Problem Statement

LoanTap, is a company providing loan products to MSME (Micro, Small and Medium Enterprises) businessmen and salaried individuals. It has a novel way of making the rather dull and complex loan disbursal process more consumer-friendly and flexible and efficient. The data science team at LoanTap is building an underwriting layer to determine the credit eligibility of MSMEs as well as individuals. LoanTap deploys loans for 4 main purposes: Personal Loan, EMI Free Loan, Personal Overdraft, and Advance Salary Loan. This case study will focus on the underwriting process behind Personal Loan only.

Given a set of attributes for an Individual, we need to determine if a credit line should be extended to them and recommend any repayment business terms.

The data columns are descibed as follows:

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

In [2]:
```
# useful imports
import numpy as np, seaborn as sns, pandas as pd, matplotlib.pyplot as plt, scipy
```

In [91]:
```
df = pd.read_csv('logistic_regression_data.csv')
df.head()
```

Out[91]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annua
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	117(
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65(
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RENT	43(
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	RENT	54(
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55(

5 rows × 27 columns

In [4]:
```
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                 394275 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
```

The numerical columns are:

In [4]:
```
df.columns[df.dtypes == "float64"]
```

Out[4]:
```
Index(['loan_amnt', 'int_rate', 'installment', 'annual_inc', 'dti', 'open_acc',
       'pub_rec', 'revol_bal', 'revol_util', 'total_acc', 'mort_acc',
       'pub_rec_bankruptcies'],
      dtype='object')
```

The categorical columns are:

In [5]:
```
df.columns[df.dtypes =="object"]
```

Out[5]:
```
Index(['term', 'grade', 'sub_grade', 'emp_title', 'emp_length',
       'home_ownership', 'verification_status', 'issue_d', 'loan_status',
       'purpose', 'title', 'earliest_cr_line', 'initial_list_status',
       'application_type', 'address'],
      dtype='object')
```

In [6]:
```
df.shape
```

Out[6]:
```
(396030, 27)
```

Shape is 3,96,030 rows and 27 columns

checking for null values

There are many missing values in employment titles, employment lengths, loan titles provided by borrower, revolving line utilization rate, number of mortgage accounts, and number of public record bankruptcies. Since they are less than 10% of the total entries, we can impute them rather than removing the columns.

In [7]:
```
df.isna().sum()*100/len(df)
```

Out[7]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
installment             0.000000
grade                   0.000000
sub_grade               0.000000
emp_title               5.789208
emp_length              4.621115
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.443148
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.069692
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                9.543469
pub_rec_bankruptcies    0.135091
address                 0.000000
dtype: float64
```

checking for duplicated values

There are no duplicate entries.

In [8]:
```
df.duplicated().sum()
```

Out[8]:
```
0
```

In [9]:
```
df.describe(include = "float64") # numerical features
```

Out[9]:

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec
count	396030.000000	396030.000000	396030.000000	3.960300e+05	396030.000000	396030.000000	396030.000000
mean	14113.888089	13.639400	431.849698	7.420318e+04	17.379514	11.311153	0.178191
std	8357.441341	4.472157	250.727790	6.163762e+04	18.019092	5.137649	0.530671
min	500.000000	5.320000	16.080000	0.000000e+00	0.000000	0.000000	0.000000
25%	8000.000000	10.490000	250.330000	4.500000e+04	11.280000	8.000000	0.000000
50%	12000.000000	13.330000	375.430000	6.400000e+04	16.910000	10.000000	0.000000
75%	20000.000000	16.490000	567.300000	9.000000e+04	22.980000	14.000000	0.000000
max	40000.000000	30.990000	1533.810000	8.706582e+06	9999.000000	90.000000	86.000000

Loan amount ranges from Rs 500 to Rs 40,000. Mean amount Rs. 14113.89 is not close to median Rs. 12000, this hints at outliers.

Interest rate ranges from 5.32% to 30.99%, with mean being 13.64% close to median 13.33%, there may not be many outliers.

Installment amount ranges from Rs. 16.08 to Rs. 1533.81. Mean Rs. 431.85 is far from median Rs. 375.43, so there are outliers.

Annual income ranges from 0 to Rs. 87,06,582. Mean income being Rs. 74,203.18 and median Rs. 64,000 so there are outliers.

Debt to income ratio(dti) ranges from 0 to 9999 and mean is 11.31. Max 9999 is so far away from median 10, so there may be some outliers.

Open accounts (or number of credit lines) ranges from 0 to 90 accounts. Mean is 11.31 accounts and median is 10 accounts. Since the max value 90 is so much higher than 75% percentile 14, there may be outliers.

Number of derogatory public records range from 0 to 86, mean is 0.18. Since max 86 is so far away from median 0, there are outliers present.

Total credit revolving balance ranges from 0 to Rs. 17,43,266, mean at Rs. 15,844.54 and median is far away at Rs. 11,181, so there may be outliers.

Revolving line utilization rate ranges from 0 to 892.3, mean is 53.79 and median is close at 54.8.

Total number of credit line accounts range from 2 to 151, mean at 25.4 and median at 24.

Number of mortgage accounts range from 0 to 34, mean is 1.81 and median is 1 much smaller than max value so there maybe outliers.

Number of public record bankruptcies range from 0 to 8, mean is 0.12 and median 0. There is wide distance from median to max number of credit lines so there may be outliers.

In [10]:

```
df.describe(include = "object")
```

Out[10]:

	term	grade	sub_grade	emp_title	emp_length	home_ownership	verification_status	issue_d	loan_sta
count	396030	396030	396030	373103	377729	396030	396030	396030	3960
unique	2	7	35	173105	11	6	3	115	
top	36 months	B	B3	Teacher	10+ years	MORTGAGE	Verified	Oct-2014	Fully P
freq	302005	116018	26655	4389	126041	198348	139563	14846	3183

Term of loan has 2 values of which 36 months is the most frequently occurring at 3,02,005 times.

Loan grade has 7 unique values, with B being most common.

Loan sub grade has 35 unique values, with B3 being the most common.

Employment title has 1,73,105 values, with Teacher being the most common title.

Employment length has 11 unique values, with 10+ years being most common.

Home ownership has 6 categories with Mortgage ownership as most common.

Verification status has 3 categories, with verified as most common.

Issue date has 115 dates, with Oct-2014 as most commonly occurring at 14,846 times.

Loan status has 2 types with fully paid as most common.

Purpose has 14 unique values, with debt consolidation as most common loan purpose.

Loan title has 48,817 unique values. Debt consolidation is the most common loan title.

Earliest credit line has 684 unique values, with Oct-2000 being when most people opened their first credit line.

Initial list status has 2 types, with F type as most common.

Application type has 3 types, with INDIVIDUAL as most common.

Address column has 393700 unique addresses, with USCGC Smith\r\nFPOAE 70466 most common at 8 times frequency.

Univariate analysis

In [205…

