In [1]:
```
# useful imports
import numpy as np, seaborn as sns, pandas as pd, matplotlib.pyplot as plt, scipy
df = pd.read_csv('logistic_regression_data.csv')
```

In [2]:
```
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
```

In [3]:
```
df.shape
```

Out[3]:
```
(396030, 27)
```

Shape is 3,96,030 rows and 27 columns

The continuous variables are:

In [4]:
```
continuous_cols = df.columns[df.dtypes != 'object']
continuous_cols
```

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 variables are:

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

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')
```

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 [6]:
```
df.isna().sum()*100/len(df)
```

Out[6]:
```
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
```

In [7]:
```
df.duplicated().sum()
```

Out[7]:
```
0
```

In [8]:
```
df.describe()
```

Out[8]:

	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.m

In [9]:

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

Out[9]:

	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	396(
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	318:

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.

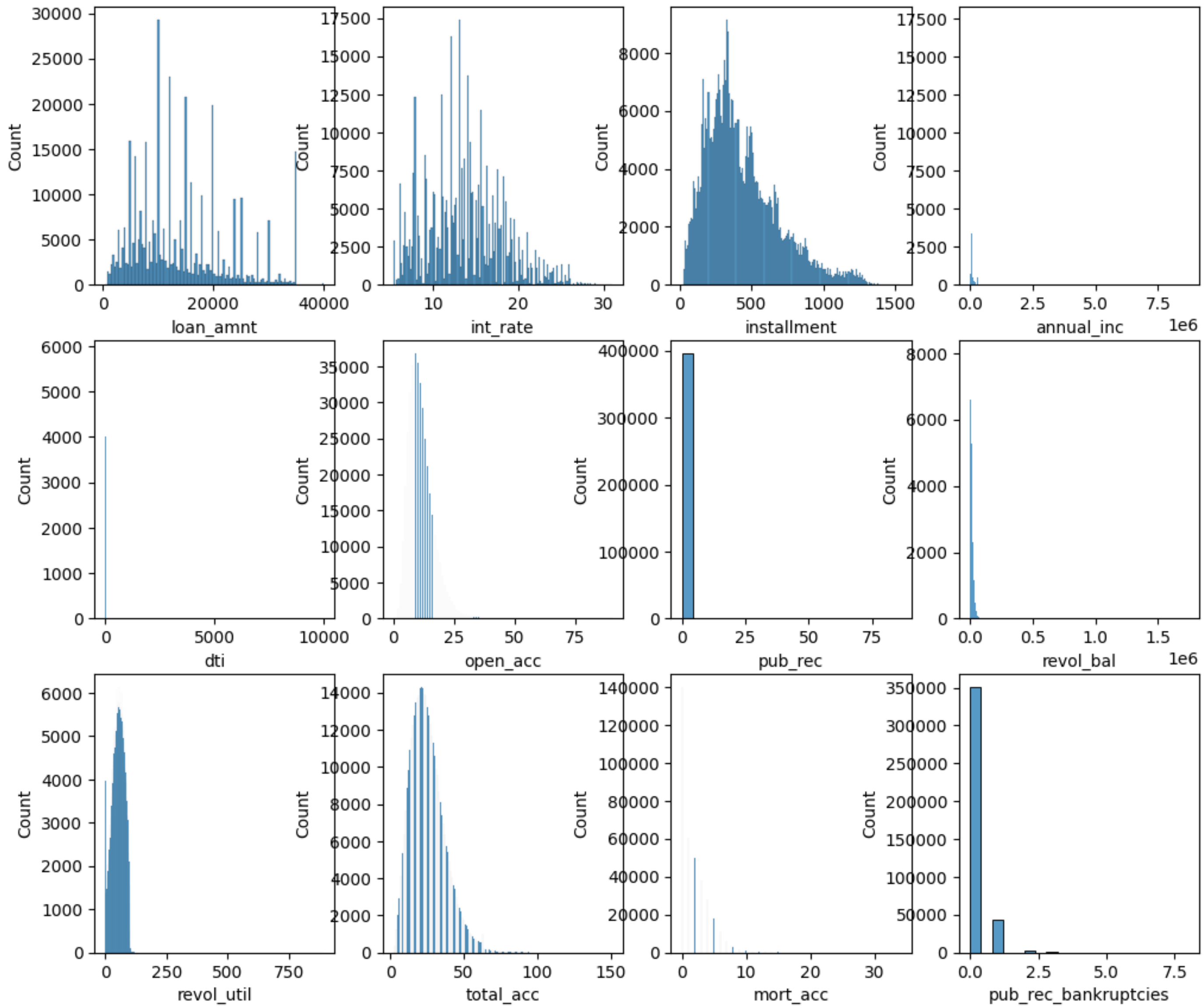
Univariate analysis

histograms

In [10]:

```
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)
```

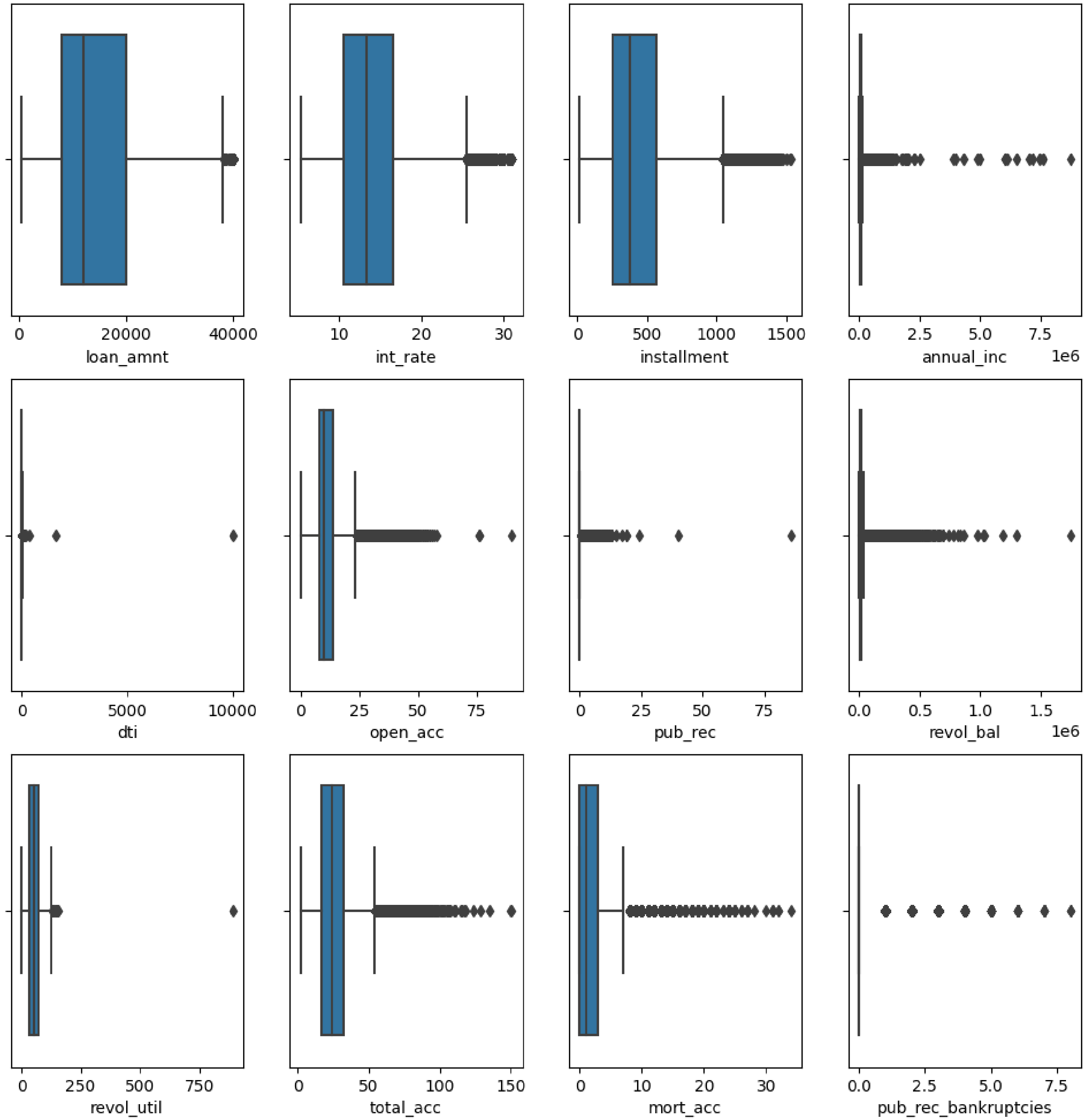
```
    sns.histplot(data=df, x=continuous_cols[i])
plt.show()
```



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

boxplots

In [11]:
```
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()
```



Most of these have the presence of outliers that need to be removed.

In [12]:

