In [2]:
```python
# useful imports
import numpy as np, seaborn as sns, pandas as pd, matplotlib.pyplot as plt, scipy
df = pd.read_csv('data.csv')
```

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

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

Shape is 3,96,030 rows and 27 columns

In [7]:
```python
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
```

In [ ]:
```python
# df.describe()
```

In [10]:
```python
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	318

In [ ]:

```
# Source Name: Split and extract features out of destination. City-place-code (State)
trip_df[['source_corridor','source_state','s']] = trip_df['source_name'].str.split(r"[\(\)]",rege
trip_df.drop(['s'],axis=1,inplace=True)
trip_df[['source_corridor','source_state']].head()
```

In [205…

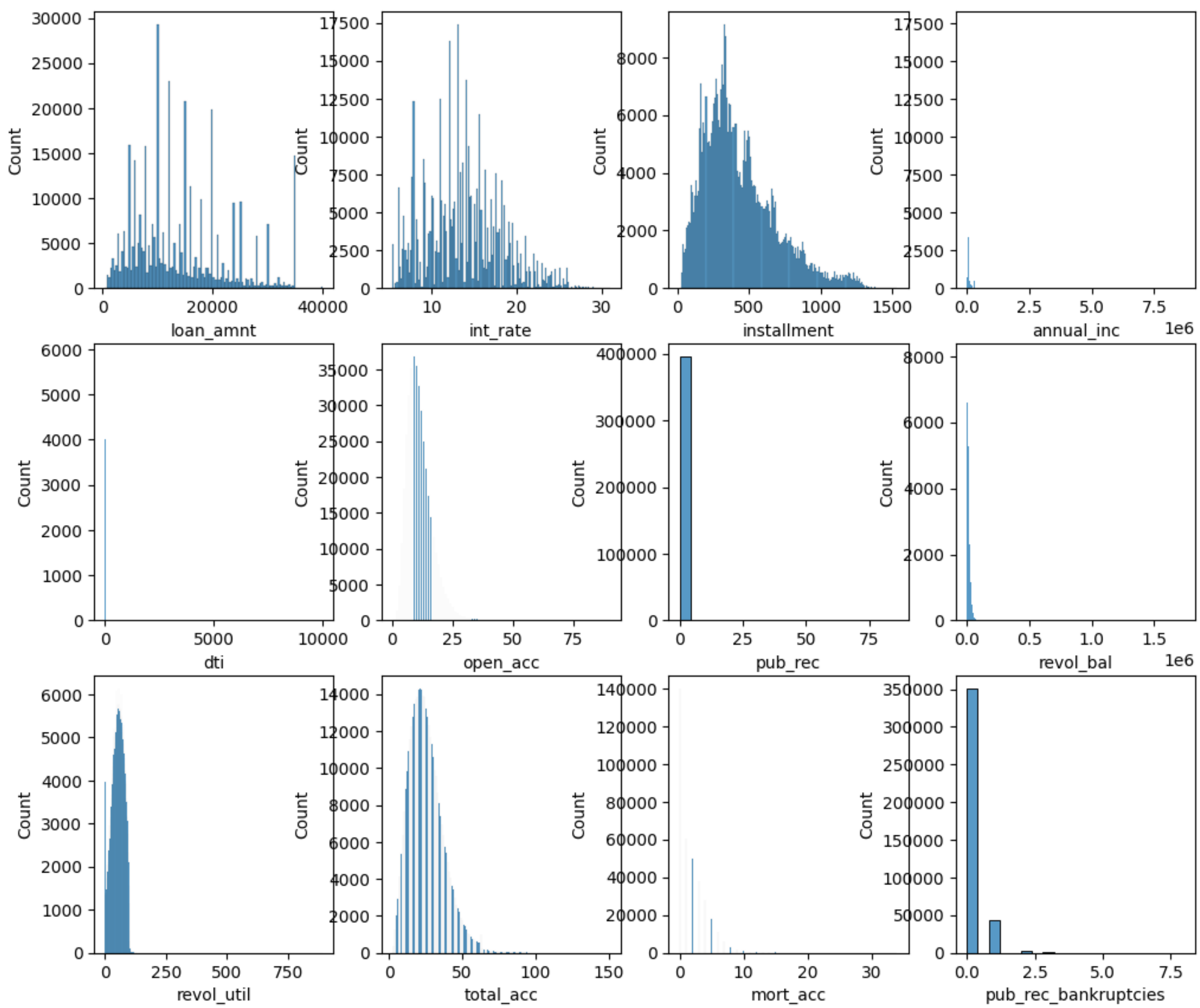
```
continuous_cols = df.columns[df.dtypes != 'object']
continuous_cols
```

Out[205]:

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

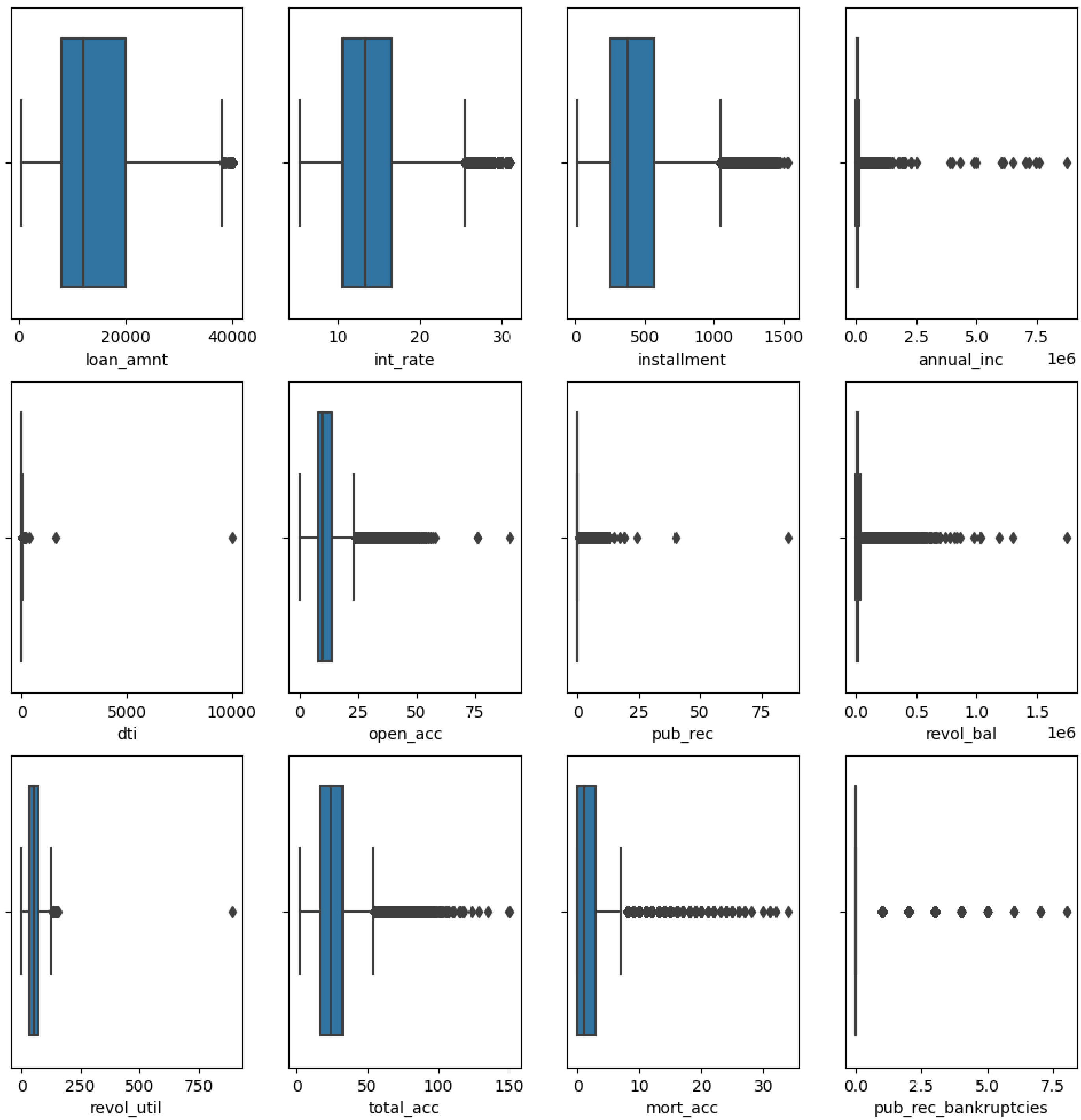
In [12]:

```
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]:
```python
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]:
```python
# We'll use IQR (inter-quartile range proximity) outlier detection method for skewed distributior
# 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]:

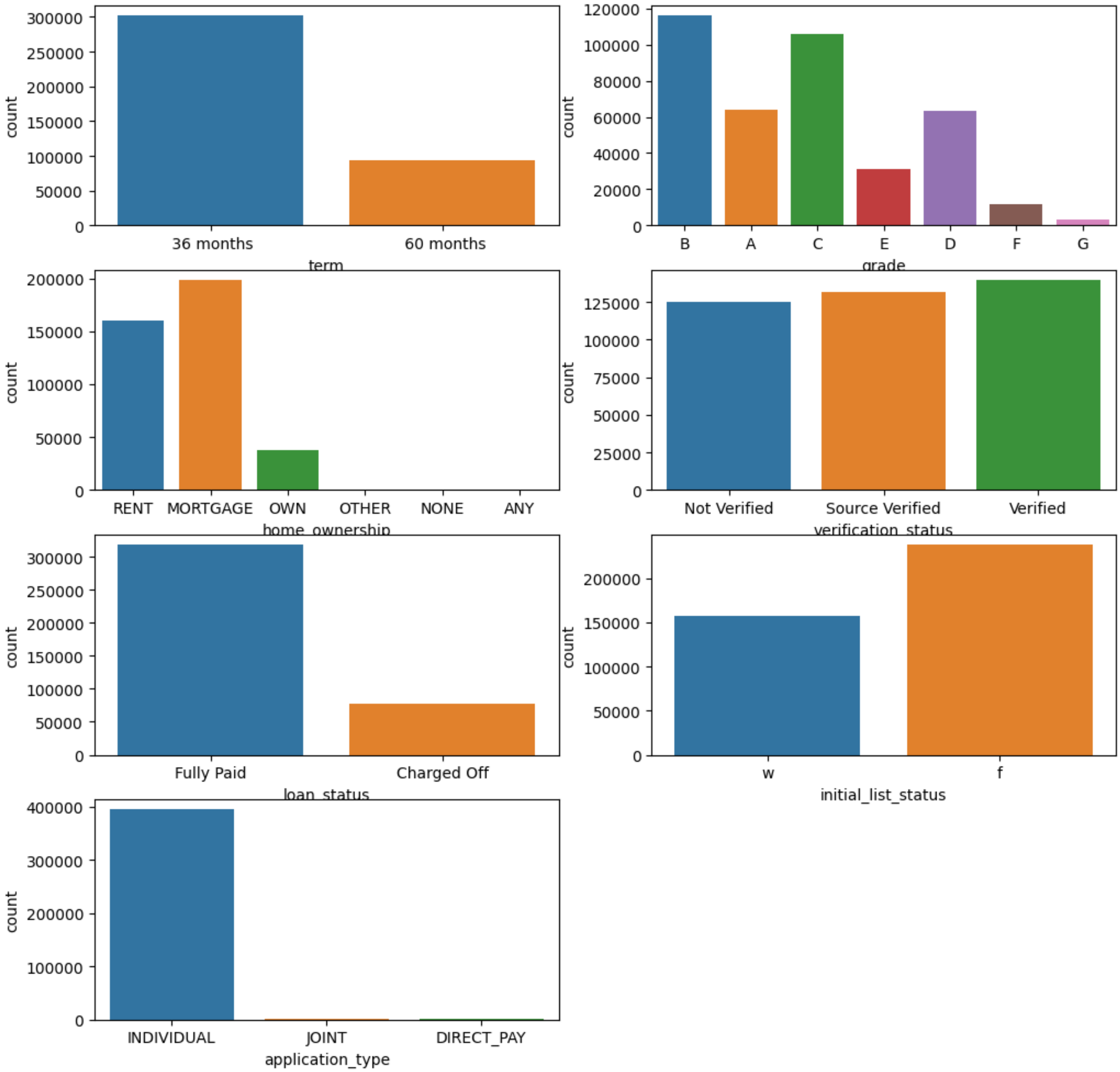
```
categorical_cols = df.columns[df.dtypes == "object"]
trim_categorical_cols = categorical_cols.drop(['sub_grade','emp_title','emp_length','issue_d','pu
                                               'title','earliest_cr_line','address'])
trim_categorical_cols
```

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()
```



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
```

