In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

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

In [3]:

```
df.head()
```

Out[3]:

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

5 rows × 27 columns

In [4]:

```
df.shape
```

Out[4]:

```
(396030, 27)
```

Their are total 396030 data points, 26 features and 1 label.

In [5]:

```
df.columns
```

Out[5]:

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

displays the columns present in the dataframe

In [6]:

```
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
```

We can observe that some features are having object data type which need to be converted to int or float data type.

In [7]:

```
df.describe()
```

Out[7]:

	loan_amnt	int_rate	installment	annual_inc	dti	open_a
count	396030.000000	396030.000000	396030.000000	3.960300e+05	396030.000000	396030.0000
mean	14113.888089	13.639400	431.849698	7.420318e+04	17.379514	11.3111
std	8357.441341	4.472157	250.727790	6.163762e+04	18.019092	5.1376
min	500.000000	5.320000	16.080000	0.000000e+00	0.000000	0.0000
25%	8000.000000	10.490000	250.330000	4.500000e+04	11.280000	8.0000
50%	12000.000000	13.330000	375.430000	6.400000e+04	16.910000	10.0000
75%	20000.000000	16.490000	567.300000	9.000000e+04	22.980000	14.0000
max	40000.000000	30.990000	1533.810000	8.706582e+06	9999.000000	90.0000

we can observe that the gap between mean and median values is drastic so loan_amnt feature is effected by outliers

int_rate is not effected by outliers as mean and the median values are almost same.

installment is

In [8]:

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

Out[8]:

	term	grade	sub_grade	emp_title	emp_length	home_ownership	verification_status
count	396030	396030	396030	373103	377729	396030	39603
unique	2	7	35	173105	11	6	3
top	36 months	B	B3	Teacher	10+ years	MORTGAGE	Verifie
freq	302005	116018	26655	4389	126041	198348	13956

In [9]:

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

Out[9]:

```
loan_amnt                   0
term                        0
int_rate                    0
installment                 0
grade                       0
sub_grade                   0
emp_title               22927
emp_length              18301
home_ownership              0
annual_inc                  0
verification_status         0
issue_d                     0
loan_status                 0
purpose                     0
title                    1755
dti                         0
earliest_cr_line            0
open_acc                    0
pub_rec                     0
revol_bal                   0
revol_util                276
total_acc                   0
initial_list_status         0
application_type            0
mort_acc                37795
pub_rec_bankruptcies      535
address                     0
dtype: int64
```

we can see emp_title, emp_length, title, revol_util, mort_acc, pub_rec_bankruptcies are having missing values need to impute data using some imputation techniques like mean, median or model based Imputing(KNNImputer).

In [10]:

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

Out[10]:

```
Fully Paid     0.803871
Charged Off    0.196129
Name: loan_status, dtype: float64
```

the 80.38% people in the data set repayed loan and 19.61% of the data points either repaying the loan or defaulters.

In [11]:

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

Out[11]:

```
Fully Paid     318357
Charged Off     77673
Name: loan_status, dtype: int64
```

In [12]:

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

Out[12]:

```
0.0     338272
1.0      49739
2.0       5476
3.0       1521
4.0        527
5.0        237
6.0        122
7.0         56
8.0         34
9.0         12
10.0        11
11.0         8
13.0         4
12.0         4
19.0         2
86.0         1
40.0         1
17.0         1
15.0         1
24.0         1
Name: pub_rec, dtype: int64
```

most of the people are having good public record only a few are having bad public record.

In [13]:

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

Out[13]:

```
0.0     139777
1.0      60416
2.0      49948
3.0      38049
4.0      27887
5.0      18194
6.0      11069
7.0       6052
8.0       3121
9.0       1656
10.0       865
11.0       479
12.0       264
13.0       146
14.0       107
15.0        61
16.0        37
17.0        22
18.0        18
19.0        15
20.0        13
24.0        10
22.0         7
21.0         4
25.0         4
27.0         3
23.0         2
32.0         2
26.0         2
31.0         2
30.0         1
28.0         1
34.0         1
Name: mort_acc, dtype: int64
```

only a few number of people are taking loan account in multiple numbers.

In [14]:

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

Out[14]:

```
0.0    350380
1.0     42790
2.0      1847
3.0       351
4.0        82
5.0        32
6.0         7
7.0         4
8.0         2
Name: pub_rec_bankruptcies, dtype: int64
```

In [15]:

```
df['pub_rec'] = df['pub_rec'].apply(lambda x:1 if x>1 else 0)
df['mort_acc'] = df['mort_acc'].apply(lambda x:1 if x>1 else 0)
df['pub_rec_bankruptcies'] = df['pub_rec_bankruptcies'].apply(lambda x:1 if x>1 else 0)
```

converting all the records with value more than 1 as 1 and else 0

In [16]:

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

Out[16]:

```
0    388011
1      8019
Name: pub_rec, dtype: int64
```

In [17]:

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

Out[17]:

```
0    237988
1    158042
Name: mort_acc, dtype: int64
```

In [18]:

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

Out[18]:

```
0    393705
1      2325
Name: pub_rec_bankruptcies, dtype: int64
```

verifing the changed records are reflected

In [19]:

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

Out[19]:

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

In [20]:

```
#df['grade'] = df['grade'].replace(df['grade'].value_counts(normalize=True).index,df['grade
```

In [21]:

```
df['grade'].replace(df['grade'].value_counts(normalize=True).index,df['grade'].value_counts
```

Out[21]:

```
0         0.292953
1         0.292953
2         0.292953
3         0.162076
4         0.267624
            ...
396025    0.292953
396026    0.267624
396027    0.292953
396028    0.267624
396029    0.267624
Name: grade, Length: 396030, dtype: float64
```

In [22]:

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

Out[22]:

```
B3    26655
B4    25601
C1    23662
C2    22580
B2    22495
B5    22085
C3    21221
C4    20280
B1    19182
A5    18526
C5    18244
D1    15993
A4    15789
D2    13951
D3    12223
D4    11657
A3    10576
A1     9729
D5     9700
A2     9567
E1     7917
E2     7431
E3     6207
E4     5361
E5     4572
F1     3536
F2     2766
F3     2286
F4     1787
F5     1397
G1     1058
G2      754
G3      552
G4      374
G5      316
Name: sub_grade, dtype: int64
```

In [23]:

```
#df['emp_title'].replace(df['emp_title'].value_counts(normalize=True).index,df['emp_title']
```

In [24]:

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

Out[24]:

```
Teacher                        0.011764
Manager                        0.011391
Registered Nurse               0.004974
RN                             0.004948
Supervisor                     0.004905
                                  ...
Teachers aide/bus monitor      0.000003
TJCross Engineers              0.000003
assitsant manager              0.000003
Applied Energy                 0.000003
Healthcare Call Center Rep     0.000003
Name: emp_title, Length: 173105, dtype: float64
```

In [25]:

```
df['emp_length'] = df['emp_length'].replace(df['emp_length'].value_counts(normalize=True).i
```

converting the feature emp_length to category

In [26]:

```
df['home_ownership'] = df['home_ownership'].replace(df['home_ownership'].value_counts(norma
```

In [27]:

```
df['verification_status'] = df['verification_status'].replace(df['verification_status'].val
```

In [28]:

```
df['issue_d'] = df['issue_d'].replace(df['issue_d'].value_counts(normalize=True).index,df['
```

In [29]:

```
#df['loan_status'] = df['loan_status'].replace(df['loan_status'].value_counts(normalize=Tru
```

