Problem Statement

- LoanTap is an online platform committed to delivering customized loan products to millennials. They innovate in an otherwise dull loan segment, to deliver instant, flexible loans on consumer friendly terms to salaried professionals and businessmen.The data science team at LoanTap is building an underwriting layer to determine the creditworthiness of MSMEs as well as individuals.
- We'll be performing EDA to check impactful data variable, feature engineering, feature selection, outlier detection and feature importance.
- Then We'll train a logistic model to answer the question of whom to give the loan.
- Evaluation metrics like precision, recall will be used to reduce false positives and also not lose on opportunity cost and answer some business problems based on our model output.

In [1324]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_selection import SelectKBest,f_classif
from sklearn.preprocessing import OneHotEncoder,StandardScaler,LabelEncoder
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score,recall_score,f1_score
from sklearn.metrics import confusion_matrix,precision_recall_curve,ConfusionMatrixD
```

In [1325]:

```
df = pd.read_csv('logistic_regression.txt')
```

In [1326]:

```
df = df[df['application_type']=='INDIVIDUAL']
```

In [1327]:

```
df.shape
```

Out[1327]:

```
(395319, 27)
```

In [1328]:

```python
df.describe(include='all')
```

Out[1328]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	en
count	395319.000000	395319	395319.000000	395319.000000	395319	395319	372466	
unique	NaN	2	NaN	NaN	7	35	172922	
top	NaN	36 months	NaN	NaN	B	B3	Teacher	
freq	NaN	301558	NaN	NaN	115900	26630	4372	
mean	14108.685770	NaN	13.633423	431.691933	NaN	NaN	NaN	
std	8354.421699	NaN	4.468309	250.646290	NaN	NaN	NaN	
min	500.000000	NaN	5.320000	16.080000	NaN	NaN	NaN	
25%	8000.000000	NaN	10.490000	250.330000	NaN	NaN	NaN	
50%	12000.000000	NaN	13.330000	375.430000	NaN	NaN	NaN	
75%	20000.000000	NaN	16.490000	567.220000	NaN	NaN	NaN	
max	40000.000000	NaN	30.990000	1533.810000	NaN	NaN	NaN	

11 rows × 27 columns

In [1329]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 395319 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             395319 non-null  float64
 1   term                  395319 non-null  object 
 2   int_rate              395319 non-null  float64
 3   installment           395319 non-null  float64
 4   grade                 395319 non-null  object 
 5   sub_grade             395319 non-null  object 
 6   emp_title             372466 non-null  object 
 7   emp_length            377092 non-null  object 
 8   home_ownership        395319 non-null  object 
 9   annual_inc            395319 non-null  float64
 10  verification_status   395319 non-null  object 
 11  issue_d               395319 non-null  object 
 12  loan_status           395319 non-null  object 
 13  purpose               395319 non-null  object 
 14  title                 393654 non-null  object 
 15  dti                   395319 non-null  float64
 16  earliest_cr_line      395319 non-null  object 
 17  open_acc              395319 non-null  float64
 18  pub_rec               395319 non-null  float64
 19  revol_bal             395319 non-null  float64
 20  revol_util            395043 non-null  float64
 21  total_acc             395319 non-null  float64
 22  initial_list_status   395319 non-null  object 
 23  application_type      395319 non-null  object 
 24  mort_acc              357524 non-null  float64
 25  pub_rec_bankruptcies  394784 non-null  float64
 26  address               395319 non-null  object 
dtypes: float64(12), object(15)
memory usage: 84.4+ MB
```

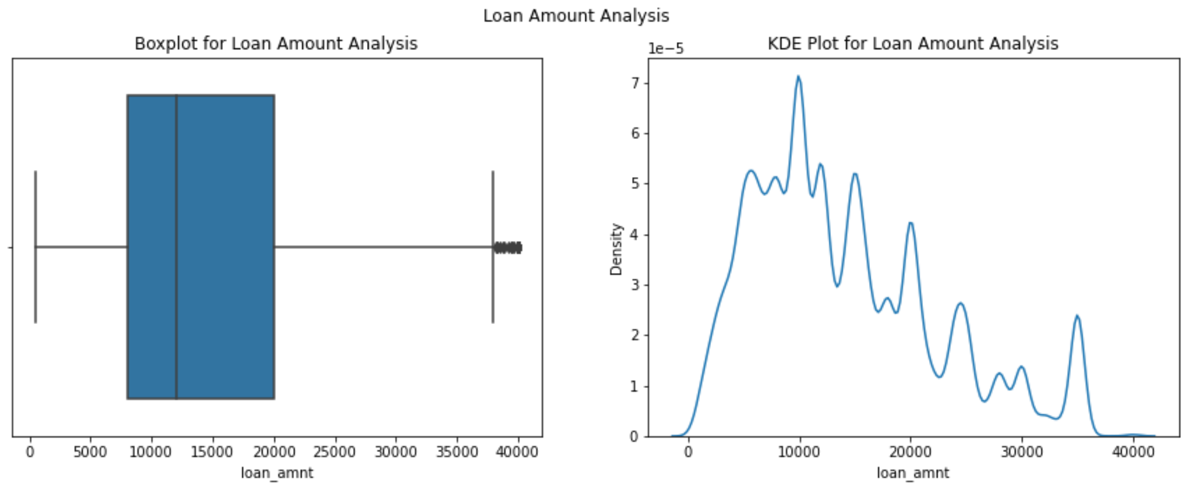
Univariate Analysis

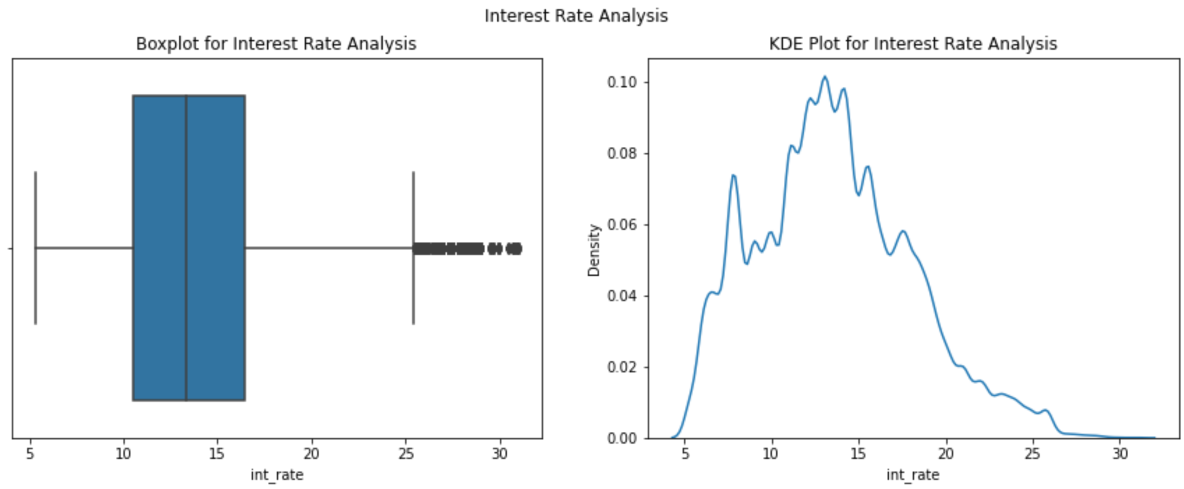
In [1330]:

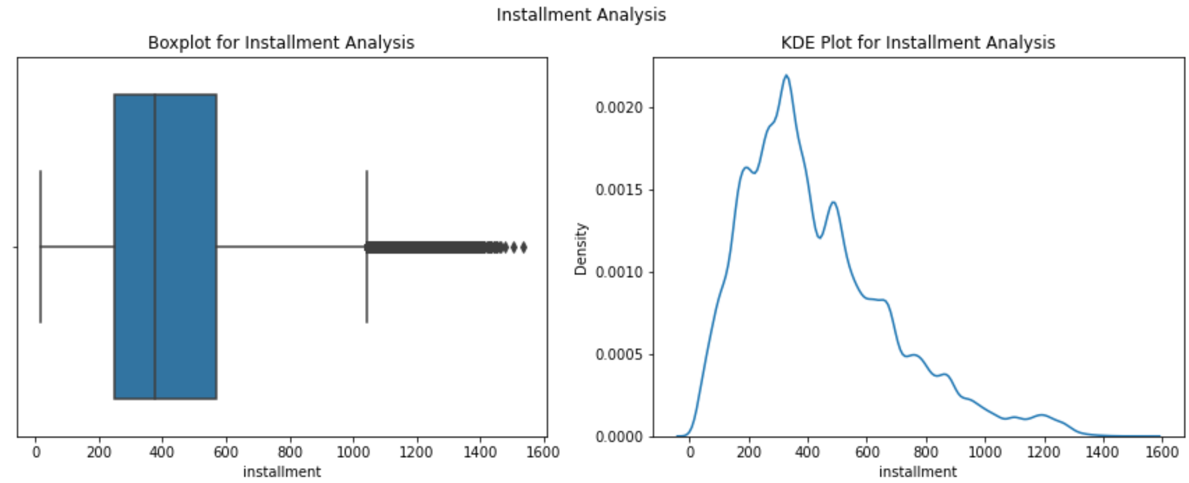
```
def plot_univariate_numerical(df,key,title):
    fig, axes = plt.subplots(1, 2, figsize=(15, 5))
    fig.suptitle(title)


    sns.boxplot(ax=axes[0], x=df[key])
    axes[0].set_title('Boxplot for {}'.format(title))


    sns.kdeplot(ax=axes[1], x=df[key])
    axes[1].set_title('KDE Plot for {}'.format(title))
```

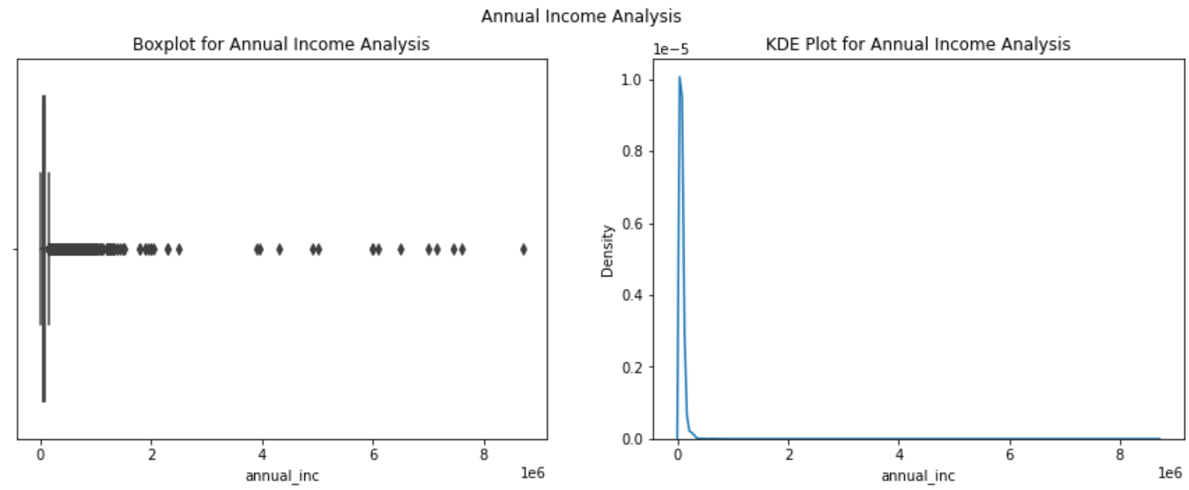
In [1331]:

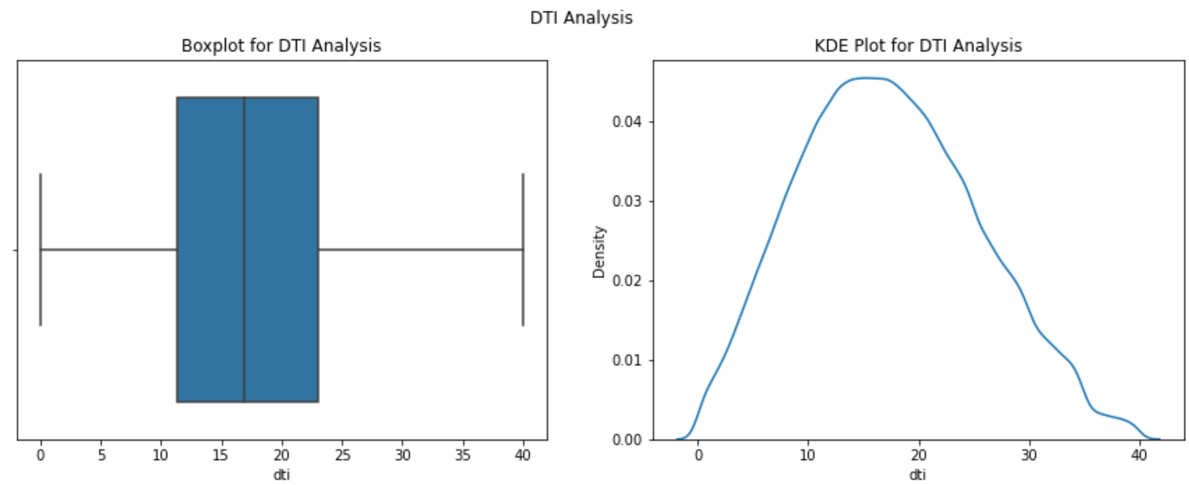
```
column = {'loan_amnt':'Loan Amount Analysis','int_rate':'Interest Rate Analysis',
          'installment':'Installment Analysis','annual_inc':'Annual Income Analysis'
         'open_acc':'Open Account Analysis','pub_rec':'Number of derogatory Public R
         'revol_util':'Revolving Line utitlization rate','revol_bal':'Total credit r
         'total_acc':'Total number of credit line',
         'mort_acc':'Number of Mortgage Account','pub_rec_bankruptcies':'Number of p


for key,desc in column.items():
    plot_univariate_numerical(df,key,desc)
```