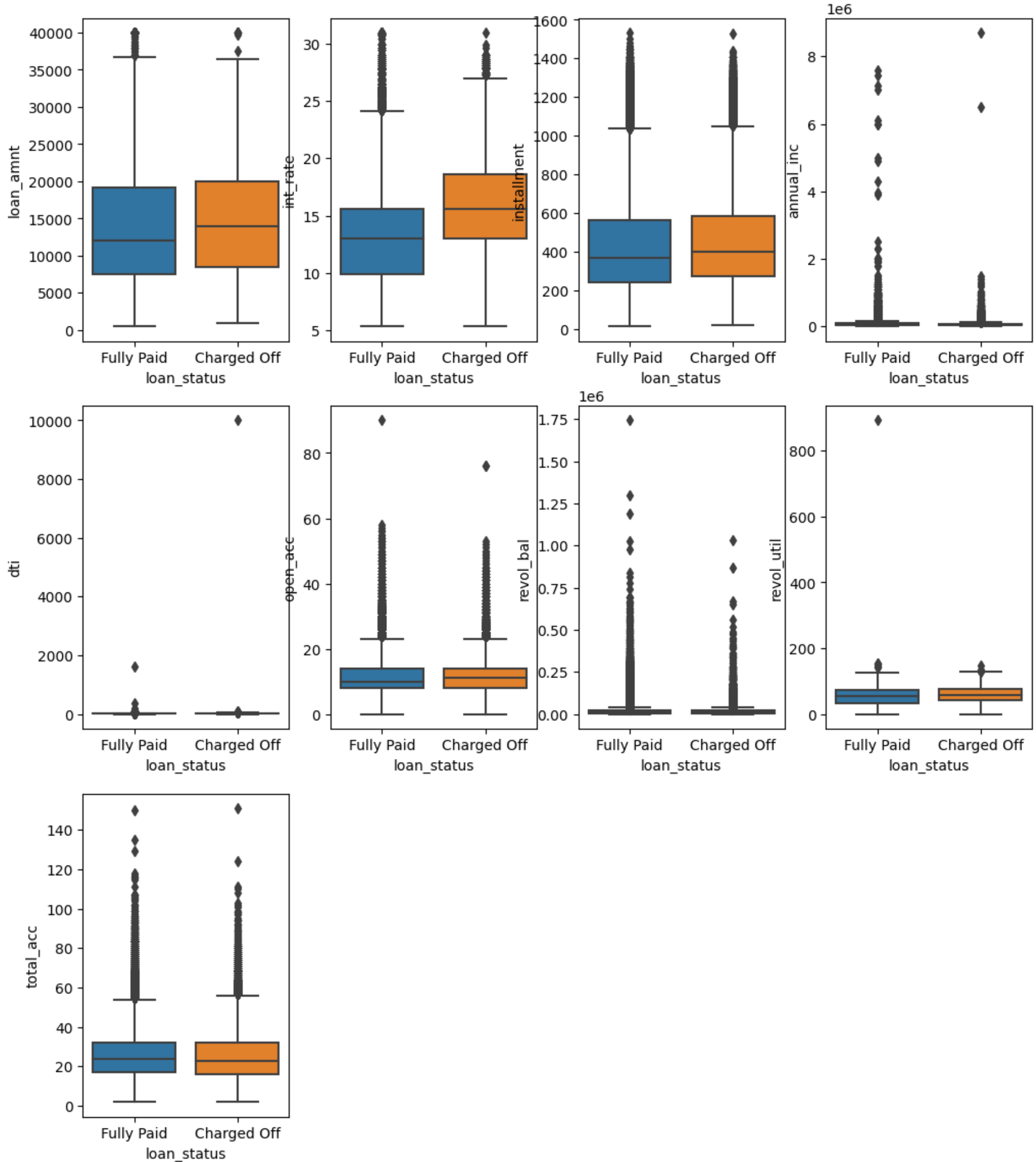
There are most number of customers with 10+ years of employment.