```
continuous_cols = df.columns[df.dtypes != 'object']
continuous_cols
```

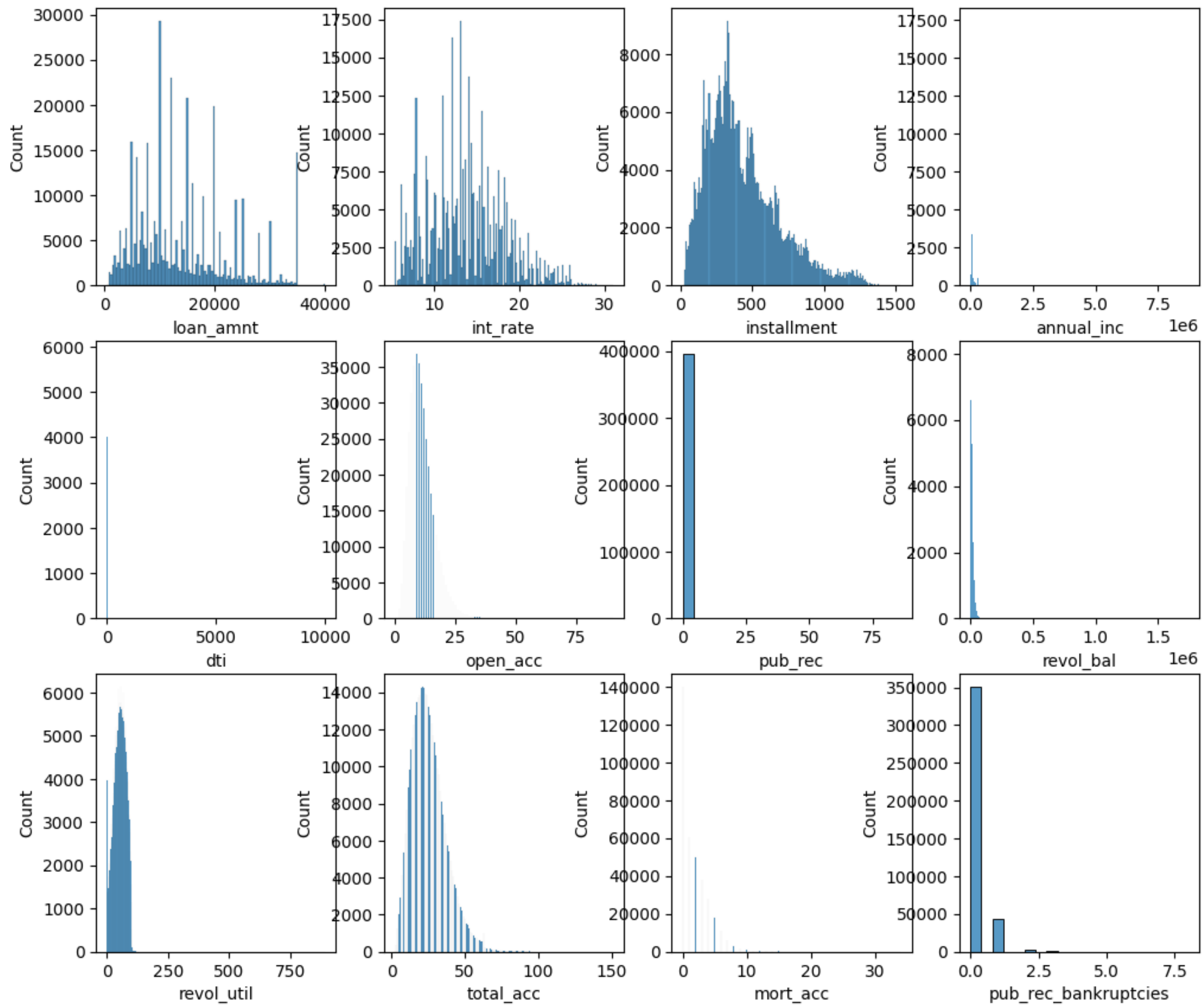
Out[205]:

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

In [12]:

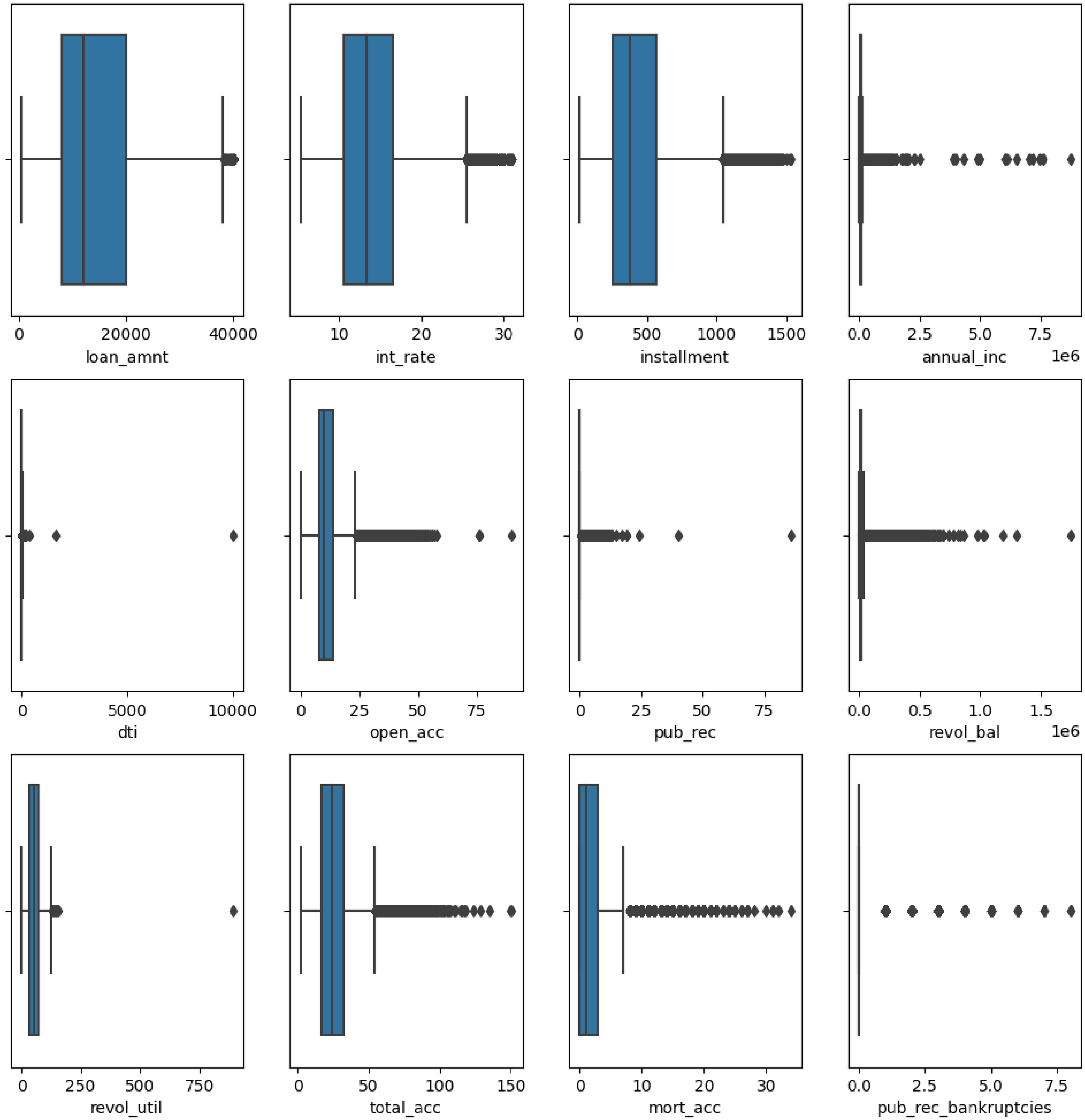
```
f = plt.figure()
f.set_figwidth(12)
f.set_figheight(14)
n = len(continuous_cols)
```

```
for i in range(n):
    plt.subplot(4,(n//4)+1,i+1)
    sns.histplot(data=df, x=continuous_cols[i])
plt.show()
```



Most of these look right skewed because of presence of outliers. If we remove the outliers, then we can get bell shaped curves.

In [29]:
```
f = plt.figure()
f.set_figwidth(12)
f.set_figheight(12)
n = len(continuous_cols)

for i in range(n):
    plt.subplot(3,4,i+1)
    sns.boxplot(data=df, x=continuous_cols[i])
plt.show()
```



In [38]:
```
# We'll use IQR (inter-quartile range proximity) outlier detection method for skewed distributio
# 3 times standard deviation rule for normal distributions. This is because most of the numerica
for i in range(len(continuous_cols)):
    iqr = scipy.stats.iqr(df[continuous_cols[i]])
    q3 = np.percentile(df[continuous_cols[i]],75)
    out = df[continuous_cols[i]][df[continuous_cols[i]] > (q3 + iqr*1.5)]
    ratio = round(len(out)*100/len(df[continuous_cols[i]]),2)
    print(f"The percentage of outliers in {continuous_cols[i]} are {ratio}%")
```

```
The percentage of outliers in loan_amnt are 0.05%
The percentage of outliers in int_rate are 0.95%
The percentage of outliers in installment are 2.84%
The percentage of outliers in annual_inc are 4.22%
The percentage of outliers in dti are 0.07%
The percentage of outliers in open_acc are 2.6%
The percentage of outliers in pub_rec are 14.58%
The percentage of outliers in revol_bal are 5.37%
The percentage of outliers in revol_util are 0.0%
The percentage of outliers in total_acc are 2.15%
The percentage of outliers in mort_acc are 0.0%
The percentage of outliers in pub_rec_bankruptcies are 0.0%
```

In [9]:

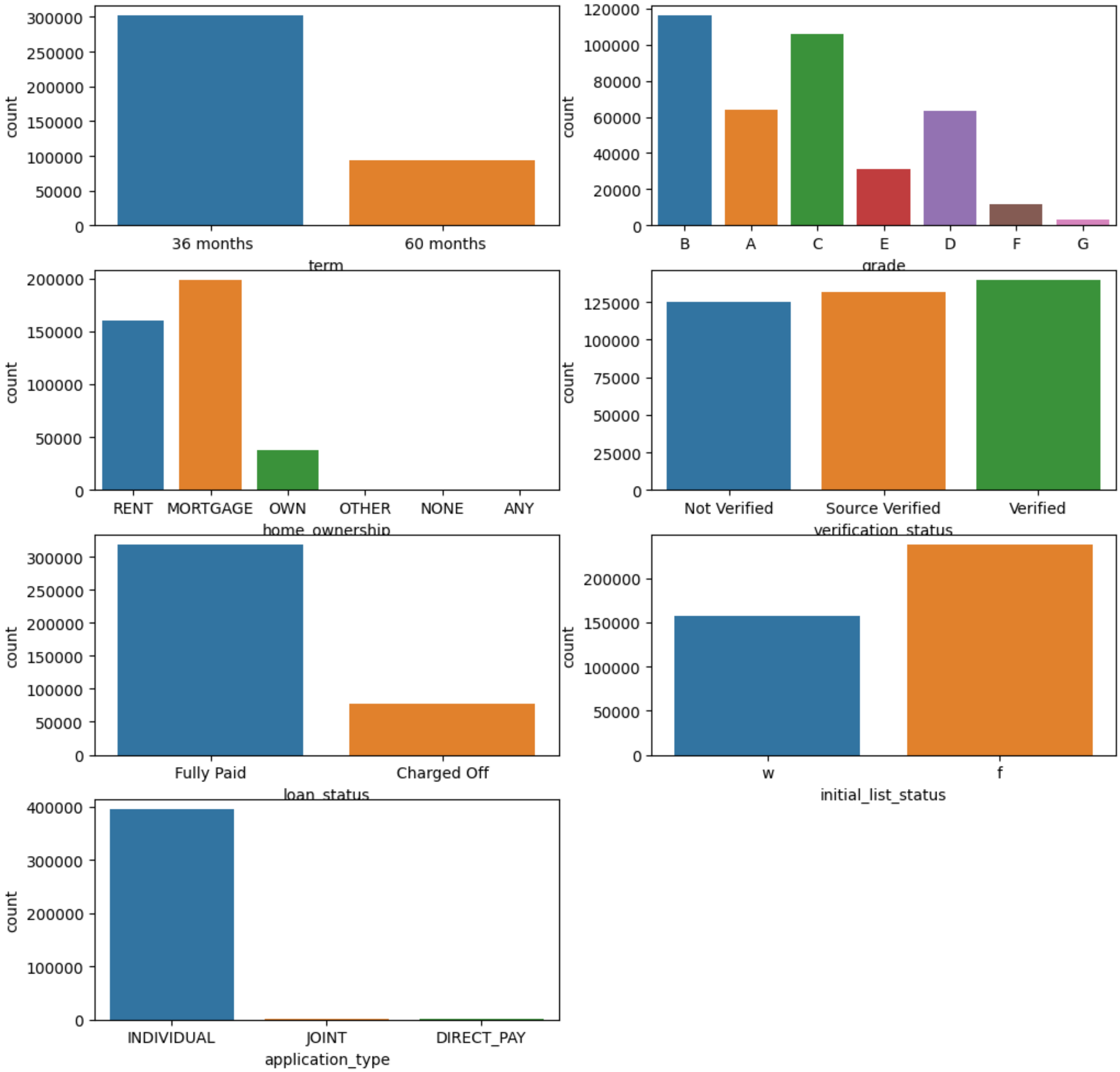
```
categorical_cols = df.columns[df.dtypes == "object"]
trim_categorical_cols = categorical_cols.drop(['sub_grade','emp_title','emp_length','issue_d','pu
                                               'title','earliest_cr_line','address'])
trim_categorical_cols
```

Out[9]:

```
Index(['term', 'grade', 'home_ownership', 'verification_status', 'loan_status',
       'initial_list_status', 'application_type'],
      dtype='object')
```

In [24]:

```
n = len(trim_categorical_cols)

f = plt.figure()
f.set_figwidth(12)
f.set_figheight(12)
for i in range(n):
    plt.subplot(4,2,i+1)
    sns.countplot(data=df, x= trim_categorical_cols[i])
plt.show()
```



Term of loan has 2 values of which 36 months is the most frequently occurring.

Loan grade has 7 unique values, with B being most common.

Verification status has 3 categories, with verified as most common.

Initial list status has 2 types, with F type as most common.

Application type has 3 types, with INDIVIDUAL as most common.

In [25]:
```
df['loan_status'].value_counts()*100/len(df)
```

Out[25]:
```
Fully Paid     80.387092
Charged Off    19.612908
Name: loan_status, dtype: float64
```

80.39% customers have paid their loans fully.

In [26]:
```
df['home_ownership'].value_counts()*100/len(df)
```

Out[26]:
```
MORTGAGE    50.084085
RENT        40.347953
OWN          9.531096
OTHER        0.028281
NONE         0.007828
ANY          0.000758
Name: home_ownership, dtype: float64
```

Most people have home_ownership as MORTAGE at about 50%. Next highest is RENT ~40%.

In [28]:
```
df['emp_title'].value_counts().head()*100/len(df)
```

Out[28]:
```
Teacher             1.108249
Manager             1.073151
Registered Nurse    0.468651
RN                  0.466126
Supervisor          0.462086
Name: emp_title, dtype: float64
```

Most common employment title is Teacher ~1.1% and then Manager ~1%.

In [176…
```
df['emp_length'].value_counts()
```

Out[176]:
```
10+ years    126041
2 years       35827
< 1 year      31725
3 years       31665
5 years       26495
1 year        25882
4 years       23952
6 years       20841
7 years       20819
8 years       19168
9 years       15314
Name: emp_length, dtype: int64
```

There are most number of customers with 10+ years of employment.

In [177…
```
df['purpose'].value_counts()
```

Out[177]:
```
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: purpose, dtype: int64
```

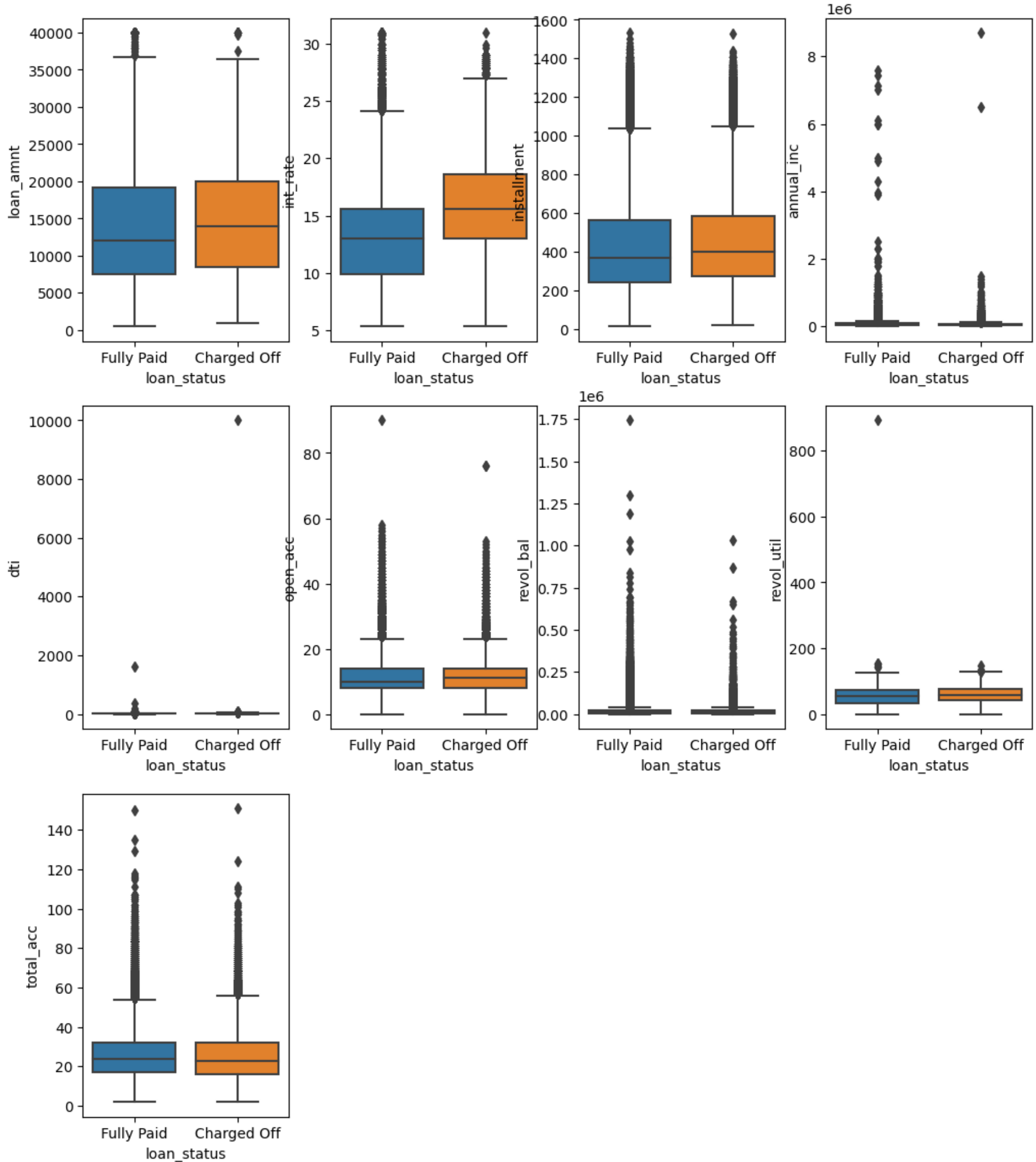
Most common purposes for getting loans were debt consolidation, credit card purchase, home improvement and other household or business purchases.

Now that we have checked the individual features, let's look at their relationship with each other.

Bivariate analysis

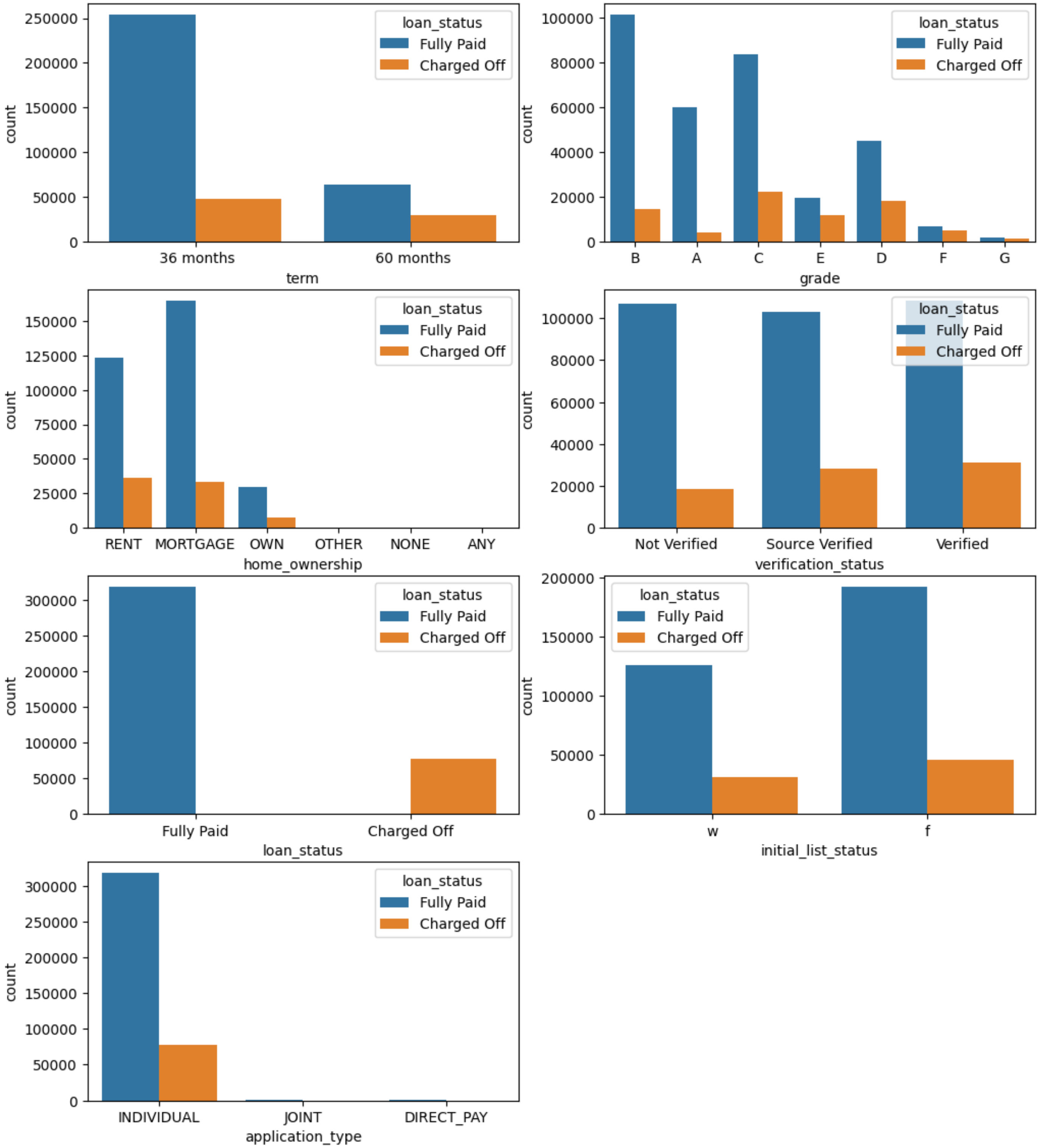
In [161…

