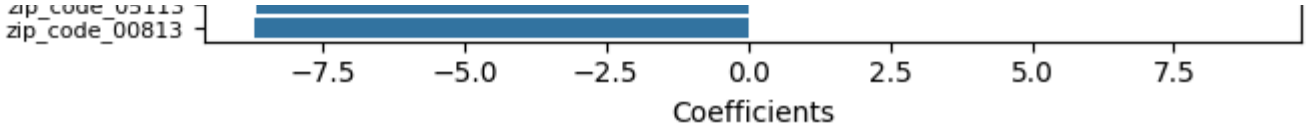
- Class Imbalance: There's a significant imbalance in the dataset, with class 0 having much higher support (59104) compared to class 1 (14512).
- Precision Disparity: Class 0 has very high precision (0.94) while class 1 has lower precision (0.49), indicating the model is more accurate in predicting class 0.
- Consistent Recall: Both classes have the same recall (0.80), suggesting the model is equally good at identifying positive instances of both classes.
- F1-Score Difference: Class 0 has a higher f1-score (0.86) compared to class 1 (0.61), indicating better overall performance for class 0. Overall Accuracy: The model achieves an accuracy of 0.80 across 73616 total instances.

Feature Importance

In [509…
```
feature_imp = pd.DataFrame({'Columns':x_train.columns, 'Coefficients':model.coef_[0]}).round(2).sort_values('Coefficie

plt.figure(figsize=(8,8))
sns.barplot(y = feature_imp['Columns'],
            x =  feature_imp['Coefficients'])
plt.title("Feature Importance for Model")
```

```
plt.yticks(fontsize=8)
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```


Feature Importance for Model
Feature
zip_code_11650
zip_code_93700
zip_code_86630
dti
open_acc
loan_amnt
revol_util
purpose_small_business
application_type_INDIVIDUAL
term
grade_G
grade_F
int_rate
grade_E
purpose_home_improvement
purpose_other
purpose_debt_consolidation
grade_D
purpose_medical
purpose_major_purchase
pub_rec
purpose_credit_card
purpose_moving
home_ownership_NONE
emp_length
home_ownership_ANY
home_ownership_RENT
verification_status_Source Verified
pub_rec_bankruptcies
purpose_educational
home_ownership_OTHER
purpose_renewable_energy
grade_C
home_ownership_OWN
purpose_vacation
verification_status_Verified
zip_code_48052
application_type_DIRECT_PAY
home_ownership_MORTGAGE
zip_code_22690
zip_code_70466
mort_acc
zip_code_30723
verification_status_Not Verified
purpose_house
purpose_car
grade_B
revol_bal
application_type_JOINT
total_acc
purpose_wedding
grade_A
annual_inc
zip_code_29597



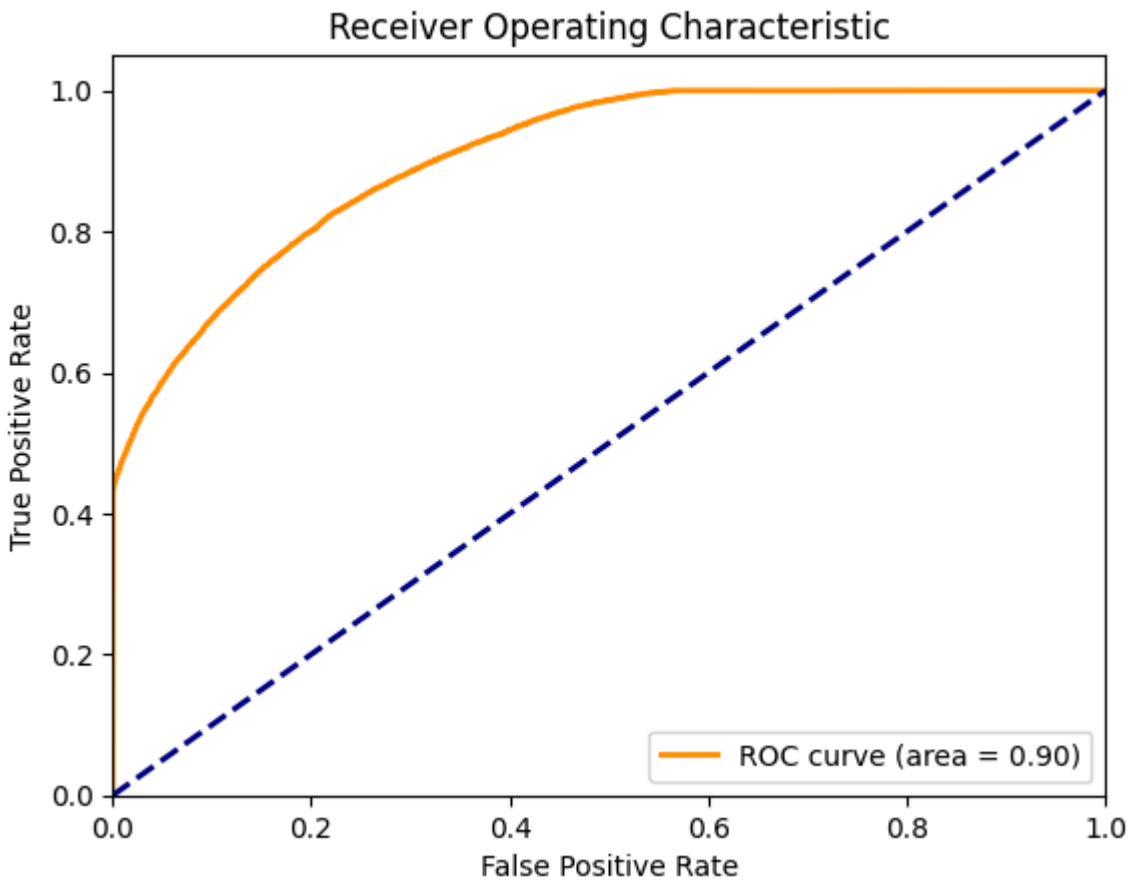
Observations on Feature Importance:

- Positive Features: The model has assigned large weightage to:
  - Zip code features (zip_code_11500, zip_code_33700, and zip_code_36610)
  - Debt-to-income ratio (dti)
  - Open accounts (open_acc)
  - Loan amount (loan_amnt)
- Negative Features: The model has assigned large negative coefficients to:
  - A few zip codes
  - Annual income (annual_inc)
  - Joint application type
- Model is heavily influenced by geographic location (zip codes), debt-to-income ratio, and loan amount, while also considering annual income and joint application type as important factors

In [510…