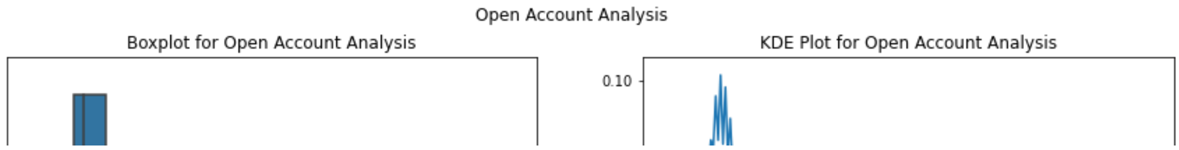


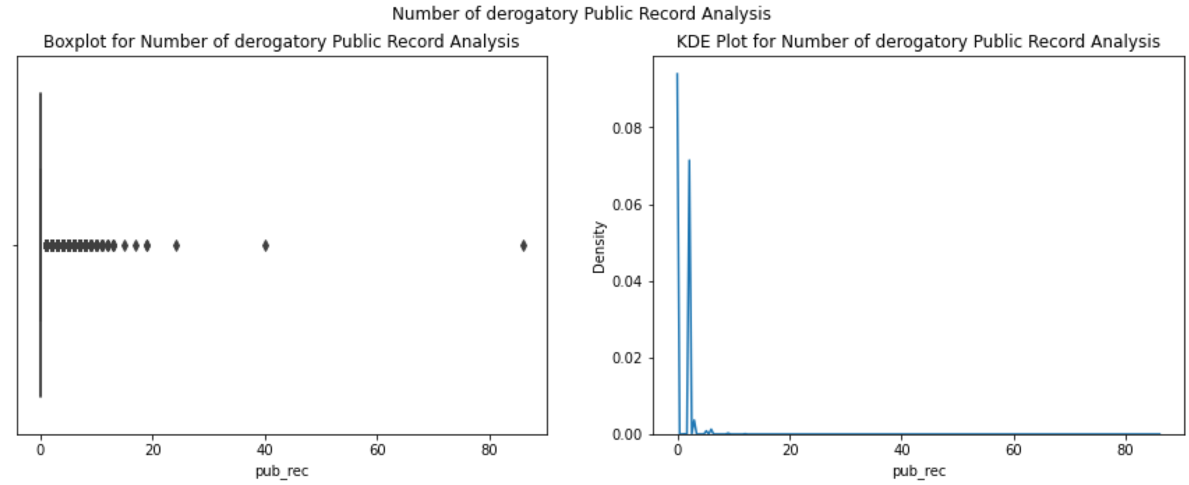


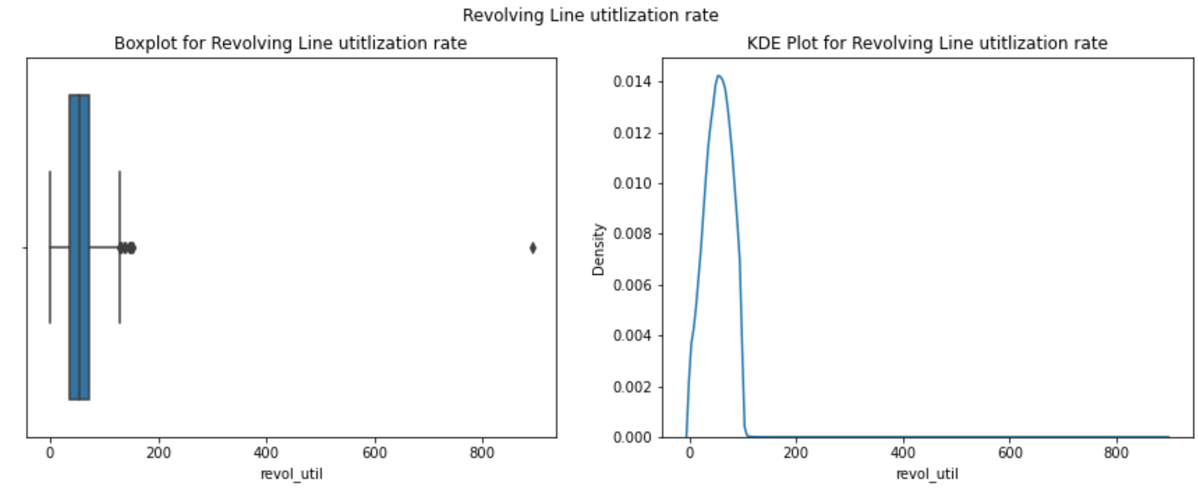

Installment Analysis
Boxplot for Installment Analysis
installment
KDE Plot for Installment Analysis
Density
installment

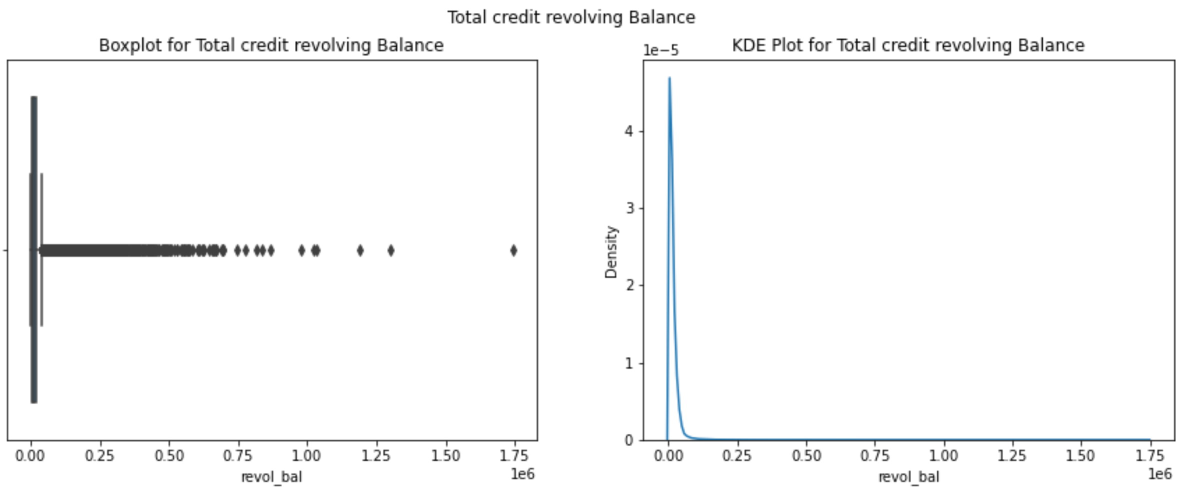

Annual Income Analysis
Boxplot for Annual Income Analysis
annual_inc
1e6
KDE Plot for Annual Income Analysis
1e−5
Density
annual_inc
1e6

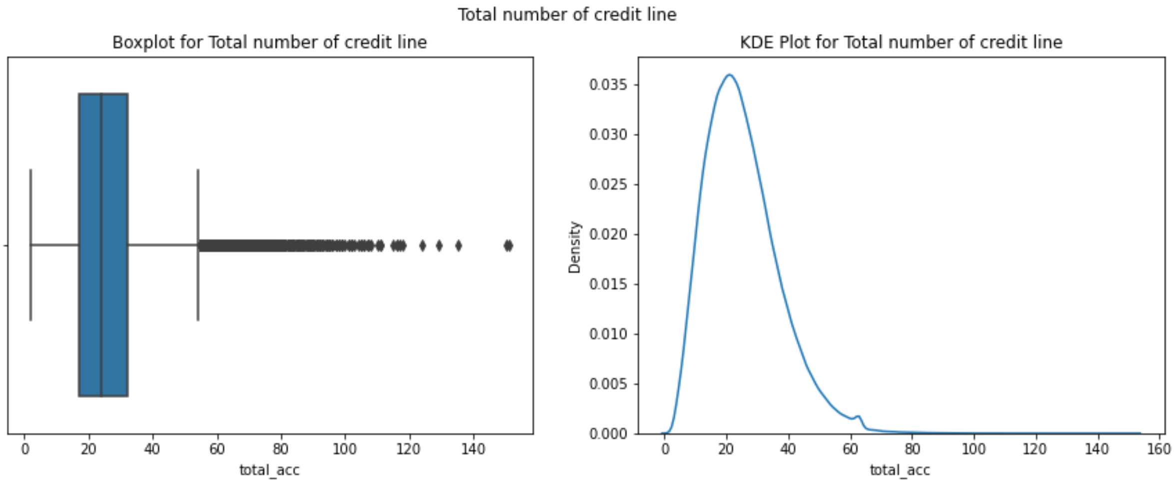

DTI Analysis
Boxplot for DTI Analysis
dti
KDE Plot for DTI Analysis
Density
dti

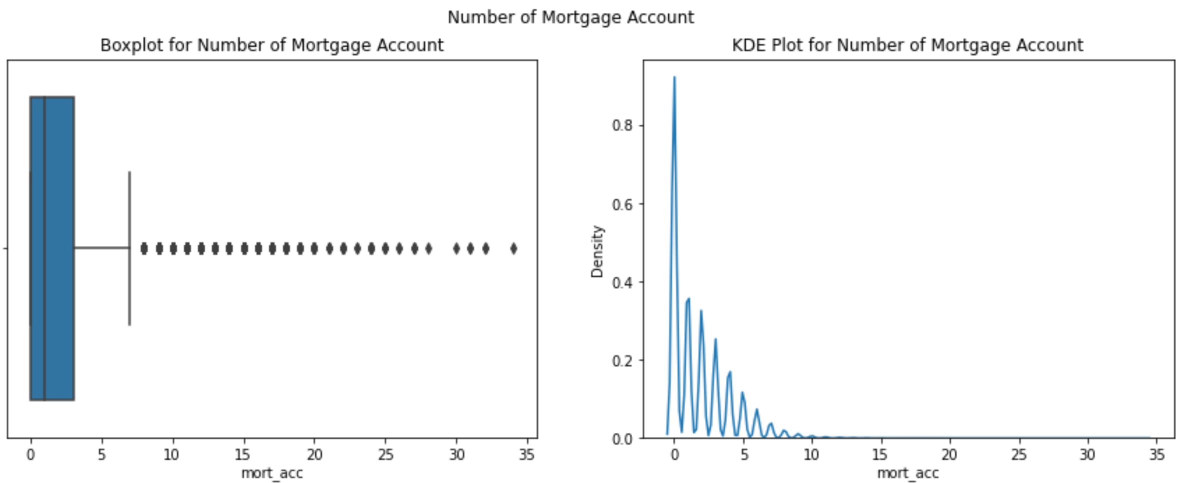

Open Account Analysis
Boxplot for Open Account Analysis
KDE Plot for Open Account Analysis
0.10

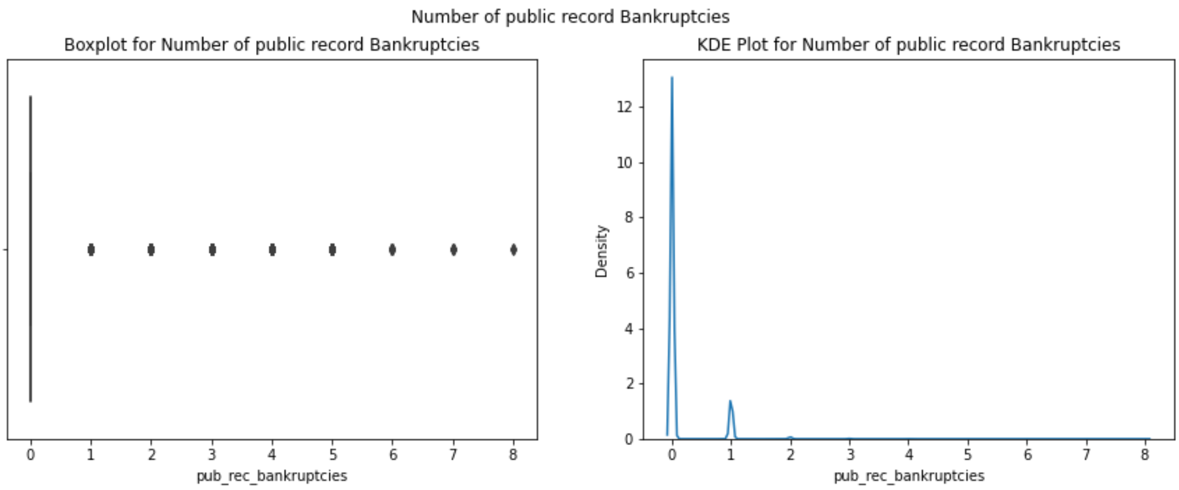

Number of derogatory Public Record Analysis
Boxplot for Number of derogatory Public Record Analysis
KDE Plot for Number of derogatory Public Record Analysis
Density
pub_rec
pub_rec


Revolving Line utitlization rate
Boxplot for Revolving Line utitlization rate
KDE Plot for Revolving Line utitlization rate
Density
revol_util
revol_util