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
```

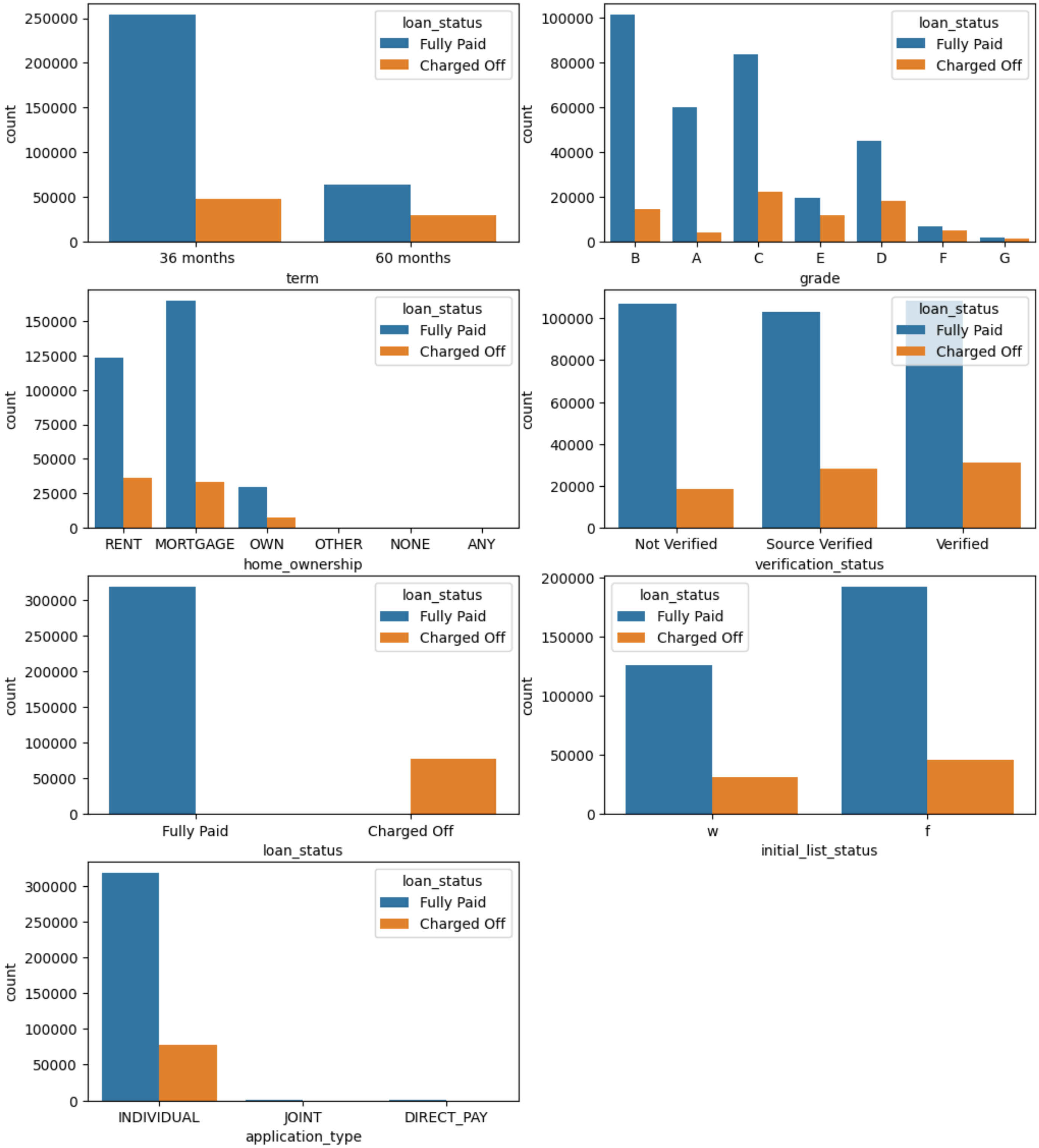
Most common purposes for getting loans were debt consolidation, credit card purchase, home improvement and other household or business purchases.

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()
```