```
f = plt.figure()
f.set_figwidth(12)
f.set_figheight(14)
n = len(continuous_cols)

for i in range(n):
    plt.subplot(3,4,i+1)
    sns.boxplot(data=df, y=continuous_cols[i],x='loan_status')
plt.show()
```



Only loan amounts and interest rates look higher for charged off customers than those who paid loans fully. Rest all features are similar for both groups.

In [11]:
```
f = plt.figure()
f.set_figwidth(12)
f.set_figheight(14)
n = len(trim_categorical_cols)

for i in range(n):
    plt.subplot(4,2,i+1)
    sns.countplot(data=df, x=trim_categorical_cols[i],hue='loan_status')
plt.show()
```



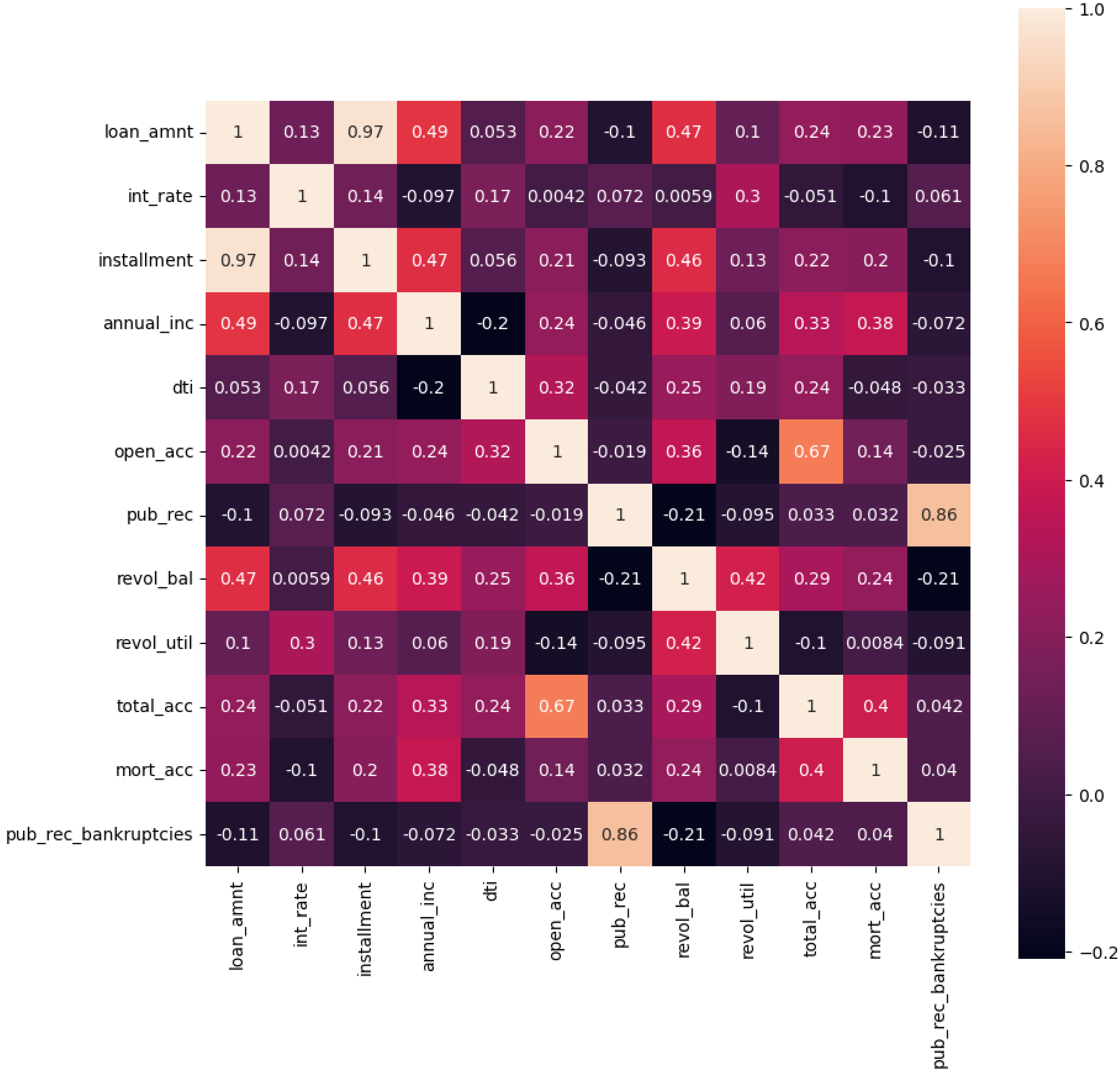
There are more customers who have fully paid their loans than those charged off in all categories.

Multivariate analysis

In [40]:
```
# Spearman's Rank Correlation Coefficient
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(method='spearman'), square=True,annot=True)
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_13772\2057271257.py:3: FutureWarning: The default va
lue of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to Fal
se. Select only valid columns or specify the value of numeric_only to silence this warning.
  sns.heatmap(df.corr(method='spearman'), square=True,annot=True)
```

Out[40]: <AxesSubplot: >



The loan amount and installment amounts are highly correlated (0.97) as per the spearman correlation coefficient as expected, because installment amounts are smaller portions of loan amount. This indicates high multicollinearity between these two features.

Number of derogatory public records are highly correlated to number of public record of bankruptcies.

Number of open credit line accounts are correlated to total number of credit accounts.

We need to remove either of the correlated variables from the pairs -(loan_amnt and installment) and (pub_rec and pub_rec_bankruptcies) in order to do regression.

In [193…
```
df = df.drop(['pub_rec_bankruptcies','installment'],axis=1)
```

Data Preprocessing

Simple Feature Engineering steps: E.g.: Creation of Flags- If value greater than 1.0 then 1 else 0. This can be done on:

1. Pub_rec
2. Mort_acc

Address column can be clipped to just use zipcode

In [92]:
```
maxele = df['pub_rec'].max()
df['pub_rec'] = pd.cut(df['pub_rec'], bins=[0,1,maxele], include_lowest=True, labels=[0,1])
maxele = df['mort_acc'].max()
df['mort_acc'] = pd.cut(df['mort_acc'], bins=[0,1,maxele], include_lowest=True, labels=[0,1])
```

In [93]:
```
df['pub_rec'].value_counts()
```

Out[93]:
```
0    388011
1      8019
Name: pub_rec, dtype: int64
```

In [94]:
```
df['mort_acc'].value_counts()
```

Out[94]:
```
0    200193
1    158042
Name: mort_acc, dtype: int64
```

In [96]:
```
df['issue_d'].head()
```

Out[96]:
```
0    Jan-2015
1    Jan-2015
2    Jan-2015
3    Nov-2014
4    Apr-2013
Name: issue_d, dtype: object
```

In [227…
```
df.duplicated().sum()
```

Out[227]:
```
0
```

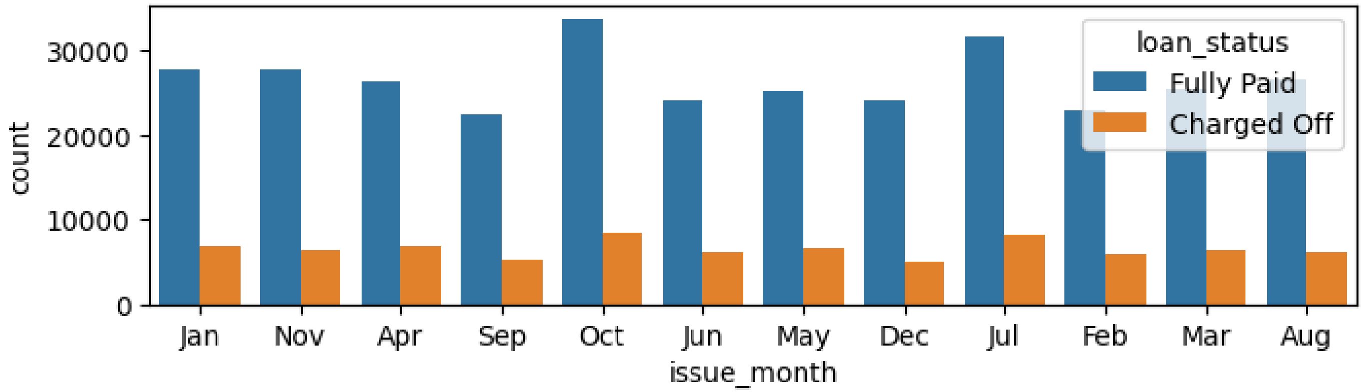
In [260…
```
df2 = df.join(df['issue_d'].str.split('-',1, expand=True).rename(columns={0:'issue_month', 1:'is
df2[['issue_d','issue_month','issue_year']].head()
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_23144\1797602395.py:1: FutureWarning: In a future ve
rsion of pandas all arguments of StringMethods.split except for the argument 'pat' will be keywo
rd-only.
  df2 = df.join(df['issue_d'].str.split('-',1, expand=True).rename(columns={0:'issue_month',
1:'issue_year'}))
```

Out[260]:

	issue_d	issue_month	issue_year
0	Jan-2015	Jan	2015
1	Jan-2015	Jan	2015
2	Jan-2015	Jan	2015
3	Nov-2014	Nov	2014
4	Apr-2013	Apr	2013

In [350…
```
f = plt.figure()
f.set_figwidth(8)
f.set_figheight(2)
sns.countplot(data=df2, x='issue_month',hue='loan_status')
plt.show()
```



There are more loans issued in the months of October and July, though not very different from other months.

In [261…