Total credit revolving Balance
Boxplot for Total credit revolving Balance
KDE Plot for Total credit revolving Balance
1e−5
Density
revol_bal
1e6
revol_bal
1e6

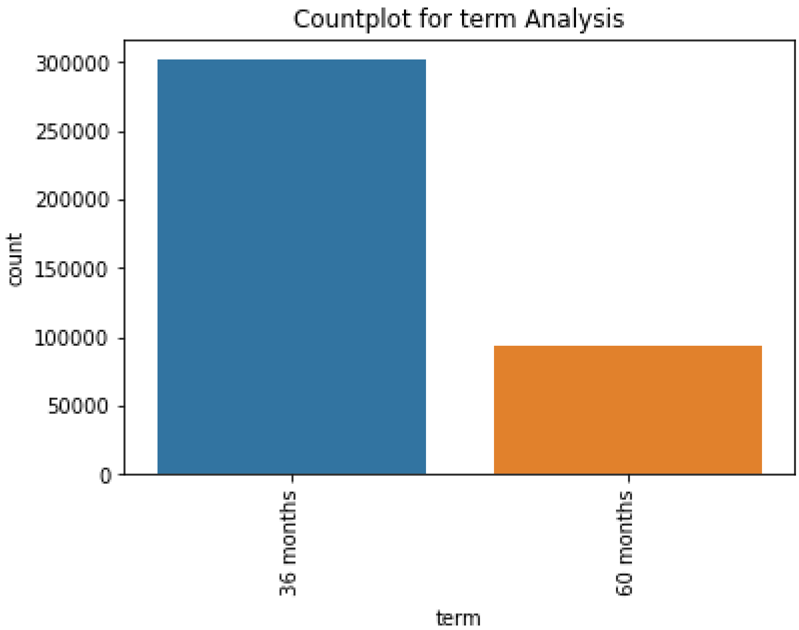

Total number of credit line
Boxplot for Total number of credit line
KDE Plot for Total number of credit line
total_acc
Density
total_acc

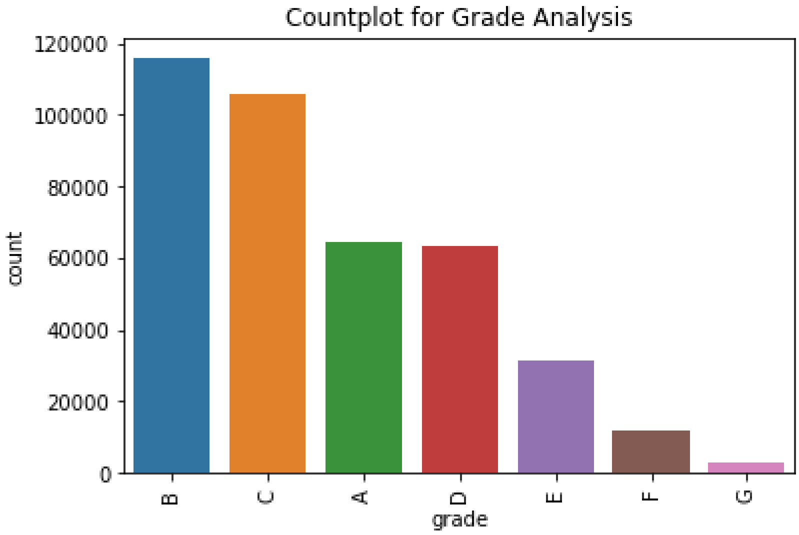

Number of Mortgage Account
Boxplot for Number of Mortgage Account
KDE Plot for Number of Mortgage Account
mort_acc
Density
mort_acc


Number of public record Bankruptcies
Boxplot for Number of public record Bankruptcies
KDE Plot for Number of public record Bankruptcies
pub_rec_bankruptcies
Density
pub_rec_bankruptcies

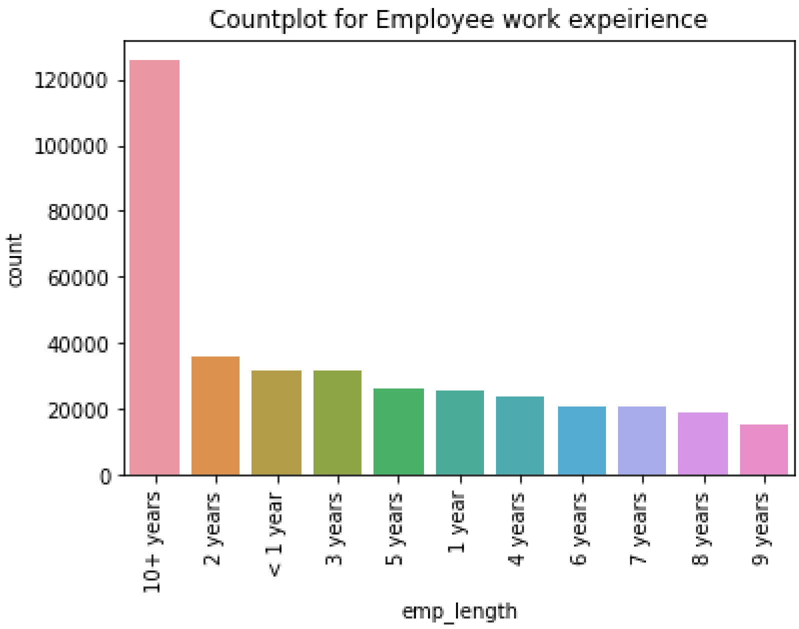
In [1332]:

```
def plot_univariate_categorical(df,key,title):

    sns.countplot(x=df[key],
                  order = df[key].value_counts().index)
    plt.title('Countplot for {}'.format(title))
    plt.xticks(rotation=90)

    plt.show()
#axes[0].set_title()

categorical_columns = {'term':'term Analysis','grade':'Grade Analysis',
                       'emp_length':'Employee work expeirience','home_ownership':'Hom
                       'verification_status':'Verification Status Analysis',
                       'loan_status':'loan status Analysis','purpose':'Purpose of Loa
                       'initial_list_status':'Initial List Status Analysis'}

for key,desc in categorical_columns.items():
    plot_univariate_categorical(df,key,desc)
```

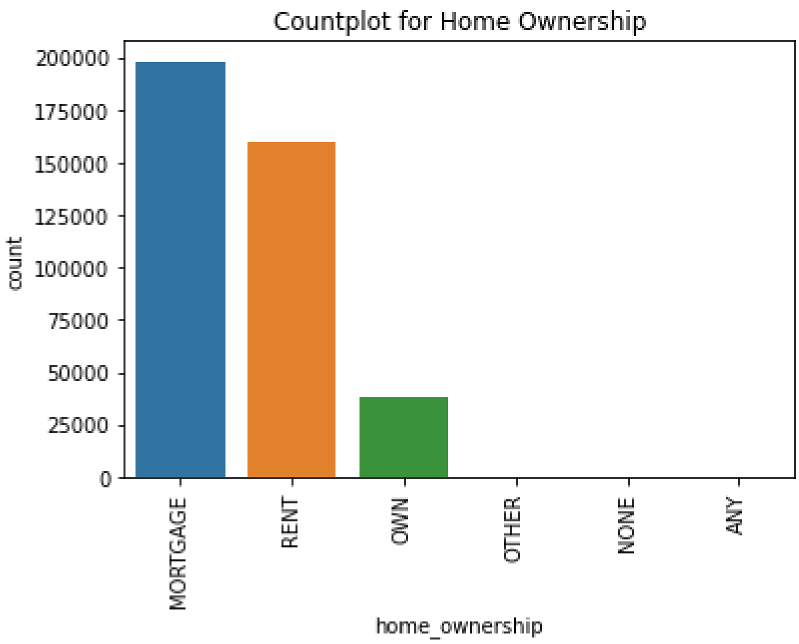




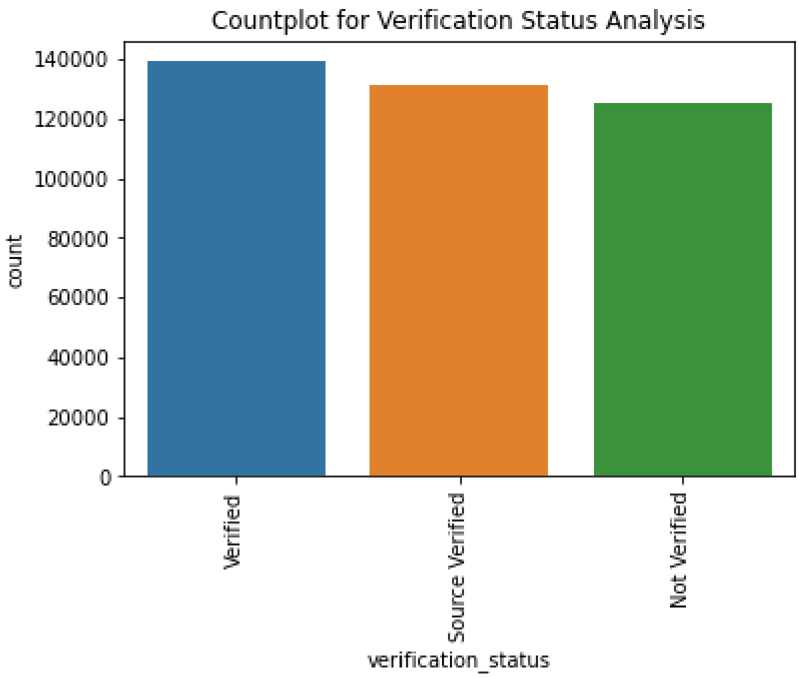
Countplot for Grade Analysis
count
0
20000
40000
60000
80000
100000
120000
B
C
A
D
E
F
G
grade

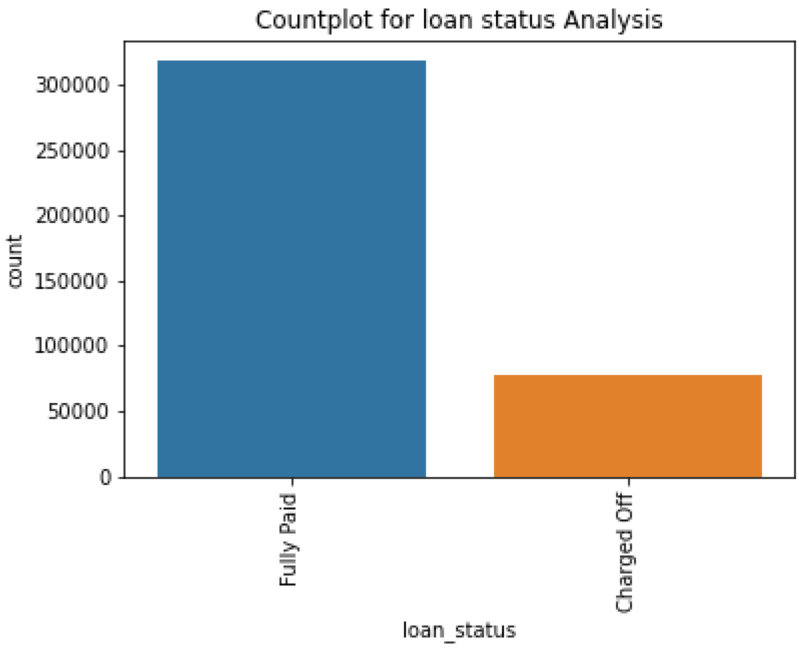


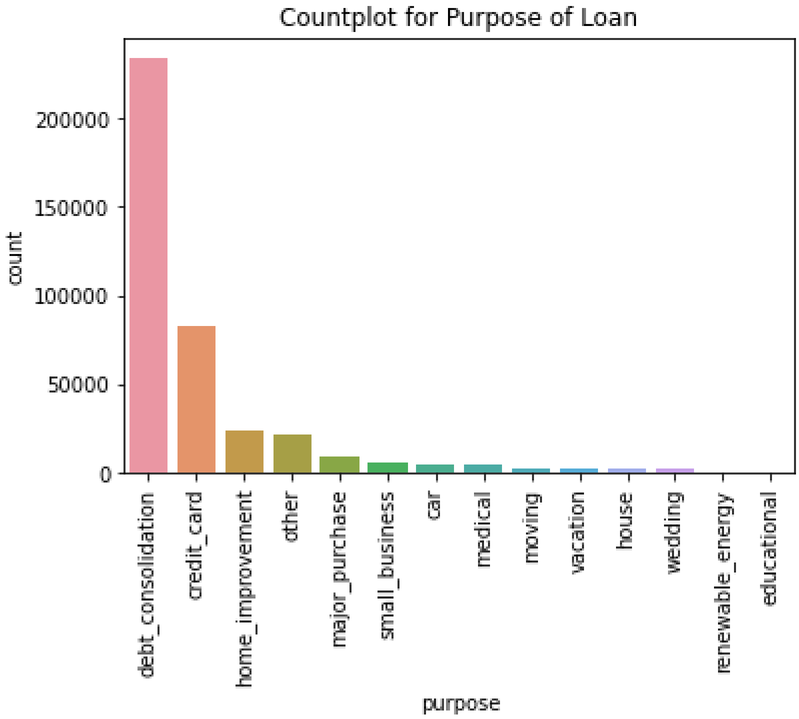
Countplot for Employee work expeirience
count
0
20000
40000
60000
80000
100000
120000
10+ years
2 years
< 1 year
3 years
5 years
1 year
4 years
6 years
7 years
8 years
9 years
emp_length