In [30]:

```
df['purpose'] = df['purpose'].replace(df['purpose'].value_counts(normalize=True).index,df['
```

In [31]:

```
#df['title'].replace(df['title'].value_counts(normalize=True).index,df['title'].value_count
```

In [32]:

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

Out[32]:

```
Debt consolidation                   152472
Credit card refinancing               51487
Home improvement                      15264
Other                                 12930
Debt Consolidation                    11608
                                      ...
Debt Consolotation                        1
My Debt Consolidation loan                1
Short term until 12/31                    1
On a Debt Free Adventure of my own!       1
cc debt consolidation                     1
Name: title, Length: 48817, dtype: int64
```

In [33]:

```
df['earliest_cr_line'] = df['earliest_cr_line'].replace(df['earliest_cr_line'].value_counts
```

In [34]:

```
df['initial_list_status'] = df['initial_list_status'].replace(df['initial_list_status'].val
```

In [35]:

```
df['application_type'] = df['application_type'].replace(df['application_type'].value_counts
```

In [36]:

```
replacer = dict({'term':{' 36 months':3,' 60 months':5},'grade':{'A':1,'B':2,'C':3,'D':4,'E
```

In [37]:

```
df['grade']
```

Out[37]:

```
0         B
1         B
2         B
3         A
4         C
         ..
396025    B
396026    C
396027    B
396028    C
396029    C
Name: grade, Length: 396030, dtype: object
```

In [38]:

```
df = df.replace(replacer)
```

as grade is cardinal and loan status is target variable and term is binary we are converting this using the replace function.

In [39]:

```
df.head()
```

Out[39]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_o
0	10000.0	3	11.44	329.48	2	B4	Marketing	0.333681	
1	8000.0	3	11.99	265.68	2	B5	Credit analyst	0.063411	
2	15600.0	3	10.49	506.97	2	B3	Statistician	0.083989	
3	7200.0	3	6.49	220.65	1	A2	Client Advocate	0.055174	
4	24375.0	5	17.27	609.33	3	C5	Destiny Management Inc.	0.040542	

5 rows × 27 columns

In [40]:

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

Out[40]:

```
2    116018
3    105987
1     64187
4     63524
5     31488
6     11772
7      3054
Name: grade, dtype: int64
```

In [41]:

```
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  int64  
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  int64  
 5   sub_grade             396030 non-null  object 
 6   emp_title             373103 non-null  object 
 7   emp_length            377729 non-null  float64
 8   home_ownership        396030 non-null  float64
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  float64
 11  issue_d               396030 non-null  float64
 12  loan_status           396030 non-null  int64  
 13  purpose               396030 non-null  float64
 14  title                 394275 non-null  object 
 15  dti                   396030 non-null  float64
 16  earliest_cr_line      396030 non-null  float64
 17  open_acc              396030 non-null  float64
 18  pub_rec               396030 non-null  int64  
 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  float64
 23  application_type      396030 non-null  float64
 24  mort_acc              396030 non-null  int64  
 25  pub_rec_bankruptcies  396030 non-null  int64  
 26  address               396030 non-null  object 
dtypes: float64(17), int64(6), object(4)
memory usage: 81.6+ MB
```

In [42]:

```
df = df.drop(columns=['sub_grade','emp_title','title','address'])
```

In [43]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 23 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  int64  
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  int64  
 5   emp_length            377729 non-null  float64
 6   home_ownership        396030 non-null  float64
 7   annual_inc            396030 non-null  float64
 8   verification_status   396030 non-null  float64
 9   issue_d               396030 non-null  float64
 10  loan_status           396030 non-null  int64  
 11  purpose               396030 non-null  float64
 12  dti                   396030 non-null  float64
 13  earliest_cr_line      396030 non-null  float64
 14  open_acc              396030 non-null  float64
 15  pub_rec               396030 non-null  int64  
 16  revol_bal             396030 non-null  float64
 17  revol_util            395754 non-null  float64
 18  total_acc             396030 non-null  float64
 19  initial_list_status   396030 non-null  float64
 20  application_type      396030 non-null  float64
 21  mort_acc              396030 non-null  int64  
 22  pub_rec_bankruptcies  396030 non-null  int64  
dtypes: float64(17), int64(6)
memory usage: 69.5 MB
```

In [44]:

```
df.isna().sum()
```

Out[44]:

```
loan_amnt                   0
term                        0
int_rate                    0
installment                 0
grade                       0
emp_length              18301
home_ownership              0
annual_inc                  0
verification_status         0
issue_d                     0
loan_status                 0
purpose                     0
dti                         0
earliest_cr_line            0
open_acc                    0
pub_rec                     0
revol_bal                   0
revol_util                276
total_acc                   0
initial_list_status         0
application_type            0
mort_acc                    0
pub_rec_bankruptcies        0
dtype: int64
```

still emp_length and revol_util are having some missing values so we have to fill the missing data with the help of some imputer.

In [45]:

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

Out[45]:

	loan_amnt	term	int_rate	installment	grade	emp_length	home_ownership	annual_in
35	5375.0	3	13.11	181.39	2	NaN	0.403480	34000.0(
36	3250.0	3	16.78	115.52	3	NaN	0.403480	22500.0(
49	15000.0	3	7.89	469.29	1	NaN	0.500841	90000.0(
58	10000.0	3	17.56	359.33	4	NaN	0.500841	32000.0(
91	30225.0	5	18.24	771.47	4	NaN	0.500841	65800.0(
...	...	...	...	...	...	...	...	..
395946	35000.0	5	16.20	854.86	3	NaN	0.500841	84000.0(
395963	7000.0	3	20.20	260.86	5	NaN	0.095311	32964.0(
395988	35000.0	5	15.59	843.53	4	NaN	0.095311	102396.0(
395999	11125.0	3	24.11	437.11	6	NaN	0.500841	31789.8
396015	4000.0	3	9.16	127.50	2	NaN	0.500841	57400.0(

18301 rows × 23 columns

In [46]:

```
X = df.drop(columns='loan_status')
Y = df['loan_status']
```

separating the larget from the main data

In [47]:

```
from sklearn.impute import SimpleImputer
```

In [48]:

```
imputer = SimpleImputer()
imputer.fit(X)
```

Out[48]:

```
SimpleImputer(add_indicator=False, copy=True, fill_value=None,
              missing_values=nan, strategy='mean', verbose=0)
```

imputing the missing vaalues with the help of mean imputer.

In [49]:

```
X.head()
```

Out[49]:

	loan_amnt	term	int_rate	installment	grade	emp_length	home_ownership	annual_inc
0	10000.0	3	11.44	329.48	2	0.333681	0.403480	117000.0
1	8000.0	3	11.99	265.68	2	0.063411	0.500841	65000.0
2	15600.0	3	10.49	506.97	2	0.083989	0.403480	43057.0
3	7200.0	3	6.49	220.65	1	0.055174	0.403480	54000.0
4	24375.0	5	17.27	609.33	3	0.040542	0.500841	55000.0

5 rows × 22 columns

In [50]:

```
def all_plots(X):
    for i in X.columns:
        sns.displot(data=X,x=i)
    plt.show()
```

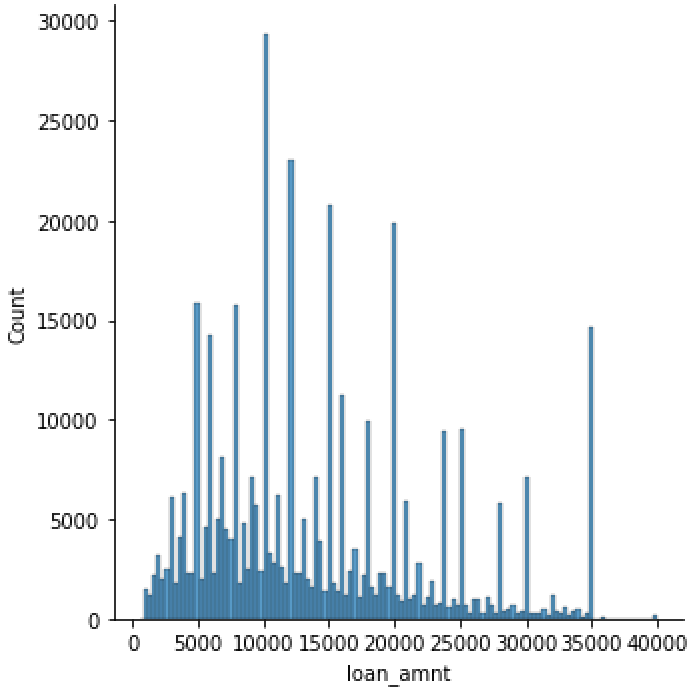
function used to plot the univariet plots.

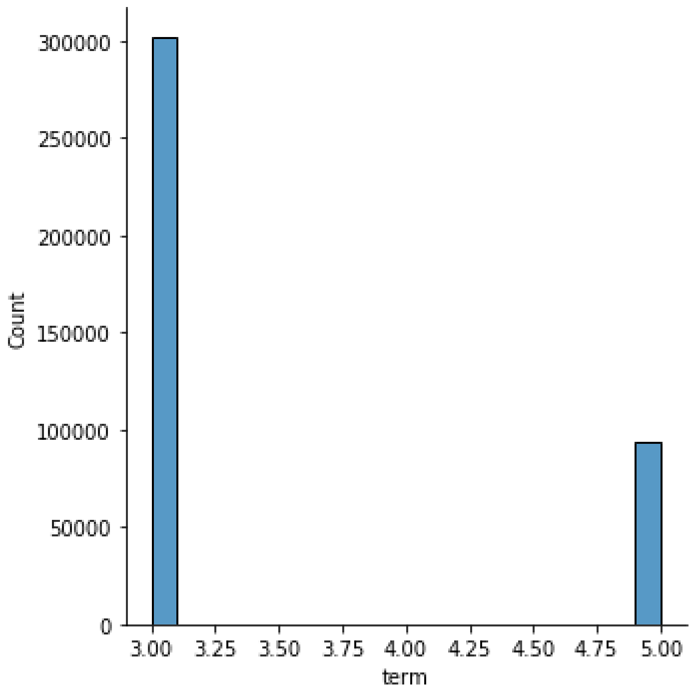
In [51]:

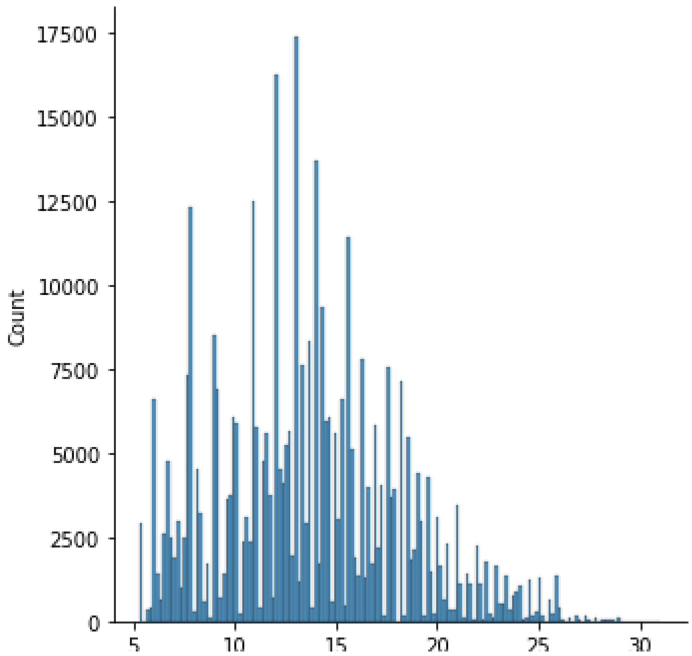
```
plt.rcParams.update({'figure.max_open_warning': 0})
```

In [52]:

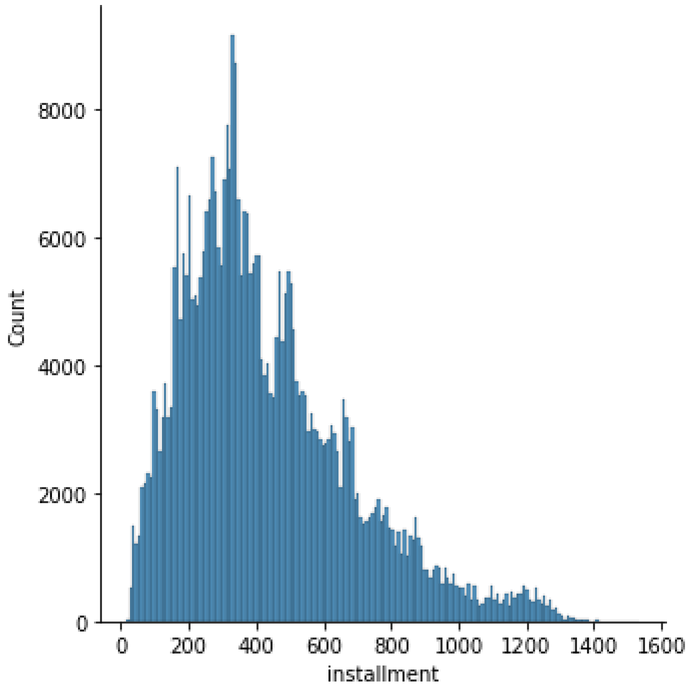
```
all_plots(X)
```



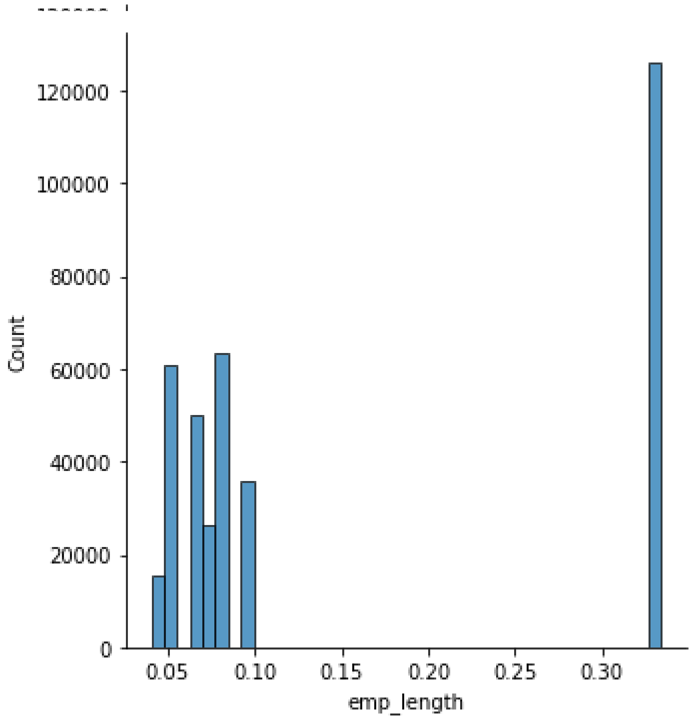




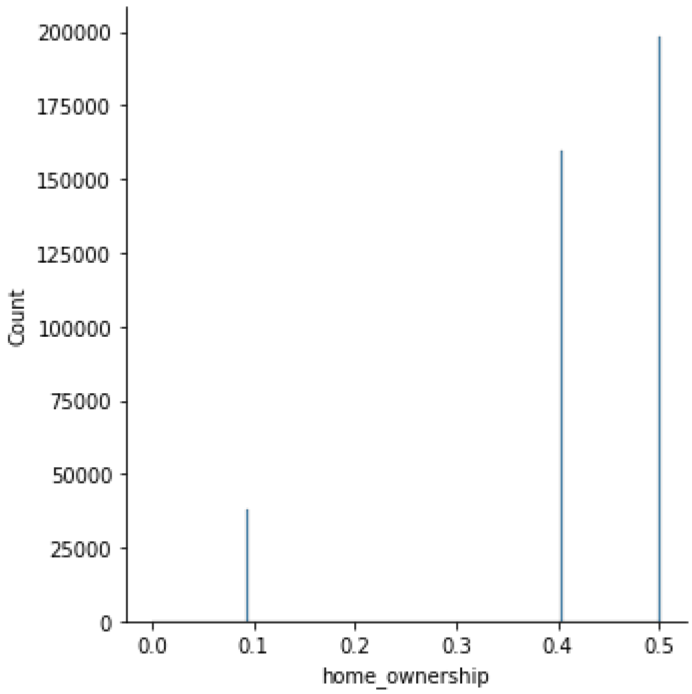
17500
15000
12500
10000
7500
5000
2500
0
Count
5
10
15
20
25
30



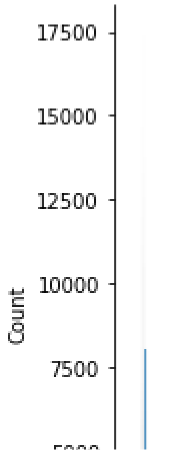