```
df2['zipcode'] = df2['address'].str[-5:]
df2['zipcode'].head()
```

Out[261]:

```
0    22690
1    05113
2    05113
3    00813
4    11650
Name: zipcode, dtype: object
```

In [262…

```
df2.drop(['address','issue_d'],axis=1,inplace=True)
df2.head()
```

Out[262]:

	loan_amnt	term	int_rate	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verifica
0	10000.0	36 months	11.44	B	B4	Marketing	10+ years	RENT	117000.0	
1	8000.0	36 months	11.99	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	
2	15600.0	36 months	10.49	B	B3	Statistician	< 1 year	RENT	43057.0	So
3	7200.0	36 months	6.49	A	A2	Client Advocate	6 years	RENT	54000.0	
4	24375.0	60 months	17.27	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000.0	

5 rows × 26 columns

Missing value treatment

In [263…

```
categ = df2[['emp_length','emp_title', 'term', 'title']].values
cont = df2[['revol_util', 'mort_acc']].values

# To calculate mean use imputer class
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer = imputer.fit(cont)
```

```
cont = imputer.transform(cont)
cont[:5]
```

Out[263]:
```
array([[41.8,  0. ],
       [53.3,  1. ],
       [92.2,  0. ],
       [21.5,  0. ],
       [69.8,  0. ]])
```

In [264…
```
imputer = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
imputer = imputer.fit(categ)
categ = imputer.transform(categ)
categ[:3]
```

Out[264]:
```
array([['10+ years', 'Marketing', ' 36 months', 'Vacation'],
       ['4 years', 'Credit analyst ', ' 36 months', 'Debt consolidation'],
       ['< 1 year', 'Statistician', ' 36 months',
        'Credit card refinancing']], dtype=object)
```

In [265…
```
df2[['emp_length','emp_title', 'term', 'title']] = categ
df2[['revol_util', 'mort_acc']] = cont
```

In [266…
```
df2.isna().sum()
```

Out[266]:
```
loan_amnt               0
term                    0
int_rate                0
grade                   0
sub_grade               0
emp_title               0
emp_length              0
home_ownership          0
annual_inc              0
verification_status     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
issue_month             0
issue_year              0
zipcode                 0
dtype: int64
```

In [267…
```
continuous_cols = df2.columns[df2.dtypes != 'object']
continuous_cols
```

Out[267]:
```
Index(['loan_amnt', 'int_rate', 'annual_inc', 'dti', 'open_acc', 'pub_rec',
       'revol_bal', 'revol_util', 'total_acc', 'mort_acc'],
      dtype='object')
```

Outlier treatment

In [201…

```
df3 = df2.copy() # will perform log transformation to treat outliers
```

In [221…

```
continuous_cols = continuous_cols.drop(labels =['pub_rec','mort_acc'])
continuous_cols
```

Out[221]:

```
Index(['loan_amnt', 'int_rate', 'annual_inc', 'dti', 'open_acc', 'revol_bal',
       'revol_util', 'total_acc'],
      dtype='object')
```

In [208…

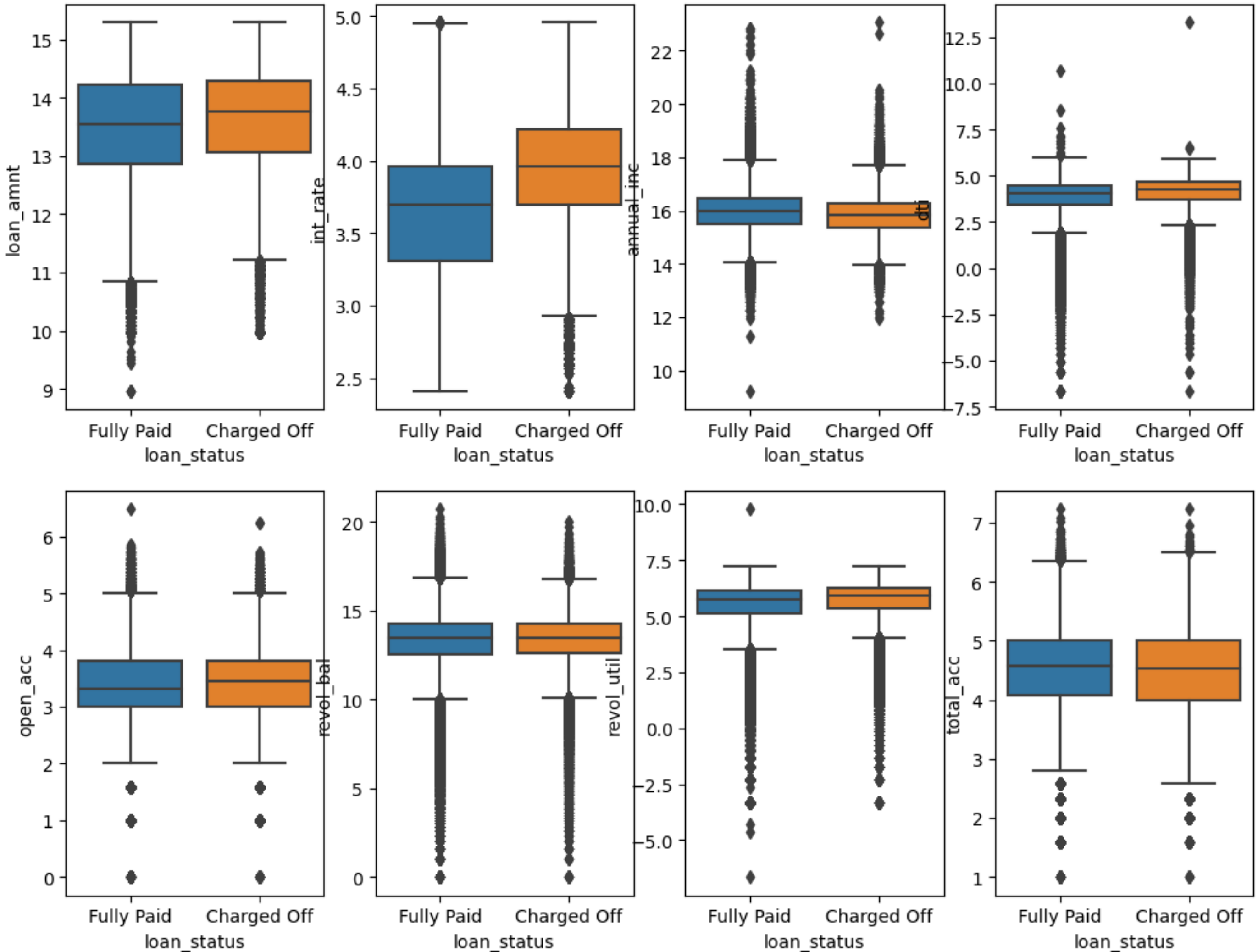
```
n = len(continuous_cols)
for i in range(n):
    df3[continuous_cols[i]] = np.log2(df3[continuous_cols[i]].values)  # log transformation

    print(continuous_cols[i], "is ", df3[continuous_cols[i]].values)
```

```
loan_amnt is  [13.28771238 12.96578428 13.92925841 ... 12.28771238 14.35810171
 10.96578428]
int_rate is  [3.51601515 3.58375975 3.39094277 ... 3.32048468 3.93640238 3.76659516]
annual_inc is  [16.836149   15.9881521  15.39396018 ... 15.78596325 15.96578428
 15.39191483]
dti is  [4.71369581 4.46270675 3.67694436 ... 4.13422094 3.98913901 3.05658353]
open_acc is  [4.         4.08746284 3.70043972 ... 3.9068906  3.169925   1.5849625 ]
revol_bal is  [15.15042164 14.29713122 13.54918302 ... 14.99717948 13.93884446
 12.06743436]
revol_util is  [5.38543104 5.73606363 6.52669485 ... 6.06393431 5.74953427 6.51254296]
total_acc is  [4.64385619 4.7548875  4.70043972 ... 4.52356196 4.32192809 4.24792751]
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_23144\4211438837.py:3: RuntimeWarning: divide by zer
o encountered in log2
  df3[continuous_cols[i]] = np.log2(df3[continuous_cols[i]].values)  # log transformation
```

In [209…

```
f = plt.figure()
f.set_figwidth(12)
f.set_figheight(14)
n = len(continuous_cols)
for i in range(n):
    plt.subplot(3,4,i+1)
    sns.boxplot(data=df3, y=continuous_cols[i],x='loan_status')
plt.show()
```



Log transformation didn't help as it introduced some other outliers.

We will try the Winsorize method to limit outliers within an upper and lower limit.

In [268...

```
from scipy.stats.mstats import winsorize
df_winsorized = df2.copy()

for i in range(len(continuous_cols)):
    df_winsorized[continuous_cols[i]] = winsorize(df2[continuous_cols[i]], (0.01,0.06))
df_winsorized.head()
```