```
# 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%
```

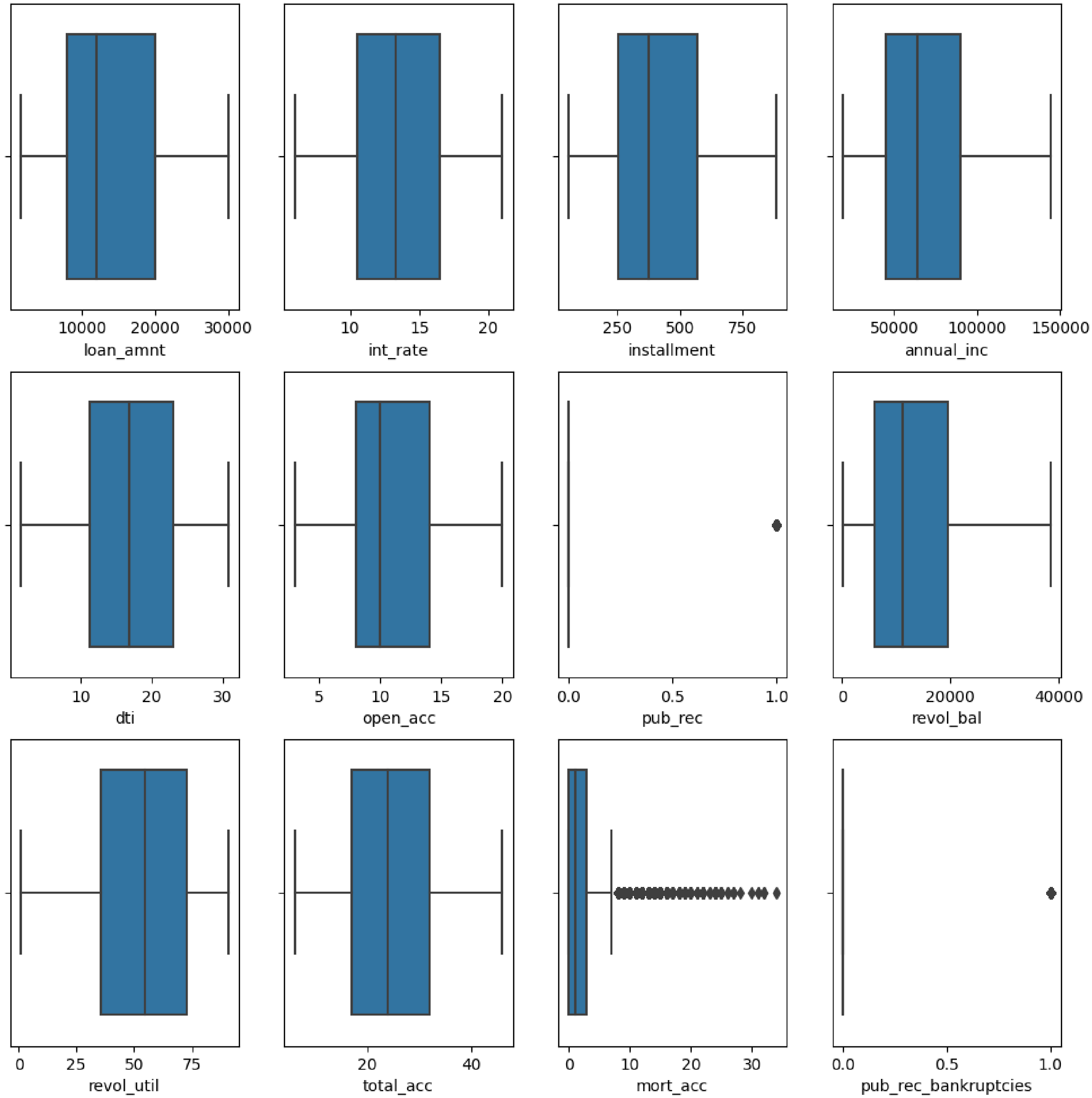
We will try the Winsorize method to limit outliers within an upper and lower limit.

In [13]:

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

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

In [14]:

```
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])
df2 = df_winsorized.copy()
plt.show()
```



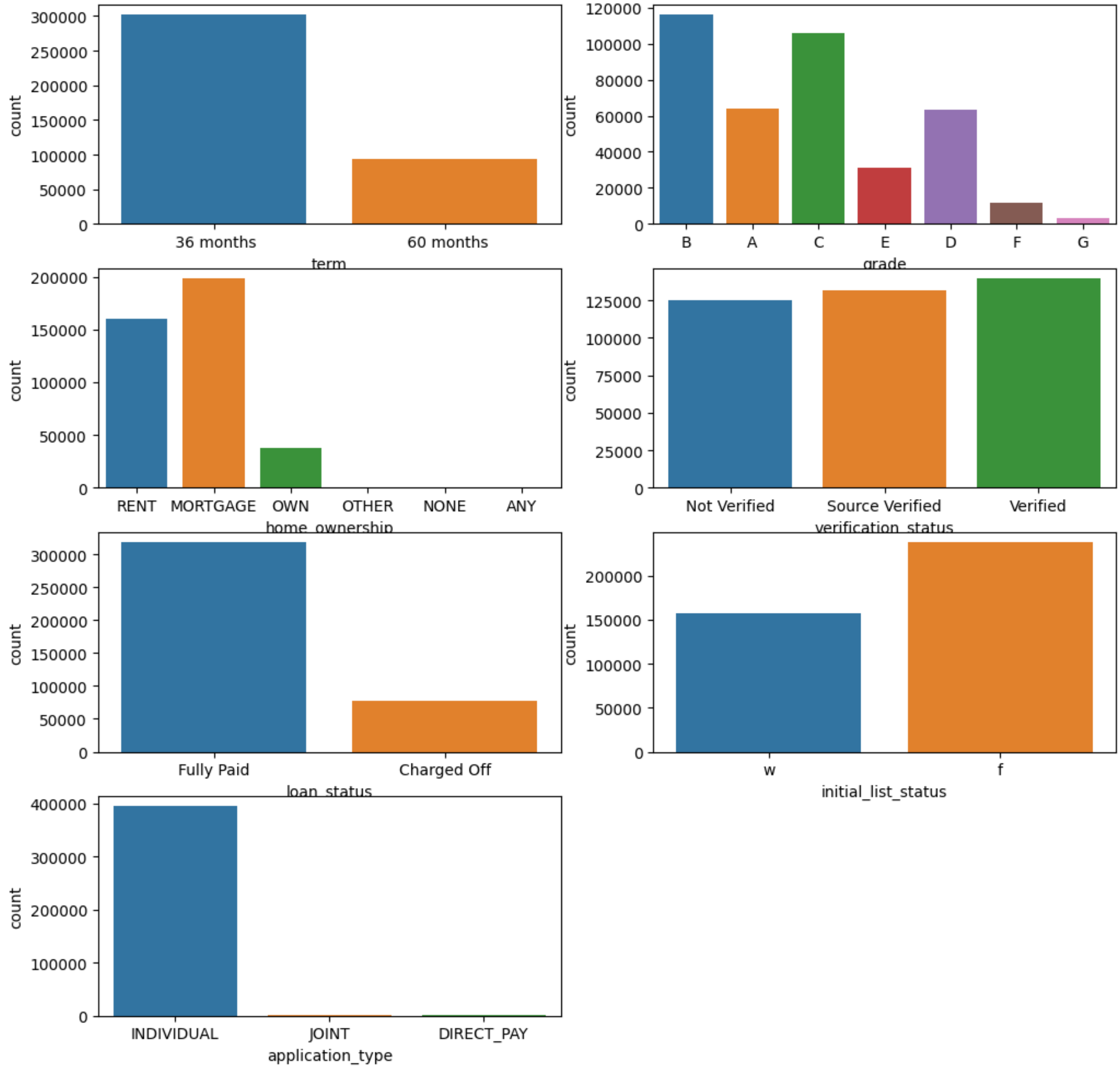
checking categorical variables

In [15]:
```
trim_categorical_cols = categorical_cols.drop(['sub_grade','emp_title','emp_length','issue_d','pu
                                               'title','earliest_cr_line','address']) # these wi
trim_categorical_cols
```

Out[15]:
```
Index(['term', 'grade', 'home_ownership', 'verification_status', 'loan_status',
       'initial_list_status', 'application_type'],
      dtype='object')
```

In [16]:
```
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)
```

```python
    sns.countplot(data=df2, 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 [17]:
```python
df2['loan_status'].value_counts()*100/len(df2)
```

Out[17]:
```
Fully Paid     80.387092
Charged Off    19.612908
Name: loan_status, dtype: float64
```

80.39% customers have paid their loans fully.

In [18]:
```
df2['home_ownership'].value_counts()*100/len(df2)
```

Out[18]:
```
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%. Let us combine the categories "OTHER","NONE" and "ANY" together.

In [19]:
```
df2.loc[(df2['home_ownership']=="NONE")|(df2['home_ownership']=="ANY"),['home_ownership']] = "OTI
df2['home_ownership'].value_counts()
```

Out[19]:
```
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          146
Name: home_ownership, dtype: int64
```

In [20]:
```
df2['purpose'].value_counts()*100/len(df2)
```

Out[20]:
```
debt_consolidation    59.214453
credit_card           20.962806
home_improvement       6.067722
other                  5.349342
major_purchase         2.219529
small_business         1.439537
car                    1.186021
medical                1.059516
moving                 0.720652
vacation               0.619145
house                  0.555766
wedding                0.457541
renewable_energy       0.083075
educational            0.064894
Name: purpose, dtype: float64
```

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

In [21]:
```
df2.loc[(df2['purpose']=="major_purchase")|(df2['purpose']=="small_business")|(df2['purpose']=="
        (df2['purpose']=="educational")|(df2['purpose']=="medical")|(df2['purpose']=="moving")|((
        |(df2['purpose']=="house")|(df2['purpose']=="wedding")|(df2['purpose']=="renewable_energ
df2['purpose'].value_counts()
```

Out[21]:
```
debt_consolidation    234507
credit_card            83019
other                  54474
home_improvement       24030
Name: purpose, dtype: int64
```

In [22]:
```
df2['title'].value_counts()*100/len(df2)
```

```
Out[22]: Debt consolidation            38.500114
         Credit card refinancing       13.000783
         Home improvement               3.854253
         Other                          3.264904
         Debt Consolidation             2.931091
                                          ...
         Graduation/Travel Expenses     0.000253
         Daughter's Wedding Bill        0.000253
         gotta move                     0.000253
         creditcardrefi                 0.000253
         Toxic Debt Payoff              0.000253
         Name: title, Length: 48817, dtype: float64
```

Loan title has similar category distribution as loan purpose, we can delete it to reduce multicollinearity.

```
In [23]: df2.drop('title', axis=1, inplace=True)
```

```
In [24]: df2['emp_title'].value_counts()*100/len(df2)
```

```
Out[24]: Teacher                    1.108249
         Manager                    1.073151
         Registered Nurse           0.468651
         RN                         0.466126
         Supervisor                 0.462086
                                      ...
         Postman                    0.000253
         McCarthy & Holthus, LLC    0.000253
         jp flooring                0.000253
         Histology Technologist     0.000253
         Gracon Services, Inc       0.000253
         Name: emp_title, Length: 173105, dtype: float64
```

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

```
In [25]: # make related employment titles in uppercase and lowercase to the same case
         df2['emp_title'] = df2['emp_title'].str.lower()
         df2['emp_title'].value_counts()*100/len(df2)
```