```
# Predict probabilities for the test set
probs = model.predict_proba(x_test)[:,1]

# Compute the false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, probs)

# Compute the area under the ROC curve
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

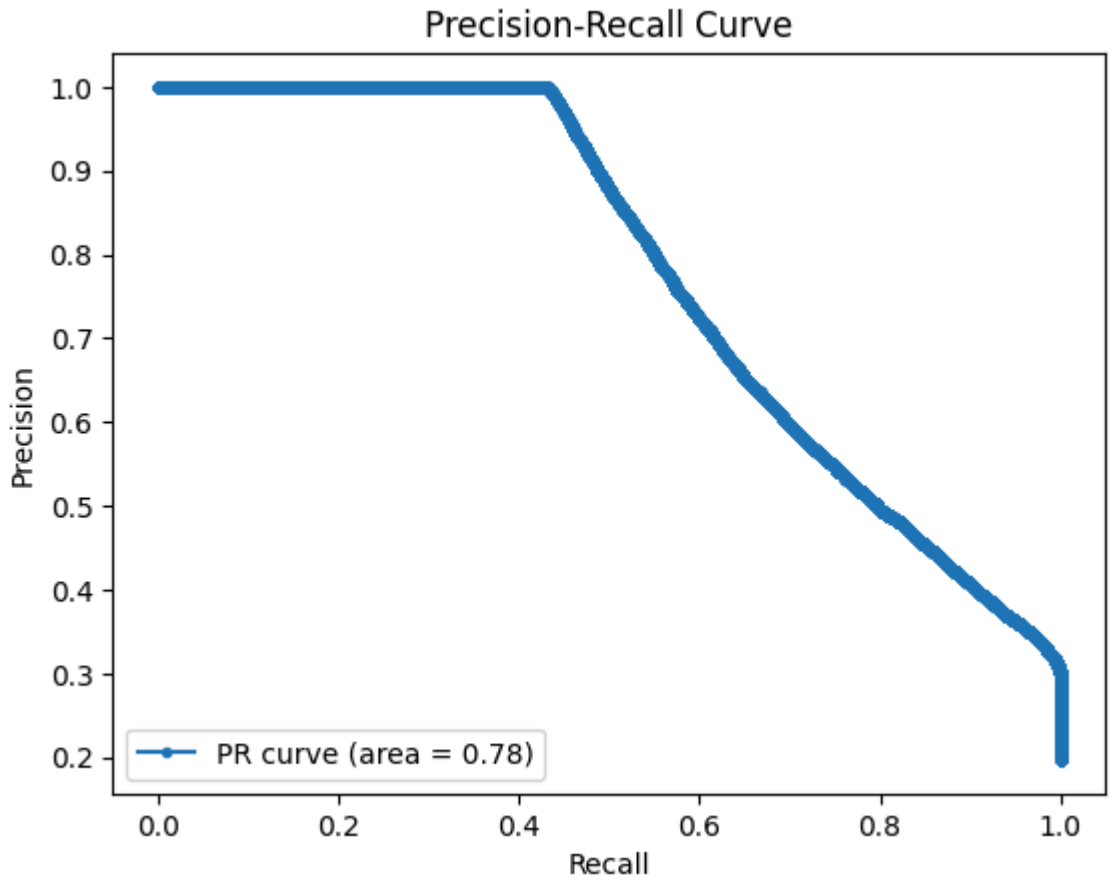


Observations on ROC - AUC

- High AUC: The area under the ROC curve is 0.90, indicating excellent model performance.
- Curve shape: The ROC curve rises steeply at first and then flattens out, suggesting the model has a good ability to distinguish between classes, especially at lower false positive rates.
- Reference line: The dashed blue diagonal line represents the performance of a random classifier, and the significant distance between this line and the ROC curve visually demonstrates the model's predictive power.

