Context:

- A Non-Banking Finance Company like 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 is building an underwriting layer to determine the creditworthiness of MSMEs as well as individuals.
- Company deploys formal credit to salaried individuals and businesses 4 main financial instruments:
  - Personal Loan
  - EMI Free Loan
  - Personal Overdraft
  - Advance Salary Loan
- This case study will focus on the underwriting process behind Personal Loan only

Problem Statement:

- Given a set of attributes for an Individual, determine if a credit line should be extended to them. If so, what should the repayment terms be in business recommendations?

Tradeoff Questions:

- How can we make sure that our model can detect real defaulters and there are less false positives? This is important as we can lose out on an opportunity to finance more individuals and earn interest on it.
- Since NPA (non-performing asset) is a real problem in this industry, it's important we play safe and shouldn't disburse loans to anyone

In [1]:

```
# Importing necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, f1_score, precision_score
from imblearn.over_sampling import SMOTE
from category_encoders import TargetEncoder
```

In [2]:

```
df = pd.read_csv("logistic_regression.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	hor
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	
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	

5 rows × 27 columns

In [4]:
```
df.shape
```

Out[4]: (396030, 27)

- **396030 data points , 26 features , 1 label.**

Missing Values Check:

In [5]:
```
# Missing Values Check
def missing_df(data):
    total_missing_df = data.isna().sum().sort_values(ascending=False)
    percentage_missing_df = ((data.isna().sum() / len(data)) * 100).so
    missingDF = pd.concat([total_missing_df, percentage_missing_df], a
    return missingDF

missing_data = missing_df(df)

```

In [6]:
```
# (df.isna().sum() / df.shape[0] ) * 100
```

Descriptive Statistics :

In [7]:
```
df.describe().round(1)
```

Out[7]:

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_ba
count	396030.0	396030.0	396030.0	396030.0	396030.0	396030.0	396030.0	396030.
mean	14113.9	13.6	431.8	74203.2	17.4	11.3	0.2	15844.
std	8357.4	4.5	250.7	61637.6	18.0	5.1	0.5	20591.
min	500.0	5.3	16.1	0.0	0.0	0.0	0.0	0.
25%	8000.0	10.5	250.3	45000.0	11.3	8.0	0.0	6025.
50%	12000.0	13.3	375.4	64000.0	16.9	10.0	0.0	11181.
75%	20000.0	16.5	567.3	90000.0	23.0	14.0	0.0	19620.
max	40000.0	31.0	1533.8	8706582.0	9999.0	90.0	86.0	1743266.

- **Loan Amount, Installments, Annual Income , revol_bal : all these columns have large differnece in mean and median . That means outliers are present in the data.**

In [8]:
```
# df.nunique()
```

In [9]:
```
# df.info()
```

In [12]:
```
columns_type = df.dtypes
```

In [13]:
```
columns_type[columns_type=="object"]
```

Out[13]:
```
term                    object
grade                   object
sub_grade               object
emp_title               object
emp_length              object
home_ownership          object
verification_status     object
issue_d                 object
loan_status             object
purpose                 object
title                   object
earliest_cr_line        object
initial_list_status     object
application_type        object
address                 object
dtype: object
```

In [14]:
```
df.describe(include="object")
```

Out[14]:

	term	grade	sub_grade	emp_title	emp_length	home_ownership	verification_sta
count	396030	396030	396030	373103	377729	396030	396
unique	2	7	35	173105	11	6	
top	36 months	B	B3	Teacher	10+ years	MORTGAGE	Veri
freq	302005	116018	26655	4389	126041	198348	139

In [15]:
```
len(columns_type[columns_type=="object"])
```

Out[15]: 15

- **15 Non-numerical (categorical/date time) features present in the dataset.**

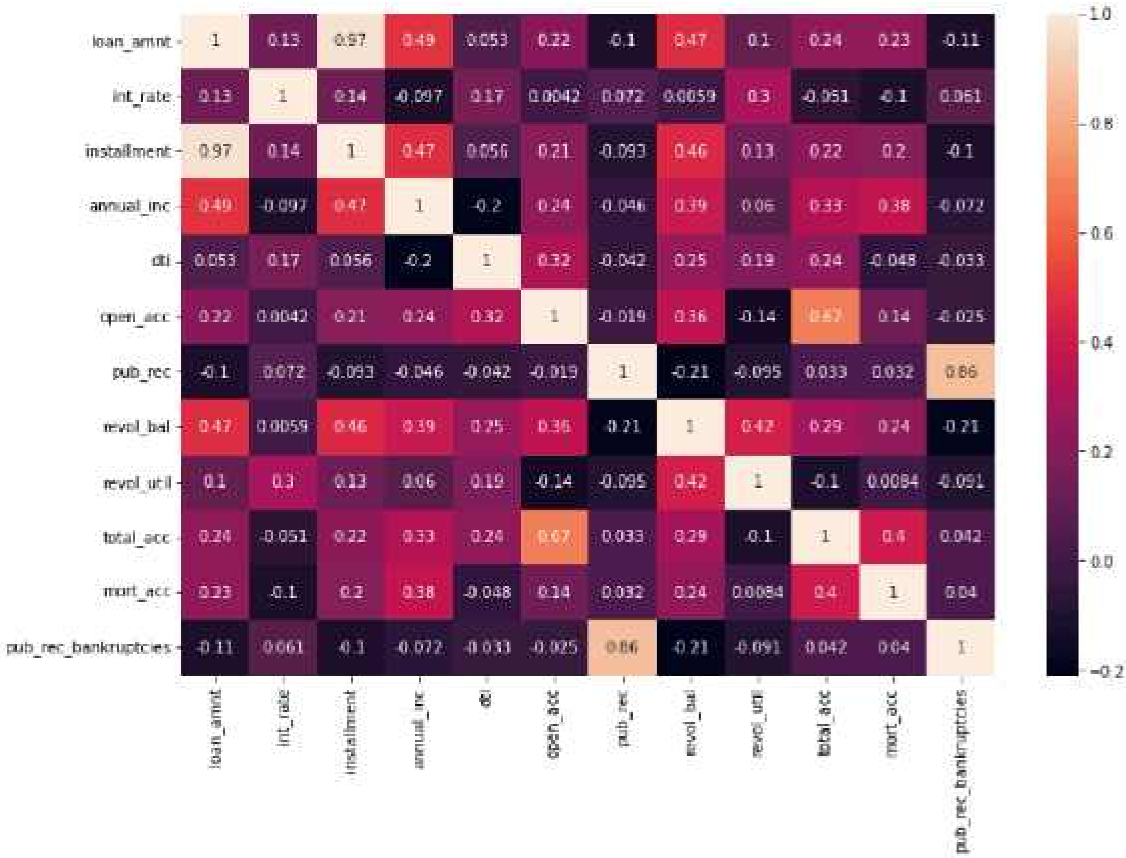
In [16]:
```
df["loan_status"].value_counts(normalize=True)*100
```

Out[16]:
```
Fully Paid     80.387092
Charged Off    19.612908
Name: loan_status, dtype: float64
```

- **As we can see, there is an imbalance in the data.**
- 80% belongs to the class 0 : which is loan fully paid.
- 20% belongs to the class 1 : which were charged off.

In [17]:

```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(method='spearman'), annot=True)
plt.show()
```



loan_amnt :

- **The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.**

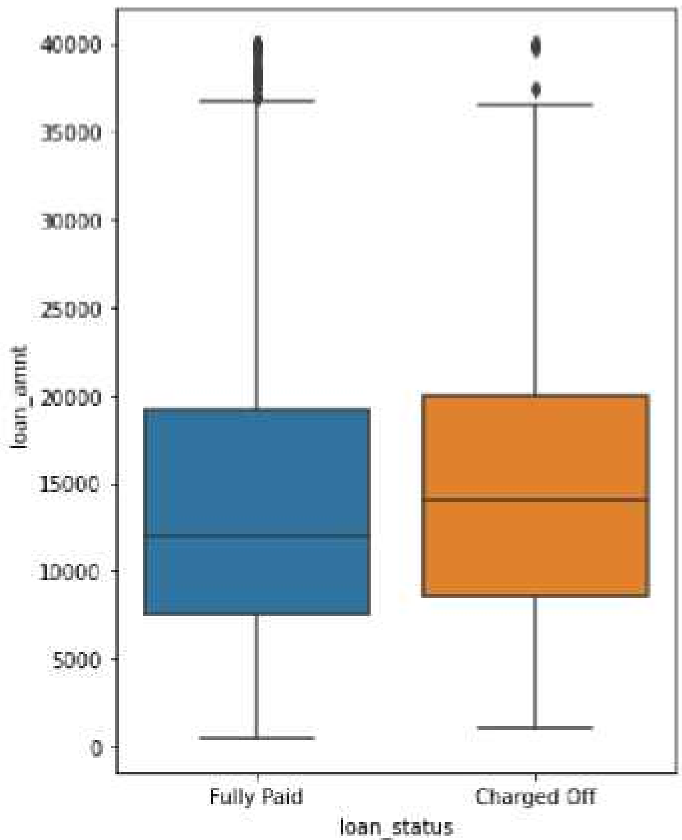
In [18]:

```
df.groupby(by = "loan_status")["loan_amnt"].describe()
```

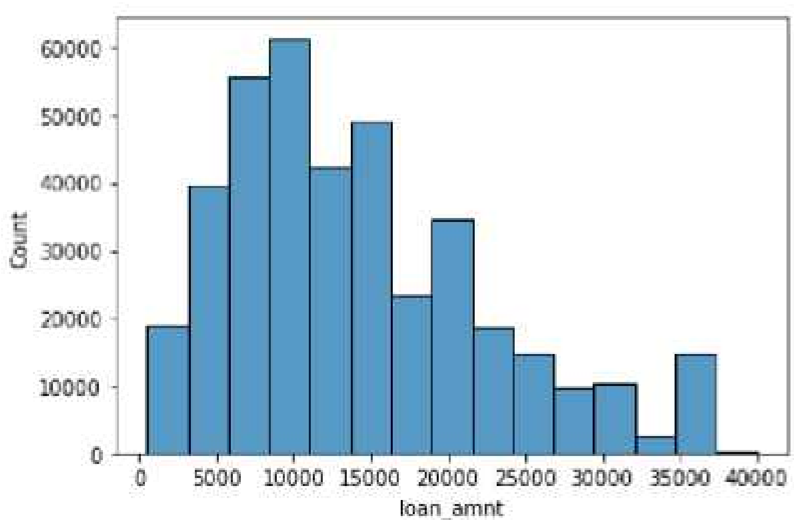
Out[18]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	15126.300967	8505.090557	1000.0	8525.0	14000.0	20000.0	40000.0
Fully Paid	318357.0	13866.878771	8302.319699	500.0	7500.0	12000.0	19225.0	40000.0

In [19]:
```
plt.figure(figsize=(5,7))
sns.boxplot(y=df["loan_amnt"],
            x=df["loan_status"])
```
Out[19]: <AxesSubplot:xlabel='loan_status', ylabel='loan_amnt'>



In [20]:
```
sns.histplot(df["loan_amnt"],bins = 15)
```
Out[20]: <AxesSubplot:xlabel='loan_amnt', ylabel='Count'>



- **for loan status Charged_off, the mean and median of loan_amount is higher than fully paid.**

- **also the distribution of loan_amnt is right skewed, which says it has outlier presence**

term :

- **The number of payments on the loan. Values are in months and can be either 36 or 60.**

In [21]:
```
df["term"].value_counts(dropna=False)
```

Out[21]:
```
 36 months    302005
 60 months     94025
Name: term, dtype: int64
```

P[loan_statis | term]

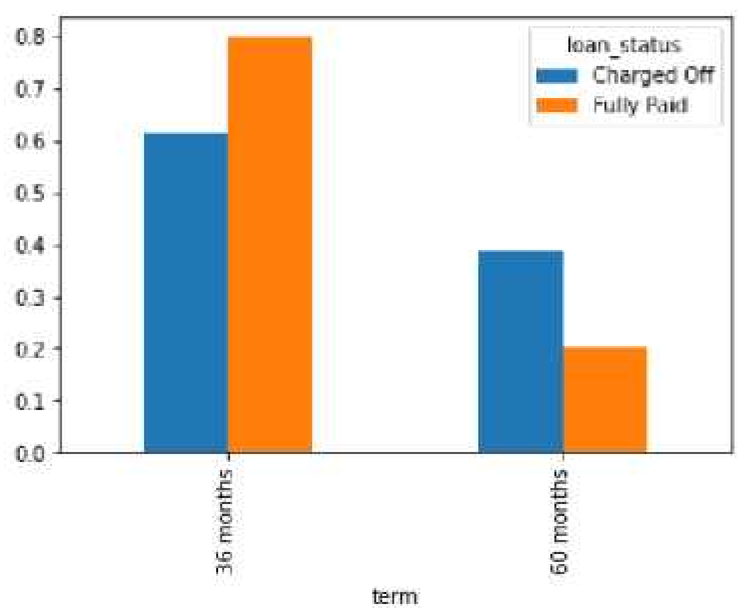
In [22]:
```
pd.crosstab(index=df["term"],
            columns=df["loan_status"], normalize="index" , margins  = 
           ) * 100
```

Out[22]:

loan_status	Charged Off	Fully Paid
term		
36 months	15.774573	84.225427
60 months	31.941505	68.058495
All	19.612908	80.387092

In [23]:
```
pd.crosstab(index=df["term"],
            columns =df["loan_status"], normalize="columns"
           ).plot(kind = "bar")
```

Out[23]: <AxesSubplot:xlabel='term'>



In [24]:
```python
# as we can observe
# the conditional probability
# of loan fully paid given that its 36 month term is higher then charge