```
Out[25]: manager                        1.423377
         teacher                        1.371108
         registered nurse               0.663334
         supervisor                     0.654243
         sales                          0.601470
                                          ...
         director of public events      0.000253
         amsec llc                      0.000253
         simon  and  schuster           0.000253
         coating specialist iii         0.000253
         gracon services, inc           0.000253
         Name: emp_title, Length: 154014, dtype: float64
```

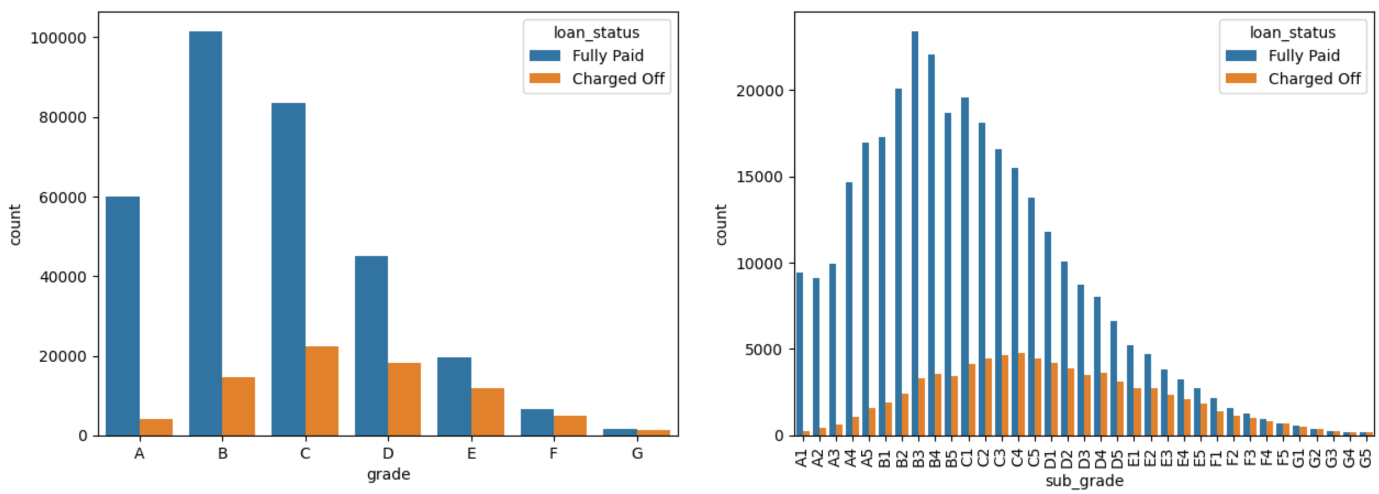
I have later done target encoding of employment title because of high cardinality.

```
In [26]: df2['emp_length'].value_counts()
```

```
Out[26]: 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.

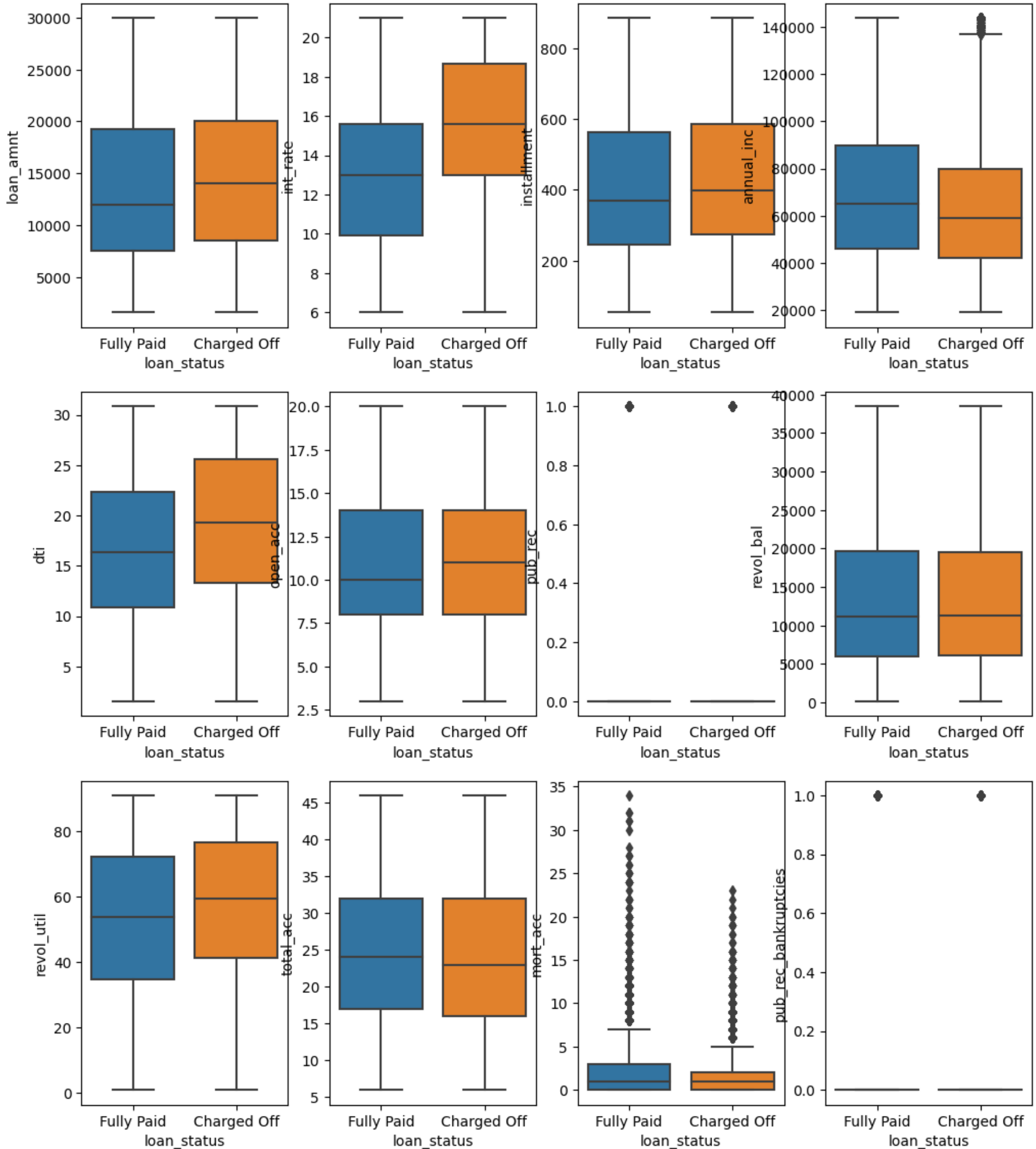
```
In [27]: f = plt.figure(figsize = (15,5))

plt.subplot(1,2,1)
grade = sorted(df2.grade.unique().tolist())
sns.countplot(data=df2, x='grade', hue='loan_status', order=grade)

plt.subplot(1,2,2)
sub_grade = sorted(df2.sub_grade.unique().tolist())
g = sns.countplot(data=df2, x='sub_grade',hue='loan_status', order=sub_grade)
g.set_xticklabels(g.get_xticklabels(), rotation=90);
```



Loan grades A, B and C have good payout rate, and so do sub grades A1-A5, B1-B5, C1-C4.