In [511…

```
# Compute the false precision and recall at all thresholds
precision, recall, thresholds = precision_recall_curve(y_test, probs)

# Area under Precision Recall Curve
auprc = average_precision_score(y_test, probs)

# Plot the precision-recall curve
plt.plot(recall, precision, marker='.', label='PR curve (area = %0.2f)' % auprc)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="lower left")
plt.show()
```



Observations on Precision and Recall:

- Good at Top Predictions: The model is very accurate (100% precision) for a significant portion of its predictions (up to 40% recall).
- Performance Drops Off: After a certain point (40% recall), the model's accuracy drops sharply, indicating a trade-off between precision and recall.

Insights:

- Geographic location matters: The model is heavily influenced by zip codes, indicating that location plays a significant role in determining creditworthiness.
- Debt-to-income ratio is crucial: The model assigns large weightage to debt-to-income ratio, suggesting that borrowers with high debt burdens are more likely to default.
- Loan amount and open accounts are important: The model considers loan amount and open accounts as important factors in determining creditworthiness.
- Annual income and joint application type are negatively correlated: The model assigns large negative coefficients to annual income and joint application type, indicating that these factors are associated with a lower likelihood of default.
- Class imbalance is significant: The dataset has a significant class imbalance, with class 0 having much higher support than class 1.
- Model performance is good: The model achieves an accuracy of 0.80 and an AUC of 0.90, indicating good performance.

Recommendations:

- Use location-based data to inform lending decisions: Consider using location-based data, such as zip codes, to inform lending decisions and adjust interest rates or loan terms accordingly.
- Monitor debt-to-income ratio closely: Closely monitor debt-to-income ratio and consider implementing stricter lending criteria for borrowers with high debt burdens.
- Optimize loan amount and open accounts: Optimize loan amount and open accounts to minimize the risk of default.
- Consider alternative data sources: Consider using alternative data sources, such as social media or online behavior, to supplement traditional credit data and improve model performance.
- Address class imbalance: Address the class imbalance by oversampling the minority class, undersampling the majority class, or using class weighting techniques.
- Continuously monitor and update the model: Continuously monitor the model's performance and update it regularly to ensure that it remains accurate and effective.

Questionnaires Answered :

1. What percentage of customers have fully paid their Loan Amount?
    - From the data provided around 80.28% of customers have fully paid there loans and 19.71% have not paid there loans

1. Comment about the correlation between Loan Amount and Installment features.
    - Loan and Installment features are highly positively correlated (1.0) hence we can remove any of the feature (in this case study i have removed installments feature) to avoid multicolinearity.
2. The majority of people have home ownership as Mortgage.
3. People with grades ‘A’ are more likely to fully pay their loan.
    - **True**
4. The top 2 affordable job titles Teacher and Manager and top 2 reasons for taking up loan are debt consolidation and credit card refinancing
5. Thinking from a bank's perspective, which metric should our primary focus be on
    - From a bank's perspective, the primary focus should be on Recall.
    - Recall measures the proportion of actual defaulters that are correctly identified by the model. In other words, it measures the bank's ability to detect and flag potential defaulters.
    - A high recall is crucial for a bank because it wants to minimize the number of false negatives (i.e., actual defaulters that are not flagged by the model). By focusing on recall, the bank can ensure that it is identifying and managing potential credit risks effectively, which is essential for maintaining a healthy loan portfolio and minimizing losses.
    - While precision is also important, a bank may be willing to tolerate some false positives (i.e., non-defaulters flagged as defaulters) in order to ensure that it is catching as many actual defaulters as possible. Therefore, recall is the more critical metric for a bank to focus on.
6. How does the gap in precision and recall affect the bank?
    - Low Precision (50%):
    - The bank may end up denying loans to deserving customers (false positives), which can lead to: Loss of potential revenue Damage to customer relationships and reputation Potential legal issues if the denial is perceived as discriminatory
    - High Recall (80%): The bank is able to identify and flag a large proportion of actual defaulters, which can help: Minimize losses due to default Maintain a healthy loan portfolio Reduce the risk of lending to high-risk borrowers

- However, the trade-off is that the bank may also flag some non-defaulters as defaulters (false positives), which can lead to the issues mentioned earlier.
- Overall, the bank needs to strike a balance between precision and recall to minimize losses while also ensuring that deserving customers are not unfairly denied loans.

7. Which were the features that heavily affected the outcome?

- Zip Code: The model assigned large weightage to zip code features, indicating that location plays a significant role in determining the likelihood of default.
- DTI (Debt-to-Income Ratio): The model also assigned significant weightage to DTI, suggesting that a borrower's debt burden is an important factor in determining their creditworthiness.
- Open Accounts: The number of open accounts was another feature that had a significant impact on the outcome, indicating that borrowers with more open accounts may be more likely to default.
- Loan Amount: The loan amount was also an important feature, suggesting that larger loan amounts may be associated with a higher risk of default.

8. Will the results be affected by geographical location?

- **(Yes)**