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()
```

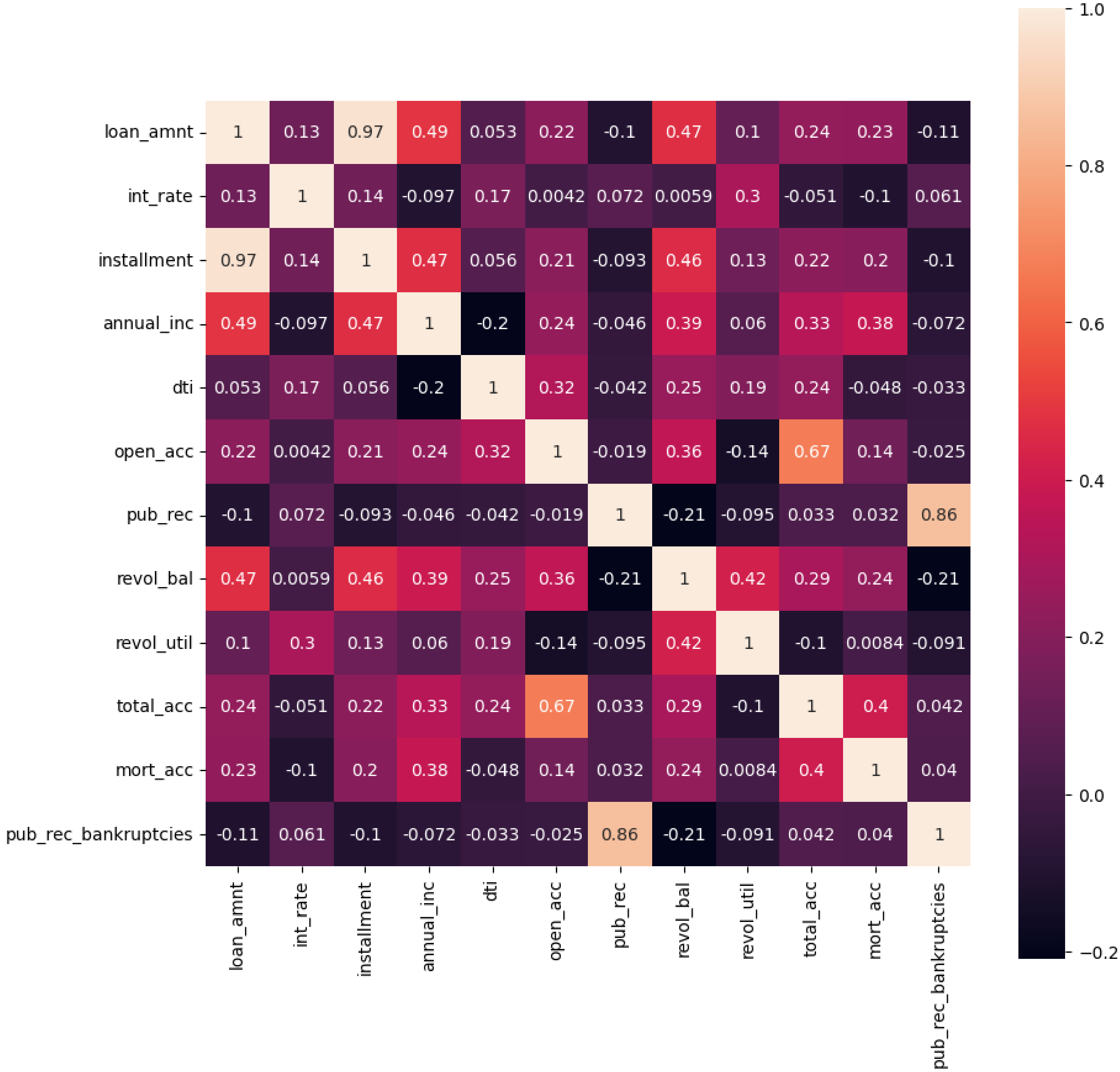


There are more customers who have fully paid their loans than those charged off in all categories.

In [40]:
```python
# 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: >



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
```

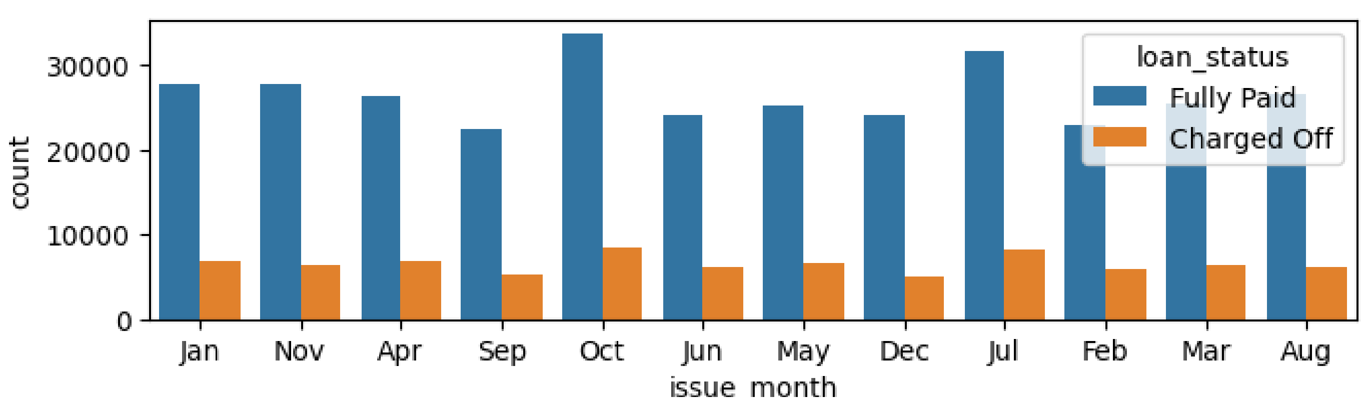
In [260…
```
df2 = df.join(df['issue_d'].str.split('-',1, expand=True).rename(columns={0:'issue_month', 1:'is
df2[['issue_d','issue_month','issue_year']].head()
```

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

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…
```python
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')
```

In [221…
```python
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 [268…
```python
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 "
```