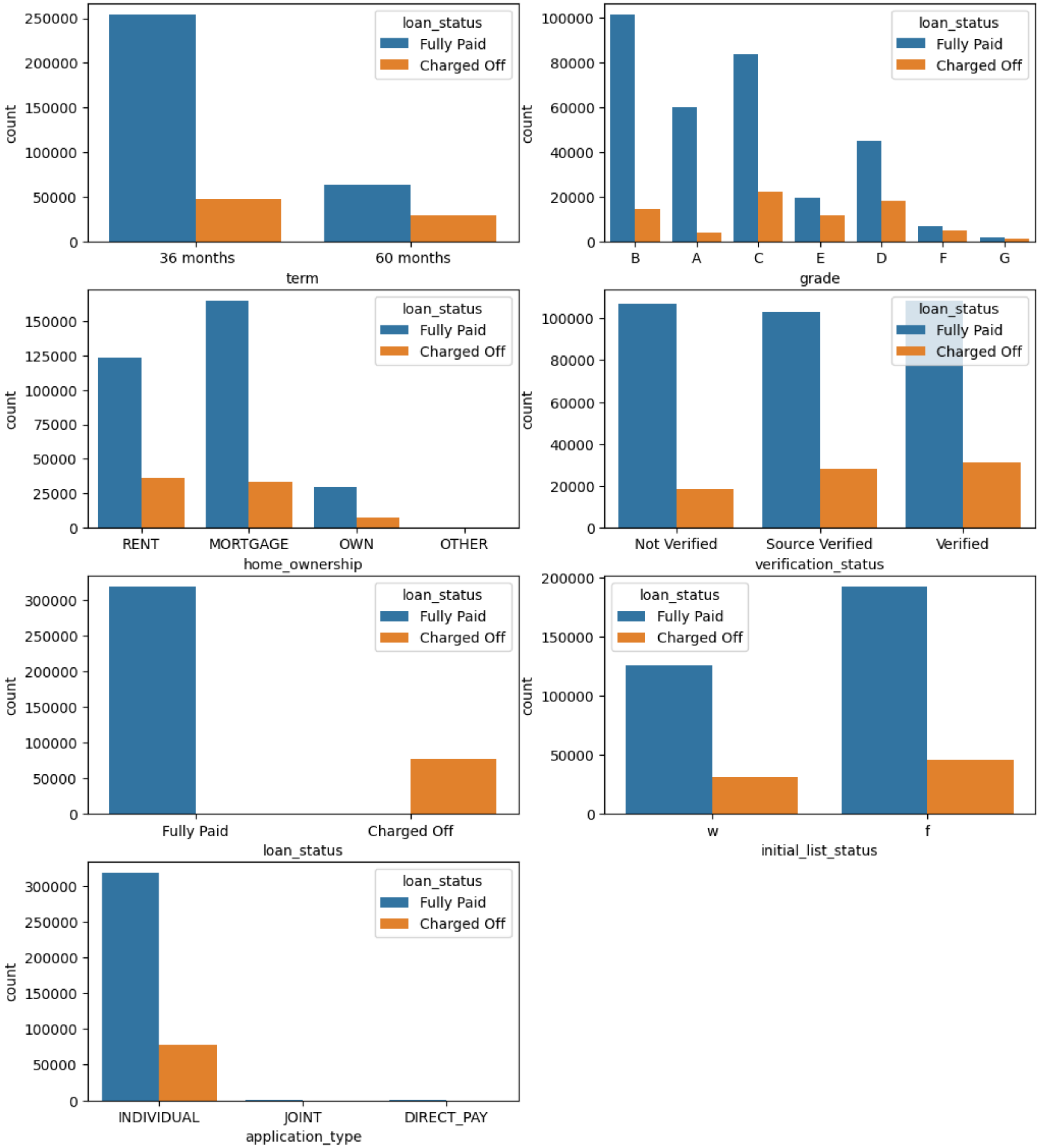
Bivariate analysis

```
In [28]: 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=df2, y=continuous_cols[i],x='loan_status')
plt.show()
```



Public derogatory records, public bankruptcy records have most records as 0, very few outliers at 1. Mortgage accounts is also mostly a small number, with some outliers have more than 5 accounts.

In [29]:
```
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=df2, x=trim_categorical_cols[i],hue='loan_status')
plt.show()
```



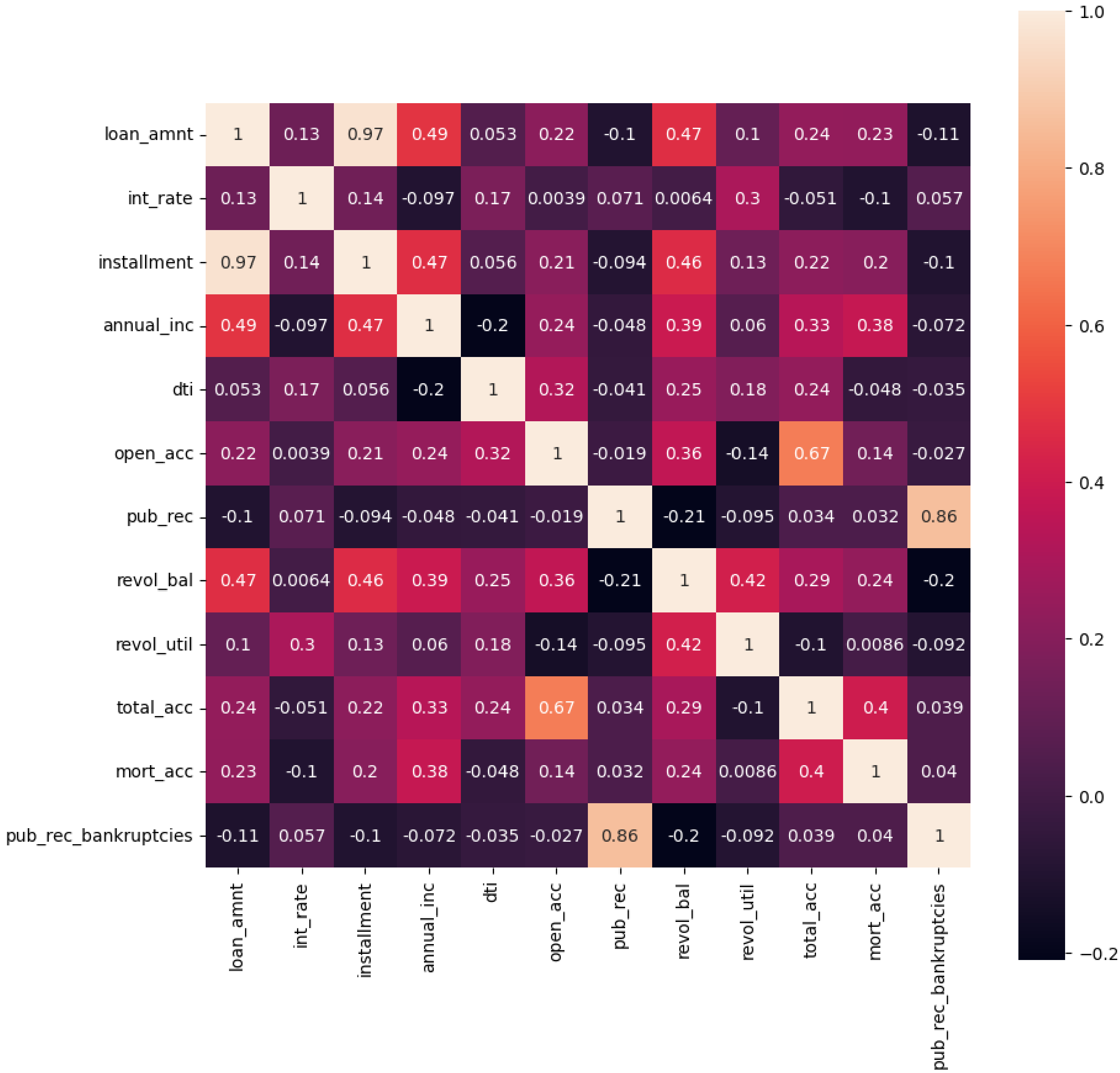
There are more customers who have fully paid their loans than those charged off in all categories.