8000
6000
4000
2000
0
Count
0
200
400
600
800
1000
1200
1400
1600
installment

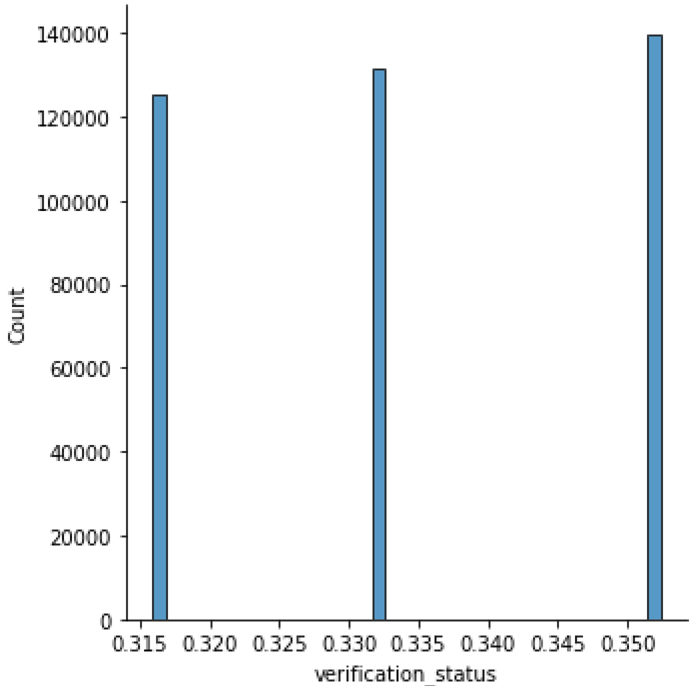


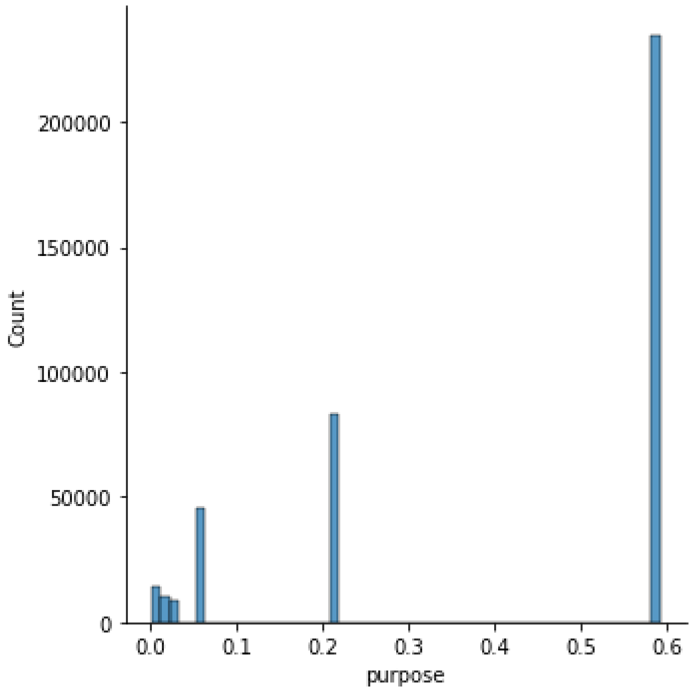
Count
emp_length



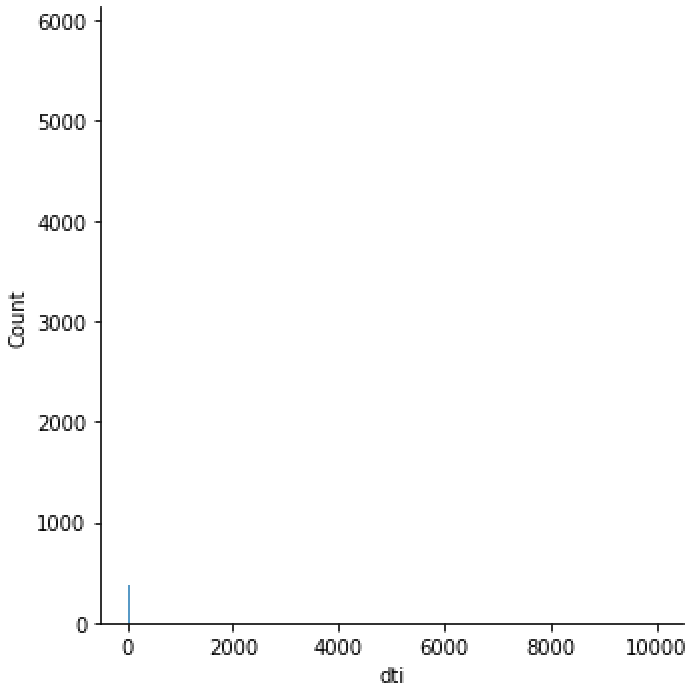
Count
home_ownership


17500
15000
12500
10000
7500
Count

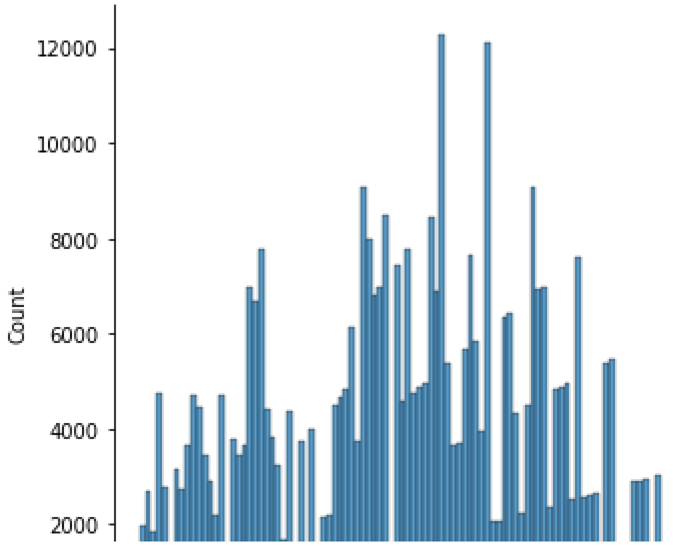

140000
120000
100000
80000
60000
40000
20000
0
Count
0.315
0.320
0.325
0.330
0.335
0.340
0.345
0.350
verification_status



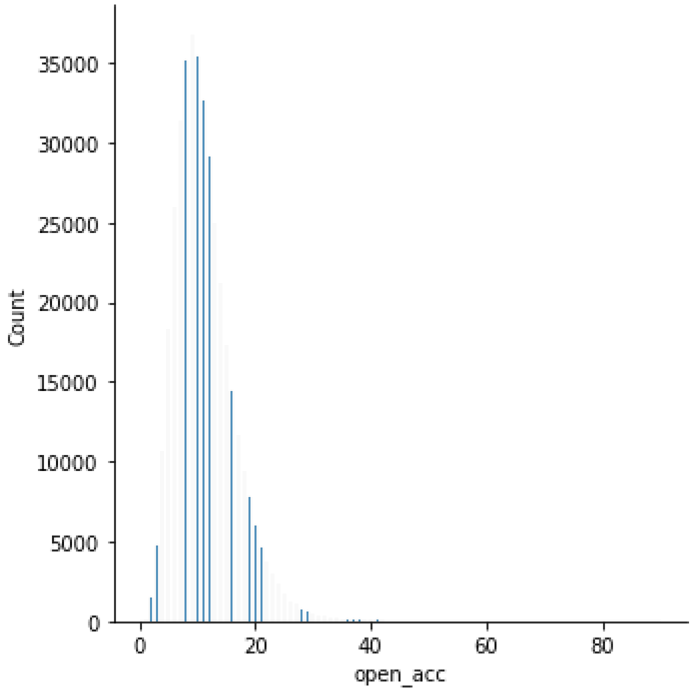
200000
150000
100000
50000
0
Count
0.0
0.1
0.2
0.3
0.4
0.5
0.6
purpose



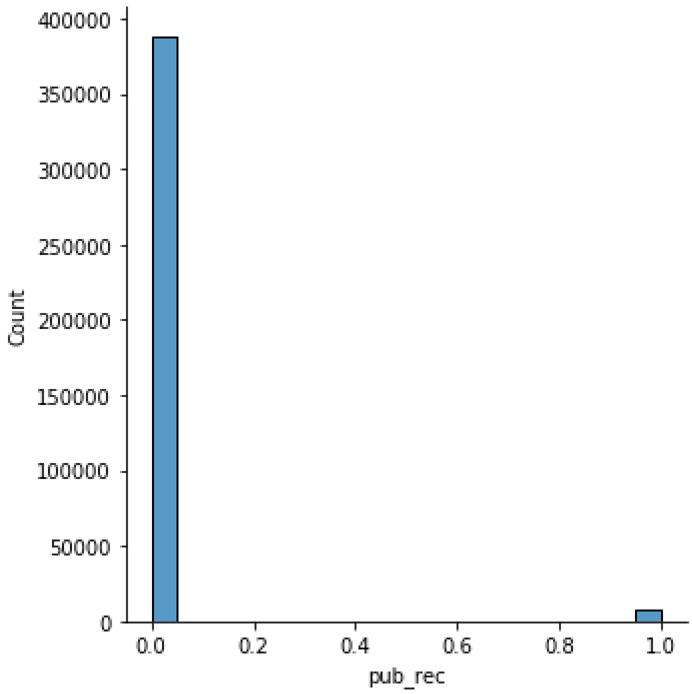