# loan fully paid probability when 60 month term is lower than charged
```

In [25]:
```python
term_values = {' 36 months': 36, ' 60 months': 60}
df['term'] = df['term'].map(term_values)

```

int_rate :

- **Interest Rate on the loan**

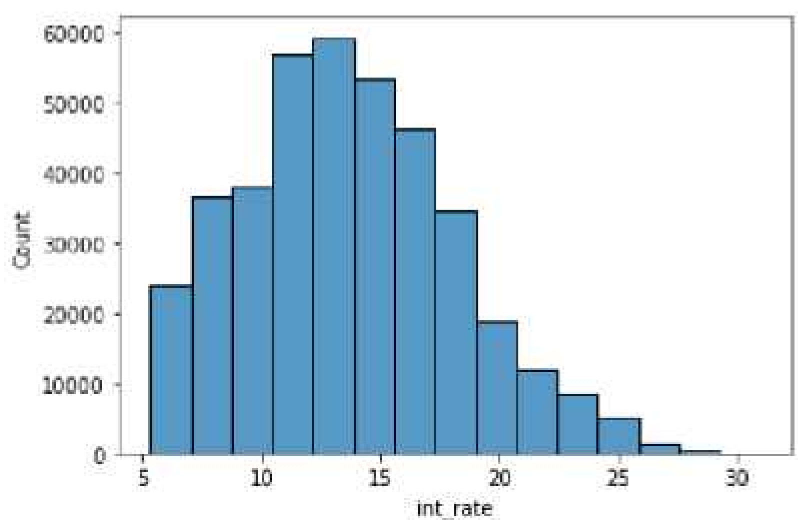
In [26]:
```python
df.groupby(by = "loan_status")["int_rate"].describe()
```

Out[26]:

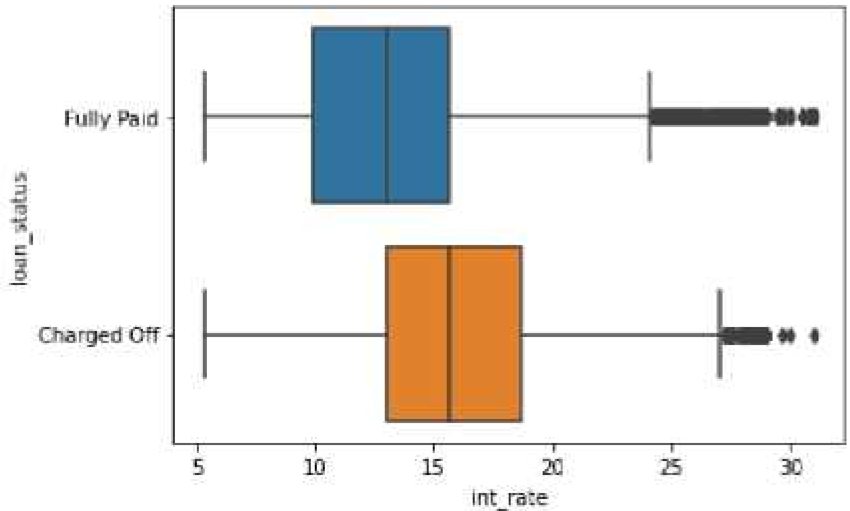
loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	15.882587	4.388135	5.32	12.99	15.61	18.64	30.99
Fully Paid	318357.0	13.092105	4.319105	5.32	9.91	12.99	15.61	30.99

In [27]:
```python
sns.histplot(df["int_rate"],bins = 15)
```

Out[27]: <AxesSubplot:xlabel='int_rate', ylabel='Count'>



In [28]:
```
sns.boxplot(x=df["int_rate"],
            y=df["loan_status"])
```
Out[28]: <AxesSubplot:xlabel='int_rate', ylabel='loan_status'>



In [29]:
```
df[df["loan_status"] == "Charged Off"]["int_rate"].median(),df[df["loar
```
Out[29]: (15.61, 15.882587256832393)

In [30]:
```
df[df["loan_status"] == "Fully Paid"]["int_rate"].median(),df[df["loan_
```
Out[30]: (12.99, 13.092105403703817)

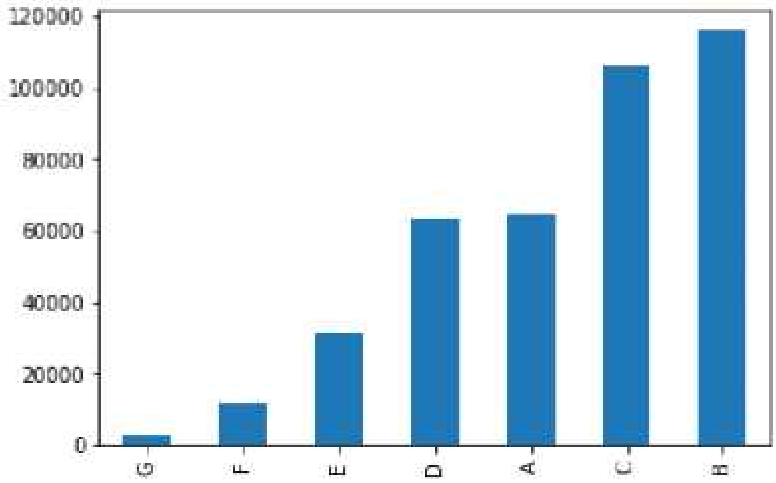
In [31]:
```
# for charge_off Loan Status ,
# interest_rate median and mean is higher than fully paid.
```

- **for loan status Charged_off, the mean and median of interest_rate is higher than fully paid.**
- **also the distribution of interest_rate is right skewed, which says it has outlier presence.**

grade :

- **LoanTap assigned loan grade**
- **Loan grades are set based on both the borrower's credit profile and the nature of the contract.**

In [32]:
```
df["grade"].value_counts().sort_values().plot(kind = "bar")
```
Out[32]: <AxesSubplot:>



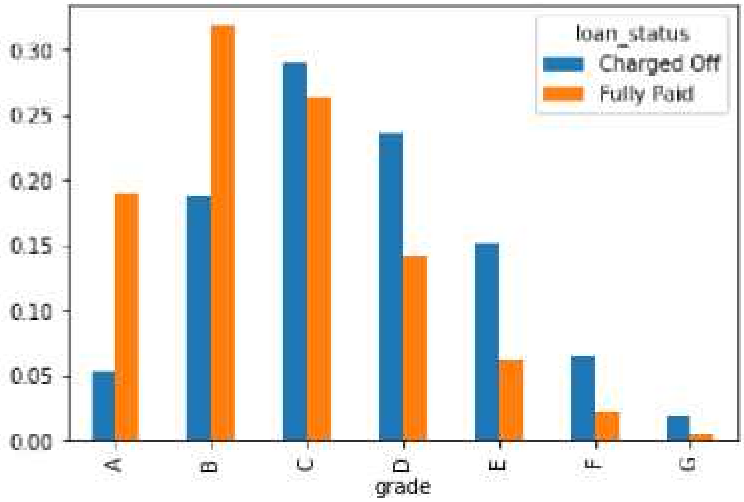
In [33]:
```
df["grade"].value_counts(dropna=False)
```
Out[33]:
```
B    116018
C    105987
A     64187
D     63524
E     31488
F     11772
G      3054
Name: grade, dtype: int64
```

In [34]:
```
pd.crosstab(index = df["grade"],
            columns= df["loan_status"],normalize= "index", margins = Tı
```
Out[34]:

loan_status	Charged Off	Fully Paid
grade		
A	0.062879	0.937121
B	0.125730	0.874270
C	0.211809	0.788191
D	0.288678	0.711322
E	0.373634	0.626366
F	0.427880	0.572120
G	0.478389	0.521611
All	0.196129	0.803871

```
In [35]:  pd.crosstab(index = df["grade"],
                        columns= df["loan_status"],normalize= "columns").plot(kind
```

Out[35]: <AxesSubplot:xlabel='grade'>




```
In [36]:  #  probability of loan_status as fully_paid decreases with grade is E,F
```

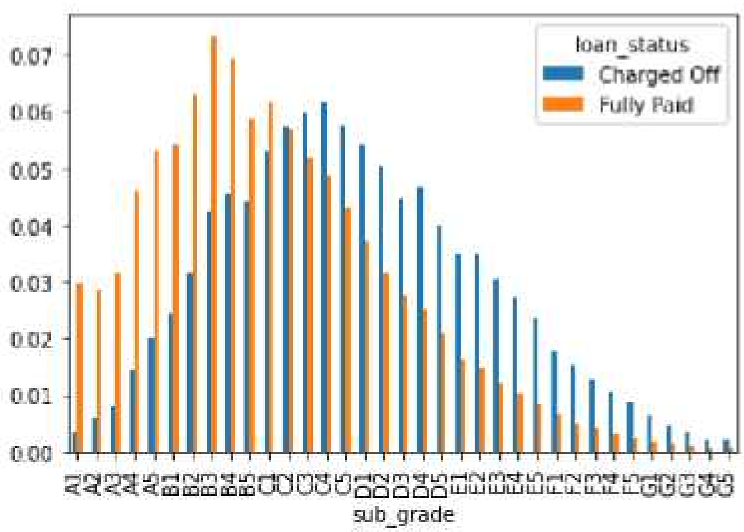
sub_grade :

- **LoanTap assigned loan subgrade**

```
In [37]:  # pd.crosstab(index = df["sub_grade"],
          #               columns= df["loan_status"],normalize= "index", margins =
```

```
In [38]:  pd.crosstab(index = df["sub_grade"],
                        columns= df["loan_status"],normalize= "columns", ).plot(kin
```

Out[38]: <AxesSubplot:xlabel='sub_grade'>



In [39]:
```
# Similar pattern is observed for sub_grade as grade .

#  later target encoding
```

emp_title :

- **The job title supplied by the Borrower when applying for the loan.***

In [40]:
```
df["emp_title"].value_counts(dropna=False).sort_values(ascending=False)
```

Out[40]:
```
NaN                   22927
Teacher                4389
Manager                4250
Registered Nurse       1856
RN                     1846
Supervisor             1830
Sales                  1638
Project Manager        1505
Owner                  1410
Driver                 1339
Office Manager         1218
manager                1145
Director               1089
General Manager        1074
Engineer                995
Name: emp_title, dtype: int64
```

In [41]:
```
df["emp_title"].nunique()
```

Out[41]:
```
173105
```

In [42]:
```
# missing values need to be treated with model based imputation .


# total unique job_titles are 173,105.
# target encoding while creating model.
```

emp_length :

- **Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.**

In [43]:
```
df["emp_length"].value_counts(dropna=False)
```

Out[43]:
```
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
NaN           18301
9 years       15314
Name: emp_length, dtype: int64
```

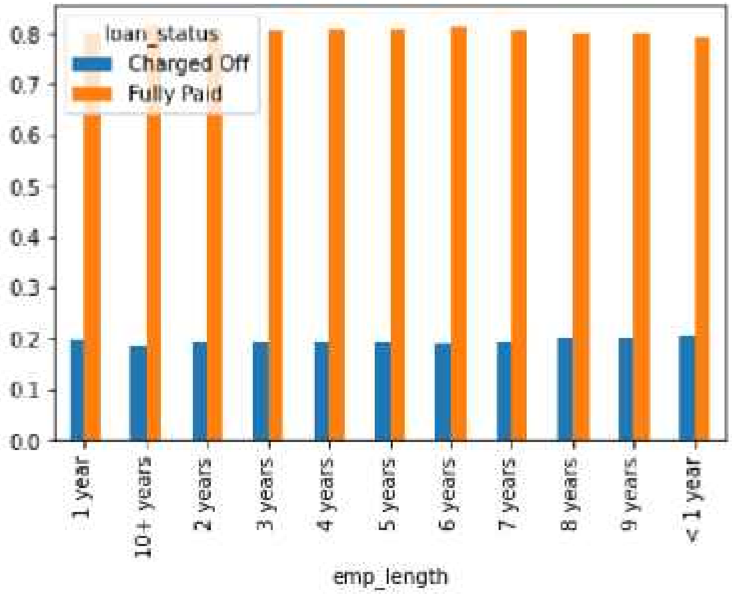
In [44]:
```
pd.crosstab(index = df["emp_length"],
            columns= df["loan_status"],normalize= "index", margins = Tr
```

Out[44]:

loan_status	Charged Off	Fully Paid
emp_length		
1 year	19.913453	80.086547
10+ years	18.418610	81.581390
2 years	19.326206	80.673794
3 years	19.523133	80.476867
4 years	19.238477	80.761523
5 years	19.218721	80.781279
6 years	18.919438	81.080562
7 years	19.477400	80.522600
8 years	19.976002	80.023998
9 years	20.047016	79.952984
< 1 year	20.687155	79.312845
All	19.229395	80.770605

In [45]:
```
pd.crosstab(index = df["emp_length"],
            columns= df["loan_status"],normalize= "index").plot(kind =
```

Out[45]: <AxesSubplot:xlabel='emp_length'>



In [46]:
```
# visually there doent seems to be much correlation between employement
# and loan_status.

```

In [47]:
```
from scipy import stats
```

In [48]:
```
stats.chi2_contingency(pd.crosstab(index = df["emp_length"],
            columns= df["loan_status"]))
```

Out[48]:
```
(122.11317384460878,
 1.88404995201913e-21,
 10,
 array([[  4976.95191526,  20905.04808474],
        [ 24236.9212716 , 101804.0787284 ],
        [  6889.31521011,  28937.68478989],
        [  6088.98780607,  25576.01219393],
        [  4605.82459912,  19346.17540088],
        [  5094.82810428,  21400.17189572],
        [  4007.59813252,  16833.40186748],
        [  4003.36766571,  16815.63233429],
        [  3685.89036055,  15482.10963945],
        [  2944.78949194,  12369.21050806],
        [  6100.52544284,  25624.47455716]]))
```

home_ownership :

- **The home ownership status provided by the borrower during registration or obtained from the credit report.**

In [49]:
```
df["home_ownership"].value_counts(dropna=False)
```

Out[49]:
```
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
ANY              3
Name: home_ownership, dtype: int64
```

In [50]:
```
df["home_ownership"] = df["home_ownership"].replace({"NONE":"OTHER", "A
```

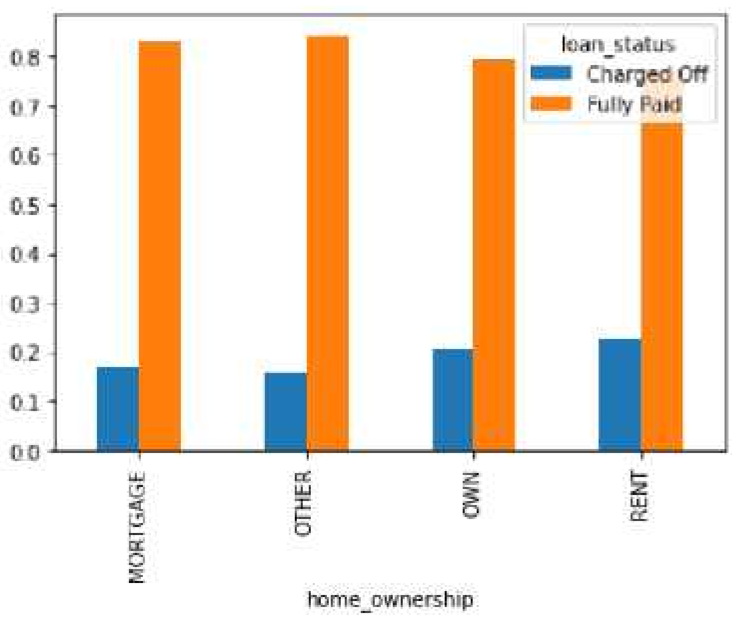
In [51]:
```
pd.crosstab(index = df["home_ownership"],
            columns= df["loan_status"],normalize= "index", margins = Tr
```

Out[51]:

loan_status home_ownership	Charged Off	Fully Paid
MORTGAGE	16.956057	83.043943
OTHER	15.753425	84.246575
OWN	20.680337	79.319663
RENT	22.662244	77.337756
All	19.612908	80.387092

In [52]:
```
pd.crosstab(index = df["home_ownership"],
            columns= df["loan_status"],normalize= "index").plot(kind= '
```

Out[52]: <AxesSubplot:xlabel='home_ownership'>



In [53]:
```
# visually there doent seems to be much correlation between home_owners
# and loan_status.
# later target encoding or label encoding .
```

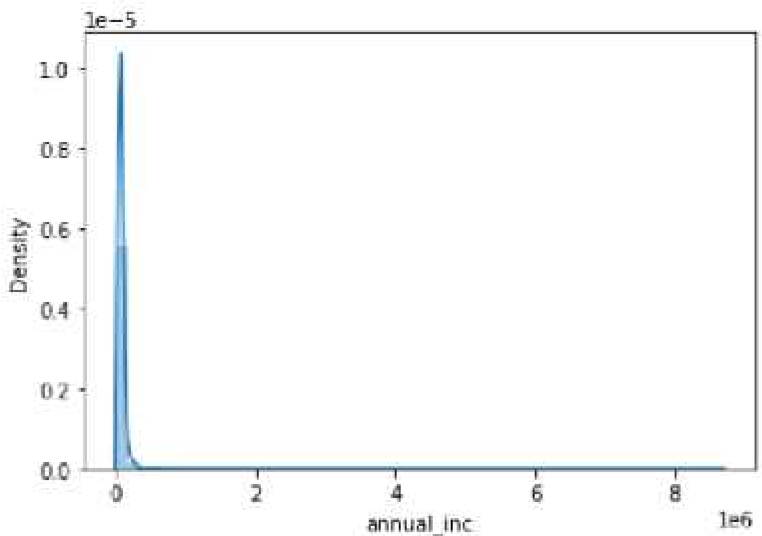
annual_inc :

- **The self-reported annual income provided by the borrower during registration.**

In [54]:
```
sns.distplot(df["annual_inc"])
```

```
C:\Users\ABC\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[54]: <AxesSubplot:xlabel='annual_inc', ylabel='Density'>