In [30]:

```python
# Spearman's Rank Correlation Coefficient
plt.figure(figsize=(10,10))
sns.heatmap(df2.corr(method='spearman'), square=True,annot=True)
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_3776\2516434410.py:3: FutureWarning: The default val
ue of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to Fals
e. Select only valid columns or specify the value of numeric_only to silence this warning.
  sns.heatmap(df2.corr(method='spearman'), square=True,annot=True)
```

Out[30]: <AxesSubplot: >



Loan amount and installment are highly correlated, as expected, so I have removed installment. Public derogatory records and public bankruptcy records are also correlated but since they refer to different types of records I will not remove them.

In [31]:
```
df2 = df2.drop(['installment'],axis=1)
```

Data Preprocessing

In [32]:
```
df2['issue_d'].head()
```

Out[32]:
```
0     Jan-2015
1     Jan-2015
2     Jan-2015
3     Nov-2014
4     Apr-2013
Name: issue_d, dtype: object
```

In [33]:
```
df2.duplicated().sum()
```

Out[33]: 0

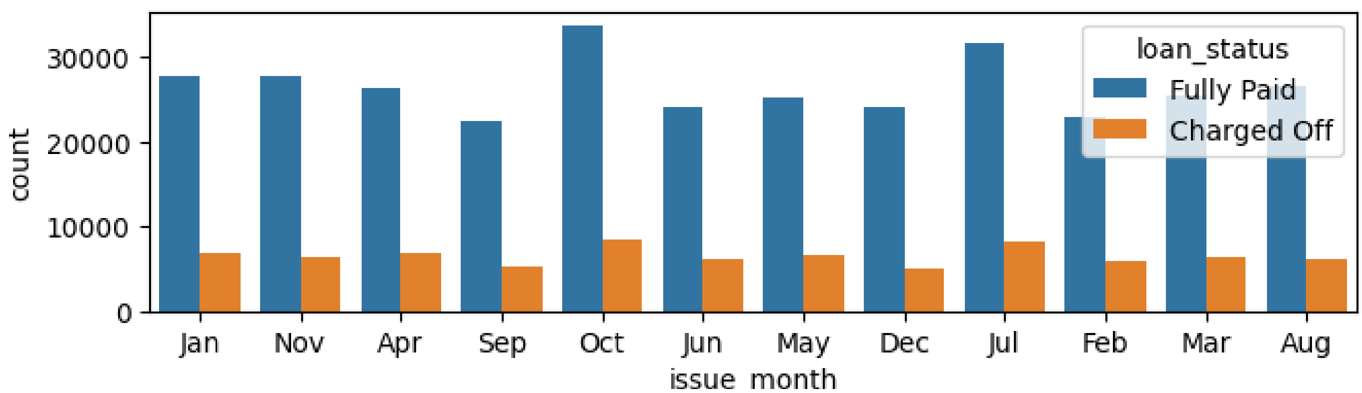
In [34]:
```python
df2 = df2.join(df2['issue_d'].str.split('-',1, expand=True).rename(columns={0:'issue_month', 1:'
df2[['issue_d','issue_month','issue_year']].head()
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_3776\2577343616.py:1: FutureWarning: In a future ver
sion of pandas all arguments of StringMethods.split except for the argument 'pat' will be keywor
d-only.
  df2 = df2.join(df2['issue_d'].str.split('-',1, expand=True).rename(columns={0:'issue_month',
1:'issue_year'}))
```

Out[34]:

	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 [35]:
```python
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.

Address column can be clipped to just use zipcode

In [36]:
```python
df2['zipcode'] = df2['address'].str[-5:]
df2['zipcode'].head()
```

Out[36]:
```
0    22690
1    05113
2    05113
3    00813
4    11650
Name: zipcode, dtype: object
```

In [37]:
```python
df2.drop(['address','issue_d','issue_month'],axis=1,inplace=True)
```

In [38]:
```python
df2['earliest_cr_line'].value_counts()*100/len(df2)
```

Out[38]:
```
Oct-2000    0.761811
Aug-2000    0.741105
Oct-2001    0.731258
Aug-2001    0.728228
Nov-2000    0.690857
              ...
Jul-1958    0.000253
Nov-1957    0.000253
Jan-1953    0.000253
Jul-1955    0.000253
Aug-1959    0.000253
Name: earliest_cr_line, Length: 684, dtype: float64
```

Extracting just the year.

In [39]:
```
df2['earliest_cr_line'] = pd.to_datetime(df2['earliest_cr_line']).dt.year
df2['earliest_cr_line'].head(2)
```

Out[39]:
```
0    1990
1    2004
Name: earliest_cr_line, dtype: int64
```

In [40]:
```
df2.isnull().sum()*100/len(df2)
```

Out[40]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                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
loan_status             0.000000
purpose                 0.000000
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.000000
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                9.543469
pub_rec_bankruptcies    0.000000
issue_year              0.000000
zipcode                 0.000000
dtype: float64
```

mean imputation of mortgage accounts based on total accounts

In [41]:
```
df2.groupby(['total_acc'])['mort_acc'].mean().head()
```

Out[41]:
```
total_acc
6.0     0.117395
7.0     0.221695
8.0     0.308422
9.0     0.365499
10.0    0.429158
Name: mort_acc, dtype: float64
```

In [42]:
```
total_acc_avg = df2.groupby(['total_acc'])['mort_acc'].mean()
def fill_mort_acc(total_acc, mort_acc):
    if np.isnan(mort_acc):
        return total_acc_avg[total_acc].round()
    else:
        return mort_acc
```

In [43]:
```
df2['mort_acc'] = df2.apply(lambda x:fill_mort_acc(x['total_acc'],x['mort_acc']), axis=1)
df2.isnull().sum()/len(df)*100
```

Out[43]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                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
loan_status             0.000000
purpose                 0.000000
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.000000
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                0.000000
pub_rec_bankruptcies    0.000000
issue_year              0.000000
zipcode                 0.000000
dtype: float64
```

In [44]:
```
categ = df2[['emp_length','emp_title']].values