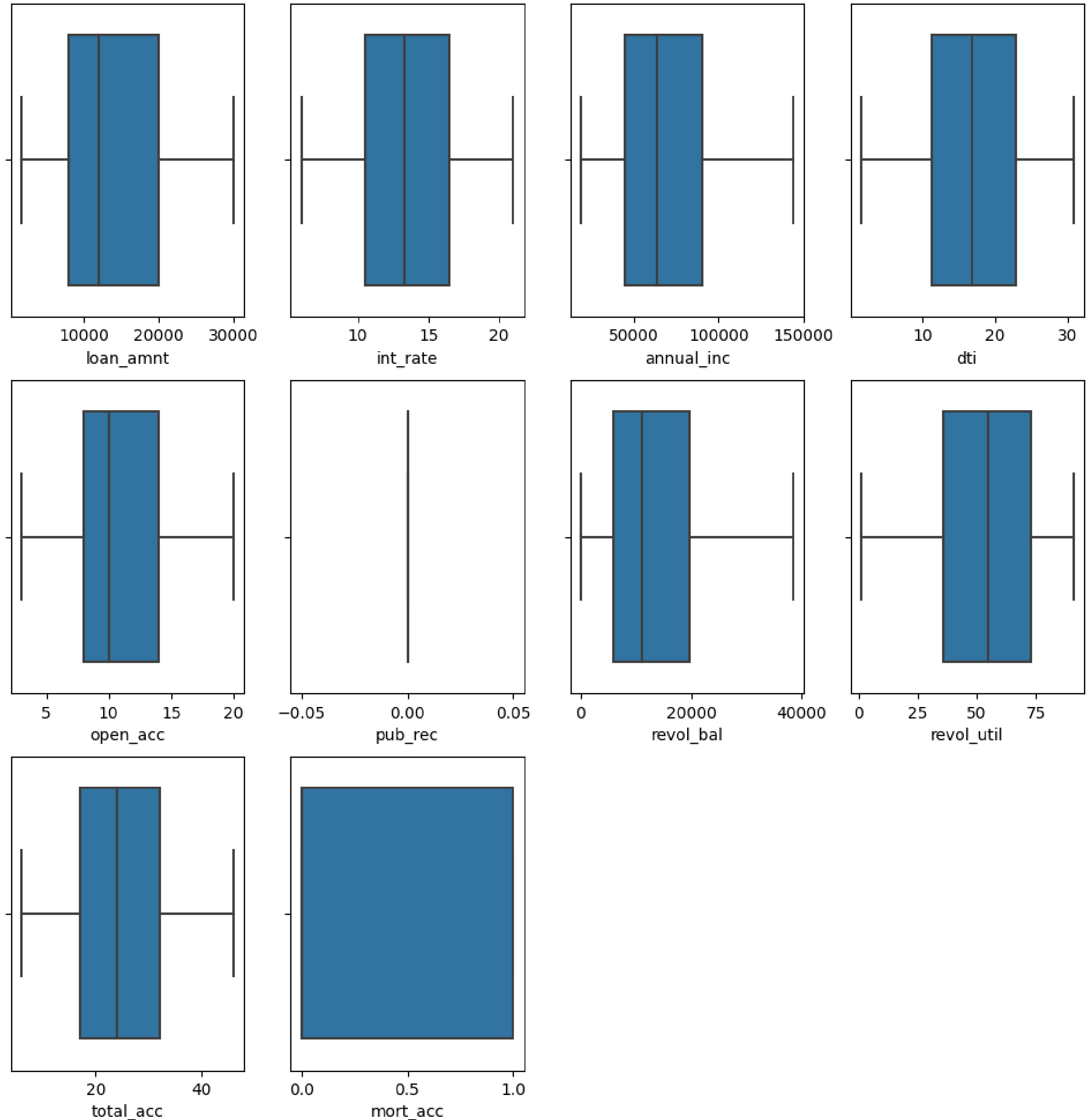
Out[268]:

	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()
```



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 [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 [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))
```

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
```

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 c
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: Us
erWarning: X does not have valid feature names, but LogisticRegression was fitted with feature n
ames
  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.

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

Out[360]:

	0	1
0	92653	2947
1	16786	6423

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
```

In [339…

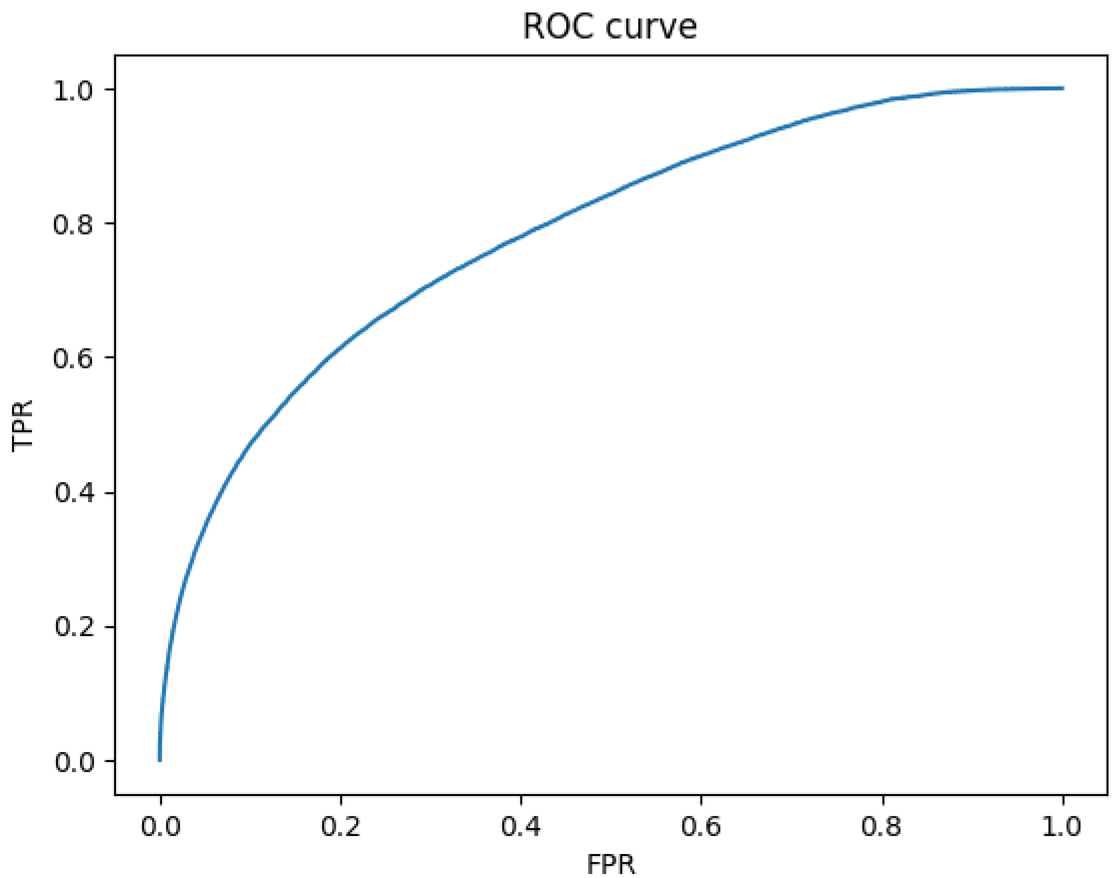
```
# from sklearn.linear_model import
y_proba = model.predict_proba(X_test)
y_proba.shape, y_test.shape
```

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



In [342…

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

Out[342]:

```
0.783562142066876
```

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

Customers from certain zipcodes are more likely to pay loans.

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