In [55]:
```
df["annual_inc"].describe()
```

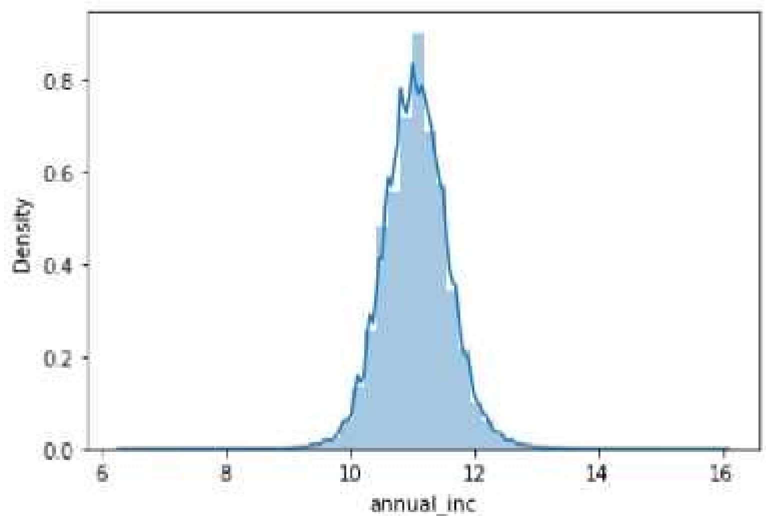
Out[55]:
```
count    3.960300e+05
mean     7.420318e+04
std      6.163762e+04
min      0.000000e+00
25%      4.500000e+04
50%      6.400000e+04
75%      9.000000e+04
max      8.706582e+06
Name: annual_inc, dtype: float64
```

In [56]:

```
1 sns.distplot(np.log(df[df["annual_inc"]>0]["annual_inc"]))
```

```
C:\Users\ABC\anaconda3\lib\site-packages\seaborn\distributions.py:2619: Fu
tureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-l
evel function with similar flexibility) or `histplot` (an axes-level funct
ion for histograms).
  warnings.warn(msg, FutureWarning)
```

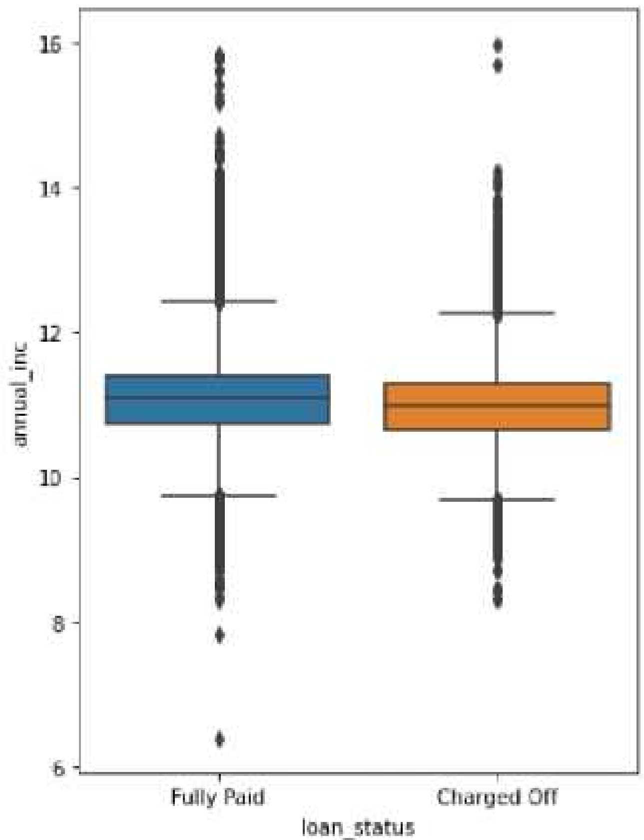
Out[56]: <AxesSubplot:xlabel='annual_inc', ylabel='Density'>



In [57]:

```
plt.figure(figsize=(5,7))
sns.boxplot(y=np.log(df[df["annual_inc"]>0]["annual_inc"]),
            x=df["loan_status"])
```

Out[57]: <AxesSubplot:xlabel='loan_status', ylabel='annual_inc'>



In [58]:

```
##from above boxplot, there seems to be no difference between annual ir
# for loan status categories

```

verification_status :

- **Indicates if income was verified by LoanTap, not verified, or if the income source was verified**

In [59]:

```
df["verification_status"].value_counts(dropna=False)
```

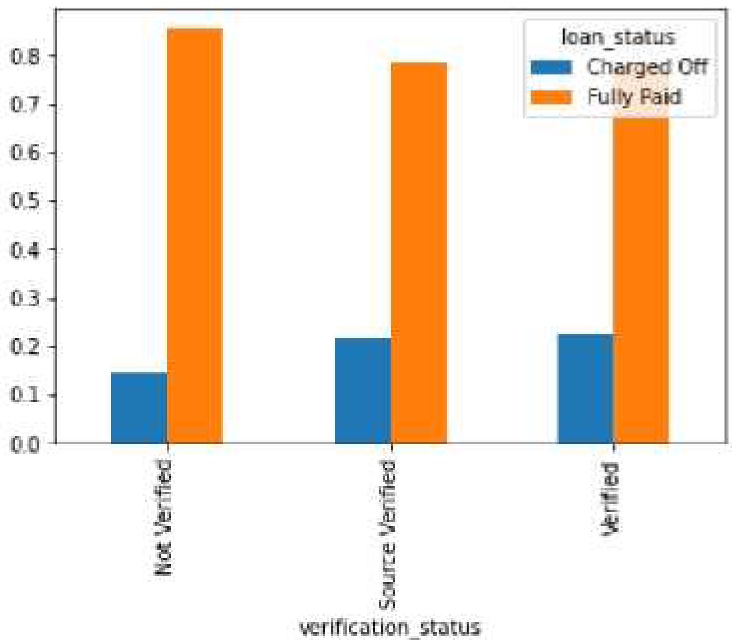
Out[59]:

```
Verified           139563
Source Verified    131385
Not Verified       125082
Name: verification_status, dtype: int64
```

In [60]:
```
pd.crosstab(index = df["verification_status"],
            columns= df["loan_status"],normalize= "index", margins = Tr
```
Out[60]:

loan_status verification_status	Charged Off	Fully Paid
Not Verified	14.635999	85.364001
Source Verified	21.474293	78.525707
Verified	22.321102	77.678898
All	19.612908	80.387092

In [61]:
```
pd.crosstab(index = df["verification_status"],
            columns= df["loan_status"],normalize= "index").plot(kind =
```
Out[61]: <AxesSubplot:xlabel='verification_status'>



In [62]:
```


# Later  label encoding
# .
# Verified           1
# Source Verified    2
# Not Verified       0

```

purpose :

- A category provided by the borrower for the loan request.

In [63]:

```
df["purpose"].nunique()
```

Out[63]: 14

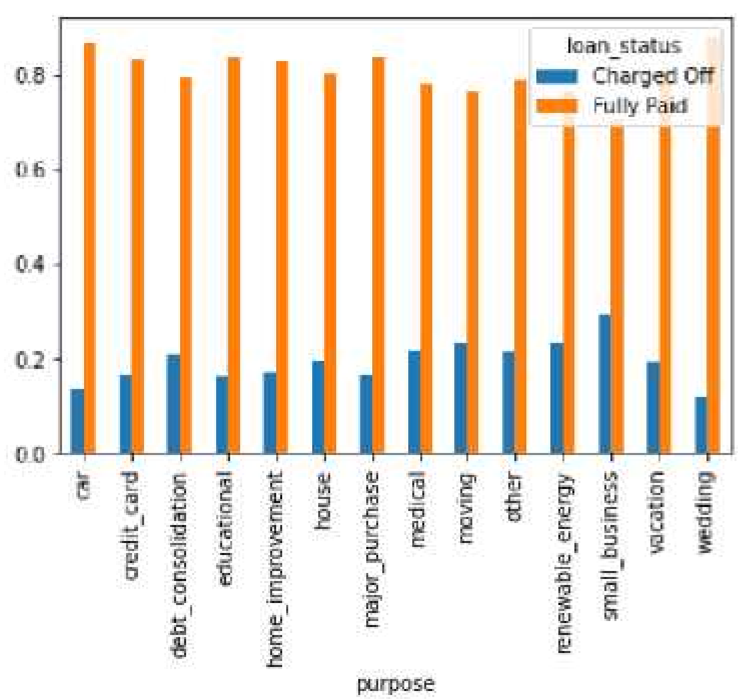
In [64]:

```
print(df["purpose"].value_counts(dropna=False))
pd.crosstab(index = df["purpose"],
            columns= df["loan_status"],normalize= "index", margins = Tr
pd.crosstab(index = df["purpose"],
            columns= df["loan_status"],normalize= "index").plot(kind = 

```

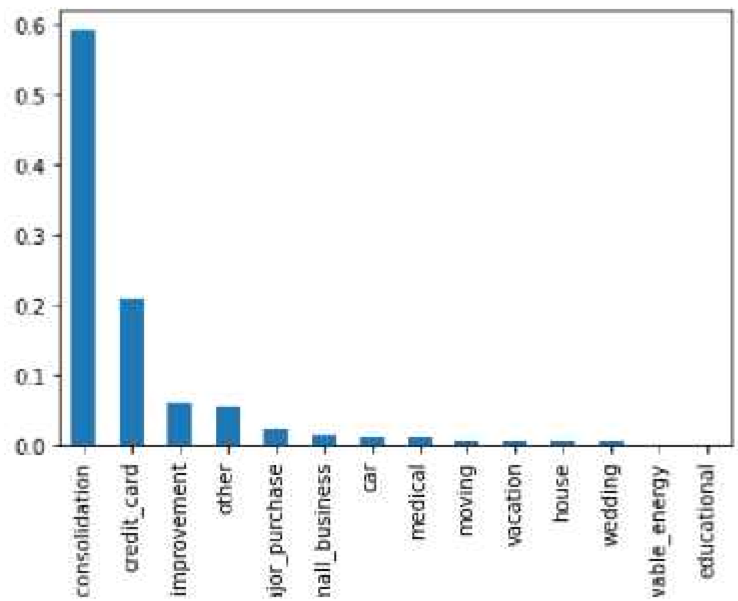
```
debt_consolidation    234507
credit_card            83019
home_improvement       24030
other                  21185
major_purchase          8790
small_business          5701
car                     4697
medical                 4196
moving                  2854
vacation                2452
house                   2201
wedding                 1812
renewable_energy         329
educational              257
Name: purpose, dtype: int64
```

Out[64]: <AxesSubplot:xlabel='purpose'>



In [65]:
```
(df["purpose"].value_counts(dropna=False,normalize=True)).plot(kind = '
```

Out[65]: <AxesSubplot:>

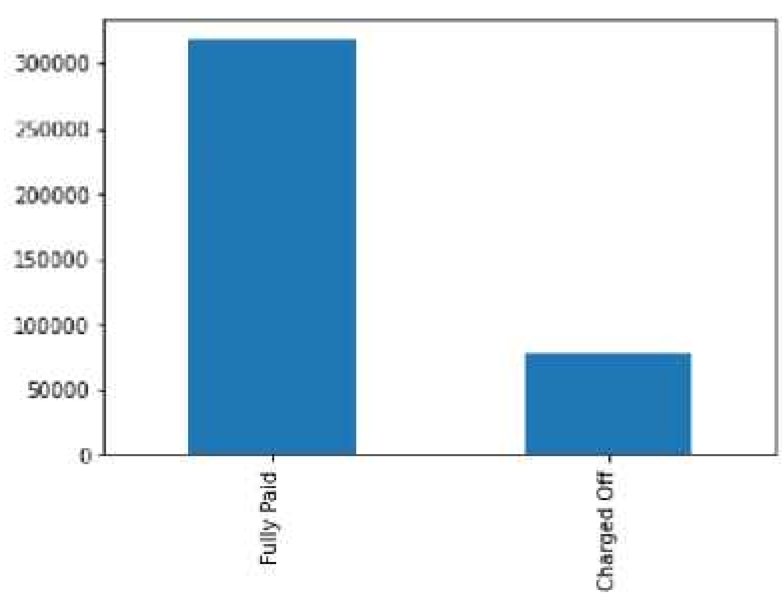


13.

loan_status : Current status of the loan - Target Variable

In [66]:
```
df["loan_status"].value_counts(dropna=False).plot(kind = "bar")
```

Out[66]: <AxesSubplot:>



In [67]:
```
df["loan_status"].value_counts(dropna=False, normalize=True)  * 100
```

Out[67]:
```
Fully Paid     80.387092
Charged Off    19.612908
Name: loan_status, dtype: float64
```

In [68]:
```
# Imbalanced data.

# 80% loans are fully paid.
# 20% loans are charged_off
```

```
## most of the loans are taken for
    debit_card,
    dept_consolidation ,
    home_improvement and others category.
## number of loan applications and amount per purpose category are hig
hest in  above category.
```

title :

- **The loan title provided by the borrower**

In [69]:
```
df["title"].nunique()
```

Out[69]:
```
48817
```

In [70]:
```
df["title"]
```

Out[70]:
```
0                          Vacation
1                Debt consolidation
2           Credit card refinancing
3           Credit card refinancing
4             Credit Card Refinance
                     ...
396025           Debt consolidation
396026           Debt consolidation
396027         pay off credit cards
396028                Loanforpayoff
396029            Toxic Debt Payoff
Name: title, Length: 396030, dtype: object
```

In [71]:
```
# title and purpose are in a way same features.
# Later needs to drop this feature.

```

```
dti = monthly total dept payment / monthly income excluding mortgages
```

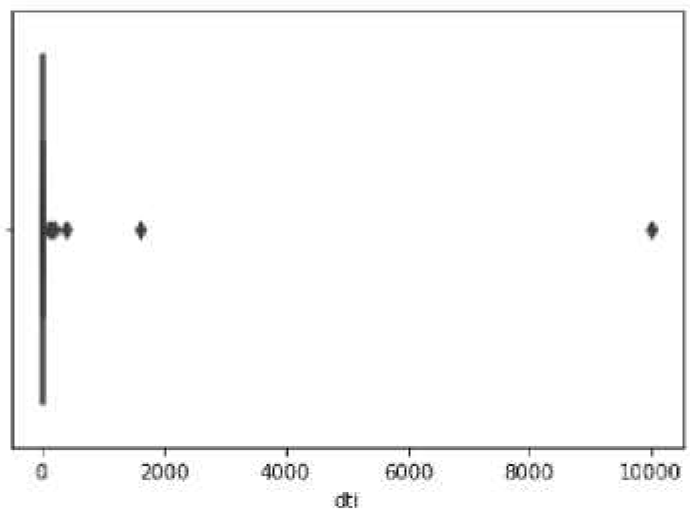
In [72]:
```
df["dti"].describe()
```

Out[72]:
```
count    396030.000000
mean         17.379514
std          18.019092
min           0.000000
25%          11.280000
50%          16.910000
75%          22.980000
max        9999.000000
Name: dti, dtype: float64
```

In [73]:
```
sns.boxenplot((df["dti"]))
```

```
C:\Users\ABC\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variable as a keyword arg: x. From version 0.1
2, the only valid positional argument will be `data`, and passing other ar
guments without an explicit keyword will result in an error or misinterpre
tation.
  warnings.warn(
```