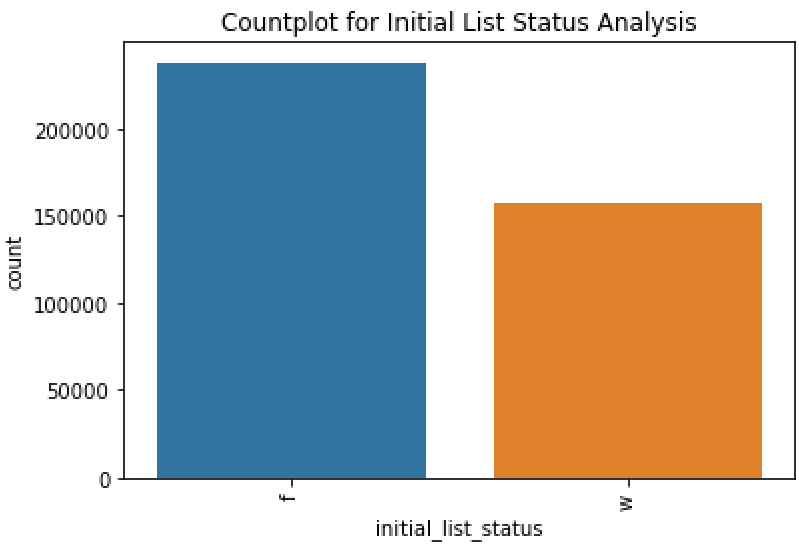


Countplot for Home Ownership
count
0
25000
50000
75000
100000
125000
150000
175000
200000
MORTGAGE
RENT
OWN
OTHER
NONE
ANY
home_ownership


Countplot for Verification Status Analysis
140000
120000
100000
80000
60000
40000
20000
0
count
Verified
Source Verified
Not Verified
verification_status


Countplot for loan status Analysis
300000
250000
200000
150000
100000
50000
0
count
Fully Paid
Charged Off
loan_status


Countplot for Purpose of Loan
count
200000
150000
100000
50000
0
debt_consolidation
credit_card
home_improvement
other
major_purchase
small_business
car
medical
moving
vacation
house
wedding
renewable_energy
educational
purpose

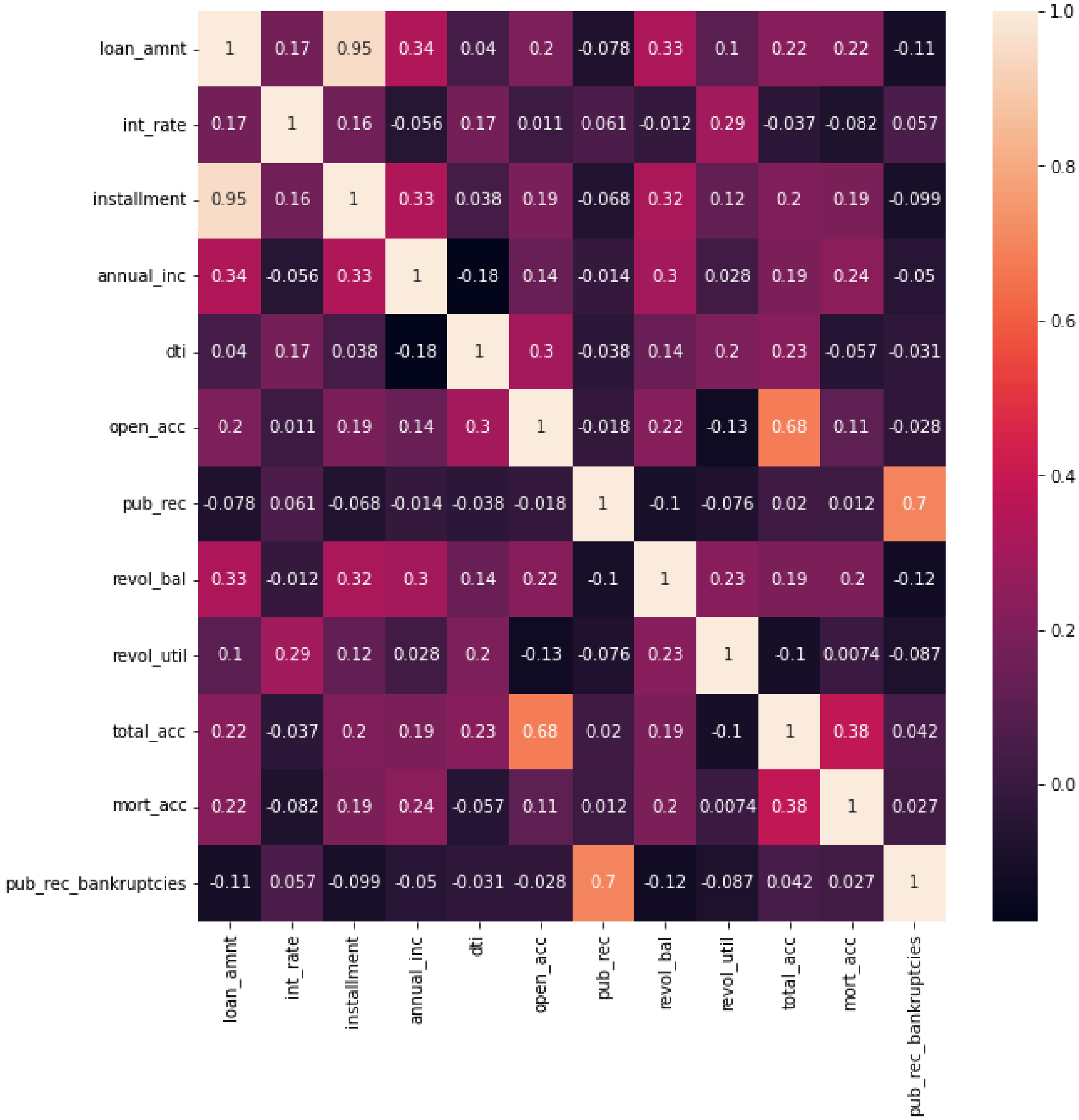

Countplot for Initial List Status Analysis
count
200000
150000
100000
50000
0
f
w
initial_list_status

In [1333]:

```
## Removing useless columns
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(),annot=True)
```

Out[1333]:

```
<AxesSubplot:>
```



In [1334]:

```
df.columns
```

Out[1334]:

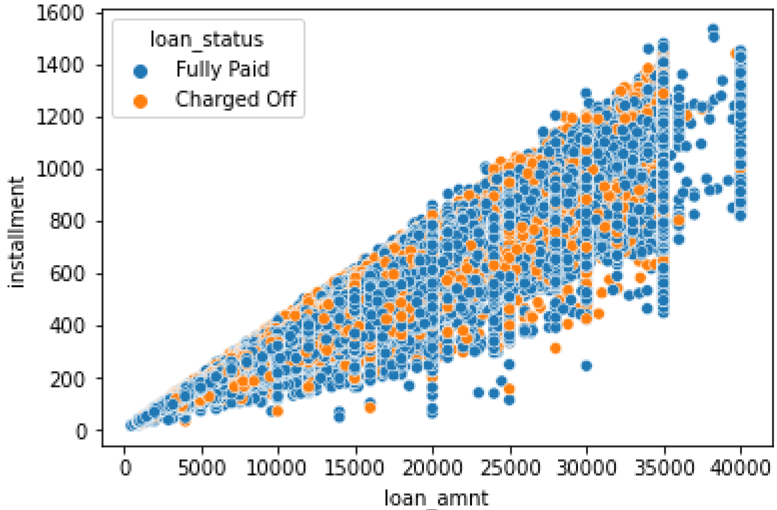
```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_g
rade',
       'emp_title', 'emp_length', 'home_ownership', 'annual_inc',
       'verification_status', 'issue_d', 'loan_status', 'purpose', 'ti
tle',
       'dti', 'earliest_cr_line', 'open_acc', 'pub_rec', 'revol_bal',
       'revol_util', 'total_acc', 'initial_list_status', 'application_
type',
       'mort_acc', 'pub_rec_bankruptcies', 'address'],
      dtype='object')
```

In [1335]:

```
sns.scatterplot(x=df['loan_amnt'],y=df['installment'],hue=df['loan_status'])
plt.plot()
```

Out[1335]:

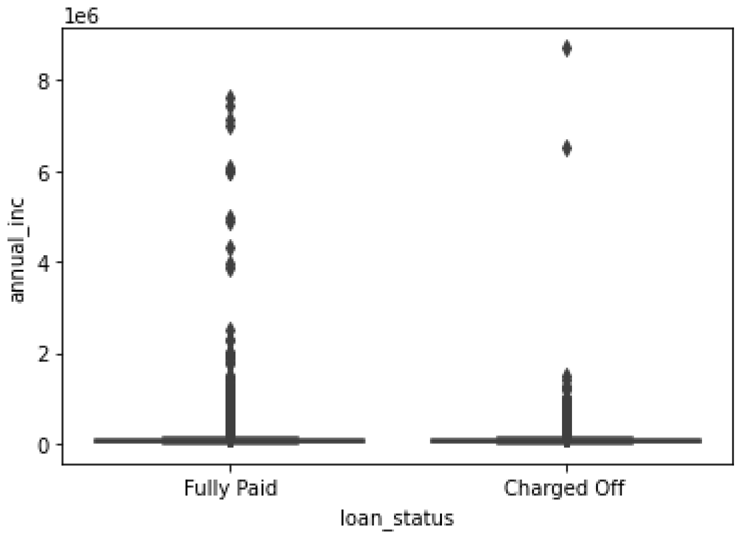
```
[]
```



In [1336]:

```
sns.boxplot(df['loan_status'],df['annual_inc'])
plt.show()
```

```
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. Fro
m version 0.12, the only valid positional argument will be `data`, and
passing other arguments without an explicit keyword will result in an
error or misinterpretation.
  warnings.warn(
```



In [1337]:

```
df.groupby(['home_ownership','loan_status'])['loan_status'].count()
```

Out[1337]:

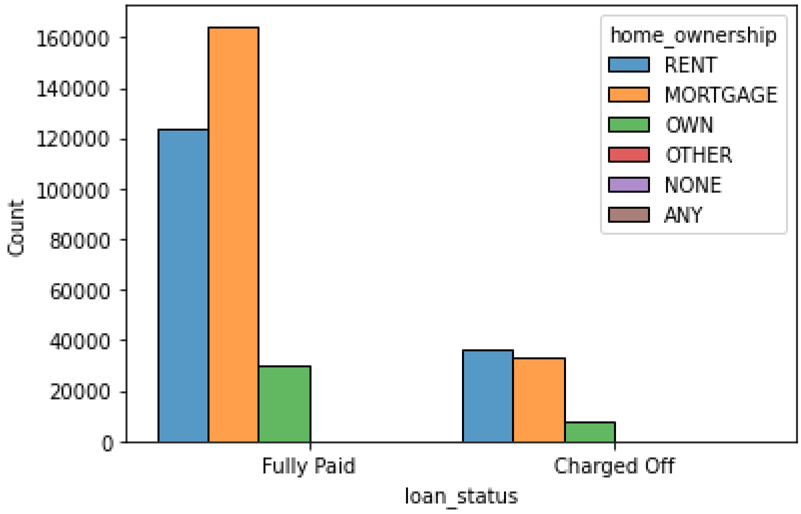
```
home_ownership  loan_status
ANY             Fully Paid          3
MORTGAGE        Charged Off     33564
                Fully Paid     164371
NONE            Charged Off         7
                Fully Paid         24
OTHER           Charged Off        16
                Fully Paid         96
OWN             Charged Off      7788
                Fully Paid      29877
RENT            Charged Off     36142
                Fully Paid     123431
Name: loan_status, dtype: int64
```

In [1338]:

```
sns.histplot(binwidth=1,
             x='loan_status',
             hue='home_ownership',
             data=df,
             stat="count",
             multiple="dodge")
plt.plot()
```

Out[1338]:

```
[]
```

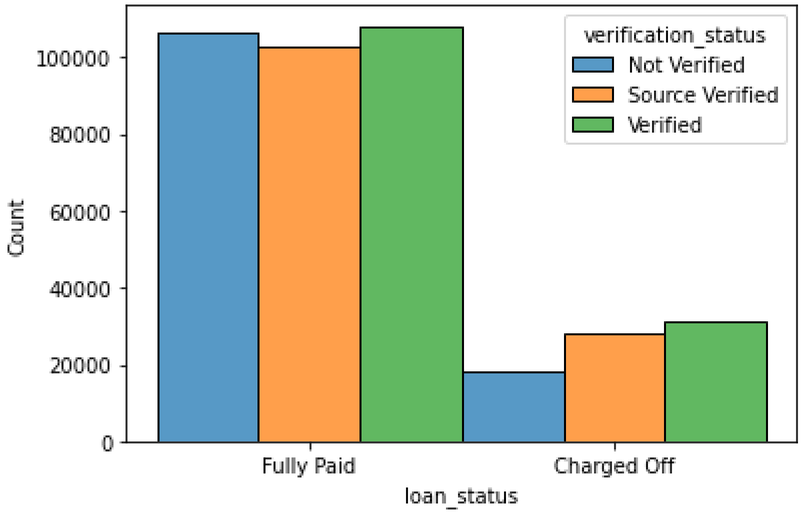


In [1339]:

```
sns.histplot(binwidth=1,
            x='loan_status',
            hue='verification_status',
            data=df,
            stat="count",
            multiple="dodge")
plt.plot()
```