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

In [45]:
```
imputer = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
imputer = imputer.fit(categ)
categ = imputer.transform(categ)
categ[:3]
```

Out[45]:
```
array([['10+ years', 'marketing'],
       ['4 years', 'credit analyst '],
       ['< 1 year', 'statistician']], dtype=object)
```

In [46]:
```
df2[['emp_length','emp_title']] = categ
```

In [47]:
```
df2.isna().sum()
```

Out[47]:
```
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
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
pub_rec_bankruptcies    0
issue_year              0
zipcode                 0
dtype: int64
```

Encoding

In [48]:
```
continuous_cols = df2.columns[df2.dtypes != 'object']
continuous_cols
```

Out[48]:
```
Index(['loan_amnt', 'int_rate', 'annual_inc', 'dti', 'earliest_cr_line',
       'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc',
       'mort_acc', 'pub_rec_bankruptcies'],
      dtype='object')
```

In [49]:
```
continuous_cols = continuous_cols.drop(labels =['pub_rec','mort_acc','pub_rec_bankruptcies'])
continuous_cols
```

Out[49]:
```
Index(['loan_amnt', 'int_rate', 'annual_inc', 'dti', 'earliest_cr_line',
       'open_acc', 'revol_bal', 'revol_util', 'total_acc'],
      dtype='object')
```

Creation of Flags

If value greater than 1.0 then 1 else 0. This can be done on:

1. Pub_rec
2. Mort_acc
3. Pub_rec_bankruptcies

In [50]:
```
def pub_rec(x):
    if x == 0.0:
        return 0
    else:
        return 1
```

```
def pub_rec_bankruptcies(x):
    if x == 0.0:
        return 0
    elif x>= 1.0:
        return 1
    else:
        return x
def mort_acc(x):
    if x <1:
        return 0
    elif x>=1.0:
        return 1
    else:
        return x
df2['pub_rec'] = df2.pub_rec.apply(pub_rec)
df2['pub_rec_bankruptcies'] = df2.pub_rec_bankruptcies.apply(pub_rec_bankruptcies)
df2['mort_acc'] = df2.mort_acc.apply(mort_acc)
```

In [51]:
```
df2['pub_rec'].value_counts()
```

Out[51]:
```
0    338272
1     57758
Name: pub_rec, dtype: int64
```

In [52]:
```
df2['pub_rec_bankruptcies'].value_counts()
```

Out[52]:
```
0    350380
1     45650
Name: pub_rec_bankruptcies, dtype: int64
```

In [53]:
```
df2['mort_acc'].value_counts()
```

Out[53]:
```
1    250817
0    145213
Name: mort_acc, dtype: int64
```

In [54]:
```
df2['term'].unique()
```

Out[54]:
```
array([' 36 months', ' 60 months'], dtype=object)
```

removing extra space and mapping term values

In [55]:
```
term_values = {' 36 months':36, ' 60 months':60}
df2['term'] = df2.term.map(term_values)
df2.term.unique()
```

Out[55]:
```
array([36, 60], dtype=int64)
```

In [56]:
```
df2['initial_list_status'].value_counts()
```

Out[56]:
```
f    238066
w    157964
Name: initial_list_status, dtype: int64
```

mapping initial list status

In [57]:
```
ls_values = {'w':0,'f':1}
df2['initial_list_status'] = df2.initial_list_status.map(ls_values)
```

target variable mapping

In [58]:
```
loan_values = {'Fully Paid':0,'Charged Off':1}
df2['loan_status'] = df2.loan_status.map(loan_values)
```

target encoding employment title due to high cardinality

In [61]:
```
import category_encoders as ce
TE = ce.TargetEncoder()
df2['emp_title'] = TE.fit_transform(df2['emp_title'],df2['loan_status'])
```

In [62]:
```
df2['emp_title'].value_counts().head(3)
```

Out[62]:
```
0.170611    103528
0.254166     28564
0.300719     23221
Name: emp_title, dtype: int64
```

In [63]:
```
X = df2.drop(['loan_status'],axis=1)
Y = np.array(df2['loan_status']).reshape(-1,1)
print(X.shape, Y.shape)
```

```
(396030, 24) (396030, 1)
```

In [86]:
```
np.unique(Y, return_counts=True)
```

Out[86]:
```
(array([0, 1], dtype=int64), array([318357,  77673], dtype=int64))
```

The dataset is imbalanced.

In [65]:
```
from sklearn.preprocessing import LabelEncoder
X = X.select_dtypes(exclude=['number']).apply(LabelEncoder().fit_transform).join(df2.select_dtyp
```

In [67]:
```
X.drop('loan_status', axis=1,inplace=True)
```

Splitting data into training and testing dataset

In [69]:
```
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
```

In [70]:
```
# 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)
```

Logistic Regression using sklearn

In [97]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(class_weight = 'balanced') # { 0:1, 1:4}) # weights are causing lower
```

```
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[97]:

```
(array([[-0.06269357,  0.49432624,  0.03700336,  0.13143811,  0.03253337,
          0.032186  , -0.02582305, -0.00832573,  0.88689972,  0.13817908,
          0.20760986,  0.00896352,  1.25987755,  0.00827205,  0.17325585,
         -0.00577007,  0.18348981,  0.05021743, -0.08691601,  0.16269549,
         -0.0662731 ,  0.02524608, -0.04090535, -0.06684011]]),
 array([-0.80088172]))
```

In [98]:

```
model.feature_names_in_
```

Out[98]:

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

In [99]:

```
features = pd.DataFrame(model.coef_.T,index=[model.feature_names_in_],columns=['coefficients']).
    by=['coefficients'],ascending=False,axis=0)
features.T
```

Out[99]:

	emp_title	zipcode	sub_grade	term	open_acc	dti	revol_util	loan_amnt	home_ownership
coefficients	1.259878	0.8869	0.494326	0.20761	0.18349	0.173256	0.162695	0.138179	0.131438

1 rows × 24 columns

The outcome was heavily affected by the features: emp_title and zipcode Top 10 most important features affecting loan payment are-employment title, zipcode, loan sub_grade, term duration (36 or 60 months), no. of open accounts, dti ratio, revolving line utilization rate, loan amount, home ownership status and number of public derogatory records.

In [100…

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

```
Train Accuracy:0.811276923465394, Test Accuracy:0.8112432559822909
```

```
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(
```

The training (0.811) and test data (0.811) accuracy score is similarly high, so we can say it is a good fit.

In [101…

```
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: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
```

Out[101]:

	0	1
0	77870	17730
1	4696	18513

In [102…

```python
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 : 0.5108020859200397
Recall score is : 0.7976646990391658
F1 score is : 0.6227881316019646
```

Precision (51.1%) and recall (79.8%) scores are low.