6000
5000
4000
3000
2000
1000
0
Count
0
2000
4000
6000
8000
10000
dti



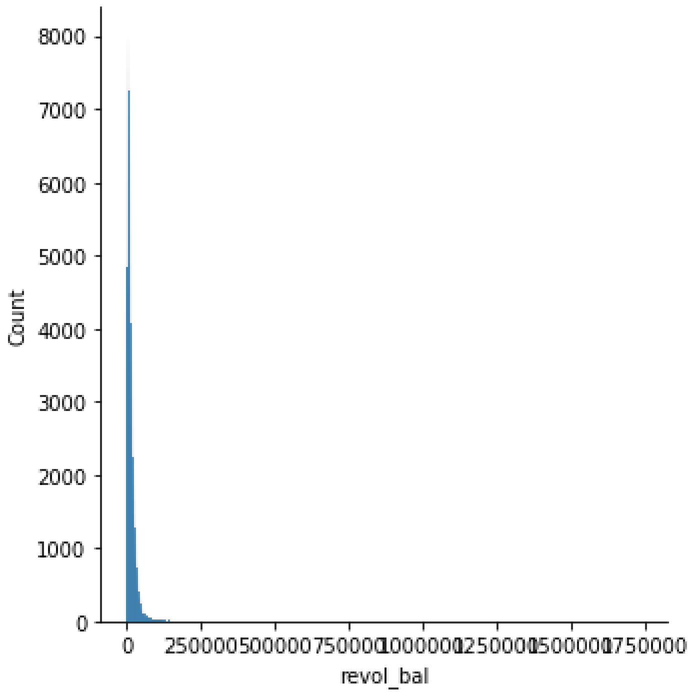
12000
10000
8000
6000
4000
2000
Count

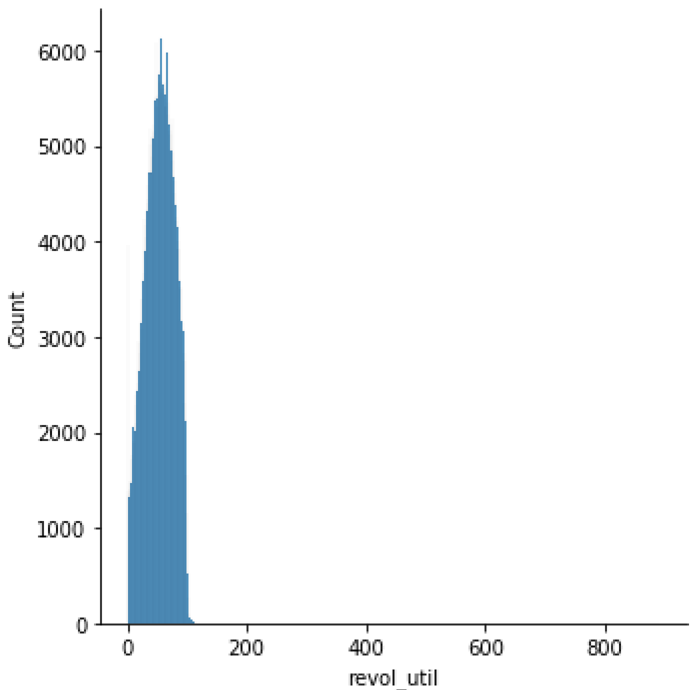


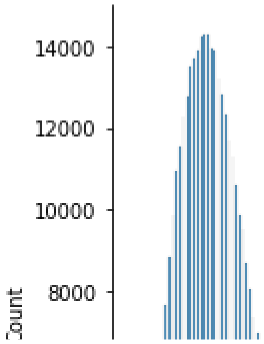
35000
30000
25000
20000
15000
10000
5000
0
Count
0
20
40
60
80
open_acc

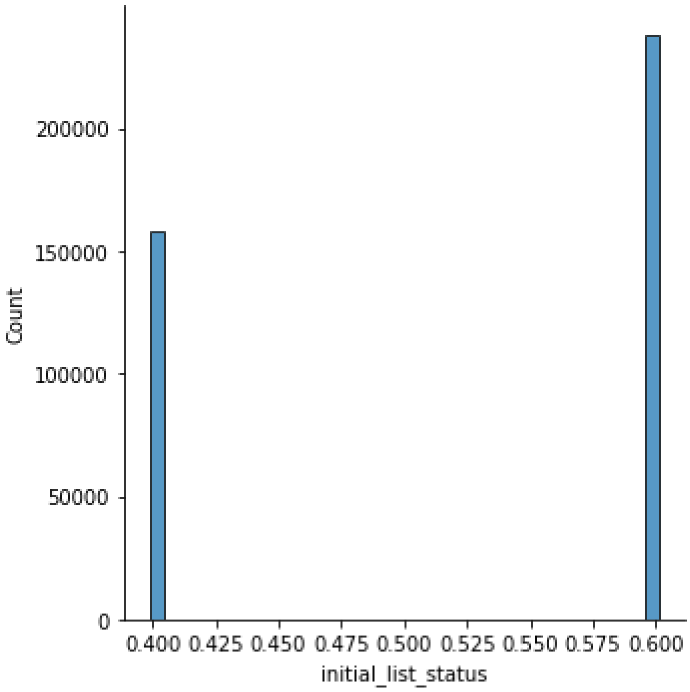


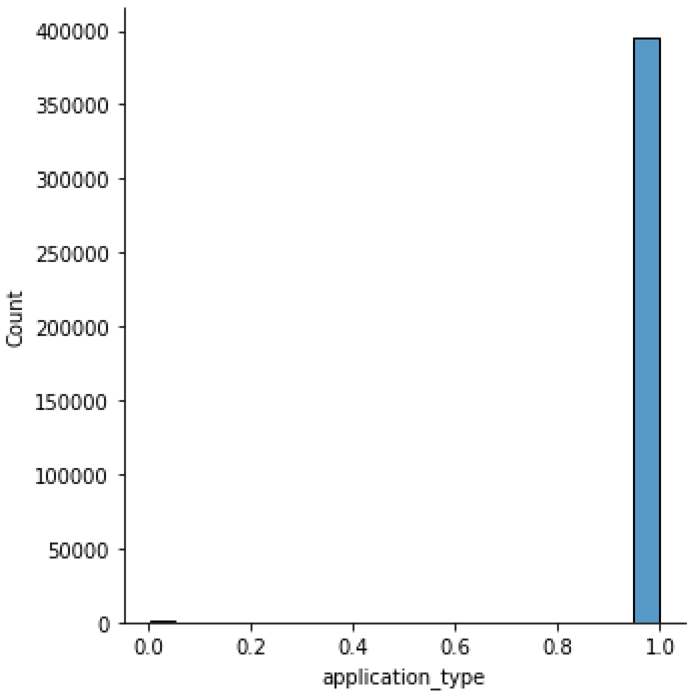
400000
350000
300000
250000
200000
150000
100000
50000
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
pub_rec

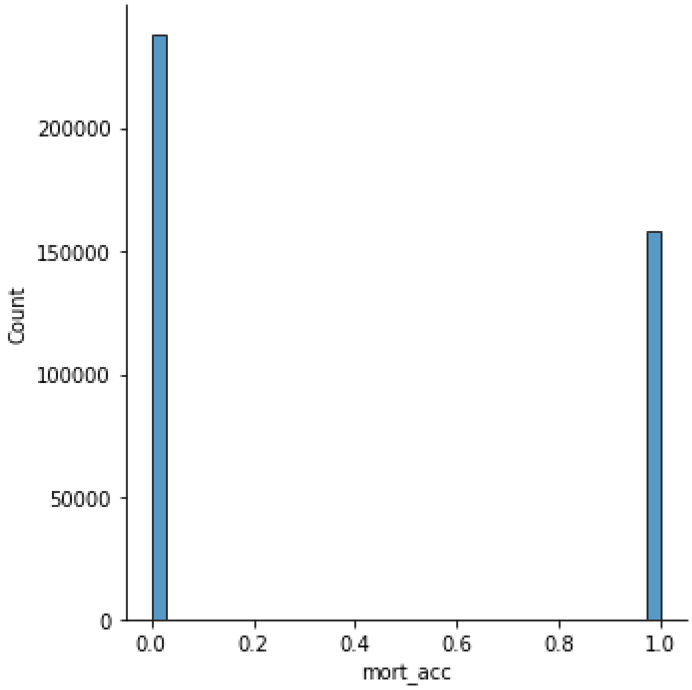

Count
revol_bal
0
250000
500000
750000
1000000