Out[1339]:

```
[]
```

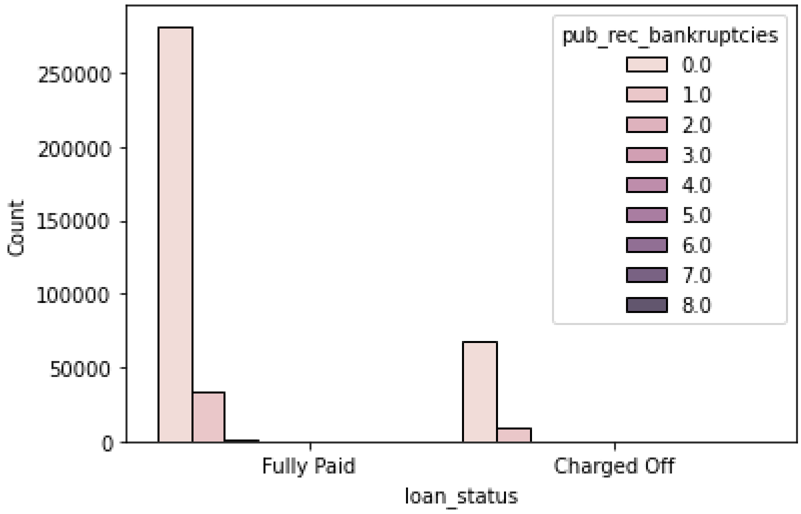


In [1340]:

```
sns.color_palette("hls", 8)
sns.histplot(binwidth=1,
            x='loan_status',
            hue='pub_rec_bankruptcies',
            data=df,
            stat="count",
            multiple="dodge"
            )
plt.plot()
```

Out[1340]:

```
[]
```



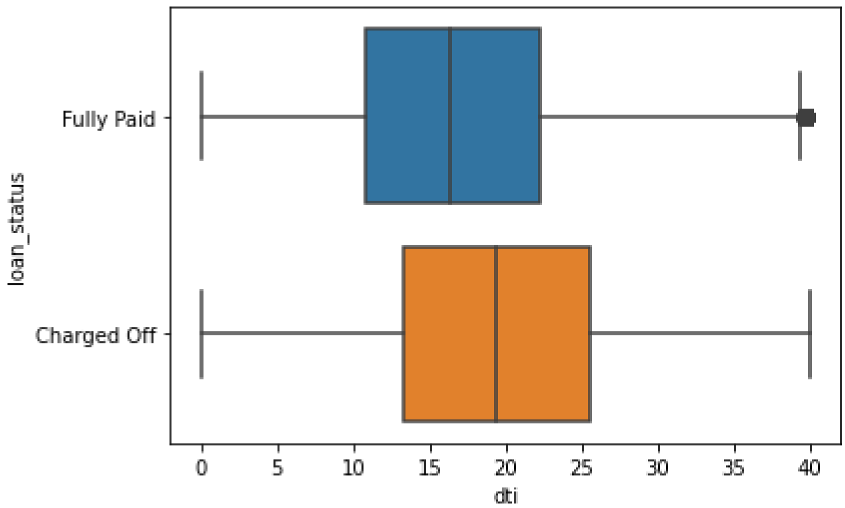
In [1341]:

```
sns.boxplot(df['dti'],df['loan_status'])
```

```
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. Fro
m version 0.12, the only valid positional argument will be `data`, and
passing other arguments without an explicit keyword will result in an
error or misinterpretation.
  warnings.warn(
```

Out[1341]:

```
<AxesSubplot:xlabel='dti', ylabel='loan_status'>
```



Comments:
- Most of the people take loan for debt consolidation, credit card payment and home improvement.
- People with High Annual income generally pay off their loan except few exceptions.
- The home_ownership is in the order of mortgage,rent and own.
- Pub_rec and pub_rec_bankruptcies are highly correlated.
- Mostly people with 10+ employment length apply for a loan.
- People with low dti are more likely to pay off the loan in comparision to people with high dti.

Data Preprocessing

In [1342]:

```
df.nunique()
```

Out[1342]:

```
loan_amnt                 1395
term                         2
int_rate                   566
installment              55624
grade                        7
sub_grade                   35
emp_title               172922
emp_length                  11
home_ownership               6
annual_inc               27147
verification_status          3
issue_d                    115
loan_status                  2
purpose                     14
title                    48817
dti                       3999
earliest_cr_line           684
open_acc                    61
pub_rec                     20
revol_bal                55592
revol_util                1226
total_acc                  118
initial_list_status          2
application_type             1
mort_acc                    33
pub_rec_bankruptcies         9
address                 392997
dtype: int64
```

In [1343]:

```
## Since application type only contains INDIVIDUAL value, we can drop it
df.drop('application_type',axis=1,inplace=True)
```

In [1344]:

```
# We'll drop address since it's not a criteria to judge whether to give loan or not
# emp_title is highly correlated to annual_inc,grade,sub_grade so we can drop it
# title column is filled by users and this can be replaced by purpose column which g
dropping_columns = ['address','emp_title','title','initial_list_status']
df.drop(dropping_columns,axis=1,inplace=True)
```

In [1345]:

```
## handling Missing values
df.isnull().sum()/len(df)*100
```

Out[1345]:

```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
installment             0.000000
grade                   0.000000
sub_grade               0.000000
emp_length              4.610707
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 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              0.069817
total_acc               0.000000
mort_acc                9.560633
pub_rec_bankruptcies    0.135334
dtype: float64
```

Handling Missing Values

In [1346]:

```
df['emp_length'].fillna('< 1 year',inplace=True)
df['pub_rec_bankruptcies'] = np.where(df['pub_rec_bankruptcies']>0, 1, 0)
df['mort_acc'] = np.where(df['mort_acc'].isnull(), 0,df['mort_acc'])
df['revol_util'] = np.where(df['revol_util'].isnull(),df['revol_util'].mean() , df['
```

In [1347]:

```
df.isnull().sum()/len(df)*100
```

Out[1347]:

```
loan_amnt               0.0
term                    0.0
int_rate                0.0
installment             0.0
grade                   0.0
sub_grade               0.0
emp_length              0.0
home_ownership          0.0
annual_inc              0.0
verification_status     0.0
issue_d                 0.0
loan_status             0.0
purpose                 0.0
dti                     0.0
earliest_cr_line        0.0
open_acc                0.0
pub_rec                 0.0
revol_bal               0.0
revol_util              0.0
total_acc               0.0
mort_acc                0.0
pub_rec_bankruptcies    0.0
dtype: float64
```

Outlier Detection

In [1348]:

```
df.info()
columns = ['loan_amnt','int_rate','installment','annual_inc','revol_bal','revol_util
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 395319 entries, 0 to 396029
Data columns (total 22 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             395319 non-null  float64
 1   term                  395319 non-null  object 
 2   int_rate              395319 non-null  float64
 3   installment           395319 non-null  float64
 4   grade                 395319 non-null  object 
 5   sub_grade             395319 non-null  object 
 6   emp_length            395319 non-null  object 
 7   home_ownership        395319 non-null  object 
 8   annual_inc            395319 non-null  float64
 9   verification_status   395319 non-null  object 
 10  issue_d               395319 non-null  object 
 11  loan_status           395319 non-null  object 
 12  purpose               395319 non-null  object 
 13  dti                   395319 non-null  float64
 14  earliest_cr_line      395319 non-null  object 
 15  open_acc              395319 non-null  float64
 16  pub_rec               395319 non-null  float64
 17  revol_bal             395319 non-null  float64
 18  revol_util            395319 non-null  float64
 19  total_acc             395319 non-null  float64
 20  mort_acc              395319 non-null  float64
 21  pub_rec_bankruptcies  395319 non-null  int64  
dtypes: float64(11), int64(1), object(10)
memory usage: 77.4+ MB
```

In [1349]:

```
def outlier_detection(df,key):
    percentile_25 = df[key].quantile(0.25)
    percentile_75 = df[key].quantile(0.75)
    iqr = percentile_75 - percentile_25
    upper_limit = percentile_75 + 1.5*iqr
    lower_limit = percentile_25 - 1.5*iqr
    upper_limit_outliers = df[df[key]>upper_limit]
    lower_limit_outliers = df[df[key]<lower_limit]
    return lower_limit,upper_limit
```

In [1350]:

```
lower_limit,upper_limit = outlier_detection(df,'annual_inc')
annual_inc_outliers = df[(df['annual_inc']>upper_limit) | (df['annual_inc']<lower_li
annual_inc_outliers
```

Out[1350]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_length	home_ownershi
87	30000.0	60 months	24.70	875.28	G	G1	5 years	MORTGAG
139	20000.0	36 months	10.37	648.83	B	B3	< 1 year	MORTGAG
195	24000.0	60 months	24.50	697.42	F	F3	10+ years	MORTGAG
221	25000.0	60 months	12.49	562.33	B	B5	10+ years	REN
228	35000.0	36 months	12.99	1179.12	C	C2	10+ years	MORTGAG
...	...	...	...	...	...	...	...	
395879	24000.0	60 months	13.99	558.32	C	C4	4 years	MORTGAG
395886	7000.0	36 months	7.90	219.04	A	A4	10+ years	MORTGAG
395892	35000.0	60 months	18.24	893.35	D	D5	10+ years	REN
395927	19600.0	36 months	11.99	650.91	B	B3	10+ years	MORTGAG
395987	14000.0	36 months	15.88	491.37	C	C4	3 years	REN

16690 rows × 22 columns

In [1351]:

```
lower_limit,upper_limit = outlier_detection(df,'int_rate')
int_rate_outliers = df[(df['int_rate']>upper_limit) | (df['int_rate']<lower_limit)]
int_rate_outliers
```

Out[1351]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_length	home_ownershi
96	12625.0	60 months	25.78	376.36	F	F5	7 years	MORTGAG
97	13400.0	60 months	25.83	399.86	G	G2	10+ years	MORTGAG
133	13075.0	60 months	27.31	401.68	G	G2	10+ years	MORTGAG
168	11800.0	60 months	28.99	374.49	G	G5	< 1 year	REN
204	34350.0	60 months	28.99	1090.13	G	G5	3 years	REN
...	...	...	...	...	...	...	...	
395425	14750.0	60 months	28.99	468.11	G	G5	10+ years	MORTGAG
395475	13075.0	60 months	26.57	395.90	F	F5	10+ years	MORTGAG
395566	10875.0	60 months	26.77	330.58	G	G1	9 years	REN
395628	14400.0	60 months	25.88	430.13	F	F4	2 years	REN
395976	16475.0	60 months	25.83	491.62	G	G2	7 years	OW

3728 rows × 22 columns

In [1352]:

```
lower_limit,upper_limit = outlier_detection(df,'installment')
installment_outliers = df[(df['installment']>upper_limit) | (df['installment']<lower
installment_outliers
```

Out[1352]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_length	home_ownershi
11	35000.0	36 months	14.64	1207.13	C	C3	8 years	MORTGAG
18	34000.0	36 months	7.90	1063.87	A	A4	10+ years	REN
57	35000.0	36 months	14.16	1198.94	C	C2	9 years	MORTGAG
95	30000.0	36 months	16.49	1061.99	D	D3	10+ years	REN
103	30000.0	36 months	15.31	1044.52	C	C2	9 years	MORTGAG
...	...	...	...	...	...	...	...	
395828	35000.0	36 months	14.09	1197.75	B	B5	10+ years	MORTGAG
395836	35000.0	36 months	12.99	1179.12	B	B4	2 years	REN
395909	32500.0	36 months	18.99	1191.16	E	E1	3 years	OW
395964	31300.0	36 months	18.85	1144.97	D	D3	6 years	REN
395968	35000.0	36 months	12.68	1173.91	C	C1	2 years	REN

11211 rows × 22 columns

In [1353]:

```
lower_limit,upper_limit = outlier_detection(df,'revol_bal')
annual_inc_outliers = df[(df['revol_bal']>upper_limit) | (df['revol_bal']<lower_limi
annual_inc_outliers
```

Out[1353]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_length	home_ownershi
11	35000.0	36 months	14.64	1207.13	C	C3	8 years	MORTGAG
13	35000.0	60 months	12.29	783.70	C	C1	10+ years	MORTGAG
51	15000.0	60 months	18.25	382.95	D	D3	8 years	MORTGAG
87	30000.0	60 months	24.70	875.28	G	G1	5 years	MORTGAG
89	23000.0	36 months	8.39	724.89	A	A5	10+ years	MORTGAG
...	...	...	...	...	...	...	...	
395902	20000.0	60 months	14.49	470.47	C	C4	10+ years	MORTGAG
395904	11200.0	60 months	16.29	274.10	D	D1	9 years	MORTGAG
395907	20000.0	36 months	12.99	673.79	C	C2	< 1 year	REN
395936	24000.0	36 months	6.49	735.47	A	A2	1 year	MORTGAG
396026	21000.0	36 months	12.29	700.42	C	C1	5 years	MORTGAG

21228 rows × 22 columns

In [1354]:

```
lower_limit,upper_limit = outlier_detection(df,'revol_util')
annual_inc_outliers = df[(df['revol_util']>upper_limit) | (df['revol_util']<lower_li
annual_inc_outliers
```