In [103…

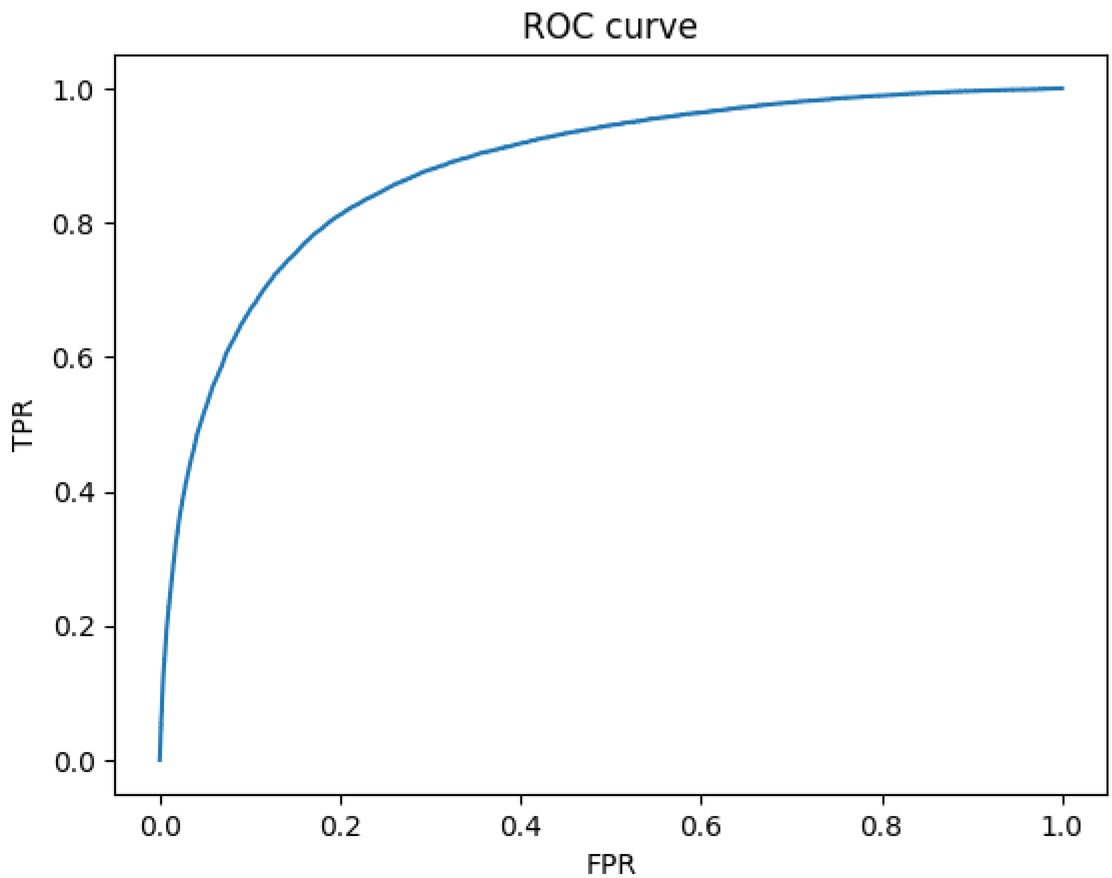
```python
# from sklearn.linear_model import
y_proba = model.predict_proba(X_test)
y_proba.shape, y_test.shape
```

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

Out[103]: ((118809, 2), (118809, 1))

In [104…

```python
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()
```



In [105…

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

Out[105]:

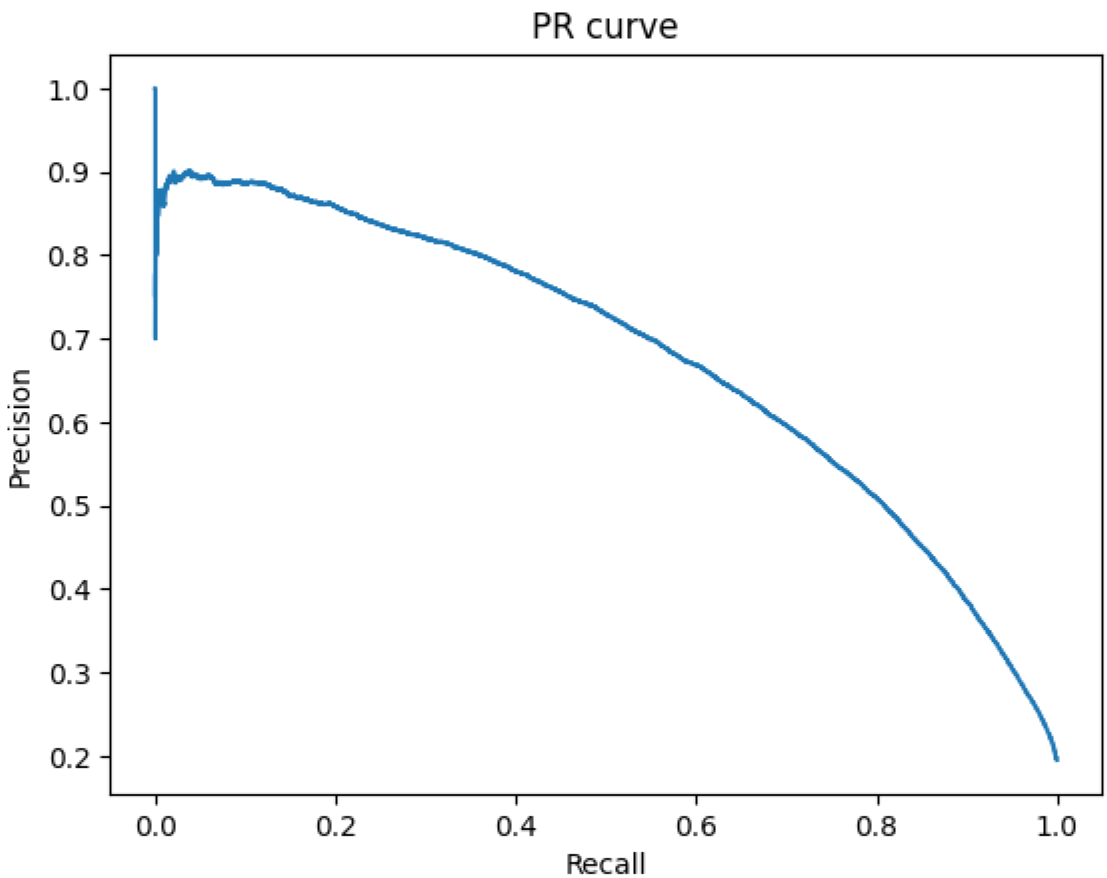
```
0.8809357325312589
```

0.88 is a good AUC score.

In [106…

```
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()
```

```
0.6795425892445759
```



Precision-recall curves are used when the imbalance in dataset is huge, but in this case since we used weights to reduce class imbalance problem we can refer to ROC curve.

Top 10 most important features affecting loan payment are-employment title, zipcode, loan sub_grade, term duration (36 or 60 months), no. of open accounts, dti ratio, revolving line utilization rate, loan amount, home ownership status and number of public derogatory records.