```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\scipy\stats\_stats_py.p
y:112: RuntimeWarning: The input array could not be properly checked for nan values. nan values
will be ignored.
  warnings.warn("The input array could not be properly "
```

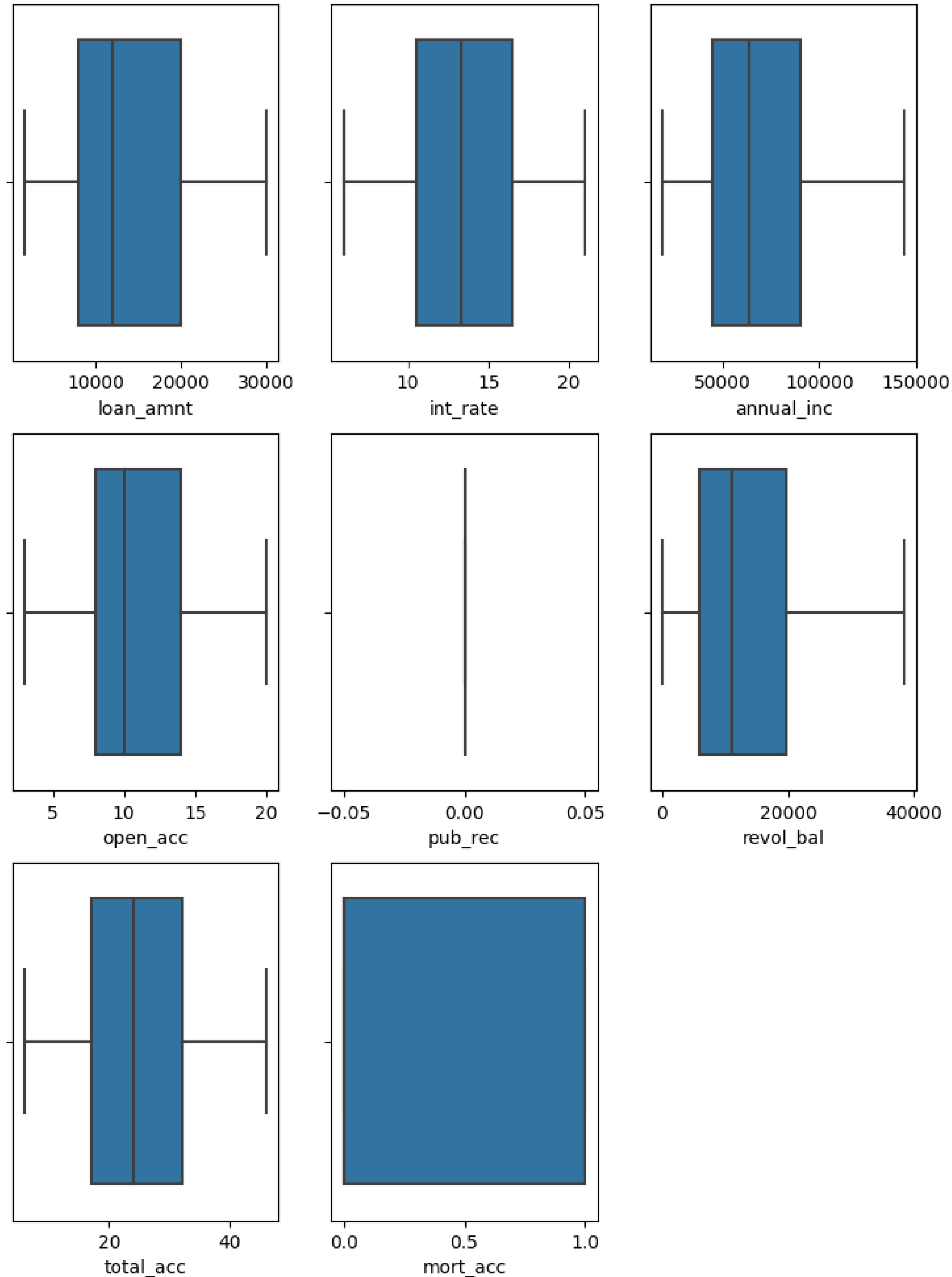
Out[268]:

	loan_amnt	term	int_rate	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verifica
0	10000.0	36 months	11.44	B	B4	Marketing	10+ years	RENT	117000.0	
1	8000.0	36 months	11.99	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	
2	15600.0	36 months	10.49	B	B3	Statistician	< 1 year	RENT	43057.0	So
3	7200.0	36 months	6.49	A	A2	Client Advocate	6 years	RENT	54000.0	
4	24375.0	60 months	17.27	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000.0	

5 rows × 26 columns

In [269…

```
f = plt.figure()
f.set_figwidth(12)
f.set_figheight(12)
n = len(continuous_cols)

for i in range(n):
    plt.subplot(3,4,i+1)
    sns.boxplot(data=df_winsorized, x=continuous_cols[i])
plt.show()
```



All outliers are winsorized effectively.

Encoding

In [270…

```
X = df_winsorized.drop(['loan_status'],axis=1)
Y = np.array(df_winsorized['loan_status']).reshape(-1,1)
print(X.shape, Y.shape)
```

```
(396030, 25) (396030, 1)
```

In [271…

```
from sklearn.preprocessing import LabelEncoder

X = X.select_dtypes(exclude=['number']).apply(LabelEncoder().fit_transform).join(df.select_dtype
X
```

Out[271]:

	term	grade	sub_grade	emp_title	emp_length	home_ownership	verification_status	purpose	title	ea
0	0	1	8	80956	1	5	0	12	36961	
1	0	1	9	33317	4	1	0	2	12926	
2	0	1	7	127182	10	5	1	1	10159	
3	0	0	1	27760	6	5	0	1	10159	
4	1	2	14	38300	9	1	2	1	9268	
...	...	...	...	...	...	...	...	...	...	
396025	1	1	8	160365	2	5	1	2	12926	
396026	0	2	10	5779	5	1	1	2	12926	
396027	0	1	5	26146	1	5	2	2	45964	
396028	1	2	11	56712	1	1	2	2	23304	
396029	0	2	11	66737	1	5	2	2	36384	

396030 rows × 23 columns

In [272…

```
X.isna().sum()
```

Out[272]:

```
term                    0
grade                   0
sub_grade               0
emp_title               0
emp_length              0
home_ownership          0
verification_status     0
purpose                 0
title                   0
earliest_cr_line        0
initial_list_status     0
application_type        0
issue_month             0
issue_year              0
zipcode                 0
loan_amnt               0
int_rate                0
annual_inc              0
dti                     0
open_acc                0
revol_bal               0
revol_util            276
total_acc               0
dtype: int64
```

There are missing values in 'revol util' which need to be treated.

In [280…

```
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer = imputer.fit(np.array(X['revol_util']).reshape([-1,1]))
X['revol_util'] = imputer.transform(np.array(X['revol_util']).reshape([-1,1]))
X['revol_util'].isna().sum()
```

Out[280]: 0

In [247…

```
Y
```

Out[247]:

```
array([['Fully Paid'],
       ['Fully Paid'],
       ['Fully Paid'],
       ...,
       ['Fully Paid'],
       ['Fully Paid'],
       ['Fully Paid']], dtype=object)
```

In [297…

```
Y[Y == 'Fully Paid'] = 0
Y[Y == 'Charged Off'] = 1
Y = np.array(Y).astype(int)
```

In [298…

```
np.unique(Y.astype(str), return_counts=True)
```

Out[298]:

```
(array(['0', '1'], dtype='<U11'), array([318357,  77673], dtype=int64))
```

In [294…

```
318357/77673
```

Out[294]:

```
4.098682940017767
```

The data has imbalance between the two classes- fully paid class and charged off class. We would need to balance them in order to get more accuracy but first let us split the data into training and testing dataset.

Splitting data into training and testing dataset

In [299…

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
# Create training and test split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=4) # 30% t
```

Column Standarization

As the different loan predictor features are in different units, we cannot fairly compare them in terms of importance. We need to scale them to a standard range called standardization.

In [300…

```
# Mean centering and Variance scaling (Standard Scaling)
from sklearn.preprocessing import StandardScaler
X_columns = X_train.columns
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_train = pd.DataFrame(X_train_std, columns=X_columns)
X_train.head()
```

Out[300]:

	term	grade	sub_grade	emp_title	emp_length	home_ownership	verification_status	purpose	ti
0	-0.558767	-0.617239	-0.922009	0.549824	-1.134132	1.090952	-1.267839	-0.293847	2.3717
1	-0.558767	1.628871	1.800817	0.837587	-0.817435	1.090952	1.179488	3.395235	-1.2732
2	-0.558767	-1.365943	-1.375814	0.176748	1.082753	1.090952	-1.267839	-0.293847	-0.7822
3	1.789655	-0.617239	-0.619473	0.974154	-0.817435	-0.987555	1.179488	0.525949	0.2212
4	-0.558767	-0.617239	-0.316937	0.420053	0.449357	1.090952	1.179488	-0.293847	-0.4123

5 rows × 23 columns

Logistic Regression using sklearn

In [357…

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression() #class_weight = { 0:1, 1:4}) # weights were causing lower accuracy o
model.fit(X_train, y_train)
model.coef_, model.intercept_
```

```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[357]:

```
(array([[ 0.16720558, -0.04356283,  0.76607169,  0.11103227,  0.00992962,
          0.14296463,  0.04336087,  0.02636221,  0.01114051, -0.01128358,
         -0.02700304, -0.01540397,  0.00659718,  0.08400227,  0.94486518,
          0.06167614, -0.26687614, -0.160677  ,  0.19410895,  0.12485265,
         -0.0741735 ,  0.09262061, -0.12569733]]),
 array([-1.82338865]))
```

In [312…

```
model.feature_names_in_
```

Out[312]:

```
array(['term', 'grade', 'sub_grade', 'emp_title', 'emp_length',
       'home_ownership', 'verification_status', 'purpose', 'title',
       'earliest_cr_line', 'initial_list_status', 'application_type',
       'issue_month', 'issue_year', 'zipcode', 'loan_amnt', 'int_rate',
       'annual_inc', 'dti', 'open_acc', 'revol_bal', 'revol_util',
       'total_acc'], dtype=object)
```

In [358…

```
features = pd.DataFrame(model.coef_.T,index=[model.feature_names_in_],columns=['coefficients']).:
features
```

Out[358]:

	coefficients
int_rate	-0.266876
annual_inc	-0.160677
total_acc	-0.125697
revol_bal	-0.074173
grade	-0.043563
initial_list_status	-0.027003
application_type	-0.015404
earliest_cr_line	-0.011284
issue_month	0.006597
emp_length	0.009930
title	0.011141
purpose	0.026362
verification_status	0.043361
loan_amnt	0.061676
issue_year	0.084002
revol_util	0.092621
emp_title	0.111032
open_acc	0.124853
home_ownership	0.142965
term	0.167206
dti	0.194109
sub_grade	0.766072
zipcode	0.944865

The outcome was heavily affected by the features: sub_grade and zipcode Top 10 most important features are 'zipcode' 'sub_grade' 'dti' 'term' 'home_ownership' 'open_acc' 'emp_title' 'revol_util' 'issue_year' 'loan_amnt'

In [359…

```
print(f'Train Accuracy:{model.score(X_train,y_train)}, Test Accuracy:{model.score(X_test,y_test)
```

```
Train Accuracy:0.8326064764213389, Test Accuracy:0.8339098889814744
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
```

The training (0.833) and test data (0.834) accuracy score is similarly high, so we can say it is a good fit.