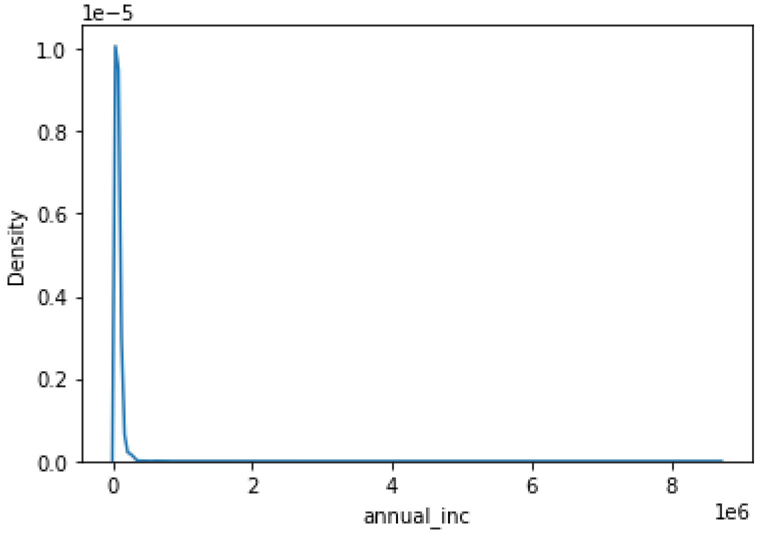
Out[1354]:

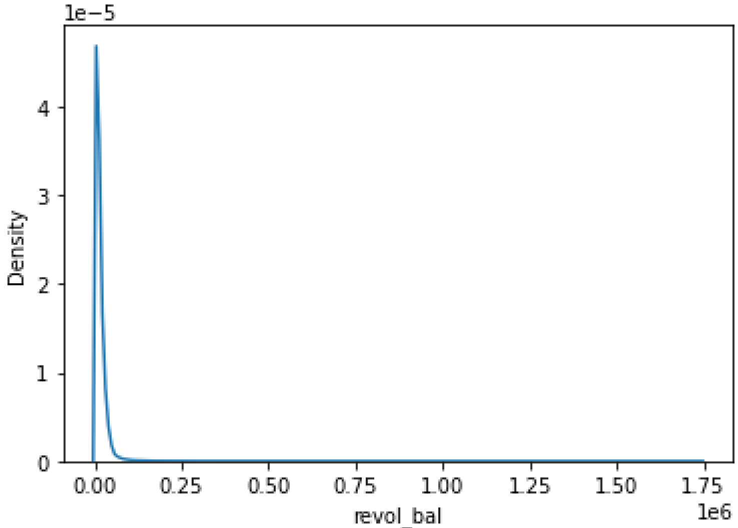
	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_length	home_ownershi
16793	18000.0	60 months	17.57	452.89	D	D4	8 years	MORTGAG
65687	10000.0	36 months	14.16	342.56	C	C2	10+ years	OW
82600	12000.0	60 months	16.55	295.34	D	D2	< 1 year	MORTGAG
108246	10000.0	36 months	17.27	357.88	D	D2	2 years	REN
137211	3500.0	36 months	12.49	117.08	B	B4	10+ years	REN
153970	12550.0	60 months	16.49	308.47	D	D3	4 years	OW
165111	12600.0	36 months	8.39	397.11	A	A5	8 years	MORTGAG
211426	9175.0	36 months	17.57	329.73	D	D4	6 years	REN
296174	12000.0	36 months	20.31	447.87	D	D5	5 years	MORTGAG
312268	8000.0	36 months	11.99	265.68	C	C1	4 years	MORTGAG
329037	35000.0	36 months	25.83	1407.01	G	G2	10+ years	REN
350333	25000.0	60 months	20.49	669.19	E	E2	5 years	REN

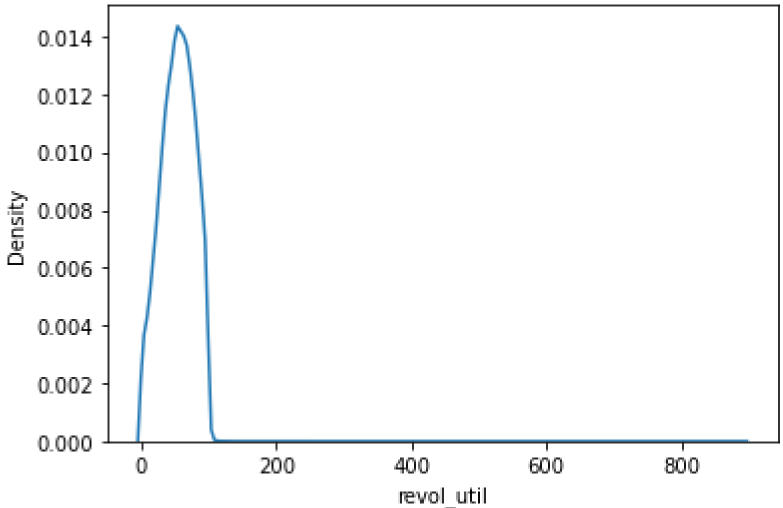
12 rows × 22 columns

In [1355]:

```
columns = ['annual_inc','revol_bal','revol_util']
for column in columns:
    sns.kdeplot(x = df[column])
    plt.show()
```





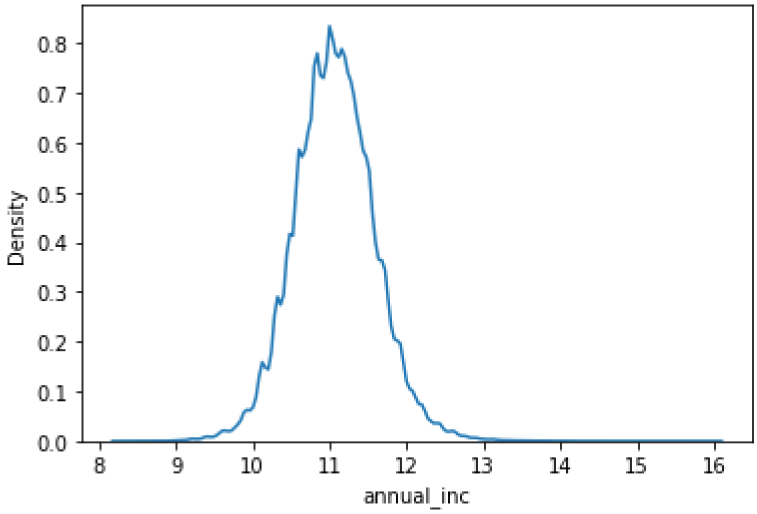


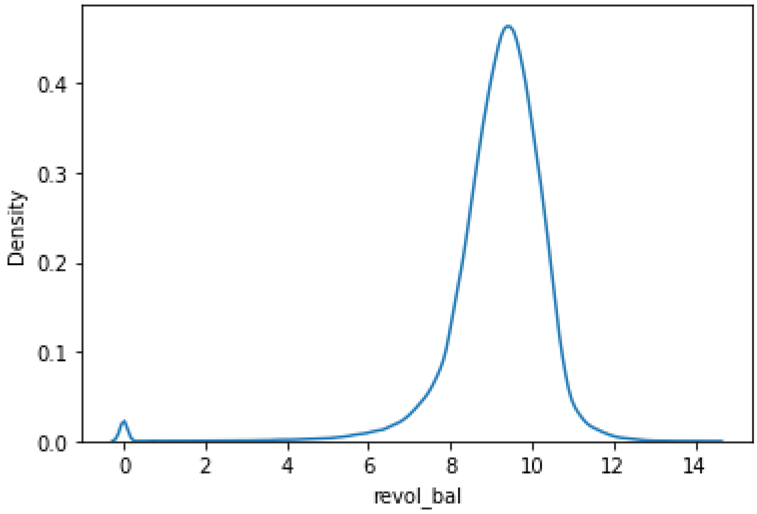
In [1356]:

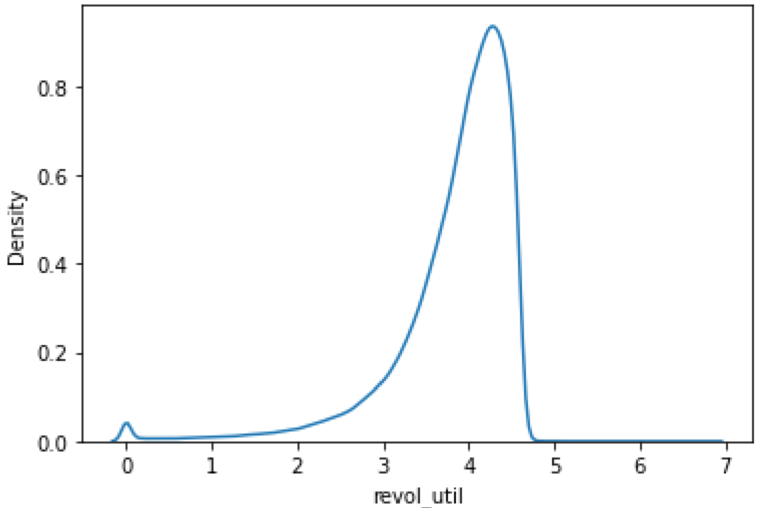
```
## Since the above 3 graphs are log normal distributed, we'll apply log function anc
df['annual_inc'] += 1
df['revol_bal'] += 1
df['revol_util'] += 1
df['annual_inc'] = np.log(df['annual_inc'])
df['revol_bal'] = np.log(df['revol_bal'])
df['revol_util'] = np.log(df['revol_util'])
```

In [1357]:

```
columns = ['annual_inc','revol_bal','revol_util']
for column in columns:
    sns.kdeplot(x = df[column])
    plt.show()
```







Feature Engineering

In [1358]:

```
df['term'] = df['term'].apply(lambda x:x.split(' ')[1])
df
```

Out[1358]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_length	home_ownership
0	10000.0	36	11.44	329.48	B	B4	10+ years	RENT
1	8000.0	36	11.99	265.68	B	B5	4 years	MORTGAGE
2	15600.0	36	10.49	506.97	B	B3	< 1 year	RENT
3	7200.0	36	6.49	220.65	A	A2	6 years	RENT
4	24375.0	60	17.27	609.33	C	C5	9 years	MORTGAGE
...	...	...	...	...	...	...	...	...
396025	10000.0	60	10.99	217.38	B	B4	2 years	RENT
396026	21000.0	36	12.29	700.42	C	C1	5 years	MORTGAGE
396027	5000.0	36	9.99	161.32	B	B1	10+ years	RENT
396028	21000.0	60	15.31	503.02	C	C2	10+ years	MORTGAGE
396029	2000.0	36	13.61	67.98	C	C2	10+ years	RENT

395319 rows × 22 columns

In [1359]:

```
df['grade'].value_counts()
```

Out[1359]:

```
B    115900
C    105833
A     64165
D     63339
E     31348
F     11707
G      3027
Name: grade, dtype: int64
```

In [1360]:

```
grade_mapping = {'A':7,'B':6,'C':5,'D':4,'E':3,'F':2,'G':1}
sub_grade_mapping = {'A1':35,'A2':34,'A3':33,'A4':32,'A5':31,'B1':30,'B2':29,'B3':28
'C2':24,'C3':23,'C4':22,'C5':21,'D1':20,'D2':19,'D3':18,'D4':17,'D5':16,'E1':15,'E2'
'F2':9,'F3':8,'F4':7,'F5':6,'G1':5,'G2':4,'G3':3,'G4':2,'G5':1}
df['sub_grade'] = df['sub_grade'].replace(sub_grade_mapping)
df['grade'] = df['grade'].replace(grade_mapping)
term_mapping = {'36':0,'60':1}
df['term'] = df['term'].replace(term_mapping)
```

In [1361]:

```
df['emp_length'].unique()
```

Out[1361]:

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

In [1362]:

```
emp_length_mapping = {'10+ years':10, '4 years':4, '< 1 year':0.5, '6 years':6, '9 y
       '2 years':2, '3 years':3, '8 years':8, '7 years':7, '5 years':5, '1 year':1}
df['emp_length'] = df['emp_length'].replace(emp_length_mapping)
```

In [1363]:

```
df['home_ownership'] = df['home_ownership'].replace({'ANY':'OTHER','NONE':'OTHER'})
home_ownership_mapping = {'OTHER':0,'RENT':1,'MORTGAGE':2,'OWN':3}
df['home_ownership'] = df['home_ownership'].replace(home_ownership_mapping)
```

In [1364]:

```
df['home_ownership'].value_counts()
```

Out[1364]:

```
2    197935
1    159573
3     37665
0       146
Name: home_ownership, dtype: int64
```

In [1365]:

```
# Considering only year of joining for 'earliest_cr_line' column.
df['earliest_cr_line'] = pd.DatetimeIndex(df['earliest_cr_line']).year
```

In [1366]:

```
# Adding new features by getting month and year from issue_d,column
# Considering the current year as 2022,we'll calculate the time period for earliest_
df['issue_d_year'] = pd.DatetimeIndex(df['issue_d']).year
df['issue_d_month'] = pd.DatetimeIndex(df['issue_d']).month

df['credit_history'] = df['issue_d_year'] - df['earliest_cr_line']
```

In [1367]:

```
df.drop(['issue_d','issue_d_month','earliest_cr_line'],axis=1,inplace=True)
```

In [1368]:

```
cap_columns = ['pub_rec_bankruptcies','mort_acc','pub_rec']
for column in cap_columns:
    df[column] = np.where(df[column]>1,1,0)
```

In [1369]:

```
df['pub_rec'].value_counts()
```

Out[1369]:

```
0    387316
1      8003
Name: pub_rec, dtype: int64
```

In [1370]:

```
df['verification_status'].value_counts()
```

Out[1370]:

```
Verified           139167
Source Verified    131211
Not Verified       124941
Name: verification_status, dtype: int64
```