1250000
1500000
1750000
8000
7000
6000
5000
4000
3000
2000
1000

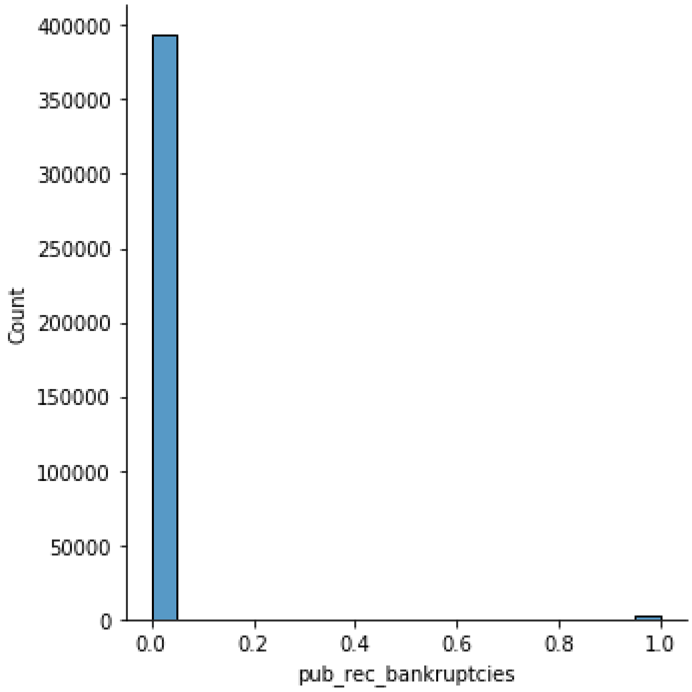

Count
revol_util
0
200
400
600
800
6000
5000
4000
3000
2000
1000


14000
12000
10000
8000
Count


200000
150000
100000
50000
0
Count
0.400 0.425 0.450 0.475 0.500 0.525 0.550 0.575 0.600
initial_list_status


400000
350000
300000
250000
200000
150000
100000
50000
0
Count
0.0 0.2 0.4 0.6 0.8 1.0
application_type

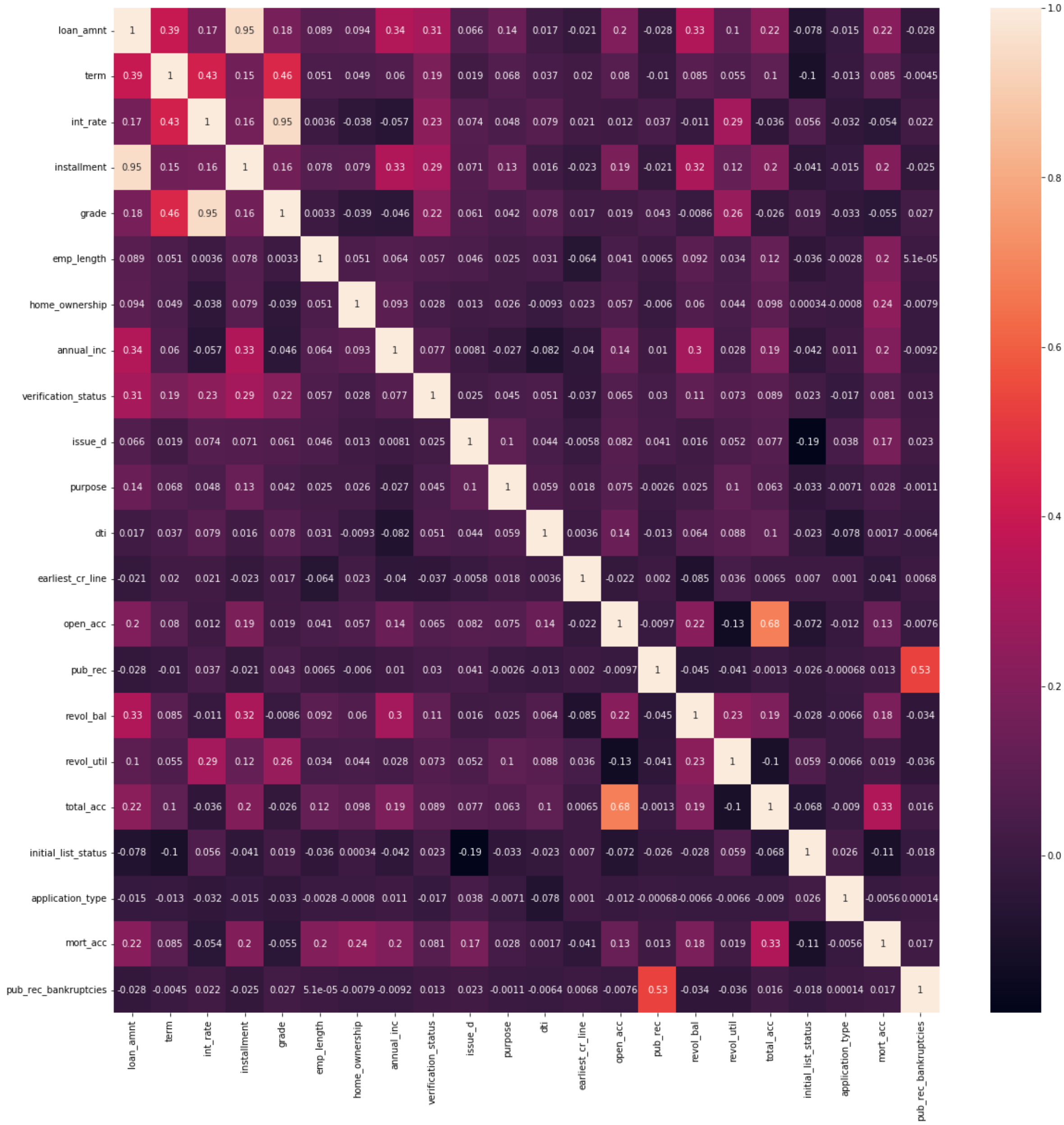




we can see the how the perticular feature is varying some are binary some are multi variet and some are numerical in nature

In [68]:

```
fig, ax = plt.subplots(figsize=(20,20))
sns.heatmap(X.corr(),annot=True,ax=ax)
plt.show()
```



loan_amount and instalment are having high corelation followed by grade and int_rate are having higher corelation pub_rec and pub_rec_bankruptcy ia also having higher corelation

In [49]:

```
X_new = imputer.transform(X)
```

In [50]:

```
from sklearn.preprocessing import StandardScaler
```

In [51]:

```
scaling = StandardScaler()
scaling.fit(X_new)
```

Out[51]:

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

standardizing the data using standerd scaler

In [52]:

```
from sklearn.model_selection import train_test_split
```

In [53]:

```
x_train,x_test,y_train,y_test = train_test_split(X_new,Y,test_size=0.20,random_state=42)
```

splitting 80% of data to train and remaining 20% data to test

In [54]:

```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[54]:

```
((316824, 22), (79206, 22), (316824,), (79206,))
```

In [55]:

```
x_train = scaling.transform(x_train)
```

In [56]:

```
x_test = scaling.transform(x_test)
```

In [57]:

```
from sklearn.linear_model import LogisticRegression
```

importing logistic regression from sklearn

In [58]:

```
model = LogisticRegression(random_state=42).fit(x_train,y_train)
```

In [59]:

```
pred = model.predict(x_test)
```

In [60]:

```
for i,j in zip(X.columns , model.coef_[0]):
    print(f"The feature  {i}  feature importance is:-  {j}")
```

```
The feature  loan_amnt  feature importance is:-  0.054416986801077025
The feature  term  feature importance is:-  -0.21231854222108398
The feature  int_rate  feature importance is:-  -0.01905039778871478
The feature  installment  feature importance is:-  -0.1116727229952053
The feature  grade  feature importance is:-  -0.4384954825225699
The feature  emp_length  feature importance is:-  0.02675927582476799
The feature  home_ownership  feature importance is:-  0.04253679424952343
The feature  annual_inc  feature importance is:-  0.18555571602713264
The feature  verification_status  feature importance is:-  -0.04099548172404
5
The feature  issue_d  feature importance is:-  -0.14512588758930342
The feature  purpose  feature importance is:-  0.0074447428705224465
The feature  dti  feature importance is:-  -0.4360867366819868
The feature  earliest_cr_line  feature importance is:-  0.0171016475512284
The feature  open_acc  feature importance is:-  -0.10054705261849302
The feature  pub_rec  feature importance is:-  -0.02447002931307284
The feature  revol_bal  feature importance is:-  0.06803010744369418
The feature  revol_util  feature importance is:-  -0.07710480695104927
The feature  total_acc  feature importance is:-  0.11389453478009894
The feature  initial_list_status  feature importance is:-  -0.00389655534819
70076
The feature  application_type  feature importance is:-  -0.03057171858511070
8
The feature  mort_acc  feature importance is:-  0.09788161464289102
The feature  pub_rec_bankruptcies  feature importance is:-  0.00101710643768
85601
```

grade is important feature followed by dti

In [61]:

```
from sklearn.metrics import f1_score,precision_score,recall_score,roc_auc_score,confusion_m
```

In [62]:

```
confusion_matrix(y_test.values,pred)
```

Out[62]:

```
array([[ 1226, 14351],
       [ 1130, 62499]], dtype=int64)
```

In [63]:

```
tn, fp, fn, tp = confusion_matrix(y_test.values,pred).flatten()
```