Out[73]: <AxesSubplot:xlabel='dti'>

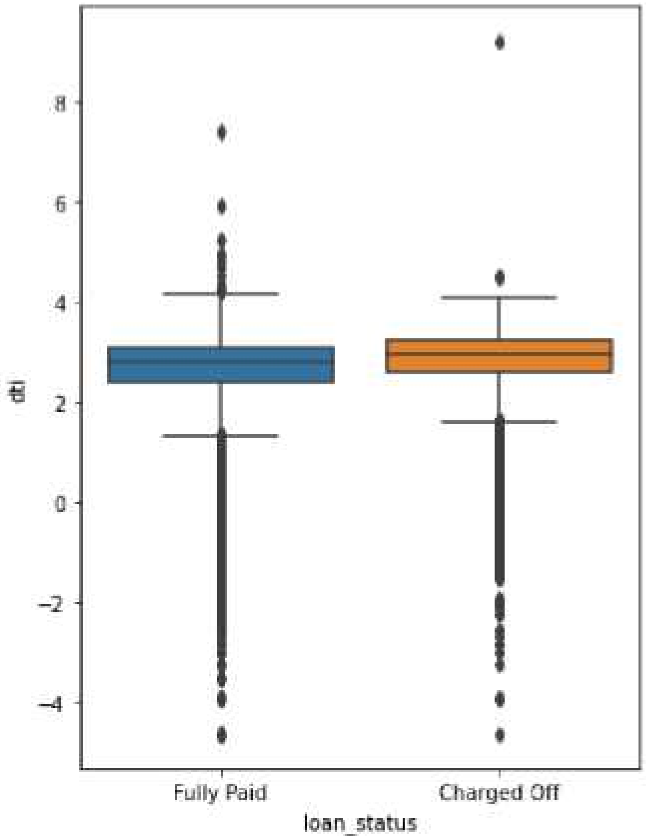


In [74]:
```
# looks like there are lots of outliers in dti column .
```

In [75]:

```
plt.figure(figsize=(5,7))
sns.boxplot(y=np.log(df[df["dti"]>0]["dti"]),
            x=df["loan_status"])
```

Out[75]: <AxesSubplot:xlabel='loan_status', ylabel='dti'>



issue_d :
The month which the loan was funded¶

issue_d :

- **The month which the loan was funded**

In [76]:

```
# df["issue_d"].value_counts(dropna=False)

# later use in feature engineering !
```

earliest_cr_line :

- **The month the borrower's earliest reported credit line was opened**

In [77]:
```
1 df["Loan_Tenure"] = ((pd.to_datetime(df["issue_d"]) -pd.to_datetime(df[
```

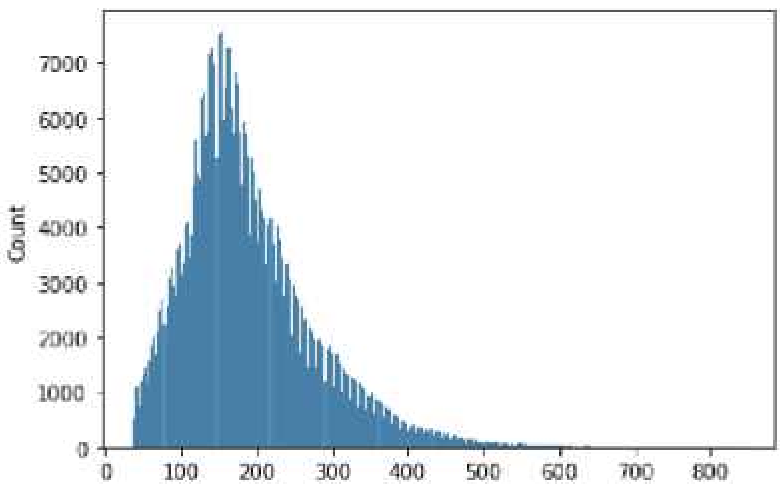
In [78]:
```
1 # pd.to_datetime(df["earliest_cr_line"])
```

In [79]:
```
1 # The month which the loan was funded
```

In [80]:
```
1 # pd.to_datetime(df["issue_d"])
```

In [81]:
```
1 sns.histplot(((pd.to_datetime(df["issue_d"]) -pd.to_datetime(df["earl
2
```

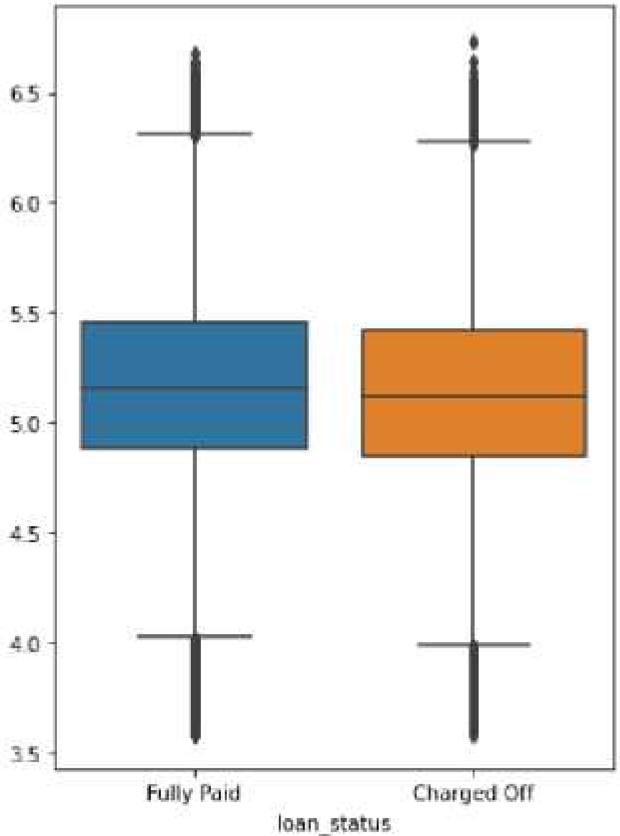
Out[81]: <AxesSubplot:ylabel='Count'>



In [82]:

```
plt.figure(figsize=(5,7))
sns.boxplot(y=np.log(((pd.to_datetime(df["issue_d"]) -pd.to_datetime(d
            x=df["loan_status"])
```

Out[82]: <AxesSubplot:xlabel='loan_status'>



open_acc :

- **The number of open credit lines in the borrower's credit file.**

In [83]:

```
df.groupby("loan_status")["open_acc"].describe()
```

Out[83]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	11.602513	5.288507	0.0	8.0	11.0	14.0	76.0
Fully Paid	318357.0	11.240067	5.097647	0.0	8.0	10.0	14.0	90.0

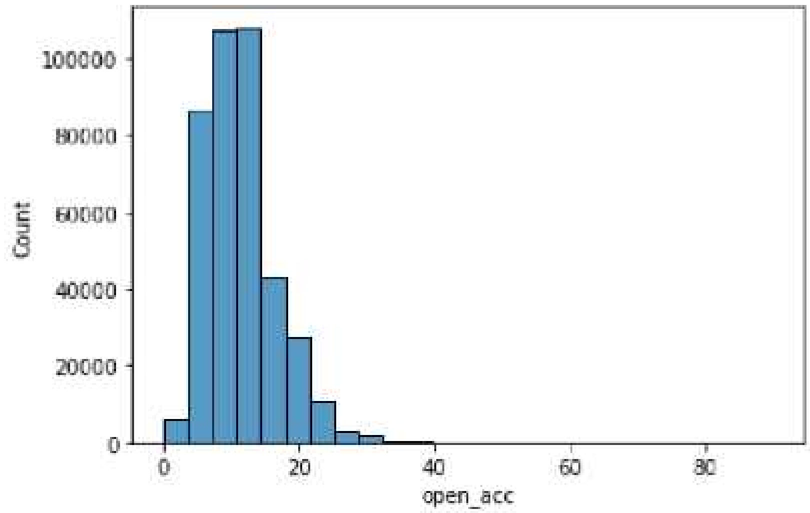
In [84]:

```
df["open_acc"].nunique()
```

Out[84]: 61

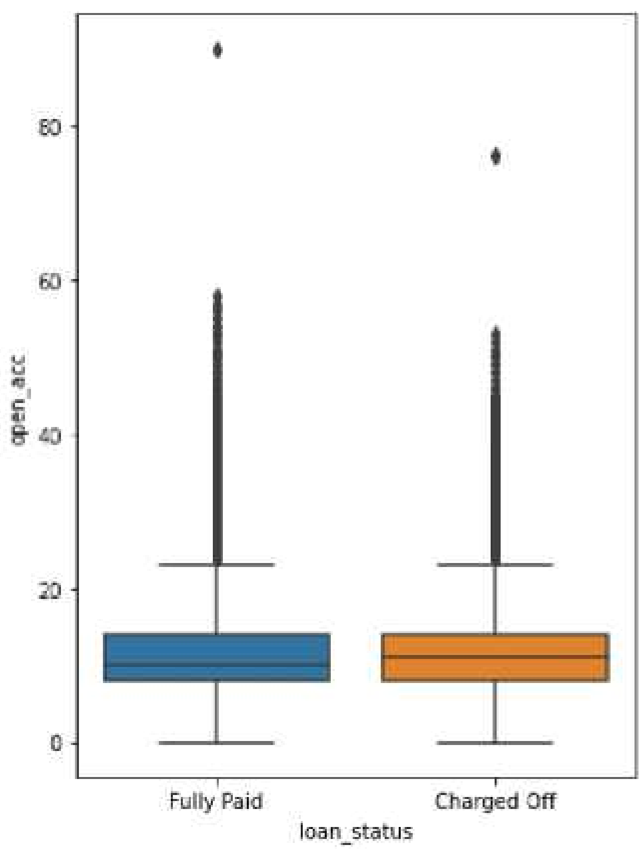
In [85]:
```
sns.histplot(df["open_acc"],bins = 25)
```

Out[85]: <AxesSubplot:xlabel='open_acc', ylabel='Count'>



In [86]:
```
plt.figure(figsize=(5,7))
sns.boxplot(y= df["open_acc"],
            x=df["loan_status"])
```

Out[86]: <AxesSubplot:xlabel='loan_status', ylabel='open_acc'>



pub_rec :

- **Number of derogatory public records**
- "Derogatory" is seen as negative to lenders, and can include late payments, collection accounts, bankruptcy, charge-offs and other negative marks on your credit report. This can impact your ability to qualify for new credit.

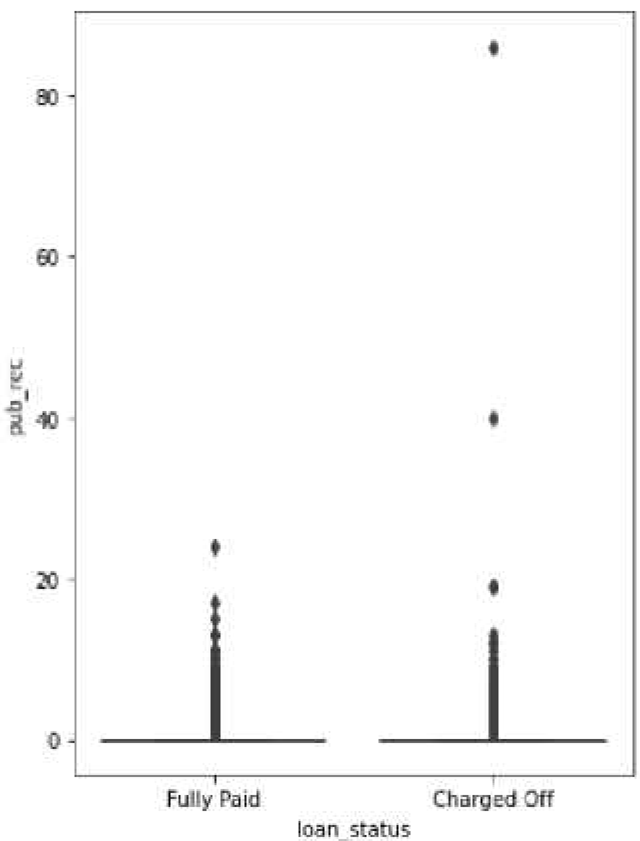
In [87]:
```
df.groupby("loan_status")["pub_rec"].describe()
```

Out[87]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	0.199606	0.648283	0.0	0.0	0.0	0.0	86.0
Fully Paid	318357.0	0.172966	0.497637	0.0	0.0	0.0	0.0	24.0

In [88]:
```
plt.figure(figsize=(5,7))
sns.boxplot(y= df["pub_rec"],
            x=df["loan_status"])
```

Out[88]: <AxesSubplot:xlabel='loan_status', ylabel='pub_rec'>

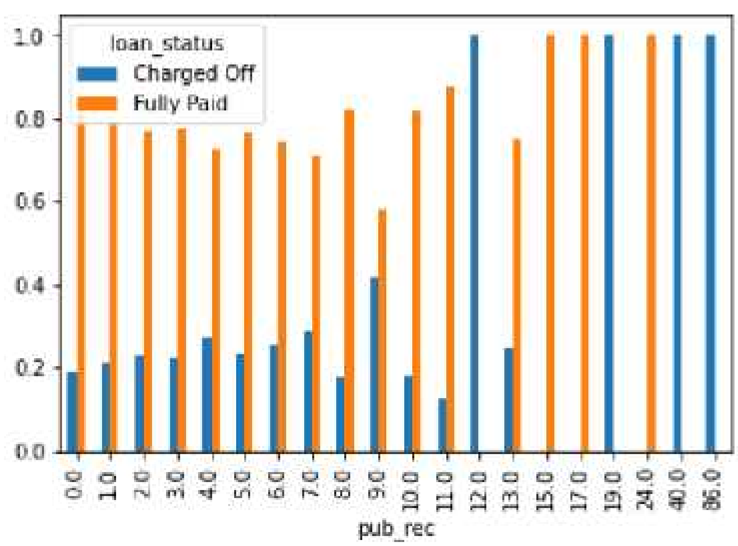


In [89]:

```
print(df["pub_rec"].value_counts(dropna=False))
pd.crosstab(index = df["pub_rec"],
            columns= df["loan_status"],normalize= "index", margins = Tı
pd.crosstab(index = df["pub_rec"],
            columns= df["loan_status"],normalize= "index").plot(kind = 
```

```
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
40.0         1
17.0         1
86.0         1
24.0         1
15.0         1
Name: pub_rec, dtype: int64
```

Out[89]: <AxesSubplot:xlabel='pub_rec'>



revol_bal :

- **Total credit revolving balance**

With revolving credit, a consumer has a line of credit he can keep using and repaying over and over. The balance that carries over from one month to the next is the revolving balance on that loan.