Results evaluation

Classification report

In [360…

```
from sklearn.metrics import confusion_matrix, precision_score, recall_score, plot_confusion_matr
y_pred = model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)

cm_df = pd.DataFrame(cm,index = np.unique(y_test), columns = np.unique(y_test) )

cm_df.head()
```

```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:450: Us
erWarning: X does not have valid feature names, but LogisticRegression was fitted with feature n
ames
  warnings.warn(
```

Out[360]:

	0	**1**
0	92653	2947
1	16786	6423

True negatives - 92653 (consumers who paid the loan as predicted)

False positives - 2947 (Consumers who paid the loan but were predicted as defaulters)

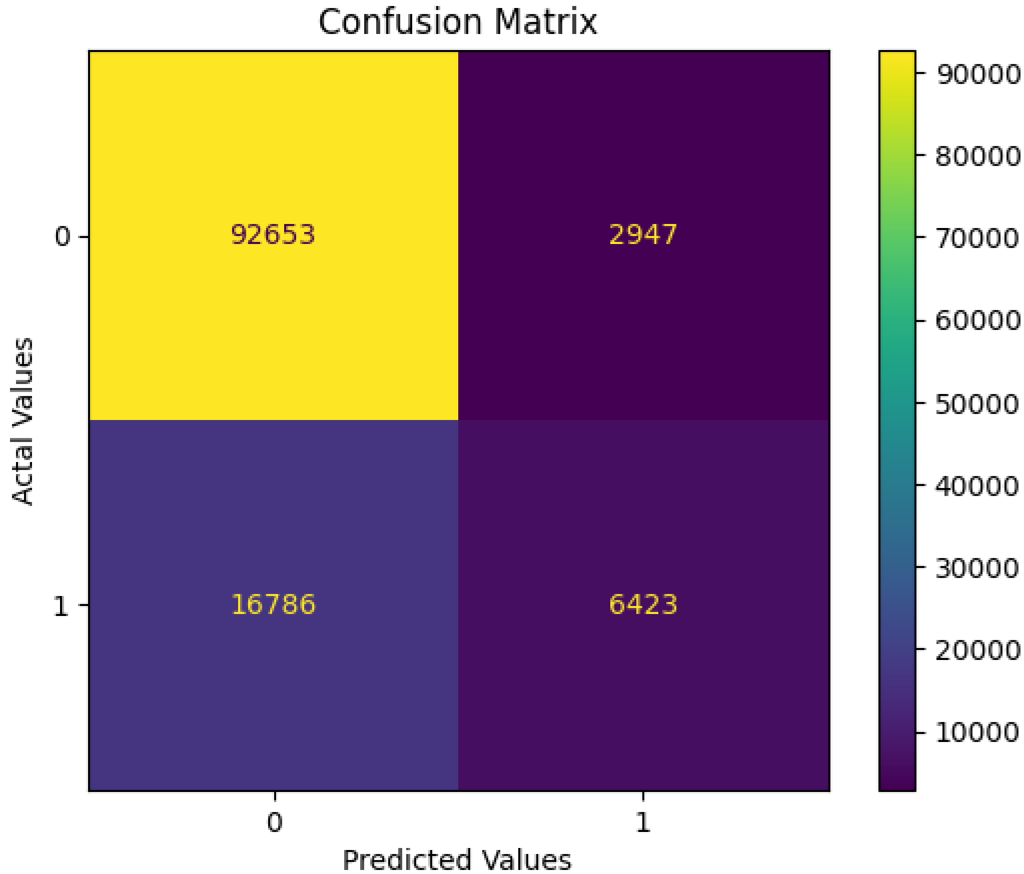
False negatives - 16786 (defaulters that were predicted to pay the loan) This is a huge financial and reputation loss for the bank

True positives - 6423 (true defaulters charged off)

In [329…

```
#Plotting the confusion matrix
plt.figure(figsize=(1,1))
plot_confusion_matrix(model,X_test,y_test)
#sns.heatmap(cm_df, annot=True,cmap='coolwarm')
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```

```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\deprecati
on.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_
matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: Confusion
MatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:450: Us
erWarning: X does not have valid feature names, but LogisticRegression was fitted with feature n
ames
  warnings.warn(
<Figure size 100x100 with 0 Axes>
```



In [361…

```
from sklearn.metrics import f1_score
print("Precision score is :",precision_score(y_test,y_pred))
print("Recall score is :",recall_score(y_test,y_pred))
print("F1 score is :",f1_score(y_test,y_pred))
```

```
Precision score is : 0.6854855923159018
Recall score is : 0.2767460898789263
F1 score is : 0.3943030786703091
```

Precision is 0.69 which is not very high as the number of positive class samples are low and there is an imbalance in the dataset, so it is hard to detect the true positives and so precision is not very high.

Recall-score: 0.28, F1-score: 0.39. Recall is very low and that is why F1-score is also low. This is because of very high number of False negatives.

In [367…

```
train_scores = []
for c in np.arange(0.1,5,0.1):
    model= LogisticRegression(penalty='l2', C=c)
    model.fit(X_train,y_train)
    tr_score = model.score(X_train,y_train)
    train_scores.append(tr_score)

import warnings
warnings.filterwarnings('ignore')
```

```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
```

```
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
```

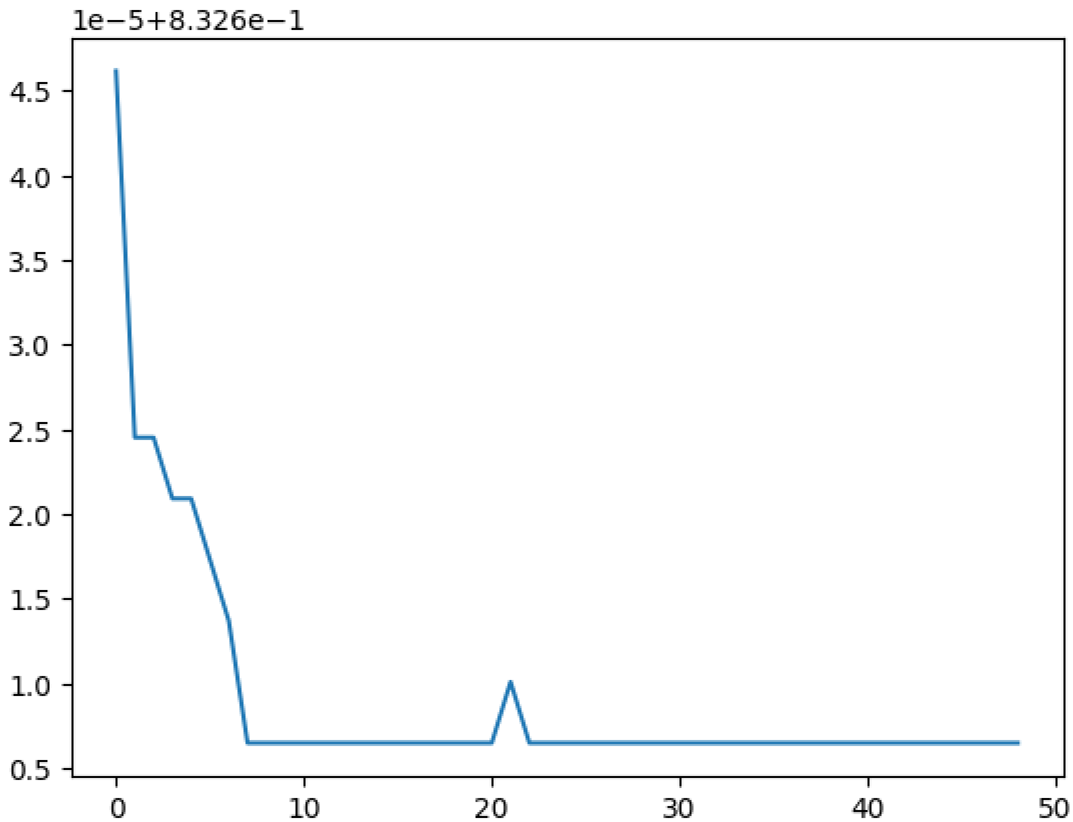
```
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
```

```
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validatio
n.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [368…

```
plt.plot(train_scores)
```

Out[368]: [<matplotlib.lines.Line2D at 0x16bdbcf1010>]



ROC (Receiver Operating Characteristic curve) and AUC (Area Under Curve)

In [339…

```
# from sklearn.linear_model import
y_proba = model.predict_proba(X_test)
y_proba.shape, y_test.shape
```