In [1371]:

```
verification_status_mapping = {'Source Verified':2,'Verified':1,'Not Verified':0}
df['verification_status'] = df['verification_status'].replace(verification_status_ma
```

In [1372]:

```
df['inst_amnt_ratio'] = df['installment']/df['loan_amnt']
```

In [1373]:

```
df.drop(['loan_amnt','installment'],axis=1,inplace=True)
```

In [1374]:

```
df['account_ratio'] = df['open_acc']/df['total_acc']
```

In [1375]:

```
df.drop(['open_acc','total_acc'],axis=1,inplace=True)
```

In [1376]:

```
le = LabelEncoder()
le.fit(df['purpose'])
df['purpose'] = le.transform(df['purpose'])
```

In [1377]:

```
df['loan_status'] = df['loan_status'].replace({'Fully Paid':0,'Charged Off':1})
df['loan_status'].value_counts()
```

Out[1377]:

```
0    317802
1     77517
Name: loan_status, dtype: int64
```

In [1378]:

```
df
```

Out[1378]:

	term	int_rate	grade	sub_grade	emp_length	home_ownership	annual_inc	verification_
0	0	11.44	6	27	10.0	1	11.669938	
1	0	11.99	6	26	4.0	2	11.082158	
2	0	10.49	6	28	0.5	1	10.670303	
3	0	6.49	7	34	6.0	1	10.896758	
4	1	17.27	5	21	9.0	2	10.915107	
...	...	...	...	...	...	...	...	
396025	1	10.99	6	27	2.0	1	10.596660	
396026	0	12.29	5	25	5.0	2	11.608245	
396027	0	9.99	6	30	10.0	1	10.942014	
396028	1	15.31	5	24	10.0	2	11.066654	
396029	0	13.61	5	24	10.0	1	10.668886	

395319 rows × 20 columns

In [1379]:

```
df = df.sort_values('issue_d_year')
```

In [1380]:

```
### Selecting Top Best features
```

In [1381]:

```
bestfeatures = SelectKBest(score_func=f_classif, k=10)
fit = bestfeatures.fit(X_train,y_train)
dfscores = pd.DataFrame(fit.scores_)
dfpvalue= pd.DataFrame(fit.pvalues_)
dfcolumns = pd.DataFrame(X_train.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfpvalue,dfscores],axis=1)
featureScores.columns = ['Specs','pvalue','Score']  #naming the dataframe columns
print(featureScores.nlargest(21,'Score'))  #print 10 best features
```

```
                   Specs         pvalue         Score
2              sub_grade   0.000000e+00  22948.824965
1               int_rate   0.000000e+00  20611.853218
0                   term   0.000000e+00  11051.488974
6                    dti   0.000000e+00   4944.628139
4             annual_inc   0.000000e+00   2078.475707
5    verification_status   0.000000e+00   1665.771009
11       inst_amnt_ratio   0.000000e+00   1528.358781
7             revol_util   0.000000e+00   1514.093525
12         account_ratio  4.783761e-235   1073.473591
8               mort_acc  1.057354e-129    587.671469
3         home_ownership   8.553678e-98    440.777593
10        credit_history   9.074413e-43    187.969468

/opt/anaconda3/lib/python3.9/site-packages/sklearn/feature_selection/_
univariate_selection.py:110: UserWarning: Features [9] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx, U
serWarning)
/opt/anaconda3/lib/python3.9/site-packages/sklearn/feature_selection/_
univariate_selection.py:111: RuntimeWarning: invalid value encountered
in true_divide
  f = msb / msw
```

In [1382]:

```
drop_columns = ['revol_bal','pub_rec','emp_length','purpose','grade','issue_d_year']
df.drop(drop_columns,axis=1,inplace=True)
```

In [1383]:

```
### We can split the data based on the issue date. The previous records can be taken
## latest records will be treated as test data
```

In [1384]:

```
split_ratio = 0.8
train_data_index = int(len(df)*0.8)
train_data_index
```

Out[1384]:

```
316255
```

In [1385]:

```
df_train = df.iloc[:train_data_index,:]
```

In [1386]:

```
df_train['loan_status'].value_counts(normalize=True)
```

Out[1386]:

```
0    0.806779
1    0.193221
Name: loan_status, dtype: float64
```

In [1387]:

```
df_test = df.iloc[train_data_index:,:]
```

In [1388]:

```
df_test['loan_status'].value_counts(normalize=True)
```

Out[1388]:

```
0    0.792447
1    0.207553
Name: loan_status, dtype: float64
```

In [1389]:

```
y_train = df_train['loan_status']
X_train = df_train.drop('loan_status',axis=1)
y_test = df_test['loan_status']
X_test = df_test.drop('loan_status',axis=1)
```

In [1390]:

```
print("X_train shape : {}".format(X_train.shape))
print("y_train shape : {}".format(y_train.shape))
print("X_test shape : {}".format(X_test.shape))
print("y_test shape : {}".format(y_test.shape))
```

```
X_train shape : (316255, 13)
y_train shape : (316255,)
X_test shape : (79064, 13)
y_test shape : (79064,)
```

In [1391]:

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_train_scaled = pd.DataFrame(X_train_scaled, columns =X_train.columns)
X_test_scaled = scaler.transform(X_test)
X_test_scaled = pd.DataFrame(X_test_scaled, columns =X_test.columns)
```

In [1392]:

```
X_train_scaled
```

Out[1392]:

	term	int_rate	sub_grade	home_ownership	annual_inc	verification_status	
0	-0.540758	-1.353441	1.361796	2.094530	-0.044831	-1.233322	-1.717
1	-0.540758	-1.426497	1.514801	2.094530	-0.652905	-1.233322	-2.013
2	-0.540758	-0.705069	-0.015253	-1.075057	-0.101488	-1.233322	0.012
3	-0.540758	-0.271300	-0.933285	0.509737	0.566135	-1.233322	-1.347
4	-0.540758	-0.921954	0.443763	-1.075057	-4.864195	-1.233322	0.123
...	...	...	...	...	...	...	
316250	-0.540758	-1.255272	0.902780	0.509737	0.425360	0.020138	0.457
316251	-0.540758	-0.316960	0.137753	-1.075057	-0.529201	1.273598	1.680
316252	1.849256	3.112103	-3.075361	-1.075057	1.849105	1.273598	0.537
316253	-0.540758	0.071150	-0.321264	-1.075057	-0.976982	0.020138	1.088
316254	1.849256	1.043707	-1.392302	-1.075057	-0.976982	1.273598	1.012

316255 rows × 13 columns

In [1393]:

```
X_train.shape
```

Out[1393]:

```
(316255, 13)
```

Model Building

In [1411]:

```
model = LogisticRegression(
        penalty='l2',
        C=0.5,
        solver = 'lbfgs',
        max_iter=300,
        n_jobs=-1,
        class_weight='balanced'
)
model.fit(X_train,y_train)
```

Out[1411]:

```
LogisticRegression

LogisticRegression(C=0.5, class_weight='balanced', max_iter=300, n_jo
bs=-1)
```

In [1412]:

```
## Model coef_ with column names
data = {'Feature':np.array(X_train.columns),'Importance':np.array(model.coef_[0]),
        'abs':np.array(np.abs(model.coef_)[0])}
feature_importance = pd.DataFrame(data)
feature_importance.sort_values('abs',ascending=False)
```

Out[1412]:

	Feature	Importance	abs
12	account_ratio	0.508208	0.508208
0	term	0.484184	0.484184
4	annual_inc	-0.371017	0.371017
11	inst_amnt_ratio	0.135093	0.135093
5	verification_status	0.105721	0.105721
2	sub_grade	-0.097565	0.097565
3	home_ownership	-0.087471	0.087471
8	mort_acc	-0.065232	0.065232
7	revol_util	0.061569	0.061569
1	int_rate	-0.028240	0.028240
6	dti	0.023060	0.023060
10	credit_history	0.004910	0.004910
9	pub_rec_bankruptcies	0.000000	0.000000

Result Evaluation

In [1428]:

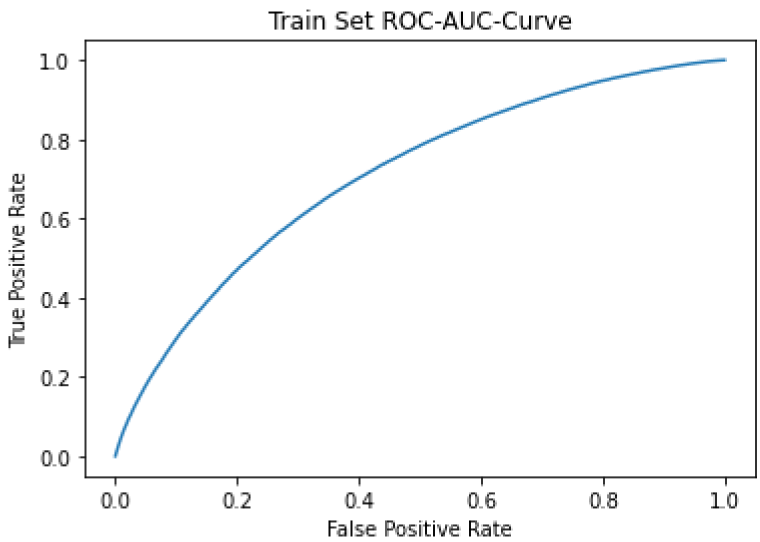
```
X_test_pred_prob = model.predict_proba(X_test)[:,1]
X_train_pred_prob = model.predict_proba(X_train)[:,1]
```

In [1429]:

```
X_test_pred = model.predict(X_test)
X_train_pred = model.predict(X_train)
```

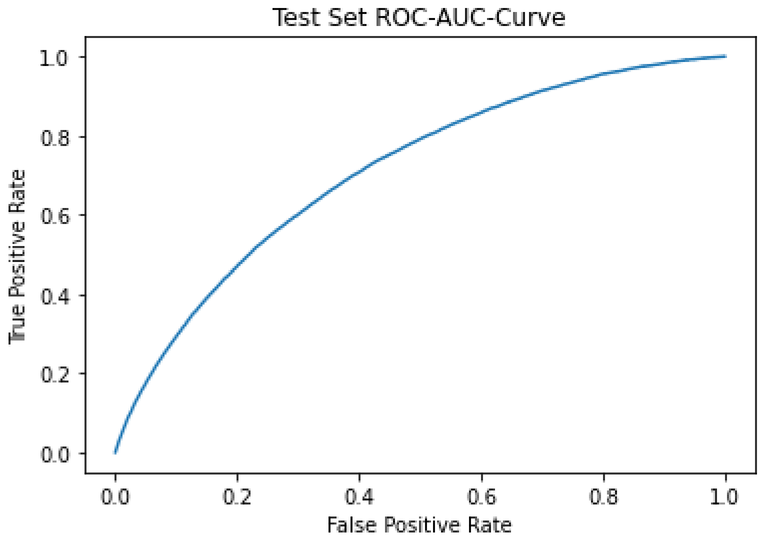
In [1439]:

```
roc_curve(y_train,train_pred_prob)
fpr,tpr,thresholds = roc_curve(y_train,train_pred_prob,pos_label=1)
plt.plot(fpr,tpr)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Train Set ROC-AUC-Curve")
plt.show()
```



In [1438]:

```
fpr,tpr,thresholds = roc_curve(y_test,test_pred_prob,pos_label=1)
plt.plot(fpr,tpr)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Test Set ROC-AUC-Curve")
plt.show()
```



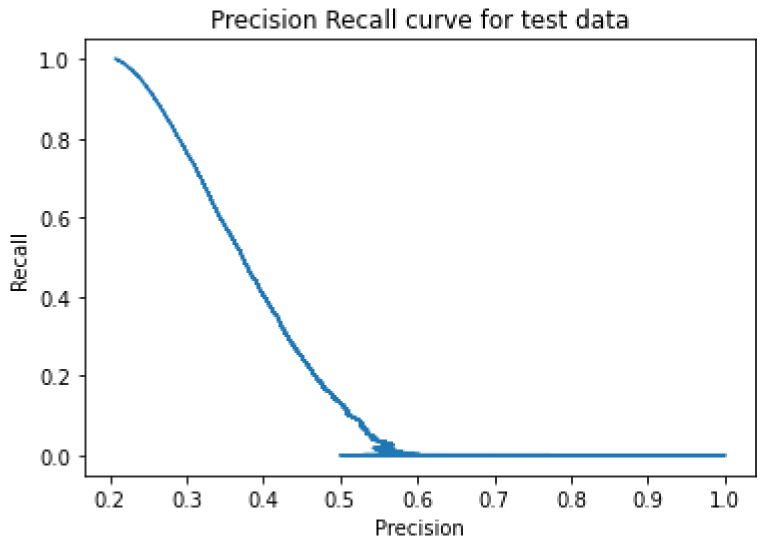
In [1430]:

```
test_precision,test_recall,_ = precision_recall_curve(y_test, X_test_pred_prob)
```

Comment: The ROC AUC Curve have a good Area Under the Curve. This means the model is performing well.

In [1440]:

```
plt.plot(test_precision,test_recall)
plt.xlabel("Precision")
plt.ylabel("Recall")
plt.title("Precision Recall curve for test data")
plt.show()
```

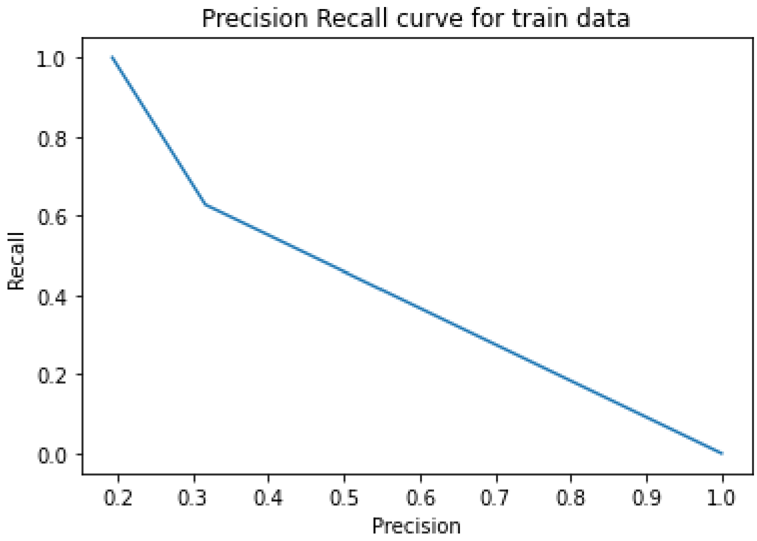


In [1441]:

```
train_precision,train_recall,_ = precision_recall_curve(y_train, X_train_pred)
```

In [1442]:

```
plt.plot(train_precision,train_recall)
plt.xlabel("Precision")
plt.ylabel("Recall")
plt.title("Precision Recall curve for train data")
plt.show()
```

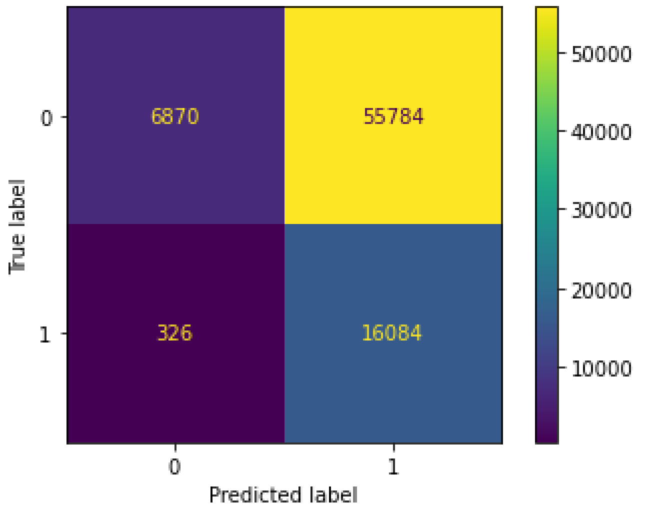


Comment: The precision recall curve shows that as the precision increased the recall starts descreasing. As per our use case we need to have high recall(So that we can predict defaulters accurately) and avoid Non-performing Asset.

In [1443]:

```
print("Confusion Matrix for Test data")
matrix = confusion_matrix(y_test,test_pred,labels=model.classes_)
disp = ConfusionMatrixDisplay(matrix,display_labels=model.classes_)
disp.plot()
plt.show()
```

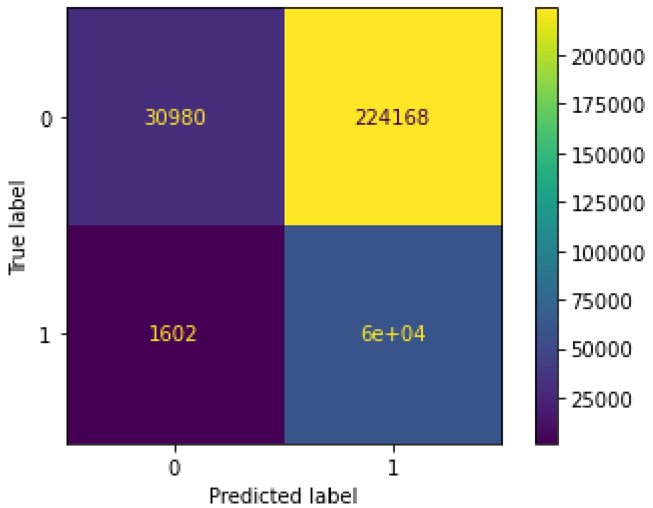
Confusion Matrix for Test data



In [1444]:

```
print("Confusion Matrix for Train data")
matrix = confusion_matrix(y_train,train_pred,labels=model.classes_)
disp = ConfusionMatrixDisplay(matrix,display_labels=model.classes_)
disp.plot()
plt.show()
```

Confusion Matrix for Train data



Comment: The confusion Matrix shows that our model it correctly identifying most of the defaulters but it also marking most of the fully paid as defaulters. This will lead to a loss of opportunity.

In [1447]:

```
print("classification report for Test set.")
print(classification_report(y_test,test_pred))
```

```
classification report for Test set.
              precision    recall  f1-score   support

           0       0.95      0.11      0.20     62654
           1       0.22      0.98      0.36     16410

    accuracy                           0.29     79064
   macro avg       0.59      0.54      0.28     79064
weighted avg       0.80      0.29      0.23     79064
```

In [1448]:

```
print("Classification report for Train set.")
print(classification_report(y_train,train_pred))
```

```
Classification report for Train set.
              precision    recall  f1-score   support

           0       0.95      0.12      0.22    255148
           1       0.21      0.97      0.35     61107

    accuracy                           0.29    316255
   macro avg       0.58      0.55      0.28    316255
weighted avg       0.81      0.29      0.24    316255
```

```
<div class="alert alert-block alert-success">
<b>Comment:</b> To make sure we have less False positives and also high recall, we can try different threshold value
    and balance those. We can also use f1-score as the metric in case both identifying a defaulter and reducing False positive is important.
</div>
```

Comment: To make sure we have high rate of identifying defaulters, we have kept a high recall for class 1(i.e. defaulter class)

Insights: Following are the insights which can be derived from the analysis
- People with 10+ years of experience usually apply for a loan.
- Most of the people applying for a loan either have a mortgage or live on rent.
- Most of the people apply for loan for either debt consolidation or credit card payment.
- People with low dti are more likely to pay off the loan in comparision to people with high dti.
- People with high annual income usually pay off their debt except few exceptions.

Business Recommendations: Following are the Business recommendations which can be derived from the analysis
- Banks should target high income people since they are more likely to pay off their loan.

- Banks should target high income people since they are more likely to pay off their loan.
- Based on the model output, Banks should also do a manual check so as to avoid opportunity cost.
- Banks should work on maintaining a balance between bad debt and opportunity cost. Playing too safe will lead to loss of business.
- Banks can target people with low dti, since they are more likely to pay off their debt.
- The above model will play a safe game and accurately predict defaulters but in the process it will also mark some potential clients as defaulters. In case some potential client is marked as defaulter, banks can set up a team to look into these kind of cases.

Questionnaire

In [1582]:

```
df = pd.read_csv('logistic_regression.txt')
df = df[df['application_type']=='INDIVIDUAL']
```

How many people fully paid their loan

In [1583]:

```
df['loan_status'].value_counts(normalize=True)
```

Out[1583]:

```
Fully Paid     0.803913
Charged Off    0.196087
Name: loan_status, dtype: float64
```

Comment about the correlation between Loan Amount and Installment features.

In [1584]:

```
## There is a high correlation between loan_amnt and installment
df[['loan_amnt','installment']].corr()
```

Out[1584]:

	loan_amnt	installment
loan_amnt	1.000000	0.953945
installment	0.953945	1.000000

The majority of people have home ownership as ___.

In [1585]:

```
## Most of the people have home ownership as mortgage
df['home_ownership'].value_counts()
```

Out[1585]:

```
MORTGAGE    197935
RENT        159573
OWN          37665
OTHER          112
NONE            31
ANY              3
Name: home_ownership, dtype: int64
```

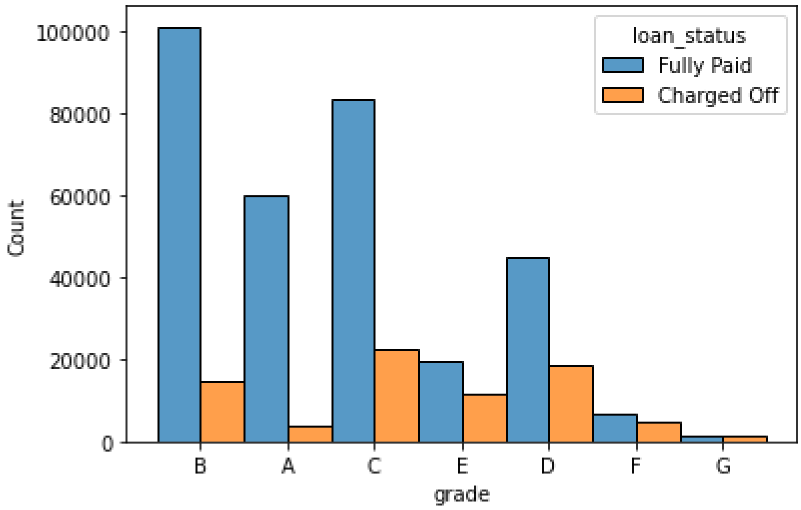
People with grades 'A' are more likely to fully pay their loan. (T/F)

In [1586]:

```
## This is False. B Grade people are more likely to pay their loan.
sns.histplot(binwidth=1,
            x='grade',
            hue='loan_status',
            data=df,
            stat="count",
            multiple="dodge")
plt.plot()
```

Out[1586]:

```
[]
```



Name the top 2 afforded job titles.

In [1587]:

```
## Teacher and Manager are the most afforded job titles
df['emp_title'].value_counts(normalize=True).sort_values(ascending=False)
```

Out[1587]:

```
Teacher                            0.011738
Manager                            0.011373
Registered Nurse                   0.004970
RN                                 0.004948
Supervisor                         0.004897
                                     ...
VIP IT Technical Lead              0.000003
Gull Lake Community Schools        0.000003
crane lear romec                   0.000003
M.C. Dean                          0.000003
Gracon Services, Inc               0.000003
Name: emp_title, Length: 172922, dtype: float64
```

Thinking from a bank's perspective, which metric should our primary focus be on..

```
-ROC AUC
-Precision
-Recall
-F1 Score
```

Banks don't want to have any bad debt and thus they need to accurately identify prospective defaulters based on their application. Thus recall will be the metric they'll focus more on.

How does the gap in precision and recall affect the bank?

Banks will have a tradeoff between opportunity cost v/s bad debt. If they have a high recall then they'll avoid bad debt and if they have high precision they can have high chances of getting a oppurtunity of lending loan but this will come with a risk of bad debt.

Which were the features that heavily affected the outcome?

The features having high impact on output are:

- term
- annual_inc
- verification_status
- sub_grade
- home_ownership

Will the results be affected by geographical location? (Yes/No)

In [1588]:

```
df['new_address'] = df['address'].apply(lambda x: x[-8:])
```

In [1589]:

```
df['location'],df['zip'] = df['new_address'].apply(lambda x:x.split(' ')[0]),df['new
```

In [1590]:

```
df['location']
```

Out[1590]:

```
0         OK
1         SD
2         WV
3         MA
4         VA
          ..
396025    DC
396026    LA
396027    NY
396028    FL
396029    AR
Name: location, Length: 395319, dtype: object
```

In [1591]:

```
data = pd.DataFrame(df.groupby(['location','loan_status']).size().sort_values(ascend
```

In [1592]:

```
new_data = pd.merge(data,data,left_on='location',right_on='location')
```

In [1593]:

```
newly_data = new_data[new_data['loan_status_x']!=new_data['loan_status_y']]
```

In [1594]:

```
newly_data.drop_duplicates(subset=['location'],inplace=True)
newly_data['ratio_paid_to_charge_off'] = newly_data['0_x']/newly_data['0_y']
```

```
/opt/anaconda3/lib/python3.9/site-packages/pandas/util/_decorators.py:
311: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
 (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)
  return func(*args, **kwargs)
/var/folders/kq/0bdtsq2j33760qf6gclvdx840000gn/T/ipykernel_1303/241809
4103.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
 (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)
  newly_data['ratio_paid_to_charge_off'] = newly_data['0_x']/newly_dat
a['0_y']
```

In [1595]:

```
df_ratio = newly_data.sort_values('ratio_paid_to_charge_off',ascending=False)
```

In [1596]:

```
df_ratio.head()
```

Out[1596]:

	location	loan_status_x	0_x	loan_status_y	0_y	ratio_paid_to_charge_off
33	MN	Fully Paid	5644	Charged Off	1246	4.529695
21	NY	Fully Paid	5687	Charged Off	1304	4.361196
77	OR	Fully Paid	5583	Charged Off	1307	4.271614
97	CA	Fully Paid	5574	Charged Off	1311	4.251716
29	VT	Fully Paid	5659	Charged Off	1331	4.251690

In [1597]:

```
df_ratio.tail()
```

Out[1597]:

	location	loan_status_x	0_x	loan_status_y	0_y	ratio_paid_to_charge_off
189	WA	Fully Paid	5496	Charged Off	1391	3.951114
61	NV	Fully Paid	5604	Charged Off	1419	3.949260
209	PA	Fully Paid	5434	Charged Off	1383	3.929140
177	WV	Fully Paid	5515	Charged Off	1415	3.897527
205	WY	Fully Paid	5481	Charged Off	1438	3.811544

There doesn't seem to be much relation between location and loan status since the ratio is somethat similar

In [ ]:

```
```