In [90]:

```
df.groupby("loan_status")["revol_bal"].describe()
```

Out[90]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	15390.454701	18203.387930	0.0	6150.0	11277.0	19485.0	1030826.0
Fully Paid	318357.0	15955.327918	21132.193457	0.0	5992.0	11158.0	19657.0	1743266.0

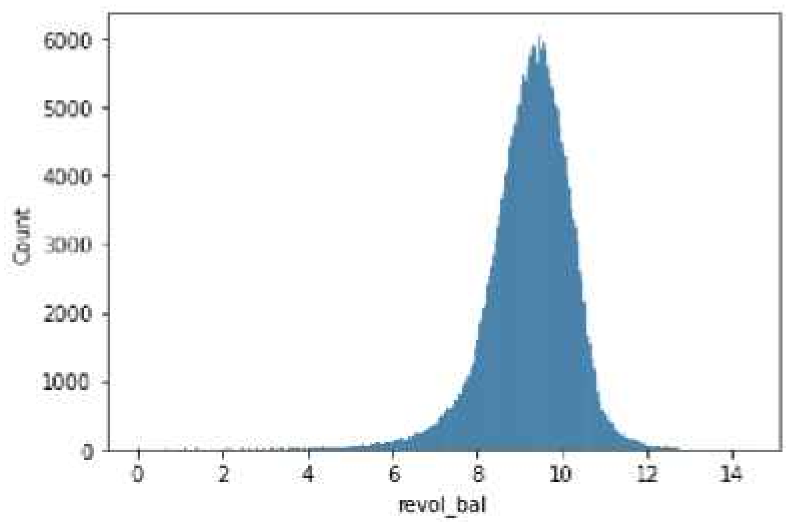
In [91]:

```
sns.histplot(np.log(df["revol_bal"]))

```

```
C:\Users\ABC\anaconda3\lib\site-packages\pandas\core\arraylike.py:397: Run
timeWarning: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Out[91]: <AxesSubplot:xlabel='revol_bal', ylabel='Count'>

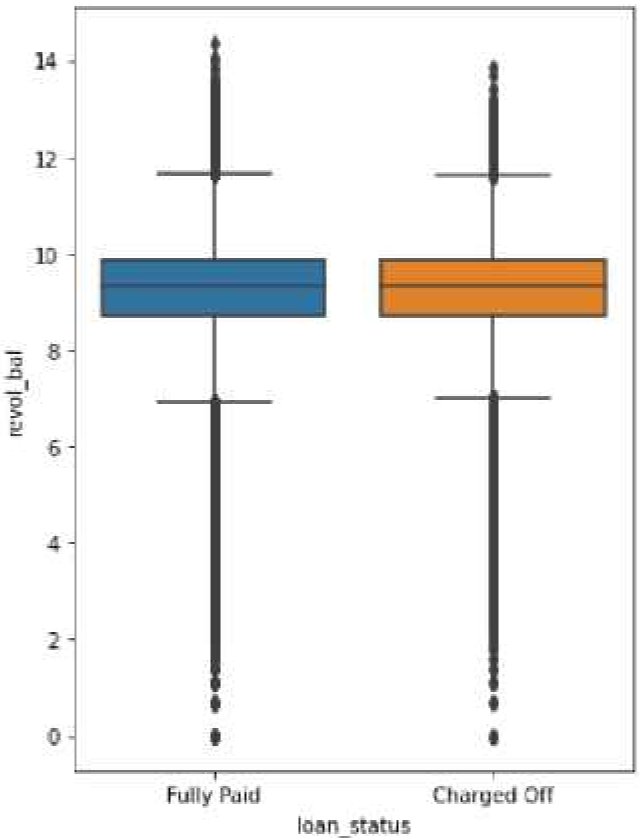


In [92]:

```
plt.figure(figsize=(5,7))
sns.boxplot(y= np.log(df["revol_bal"]),
            x=df["loan_status"])
```

```
C:\Users\ABC\anaconda3\lib\site-packages\pandas\core\arraylike.py:397: Run
timeWarning: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Out[92]: <AxesSubplot:xlabel='loan_status', ylabel='revol_bal'>



revol_util :

- **Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.**

Your credit utilization rate, sometimes called your credit utilization ratio, is the amount of revolving credit you're currently using divided by the total amount of revolving credit you have available. In other words, it's how much you currently owe divided by your credit limit. It is generally expressed as a percent.

In [93]:

```
df.groupby("loan_status")["revol_util"].describe()
```

Out[93]:

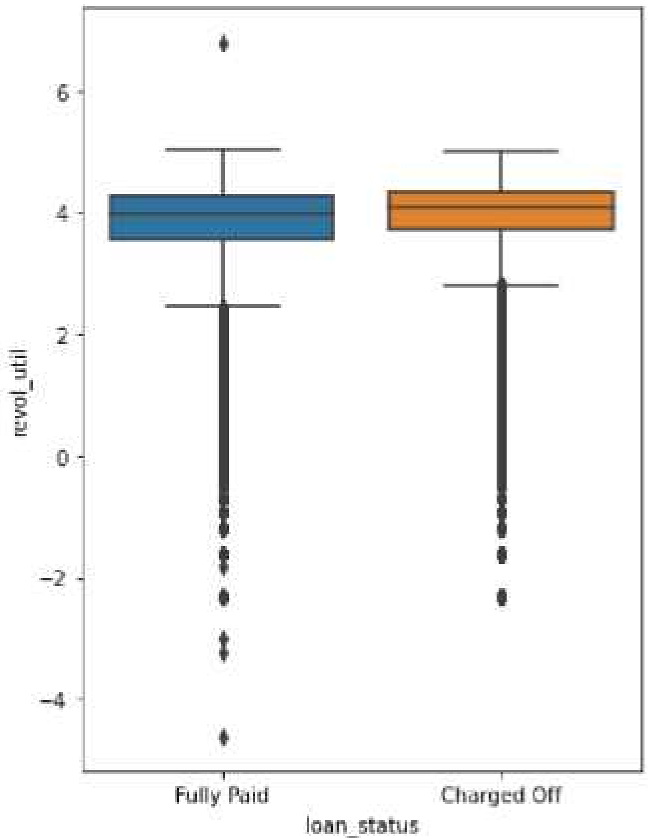
loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77610.0	57.869824	23.492176	0.0	41.2	59.3	76.2	148.0
Fully Paid	318144.0	52.796918	24.578304	0.0	34.6	53.7	72.0	892.3

In [94]:

```
plt.figure(figsize=(5,7))
sns.boxplot(y= np.log(df["revol_util"]),
            x=df["loan_status"])
```

```
C:\Users\ABC\anaconda3\lib\site-packages\pandas\core\arraylike.py:397: Run
timeWarning: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Out[94]: <AxesSubplot:xlabel='loan_status', ylabel='revol_util'>



total_acc :

- **The total number of credit lines currently in the borrower's credit file**

In [95]:

```
# df["total_acc"].value_counts()
```

In [96]:

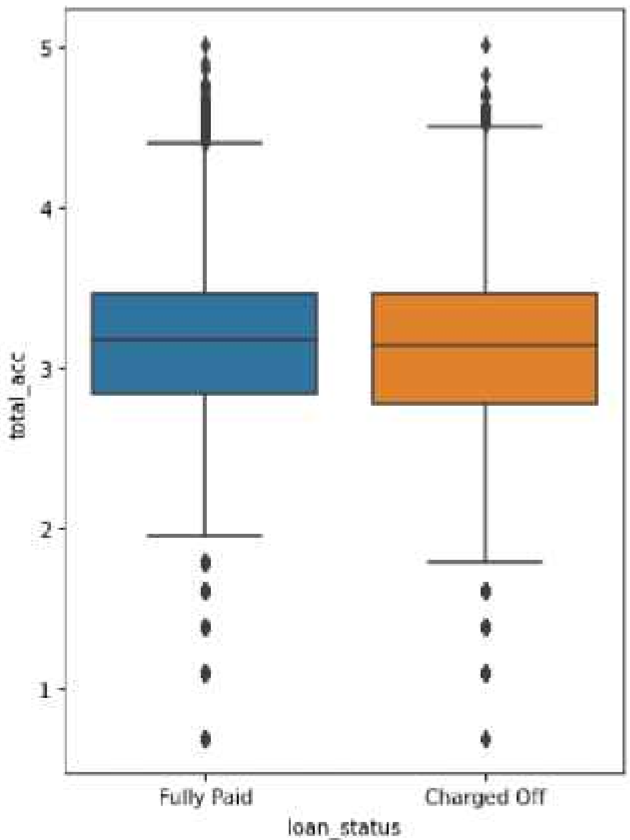
```
df.groupby("loan_status")["total_acc"].describe()
```

Out[96]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	24.984152	11.913692	2.0	16.0	23.0	32.0	151.0
Fully Paid	318357.0	25.519800	11.878117	2.0	17.0	24.0	32.0	150.0

In [97]:
```
plt.figure(figsize=(5,7))
sns.boxplot(y= np.log(df["total_acc"]),
            x=df["loan_status"])
```

Out[97]: <AxesSubplot:xlabel='loan_status', ylabel='total_acc'>



initial_list_status :

- **The initial listing status of the loan. Possible values are – W, F**

In [98]:
```
df["initial_list_status"].value_counts()
```

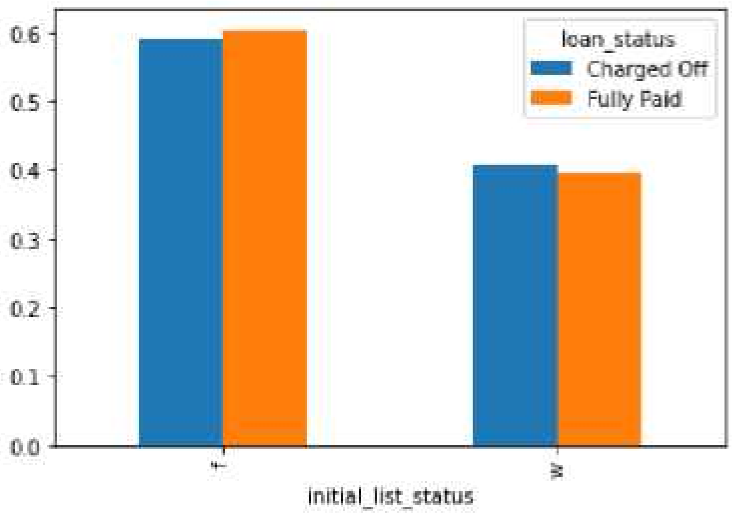
Out[98]:
```
f    238066
w    157964
Name: initial_list_status, dtype: int64
```

In [99]:
```
print(df["initial_list_status"].value_counts(dropna=False))

pd.crosstab(index = df["initial_list_status"],
            columns= df["loan_status"],normalize= "columns").plot(kind
```

```
f    238066
w    157964
Name: initial_list_status, dtype: int64
```

Out[99]: <AxesSubplot:xlabel='initial_list_status'>



application_type :

- **Indicates whether the loan is an individual application or a joint application with two co-borrowers**

In [100]:
```
df["application_type"].value_counts()
```

Out[100]:
```
INDIVIDUAL    395319
JOINT            425
DIRECT_PAY       286
Name: application_type, dtype: int64
```

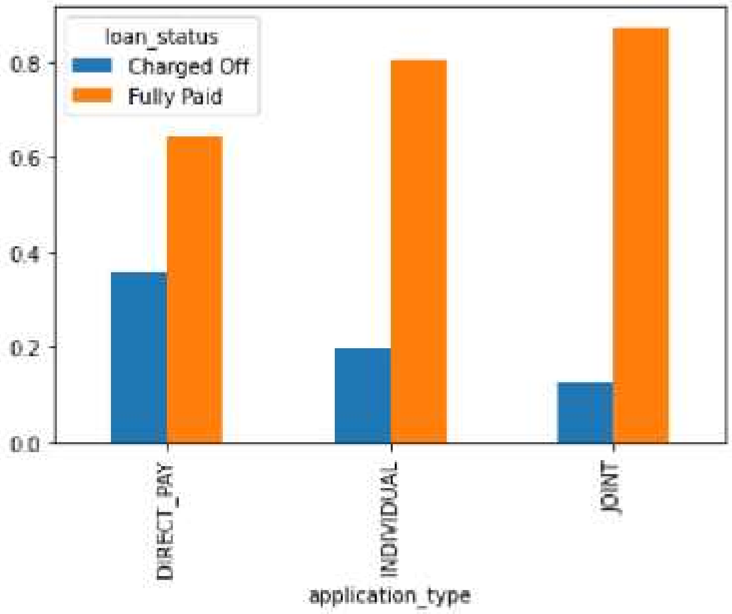
In [101]:

```
print(df["application_type"].value_counts(dropna=False))

pd.crosstab(index = df["application_type"],
            columns= df["loan_status"],normalize= "index").plot(kind =
```

```
INDIVIDUAL    395319
JOINT            425
DIRECT_PAY       286
Name: application_type, dtype: int64
```

Out[101]: <AxesSubplot:xlabel='application_type'>



mort_acc :

- **Number of mortgage accounts.**

In [102]:

```
# df["mort_acc"].value_counts(dropna=False)
```

In [103]:

```
df.groupby("loan_status")["mort_acc"].describe()
```

Out[103]:

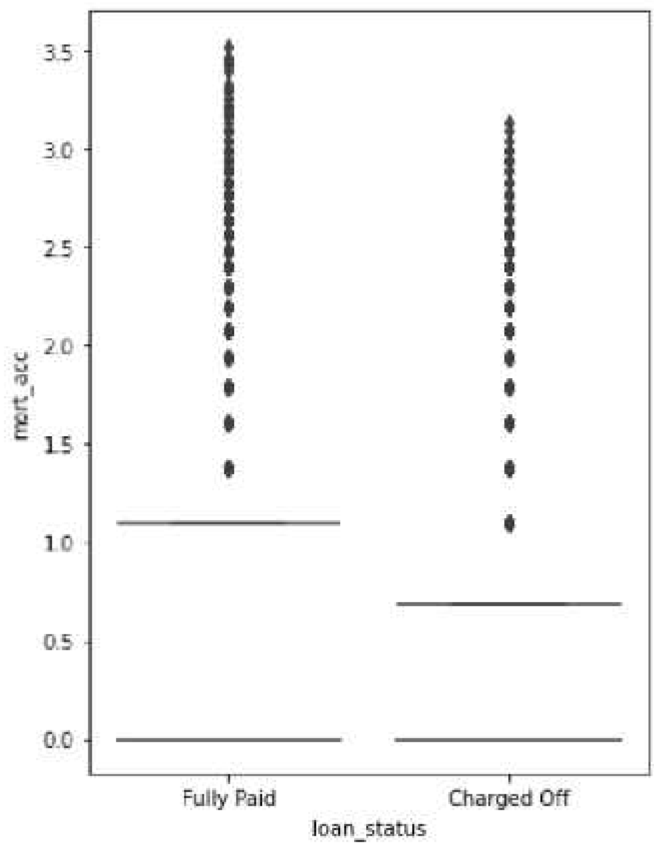
loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	72123.0	1.501213	1.974353	0.0	0.0	1.0	2.0	23.0
Fully Paid	286112.0	1.892836	2.182456	0.0	0.0	1.0	3.0	34.0

In [104]:

```
plt.figure(figsize=(5,7))
sns.boxplot(y= np.log(df["mort_acc"]),
            x=df["loan_status"])
```

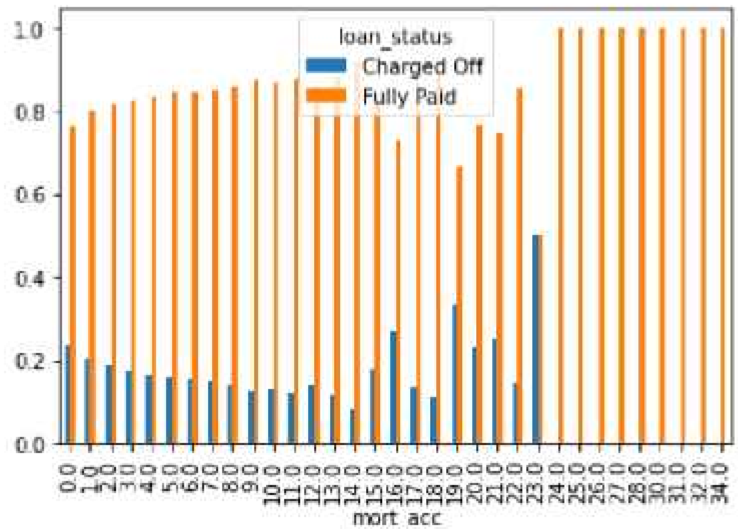
```
C:\Users\ABC\anaconda3\lib\site-packages\pandas\core\arraylike.py:397: Run
timeWarning: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
C:\Users\ABC\anaconda3\lib\site-packages\numpy\lib\function_base.py:4486:
RuntimeWarning: invalid value encountered in subtract
  diff_b_a = subtract(b, a)
C:\Users\ABC\anaconda3\lib\site-packages\numpy\lib\function_base.py:4486:
RuntimeWarning: invalid value encountered in subtract
  diff_b_a = subtract(b, a)
```

Out[104]: <AxesSubplot:xlabel='loan_status', ylabel='mort_acc'>



In [105]:
```
pd.crosstab(index = df["mort_acc"],
                columns= df["loan_status"],normalize= "index").plot(kind =
```

Out[105]: <AxesSubplot:xlabel='mort_acc'>



pub_rec_bankruptcies :

- **Number of public record bankruptcies**

In [106]:
```
df["pub_rec_bankruptcies"].value_counts()
```

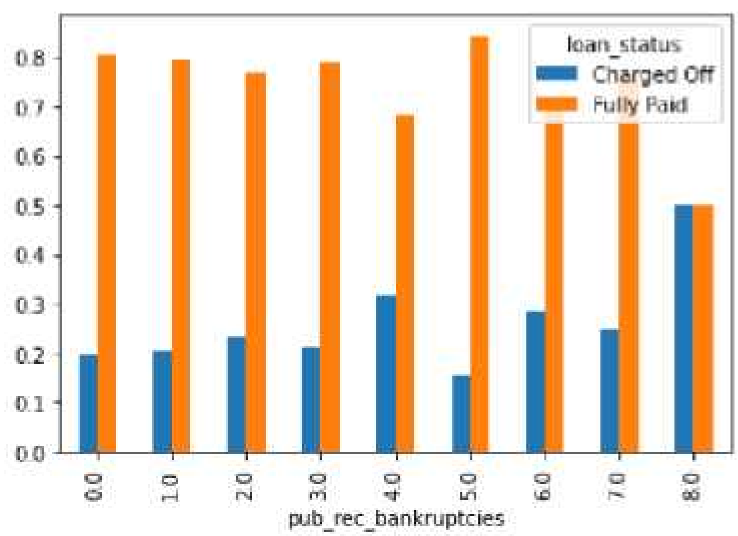
Out[106]:
```
0.0    350380
1.0     42790
2.0      1847
3.0       351
4.0        82
5.0        32
6.0         7
7.0         4
8.0         2
Name: pub_rec_bankruptcies, dtype: int64
```

In [107]:

```
print(df["pub_rec_bankruptcies"].value_counts(dropna=False))
print(pd.crosstab(index = df["pub_rec_bankruptcies"],
            columns= df["loan_status"],normalize= "index", margins = Tr
pd.crosstab(index = df["pub_rec_bankruptcies"],
            columns= df["loan_status"],normalize= "index").plot(kind = 
```

```
0.0    350380
1.0     42790
2.0      1847
NaN       535
3.0       351
4.0        82
5.0        32
6.0         7
7.0         4
8.0         2
Name: pub_rec_bankruptcies, dtype: int64
loan_status           Charged Off  Fully Paid
pub_rec_bankruptcies
0.0                     19.499115   80.500885
1.0                     20.394952   79.605048
2.0                     23.226854   76.773146
3.0                     21.082621   78.917379
4.0                     31.707317   68.292683
5.0                     15.625000   84.375000
6.0                     28.571429   71.428571
7.0                     25.000000   75.000000
8.0                     50.000000   50.000000
All                     19.617441   80.382559
```

Out[107]: <AxesSubplot:xlabel='pub_rec_bankruptcies'>



Address:

- **Address of the individual**

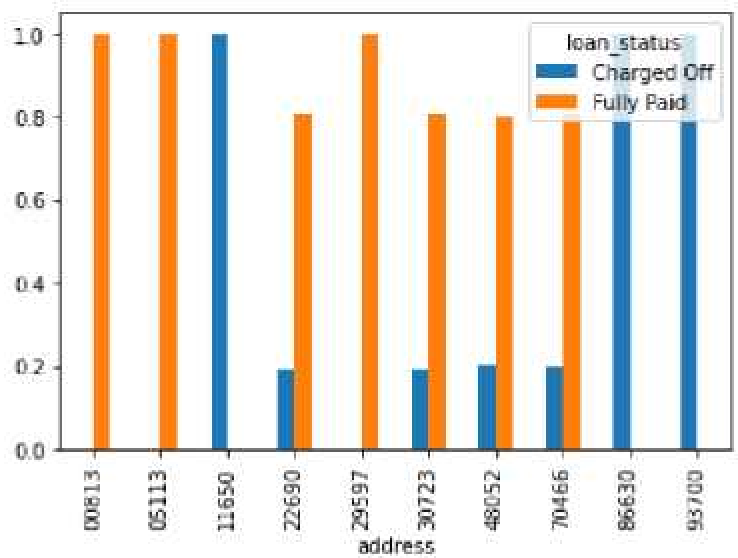
In [108]:
```
df["address"][10]
```
Out[108]: '40245 Cody Drives\r\nBartlettfort, NM 00813'

In [109]:
```
df["address"] = df["address"].str.split().apply(lambda x:x[-1])
```

In [110]:
```
df["address"].value_counts()
```
Out[110]:
```
70466    56985
30723    56546
22690    56527
48052    55917
00813    45824
29597    45471
05113    45402
11650    11226
93700    11151
86630    10981
Name: address, dtype: int64
```

In [111]:
```
pd.crosstab(index = df["address"],
            columns= df["loan_status"],normalize= "index").plot(kind =

```
Out[111]: <AxesSubplot:xlabel='address'>



In [112]:
```
df["pin_code"] = df["address"]
df.drop(["address"],axis = 1  ,inplace=True)
```

In [113]:
```
df.drop(["title","issue_d","earliest_cr_line","initial_list_status"],ax
```

In [114]:
```
df.drop(["pin_code"],axis=1,inplace=True)
```

In [115]:
```
df.drop(["Loan_Tenure"],axis=1,inplace=True)
```

Missing value treatment

In [116]:
```
missing_data[missing_data["Percent"]>0]
```

Out[116]:

	Total	Percent
mort_acc	37795	9.543469
emp_title	22927	5.789208
emp_length	18301	4.621115
title	1755	0.443148
pub_rec_bankruptcies	535	0.135091
revol_util	276	0.069692

In [117]:
```
from sklearn.impute import SimpleImputer
Imputer = SimpleImputer(strategy="most_frequent")
df["mort_acc"] = Imputer.fit_transform(df["mort_acc"].values.reshape(-1
```

In [118]:
```
df.dropna(inplace=True)
```

In [119]:
```
missing_df(df)
```

Out[119]:

	Total	Percent
loan_amnt	0	0.0
term	0	0.0
mort_acc	0	0.0
application_type	0	0.0
total_acc	0	0.0
revol_util	0	0.0
revol_bal	0	0.0
pub_rec	0	0.0
open_acc	0	0.0
dti	0	0.0
purpose	0	0.0
loan_status	0	0.0
verification_status	0	0.0
annual_inc	0	0.0
home_ownership	0	0.0
emp_length	0	0.0
emp_title	0	0.0
sub_grade	0	0.0
grade	0	0.0
installment	0	0.0
int_rate	0	0.0
pub_rec_bankruptcies	0	0.0

Pre-proccessing :

Feature Engineering

In [120]:
```
from category_encoders import TargetEncoder

```

In [121]:
```
TE = TargetEncoder()
```

In [122]:
```
df["loan_status"].replace({"Fully Paid":0,
                            "Charged Off" : 1},inplace=True)
```

In [123]:
```python
df.sample(3)
```

Out[123]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	h
259644	4000.0	60	13.43	91.90	C	C3	jp morgan chase	2 years	
359463	35000.0	60	23.99	1006.68	E	E2	Cardiologist	5 years	
35603	15000.0	36	13.67	510.27	B	B5	FOREMAN	8 years	

3 rows × 22 columns

In [124]:
```python
df.columns
```

Out[124]:
```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grad
e',
       'emp_title', 'emp_length', 'home_ownership', 'annual_inc',
       'verification_status', 'loan_status', 'purpose', 'dti', 'open_acc',
       'pub_rec', 'revol_bal', 'revol_util', 'total_acc', 'application_typ
e',
       'mort_acc', 'pub_rec_bankruptcies'],
      dtype='object')
```

In [125]:
```python
target_enc = ["sub_grade","grade",'term', 'emp_title', 'emp_length', 'h
```

In [126]:
```python
for col in target_enc:
    from category_encoders import TargetEncoder
    TEncoder = TargetEncoder()

    df[col] = TEncoder.fit_transform(df[col],df["loan_status"])
```

```
Warning: No categorical columns found. Calling 'transform' will only retur
n input data.
```

In [127]:

```
df
```

Out[127]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length
0	10000.0	36	11.44	329.48	0.121856	0.134935	0.247136	0.184208
1	8000.0	36	11.99	265.68	0.121856	0.150496	0.214018	0.191896
2	15600.0	36	10.49	506.97	0.121856	0.119644	0.189214	0.206840
3	7200.0	36	6.49	220.65	0.059785	0.044741	0.167211	0.189319
4	24375.0	60	17.27	609.33	0.207325	0.239437	0.297320	0.200951
...	...	...	...	...	...	...	...	...
396025	10000.0	60	10.99	217.38	0.121856	0.134935	0.167211	0.193219
396026	21000.0	36	12.29	700.42	0.207325	0.168489	0.220430	0.191915
396027	5000.0	36	9.99	161.32	0.121856	0.094672	0.267968	0.184208
396028	21000.0	60	15.31	503.02	0.207325	0.192642	0.167211	0.184208
396029	2000.0	36	13.61	67.98	0.207325	0.192642	0.217205	0.184208

372161 rows × 22 columns

Outlier treatment :

In [128]:

```
def outlier_remover(a,df):

    q1 = a.quantile(.25)
    q3 = a.quantile(.75)
    iqr = q3 - q1

    maxx = q3 + 1.5 * iqr
    minn = q1 - 1.5 * iqr

    return df.loc[(a>=minn) & (a<=maxx)]
```

In [129]:

```
floats = ['loan_amnt', 'int_rate', 'annual_inc', 'dti', 'open_acc','rev

```

In [130]:

```
df.sample(3)
```

Out[130]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length
274744	12000.0	36	11.99	398.52	0.121856	0.119644	0.282723	0.184208
288790	4000.0	36	8.18	125.68	0.121856	0.094672	0.381888	0.194577
258111	9000.0	36	7.69	280.75	0.059785	0.067578	0.235879	0.198859

3 rows × 22 columns

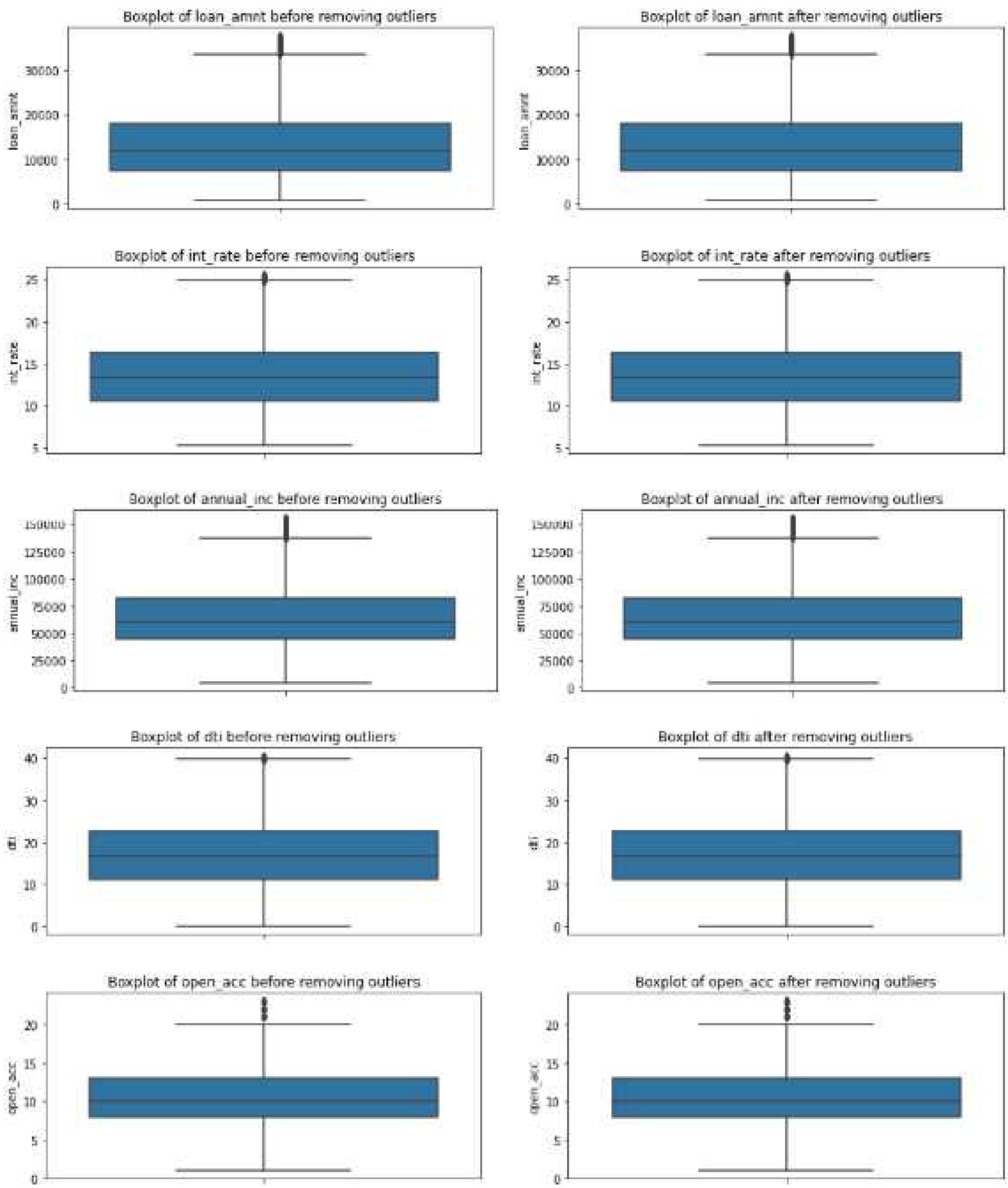
In [131]:

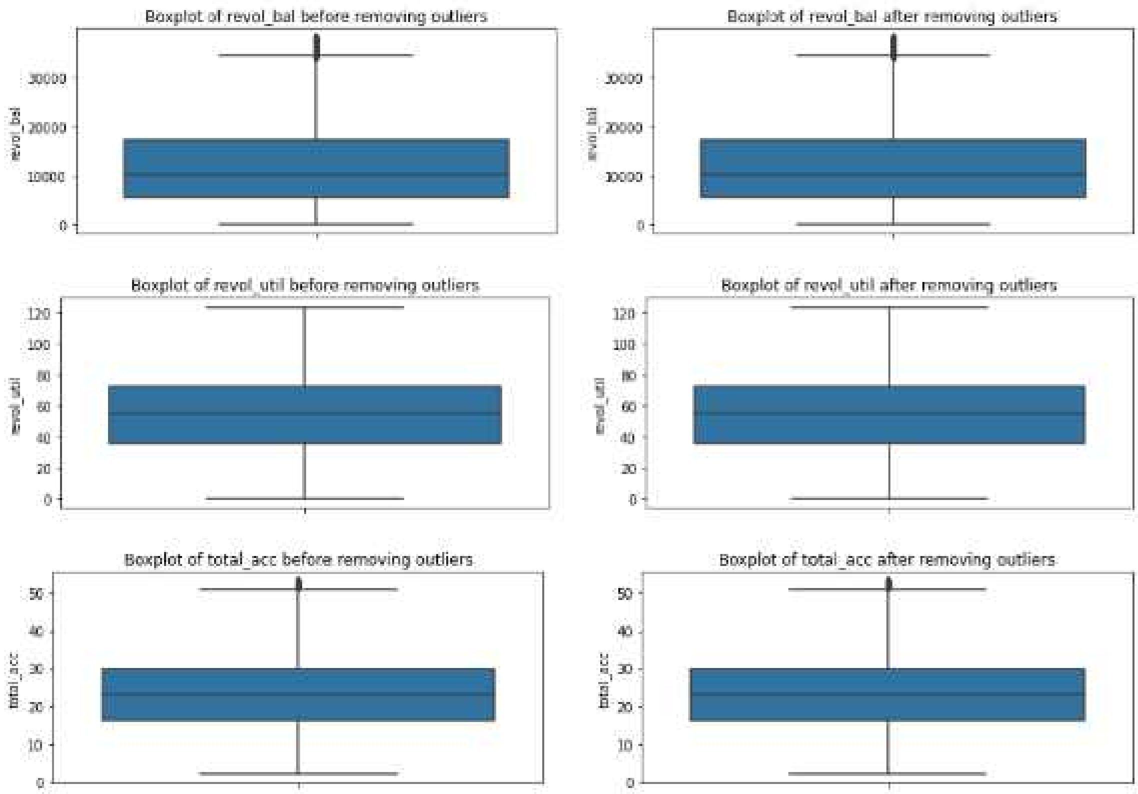
```
for i in floats:
    df = outlier_remover(df[i],df)
```

In [132]:

```
for i in floats:
    plt.figure(figsize=(15, 3))
    plt.subplot(121)
    sns.boxplot(y=df[i])
    plt.title(f"Boxplot of {i} before removing outliers")
    plt.subplot(122)
    sns.boxplot(y=df[i])
    plt.title(f"Boxplot of {i} after removing outliers")

    plt.show()
```





Missing value check :

In [133]:
```
def missing_df(data):
    total_missing_df = data.isna().sum().sort_values(ascending = False)
    percentage_missing_df = ((data.isna().sum()/len(data)*100)).sort_va
    missingDF = pd.concat([total_missing_df, percentage_missing_df],axi
    return missingDF


missing_data = missing_df(df)
missing_data[missing_data["Total"]>0]

```

Out[133]:

Total	Percent

In [134]:
```
df.columns
```

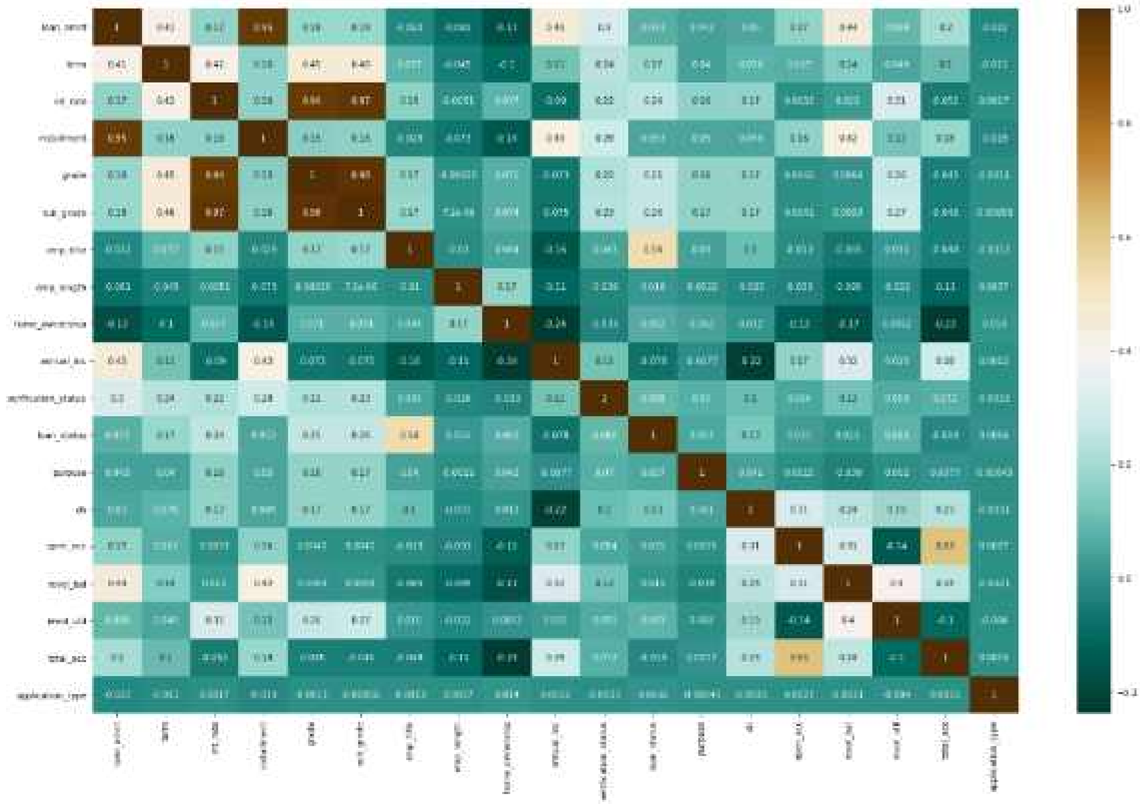
Out[134]:
```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grad
e',
       'emp_title', 'emp_length', 'home_ownership', 'annual_inc',
       'verification_status', 'loan_status', 'purpose', 'dti', 'open_acc',
       'pub_rec', 'revol_bal', 'revol_util', 'total_acc', 'application_typ
e',
       'mort_acc', 'pub_rec_bankruptcies'],
      dtype='object')
```

In [135]:
```python
df.drop(["mort_acc","pub_rec_bankruptcies"],axis = 1 , inplace=True)
```

In [136]:
```python
df.drop(["pub_rec"],axis = 1 , inplace=True)
```

In [137]:
```python
plt.figure(figsize=(24,15))
sns.heatmap(df.corr(),annot=True,cmap='BrBG_r')

plt.show()
```



Train-test split :

In [138]:
```python
X = df.drop(["loan_status"],axis = 1)
y = df["loan_status"]
```

In [139]:
```python
from sklearn.model_selection import train_test_split
```

In [140]:
```python
X_train , X_test , y_train , y_test = train_test_split(X,y,
                                                       random_state=3,
                                                       test_size=0.2)
```

Logistic Regression on Non-Standardised Data :

In [141]:
```python
from sklearn.linear_model import LogisticRegression
LR1st = LogisticRegression(class_weight="Auto")
```

In [142]:

```
LR1st = LogisticRegression(class_weight='balanced')

```

In [143]:

```
LR1st.fit(X_train,y_train)
```

```
C:\Users\ABC\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:469: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

Out[143]:

```
LogisticRegression(class_weight='balanced')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [144]:

```
LR1st.score(X_test,y_test)
```

Out[144]:

```
0.5595138176505331
```

In [145]:

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

In [146]:

```
f1_score(y_test,LR1st.predict(X_test))
```

Out[146]:

```
0.3780177332982179
```

In [147]:

```
recall_score(y_test,LR1st.predict(X_test))
```

Out[147]:

```
0.6964820056611403
```

In [148]:

```
precision_score(y_test,LR1st.predict(X_test))
```

Out[148]:

```
0.2594054037772222
```

Standardizing - preprocessing

In [149]:

```
from sklearn.preprocessing import StandardScaler
StandardScaler = StandardScaler()


```

In [150]:
```
StandardScaler.fit(X_train)
```

Out[150]:
```
StandardScaler()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [151]:
```
X_train = StandardScaler.transform(X_train)
X_test = StandardScaler.transform(X_test)

```

In [152]:
```
from sklearn.linear_model import LogisticRegression
LR_Std = LogisticRegression(C=1.0)
LR_Std.fit(X_train,y_train)
print("Accuracy: ",LR_Std.score(X_test,y_test))
print("f1_score: ",f1_score(y_test,LR_Std.predict(X_test)))
print("recall_score: ",recall_score(y_test,LR_Std.predict(X_test)))
print("precision_score: ",precision_score(y_test,LR_Std.predict(X_test
```

```
Accuracy:  0.8678541452951599
f1_score:  0.6058048961424333
recall_score:  0.5283461382935706
precision_score:  0.7098772139519722
```

In [153]:

```
pd.DataFrame(data=LR_Std.coef_,columns=X.columns).T
```

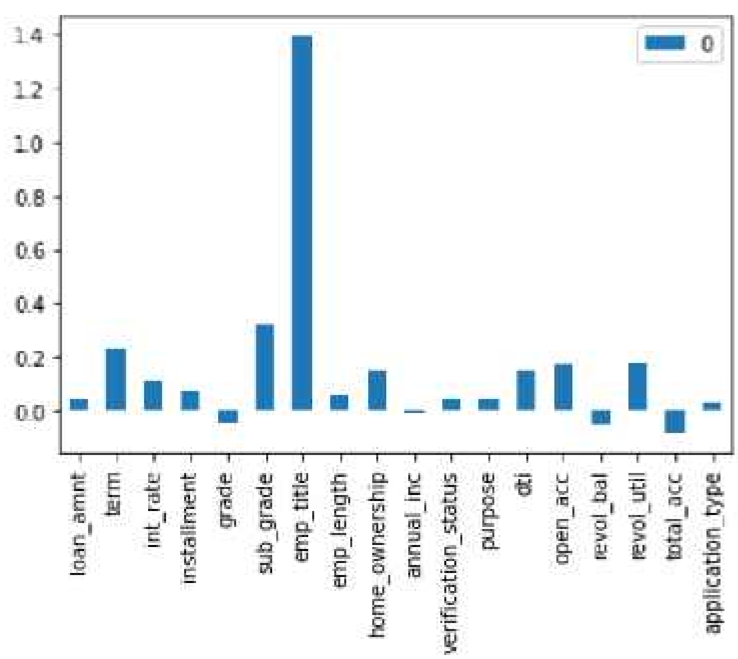
Out[153]:

	0
loan_amnt	0.047677
term	0.231816
int_rate	0.111797
installment	0.077540
grade	-0.042886
sub_grade	0.322222
emp_title	1.391361
emp_length	0.060433
home_ownership	0.149167
annual_inc	-0.009222
verification_status	0.044294
purpose	0.044511
dti	0.146886
open_acc	0.172216
revol_bal	-0.050793
revol_util	0.178830
total_acc	-0.080503
application_type	0.027792

In [154]:

```
pd.DataFrame(data=LR_Std.coef_,columns=X.columns).T.plot(kind = "bar")
```

Out[154]: <AxesSubplot:>



Data Balancing :

In [155]:
```
from imblearn.over_sampling import SMOTE
```

In [156]:
```
SmoteBL = SMOTE(k_neighbors=7)
```

In [157]:
```
X_smote , y_smote = SmoteBL.fit_resample(X_train,y_train)
```

In [158]:
```
X_smote.shape,  y_smote.shape
```

Out[158]: ((416188, 18), (416188,))

In [159]:
```
# y_smote.value_counts()

```

In [160]:
```
from sklearn.linear_model import LogisticRegression
```

In [161]:
```
LogReg = LogisticRegression(max_iter=1000,class_weight="balanced")
```

In [162]:
```
from sklearn.model_selection import cross_val_score
```

In [163]:
```
cross_val_score(estimator = LogReg,
                cv=5,
                X = X_smote,
                y = y_smote,
                scoring= "f1"

        )
```

Out[163]: array([0.81043817, 0.81814618, 0.81727811, 0.81854   , 0.81869803])

In [164]:
```
cross_val_score(estimator = LogReg,
                cv=5,
                X = X_smote,
                y = y_smote,
                scoring= "precision"

        )
```

Out[164]: array([0.83252255, 0.83462894, 0.83420078, 0.83394834, 0.83238975])

In [165]:
```
cross_val_score(estimator = LogReg,
                cv=5,
                X = X_smote,
                y = y_smote,
                scoring= "accuracy"

        )
```

Out[165]: array([0.81533675, 0.82166799, 0.82091112, 0.82183404, 0.8216298])

In [166]:

```
cross_val_score(estimator = LogReg,
                cv=5,
                X = X_train,
                y = y_train,
                scoring= "precision"

        )
```

Out[166]:

```
array([0.53076607, 0.53778475, 0.53549319, 0.5352065 , 0.52933151])
```

In [167]:

```
from sklearn.linear_model import LogisticRegression
LogReg = LogisticRegression(max_iter=1000,class_weight="balanced")
```

In [168]:

```
LogReg.fit(X= X_train ,y = y_train)
```

Out[168]:

```
LogisticRegression(class_weight='balanced', max_iter=1000)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [169]:

```
LogReg.score(X_test,y_test)
```

Out[169]:

```
0.8261835928999969
```

In [170]:

```
LogReg.coef_.round(2)
```

Out[170]:

```
array([[ 0.09,  0.21, -0.07,  0.05, -0.05,  0.53,  1.47,  0.05,  0.14,
         0.01,  0.06,  0.05,  0.17,  0.16, -0.06,  0.16, -0.07,  0.04]])
```

In [171]:

```
from sklearn.metrics  import confusion_matrix, f1_score, precision_scor
print(confusion_matrix(y_test, LogReg.predict(X_test)))
print(precision_score(y_test ,LogReg.predict(X_test)))
print(recall_score(y_test ,LogReg.predict(X_test)))
print(f1_score(y_test ,LogReg.predict(X_test)))


```

```
[[43356  8617]
 [ 2566  9799]]
0.5320916594265855
0.7924787707238172
0.6366914655144408
```

In [172]:

```
LogReg.coef_
```

Out[172]:

```
array([[ 0.08572422,  0.21400667, -0.07272492,  0.04665781, -0.04789813,
         0.53227735,  1.4680174 ,  0.05191716,  0.137461  ,  0.01155915,
         0.05613236,  0.04579973,  0.16682527,  0.160338  , -0.05560624,
         0.15543841, -0.07028592,  0.03507352]])
```

In [173]:
```
df.drop(["loan_status"], axis = 1).columns
```

Out[173]:
```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grad
e',
       'emp_title', 'emp_length', 'home_ownership', 'annual_inc',
       'verification_status', 'purpose', 'dti', 'open_acc', 'revol_bal',
       'revol_util', 'total_acc', 'application_type'],
      dtype='object')
```

In [174]:
```
feature_importance = pd.DataFrame(index = df.drop(["loan_status"],
                                                    axis = 1).columns,
                                      data = LogReg.coef_.ravel()).reset_ir
feature_importance
```

Out[174]:

	index	0
0	loan_amnt	0.085724
1	term	0.214007
2	int_rate	-0.072725
3	installment	0.046658
4	grade	-0.047898
5	sub_grade	0.532277
6	emp_title	1.468017
7	emp_length	0.051917
8	home_ownership	0.137461
9	annual_inc	0.011559
10	verification_status	0.056132
11	purpose	0.045800
12	dti	0.166825
13	open_acc	0.160338
14	revol_bal	-0.055606
15	revol_util	0.155438
16	total_acc	-0.070286
17	application_type	0.035074

In [176]:
```
LogReg.score(X_train,y_train)
```

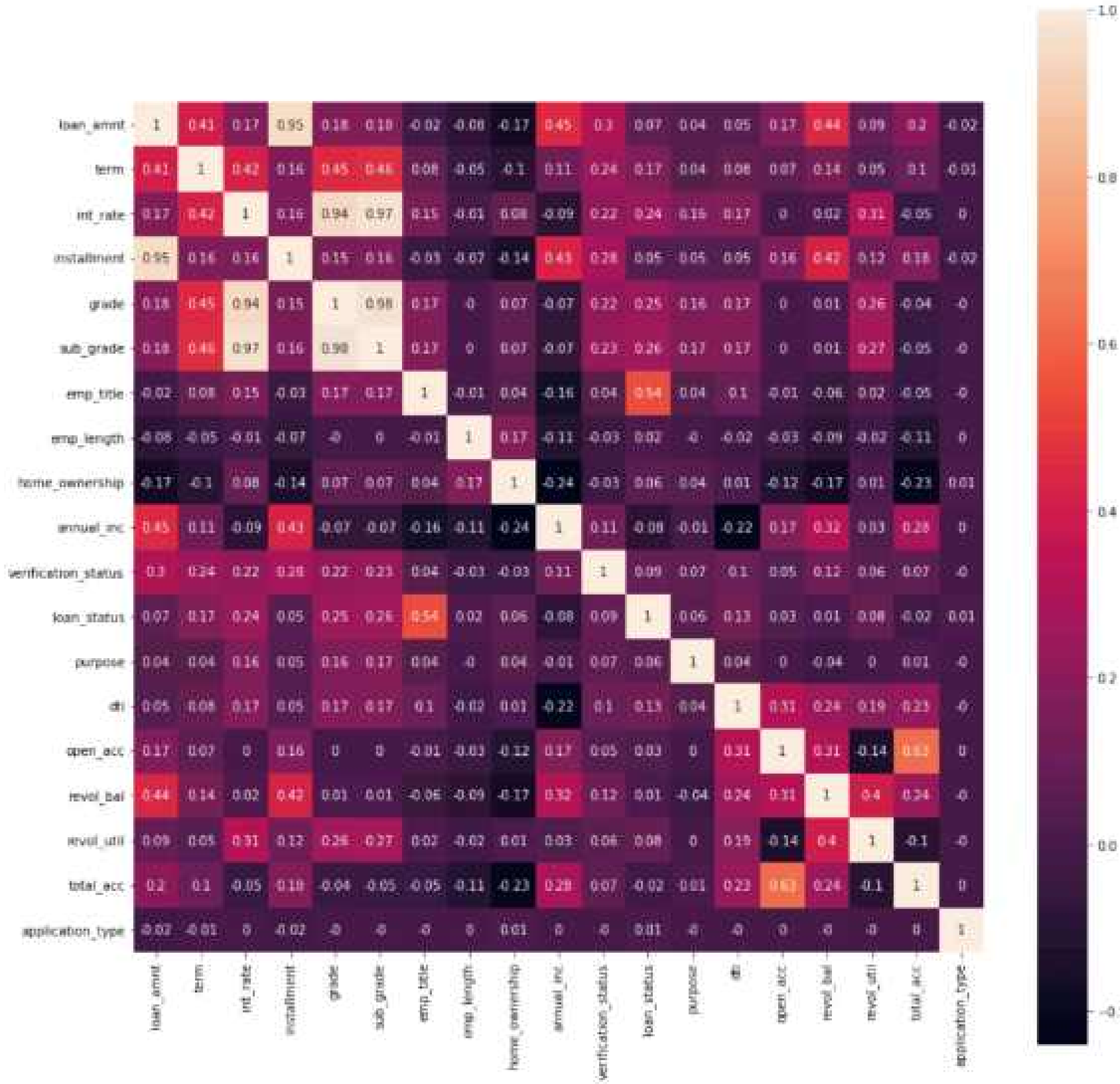
Out[176]: 0.8278298037691859

In [177]:
```
LogReg.score(X_test,y_test)
```

Out[177]: 0.8261835928999969

In [178]:
```python
plt.figure(figsize=(15,15))