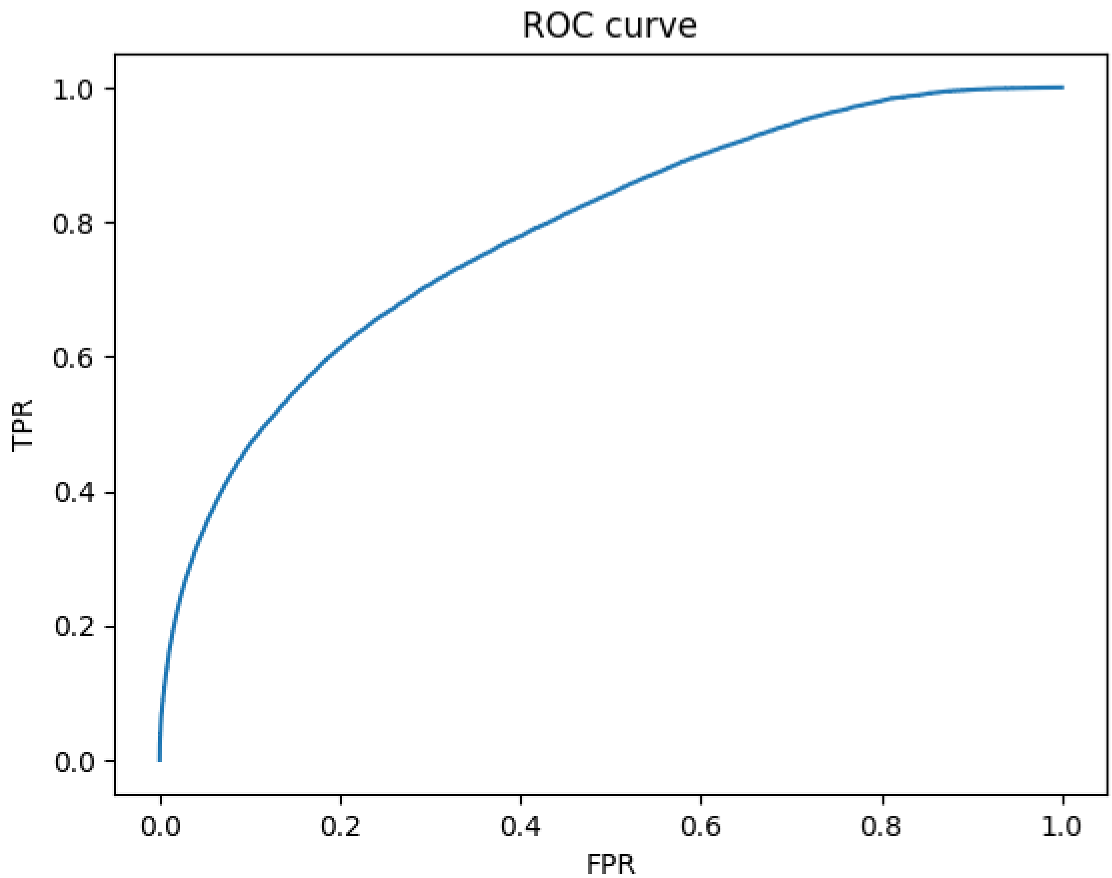
```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:450: Us
erWarning: X does not have valid feature names, but LogisticRegression was fitted with feature n
ames
  warnings.warn(
```

Out[339]: ((118809, 2), (118809, 1))

In [340…

```
from sklearn.metrics import roc_curve, roc_auc_score
fpr, tpr, thr = roc_curve(y_test, y_proba[:,1])
plt.plot(fpr,tpr)
plt.title('ROC curve')
plt.xlabel('FPR')
```

```
plt.ylabel('TPR')
plt.show()
```



There are more true positives than False positives, hence the ROC curve has more area under curve than 0.5.

In [342…

```
roc_auc_score(y_test,y_proba[:,1])
```

Out[342]:

```
0.783562142066876
```

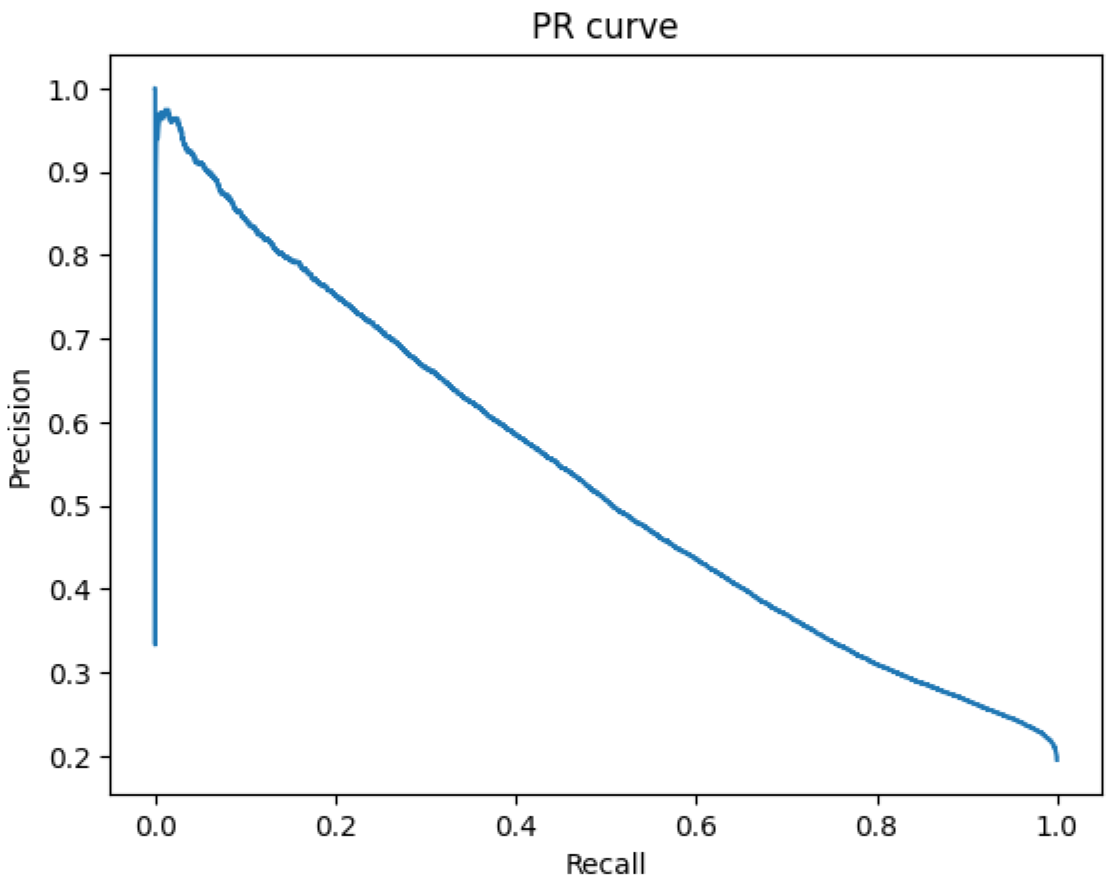
0.78 is decent area under curve.

Precision recall curve

In [347…

```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc
precision, recall, thr = precision_recall_curve(y_test, y_proba[:,1])
print(auc(recall, precision))
plt.plot(recall, precision)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PR curve')
plt.show()
```

```
0.5326813316214373
```



0.53 is not a good AUC score. This is because the recall is very low due to large number of False Negatives and causing the PR AUC to be small.

Precision and Recall Tradeoff Questions:

How can we make sure that our model can detect real defaulters and there are less false positives? This is important as we can lose out on an opportunity to finance more individuals and earn interest on it. We need to make sure that precision is high (close to 1) so that there are less false positives and we are not marking any loan eligible individuals as defaulters and lose out on financial opportunities. Precision is the ratio of true positives predicted (true loan defaulters) over all positives predicted (true positive + false positive) predicted as defaulters.

Since NPA (non-performing asset) is a real problem in this industry, it’s important we play safe and shouldn’t disburse loans to anyone Since NPA can cause huge financial and reputation losses, banks need to make sure that their model's recall score is high (close to 1), so that there are no false negatives i.e. no defaulting customers are predicted as loan eligible. Recall is the ratio of true positives predicted and all positives (true positives and false negatives).

Business Insights

There are most number of customers with 10+ years of employment.

Most common purposes for getting loans were debt consolidation, credit card purchase, home improvement and other household or business purchases.

What percentage of customers have fully paid their Loan Amount? 80.39% customers have paid their loans fully.

Comment about the correlation between Loan Amount and Installment features The loan amount and installment amounts are highly correlated (0.97) as per the spearman correlation coefficient.

The majority of people have home ownership as: MORTAGE at about 50%. Next highest is RENT ~40%..

People with grades 'A' are more likely to fully pay their loan. True, people with loan grade A are more likely to pay their loan.

Name the top 2 afforded job titles. Most common employment title is Teacher ~1.1% and then Manager ~1%.

Thinking from a bank's perspective, which metric should our primary focus be on.. ROC AUC Precision Recall F1 Score We should focus on Precision as this is a newer brand in this sector and needs to grow and get financial profits through as many loan consumers as possible, so we want to reduce False Positives who are loan-eligible customers marked as defaulters who could have paid their loans and provided the business its growth avenue. F1-score: it is a harmonic mean of Precision and Recall. ROC-AUC : Not good metric to consider as we have highly imbalanced data. Recall: Consider when we do not want NPAs which are not as important for a newer brand compared to industry veterans.

How does the gap in precision and recall affect the bank? The gap denotes that since recall is lower, there are more false negatives which are defaulters that were marked as eligible for loans, and have caused the bank loss through NPAs.

Which were the features that heavily affected the outcome? The outcome was heavily affected by the features: sub_grade and zipcode Top 10 most important features are 'zipcode' 'sub_grade' 'dti' 'term' 'home_ownership' 'open_acc' 'emp_title' 'revol_util' 'issue_year' 'loan_amnt'

Will the results be affected by geographical location? Yes! Zipcode or the geographical location is the most important feature affecting the outcome.

The training (0.833) and test data (0.834) accuracy score is similarly high, so we can say it is a good fit.

True negatives - 92653 (consumers who paid the loan as predicted)

False positives - 2947 (Consumers who paid the loan but were predicted as defaulters)

False negatives - 16786 (defaulters that were predicted to pay the loan) This is a huge financial and reputation loss for the bank

True positives - 6423 (true defaulters charged off)

Precision is 0.69 which is not very high as the number of positive class samples are low and there is an imbalance in the dataset, so it is hard to detect the true positives and so precision is not very high.

Recall-score: 0.28, F1-score: 0.39. Recall is very low and that is why F1-score is also low. This is because of very high number of False negatives.

Recommendations

Customers from certain zipcodes are more likely to pay loans so they should be given incentives and targeted for marketing.

Most customers have 10+ years of employment, so they are more likely to pay their loans and should be marketed for other loan categories.

Loan amount, interest rate and installment amount should be kept low as then the customers are likely to pay back the loan.

Grade A, B and C loans should be provided more incentives and marketing as they are more likely to be paid back.

Term of 36 months loan should be used more often as it more likely to be fully paid by customers.

More data should be collected to improve model prediction accuracy higher than 83.3% and to catch more defaulters.

Mortgage and rent were the most common loan purposes, so such loan categories should be provided with lesser interest rate to attract more loan payers and less defaulters.

```
In [ ]:
```