In [64]:

```
precision_score(y_test.values,pred),tp/(tp+fp)
```

Out[64]:

```
(0.8132595966167859, 0.8132595966167859)
```

our model is giving an precision of 81.32%

In [65]:

```
recall_score(y_test.values,pred),tp/(tp+fn)
```

Out[65]:

```
(0.9822408021499631, 0.9822408021499631)
```

our model is giving an recall of 98.22%

In [66]:

```
f1_score(y_test.values,pred)
```

Out[66]:

```
0.8897984752169363
```

our model is giving an F1 score of 88.97%

In [67]:

```
roc_auc_score(y_test.values,pred)
```

Out[67]:

```
0.5304732931594651
```

In [68]:

```
pred_proba = model.predict_proba(x_test)
```

In [69]:

```
pred_proba
```

Out[69]:

```
array([[0.27767617, 0.72232383],
       [0.39748346, 0.60251654],
       [0.32558046, 0.67441954],
       ...,
       [0.14974418, 0.85025582],
       [0.09429087, 0.90570913],
       [0.40694468, 0.59305532]])
```

predicting the probabilities of both teh class 0 and class 1

In [70]:

```
fpr, tpr, threshold = roc_curve(y_test.values,pred_proba[:,1])
```

In [71]:

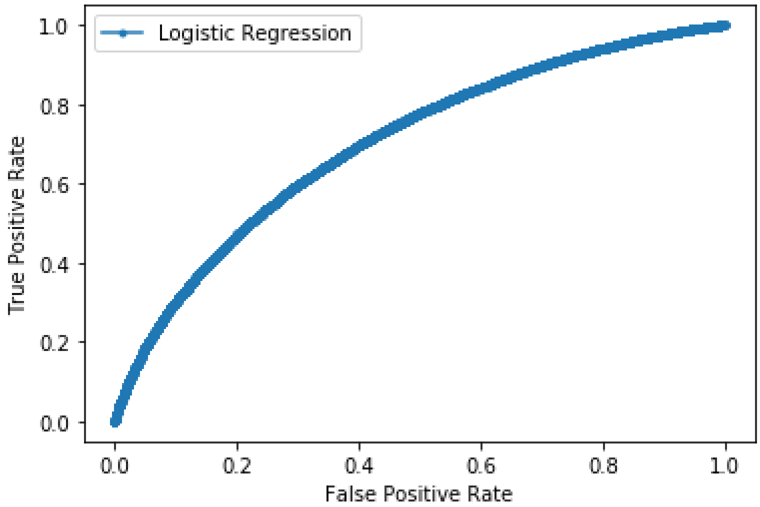
```
fpr, tpr, threshold
```

Out[71]:

```
(array([0.00000000e+00, 6.41972138e-05, 6.41972138e-05, ...,
        9.99871606e-01, 1.00000000e+00, 1.00000000e+00]),
 array([0.00000000e+00, 0.00000000e+00, 1.25728834e-04, ...,
        9.99984284e-01, 9.99984284e-01, 1.00000000e+00]),
 array([2.        , 1.        , 0.99795403, ..., 0.24356935, 0.23205378,
        0.18039997]))
```

In [72]:

```
plt.plot(fpr,tpr,marker='.',label='Logistic Regression')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```



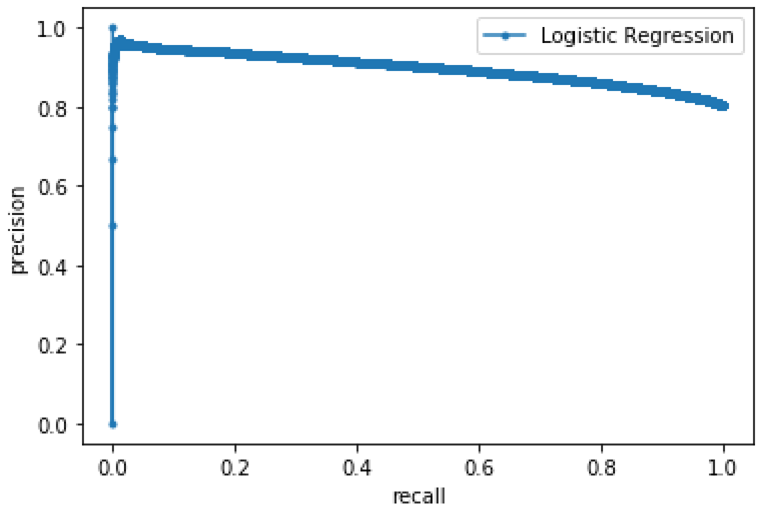
plotting the ROC-AUC curve

In [73]:

```
precision_val, recall_val, threshold  = precision_recall_curve(y_test,pred_proba[:,1])
```

In [74]:

```
plt.plot(recall_val,precision_val,marker='.',label='Logistic Regression')
plt.xlabel('recall')
plt.ylabel('precision')
plt.legend()
plt.show()
```



plotting the precision recall curve we can observe that if we want to get more precision (less FP) thenn we have to sacrifice the recall score.

In [75]:

```
max(precision_val),threshold
```

Out[75]:

```
(1.0,
 array([0.18039997, 0.23205378, 0.24101828, ..., 0.99999912, 1.        ,
        1.        ]))
```

In [76]:

```
recall_val
```

Out[76]:

```
array([1.00000000e+00, 9.99984284e-01, 9.99984284e-01, ...,
       1.57161043e-05, 0.00000000e+00, 0.00000000e+00])
```

In [77]:

```
from sklearn.model_selection import GridSearchCV
```

In [78]:

```
model_h_tuned = LogisticRegression(random_state=42)
```

In [96]:

```
hyperparameters = {'C':np.arange(0.1,3.4,0.1)}
best_cv = GridSearchCV(estimator=model_h_tuned,param_grid=hyperparameters,n_jobs=-1,scoring
```

In [97]:

```
best_cv.fit(x_train,y_train)
```

Out[97]:

```
GridSearchCV(cv=None, error_score=nan,
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fal
se,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=Non
e,
                                          max_iter=100, multi_class='auto',
                                          n_jobs=None, penalty='l2',
                                          random_state=42, solver='lbfgs',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'C': array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
0.9, 1. , 1.1, 1.2, 1.3,
       1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6,
       2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='precision', verbose=0)
```

hyperparameter tuning the model to get the best value of C

In [98]:

```
best_cv.best_estimator_
```

Out[98]:

```
LogisticRegression(C=0.8, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=42, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [99]:

```
pred_proba = best_cv.predict_proba(x_test)
```

In [100]:

```
precision_score(y_test.values,best_cv.predict(x_test))
```

Out[100]:

```
0.8132595966167859
```

In [101]:

```
precision_score(y_test.values,pred_proba[:,1]>0.96)
```

Out[101]:

```
0.9625
```

the best posible value of precision is 0.9625 and we are getting this value at the threshold 0.96 insted of 0.5 at the same place our recall score is only 0.84%

In [102]:

```
recall_score(y_test.values,pred_proba[:,1]>0.96)
```

Out[102]:

```
0.008470980213424696
```

Actionable Insights & Recommendations

if we want to balance between bad loan and at the same time we have to give loans to the people we are repaying correctly we have to use a threshold which balences out both th eprecision and recall score which would be some where around 0.5 and 0.96.

as grade and dti are contrubuting the most to the prediction we can fine tune and get the exact values which will defenetly help in preventing the bad loans.

more than 80% of the loan given by the bank is repayed so we can improve offer some reduction in the interest for the people who are repaying the loan on time will decrease the bad loan.

In [ ]:

```

```