sns.heatmap(df.corr().round(2),annot=True,square=True)
```
Out[178]: <AxesSubplot:>



Metrics :

In [179]:
```python
from sklearn.metrics  import confusion_matrix, f1_score, precision_scor
confusion_matrix(y_test, LogReg.predict(X_test))


```
Out[179]:
```
array([[43356,  8617],
       [ 2566,  9799]], dtype=int64)
```

In [180]:
```python
precision_score(y_test ,LogReg.predict(X_test))
```
Out[180]: 0.5320916594265855

In [181]:
```
recall_score(y_test ,LogReg.predict(X_test))
```
Out[181]: 0.7924787707238172

In [182]:
```
pd.crosstab(y_test ,LogReg.predict(X_test))
```
Out[182]:

col_0	0	1
loan_status		
0	43356	8617
1	2566	9799

In [183]:
```
recall_score(y_train ,LogReg.predict(X_train))
```
Out[183]: 0.793304369010882

In [184]:
```
recall_score(y_test ,LogReg.predict(X_test))
```
Out[184]: 0.7924787707238172

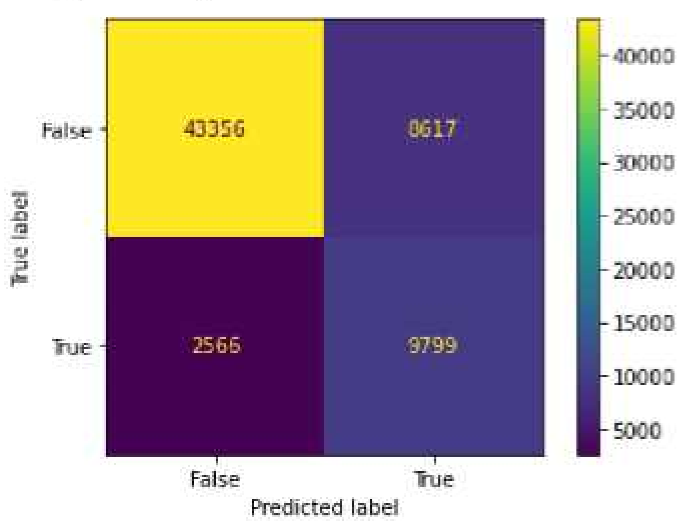
In [185]:
```
f1_score(y_test ,LogReg.predict(X_test))
```
Out[185]: 0.6366914655144408

In [186]:
```
f1_score(y_train ,LogReg.predict(X_train))
```
Out[186]: 0.6381779875549168

In [187]:
```
from sklearn.metrics import ConfusionMatrixDisplay
```

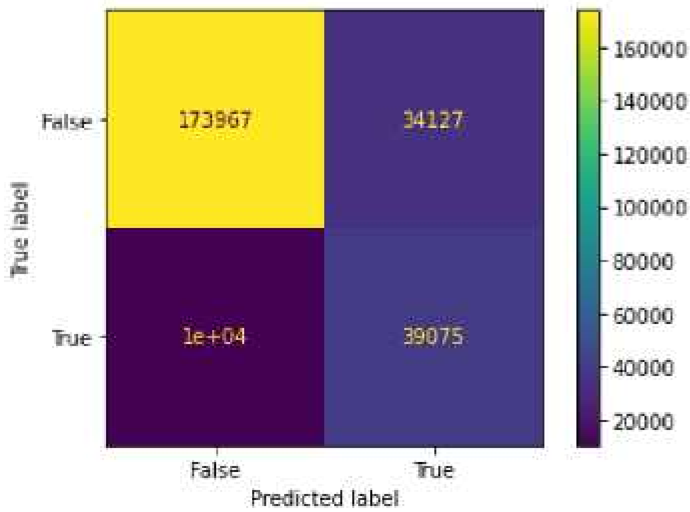
In [188]:
```
from sklearn.metrics import fbeta_score
```

In [189]:
```
cm_display  = ConfusionMatrixDisplay(confusion_matrix= confusion_matri
                                                           LogReg.predi
cm_display.plot()
plt.show()
```



In [190]:
```
# fbeta_score
```

In [191]:
```
cm_display  = ConfusionMatrixDisplay(confusion_matrix= confusion_matri
                                                         LogReg.predic
cm_display.plot()
plt.show()
```



In [192]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [193]:
```
DecisionTreeClassifier = DecisionTreeClassifier(max_depth=5, splitter='
                                    criterion="entropy",clas
```

In [194]:
```
DecisionTreeClassifier.fit(X_train,y_train)
```

Out[194]:
```
DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
                       max_depth=5)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [195]:
```
DecisionTreeClassifier.score(X_test,y_test)
```

Out[195]:
```
0.7946936491653456
```

In [196]:
```
# DecisionTreeClassifier.score(X_smote,y_smote)
```

In [197]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [198]:
```
RF = RandomForestClassifier(n_estimators=30,max_depth=10,class_weight='
```

In [199]:
```
RF.fit(X_train,y_train)
```
Out[199]: RandomForestClassifier(class_weight='balanced', max_depth=10, n_estimators=30)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [200]:
```
RF.score(X_test,y_test)
```
Out[200]: 0.8114799962697007

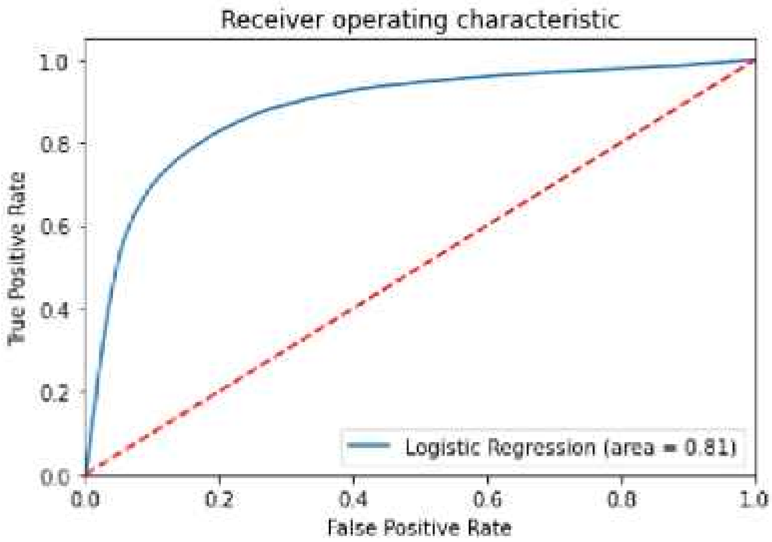
In [201]:
```
feature_importance = pd.DataFrame(index = df.drop(["loan_status"],
                                                  axis = 1).columns,
                                  data = RF.feature_importances_.ravel(
feature_importance
```
Out[201]:

	index	0
0	loan_amnt	0.008274
1	term	0.015486
2	int_rate	0.046712
3	installment	0.008575
4	grade	0.060045
5	sub_grade	0.061285
6	emp_title	0.739095
7	emp_length	0.002199
8	home_ownership	0.004068
9	annual_inc	0.010301
10	verification_status	0.004592
11	purpose	0.002172
12	dti	0.015531
13	open_acc	0.003705
14	revol_bal	0.005976
15	revol_util	0.007677
16	total_acc	0.004228
17	application_type	0.000080

In [203]:
```
from sklearn.metrics import precision_recall_curve
```
In [206]:
```
from sklearn.metrics import roc_auc_score,roc_curve
```

In [207]:

```
logit_roc_auc = roc_auc_score(y_test, LogReg.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, LogReg.predict_proba(X_test)[
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



In [208]:

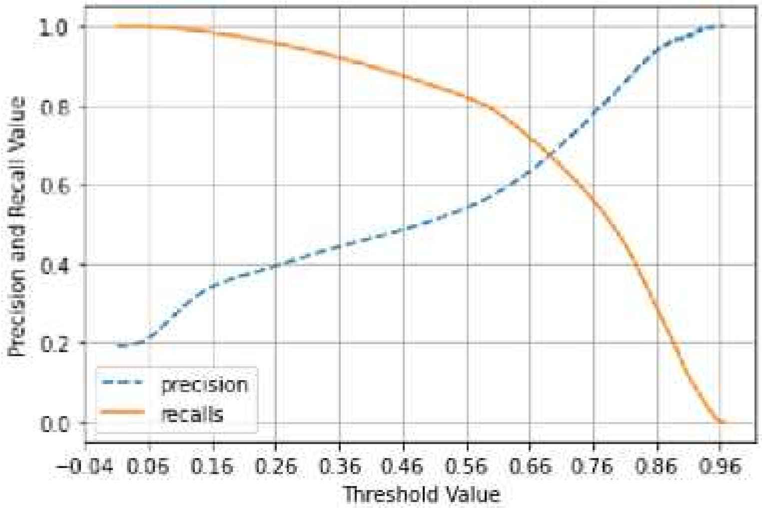
```
LogReg.predict_proba(X_test)
```

Out[208]:

```
array([[0.73865924, 0.26134076],
       [0.85546227, 0.14453773],
       [0.53549564, 0.46450436],
       ...,
       [0.61298399, 0.38701601],
       [0.80033023, 0.19966977],
       [0.7290943 , 0.2709057 ]])
```

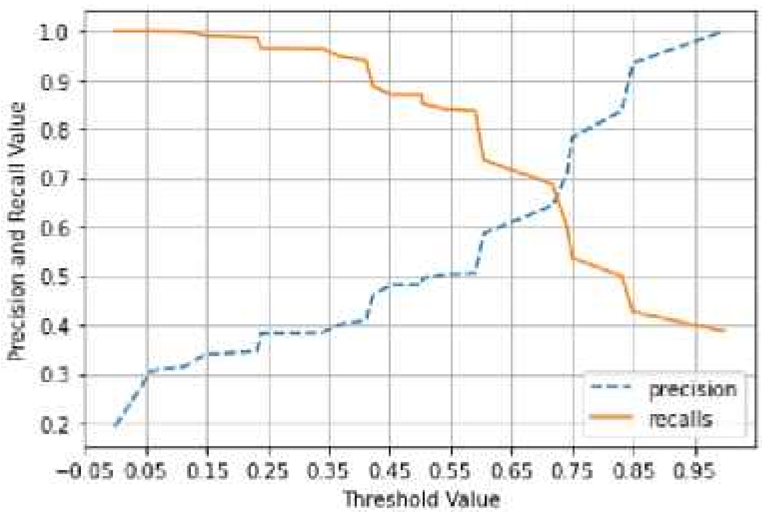
In [209]:
```
precision_recall_curve_plot(y_test, RF.predict_proba(X_test)[:,1])

```



In [210]:
```
precision_recall_curve_plot(y_test, DecisionTreeClassifier.predict_prob

```



In [211]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(class_weight="balanced")
model.fit(X_train, y_train)
```

Out[211]:
```
LogisticRegression(class_weight='balanced')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [212]:
```
def custom_predict(X, threshold):
    probs = model.predict_proba(X)
    return (probs[:, 1] > threshold).astype(int)
```

In [213]:
```
new_preds = custom_predict(X=X_test, threshold=0.75)
```

In [214]:
```
model.score(X_test,y_test)
```

Out[214]: 0.8261835928999969

In [215]:
```
precision_score(y_test,new_preds)
```

Out[215]: 0.6747758146142235

Inferences and Report :

- 396030 data points , 26 features , 1 label.

- 80% belongs to the class 0 : which is loan fully paid.
- 20% belongs to the class 1 : which were charged off.

- Loan Amount distribution / media is slightly higher for Charged_off loanStatus.

- Probability of CHarged_off status is higher in case of 60 month term.

- Interest Rate mean and media is higher for Charged_off LoanStatus.

- Probability of Charged_off LoanStatus is higher for Loan Grades are E ,F, G.
- G grade has the highest probability of having defaulter.
- Similar pattern is visible in sub_grades probability plot.

- Employement Length has overall same probability of Loan_status as fully paid and defaulter.
- That means Defaulters has no relation with their Emoployement length.

- For those borrowers who have rental home, has higher probability of defaulters.
- borrowers having their home mortgage and owns have lower probability of defaulter.

- Annual income median is lightly higher for those who's loan status is as fully paid.

- Somehow , verified income borrowers probability of defaulter is higher than those who are not verified by loan tap.

- Most of the borrowers take loans for dept-consolidation and credit card payoffs.
- the probability of defaulters is higher in the small_business owner borrowers.

- debt-to-income ratio is higher for defaulters.

- number of open credit lines in the borrowers credit file is same as for loan status as fully paid and defaulters.

- Number of derogatory public records increases , the probability of borrowers declared as defaulters also increases
- aspecially for those who have higher than 12 public_records.

- Total credit revolving balance is almost same for both borrowers who had fully paid loan and declared defaulter
- but Revolving line utilization rate is higher for defaulter borrowers.

- Application type Direct-Pay has higher probability of defaulter borrowers than individual and joint.

- Number of public record bankruptcies increasaes , higher the probability of defaulters.

- Most important features/ data for prediction , as per Logistic Regression, Decision tree classifier and Random Forest model are : Employee Title, Loan Grade and Sub-Grade,

Actionable Insights & Recommendations

- We should try to keep the precision higher as possible compare to recall , and keep the false positive low.
- that will help not to missout the opportopportunity to finance more individuals and earn interest on it. This we can achieve by setting up the higher threshold.
- Giving loans to those even having slightly higher probability of defaulter, we can maximise the earning , by this risk taking method.
- and Since NPA is a real problem in the industry , Company should more investigate and check for the proof of assets. Since it was observed in probability plot, verified borrowers had higher probability of defaulters than non-varified.
- Giving loans to those who have no mortgage house of any owned property have higher probability of defaulter , giving loan to this category borrowers can be a problem of NPA.

In [ ]:

```
1
```