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

Data dictionary:

1. 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.
2. term : The number of payments on the loan. Values are in months and can be either 36 or 60.
3. int_rate : Interest Rate on the loan
4. installment : The monthly payment owed by the borrower if the loan originates.
5. grade : Institution assigned loan grade
6. sub_grade : Institution assigned loan subgrade
7. emp_title :The job title supplied by the Borrower when applying for the loan.*

8. 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.
9. home_ownership : The home ownership status provided by the borrower during registration or obtained from the credit report.
10. annual_inc : The self-reported annual income provided by the borrower during registration.
11. verification_status : Indicates if income was verified by Institution, not verified, or if the income source was verified
12. issue_d : The month which the loan was funded
13. loan_status : Current status of the loan - Target Variable
14. purpose : A category provided by the borrower for the loan request.
15. title : The loan title provided by the borrower
16. dti : A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested Institution loan, divided by the borrower's self-reported monthly income.
17. earliest_cr_line :The month the borrower's earliest reported credit line was opened
18. open_acc : The number of open credit lines in the borrower's credit file.
19. pub_rec : Number of derogatory public records
20. revol_bal : Total credit revolving balance
21. revol_util : Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.
22. total_acc : The total number of credit lines currently in the borrower's credit file
23. initial_list_status : The initial listing status of the loan. Possible values are – W, F
24. application_type : Indicates whether the loan is an individual application or a joint application with two co-borrowers
25. mort_acc : Number of mortgage accounts.
26. pub_rec_bankruptcies : Number of public record bankruptcies
27. Address: Address of the individual

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import figure

import statsmodels.api as sm
from scipy.stats import norm
from scipy.stats import t

import warnings
warnings.filterwarnings('ignore')

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

In [3]:
```
df = pd.read_csv("logistic_regression.csv")
```

In [5]:
```
df.head()
```

Out[5]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_owne
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	MORT
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	MORT

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

Out[4]:
```
(396030, 27)
```

Missing Value Check :

In [5]:
```
def missing_df(data):
    total_missing_df = data.isna().sum().sort_values(ascending = False)
    percentage_missing_df = ((data.isna().sum()/len(data)*100)).sort_values(ascending
    missingDF = pd.concat([total_missing_df, percentage_missing_df],axis = 1, keys=['T
    return missingDF


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

Out[5]:

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

Out[6]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
installment             0.000000
grade                   0.000000
sub_grade               0.000000
emp_title               5.789208
emp_length              4.621115
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.443148
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.069692
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                9.543469
pub_rec_bankruptcies    0.135091
address                 0.000000
dtype: float64
```

Descriptive Statistics :

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

Out[7]:

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_bal	revol_u
count	396030.0	396030.0	396030.0	396030.0	396030.0	396030.0	396030.0	396030.0	395754
mean	14113.9	13.6	431.8	74203.2	17.4	11.3	0.2	15844.5	53
std	8357.4	4.5	250.7	61637.6	18.0	5.1	0.5	20591.8	24
min	500.0	5.3	16.1	0.0	0.0	0.0	0.0	0.0	0
25%	8000.0	10.5	250.3	45000.0	11.3	8.0	0.0	6025.0	35
50%	12000.0	13.3	375.4	64000.0	16.9	10.0	0.0	11181.0	54
75%	20000.0	16.5	567.3	90000.0	23.0	14.0	0.0	19620.0	72
max	40000.0	31.0	1533.8	8706582.0	9999.0	90.0	86.0	1743266.0	892

- #### 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()
```

Out[8]:
```
loan_amnt                 1397
term                         2
int_rate                   566
installment              55706
grade                        7
sub_grade                   35
emp_title               173105
emp_length                  11
home_ownership               6
annual_inc               27197
verification_status          3
issue_d                    115
loan_status                  2
purpose                     14
title                    48817
dti                       4262
earliest_cr_line           684
open_acc                    61
pub_rec                     20
revol_bal                55622
revol_util                1226
total_acc                  118
initial_list_status          2
application_type             3
mort_acc                    33
pub_rec_bankruptcies         9
address                 393700
dtype: int64
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  object 
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  object 
 5   sub_grade             396030 non-null  object 
 6   emp_title             373103 non-null  object 
 7   emp_length            377729 non-null  object 
 8   home_ownership        396030 non-null  object 
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  object 
 11  issue_d               396030 non-null  object 
 12  loan_status           396030 non-null  object 
 13  purpose               396030 non-null  object 
 14  title                 394275 non-null  object 
 15  dti                   396030 non-null  float64
 16  earliest_cr_line      396030 non-null  object 
 17  open_acc              396030 non-null  float64
 18  pub_rec               396030 non-null  float64
 19  revol_bal             396030 non-null  float64
 20  revol_util            395754 non-null  float64
 21  total_acc             396030 non-null  float64
 22  initial_list_status   396030 non-null  object 
 23  application_type      396030 non-null  object 
 24  mort_acc              358235 non-null  float64
 25  pub_rec_bankruptcies  395495 non-null  float64
 26  address               396030 non-null  object 
dtypes: float64(12), object(15)
memory usage: 81.6+ MB
```

In [10]:

```
columns_type = df.dtypes
columns_type[columns_type=="object"]
```

Out[10]:

```
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 [11]:

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

Out[11]:

	term	grade	sub_grade	emp_title	emp_length	home_ownership	verification_status	issue
count	396030	396030	396030	373103	377729	396030	396030	396(
unique	2	7	35	173105	11	6	3	'
top	36 months	B	B3	Teacher	10+ years	MORTGAGE	Verified	C 2(
freq	302005	116018	26655	4389	126041	198348	139563	148

In [12]:
```
len(columns_type[columns_type=="object"])
```

Out[12]:
```
15
```

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

In [ ]:

In [ ]:

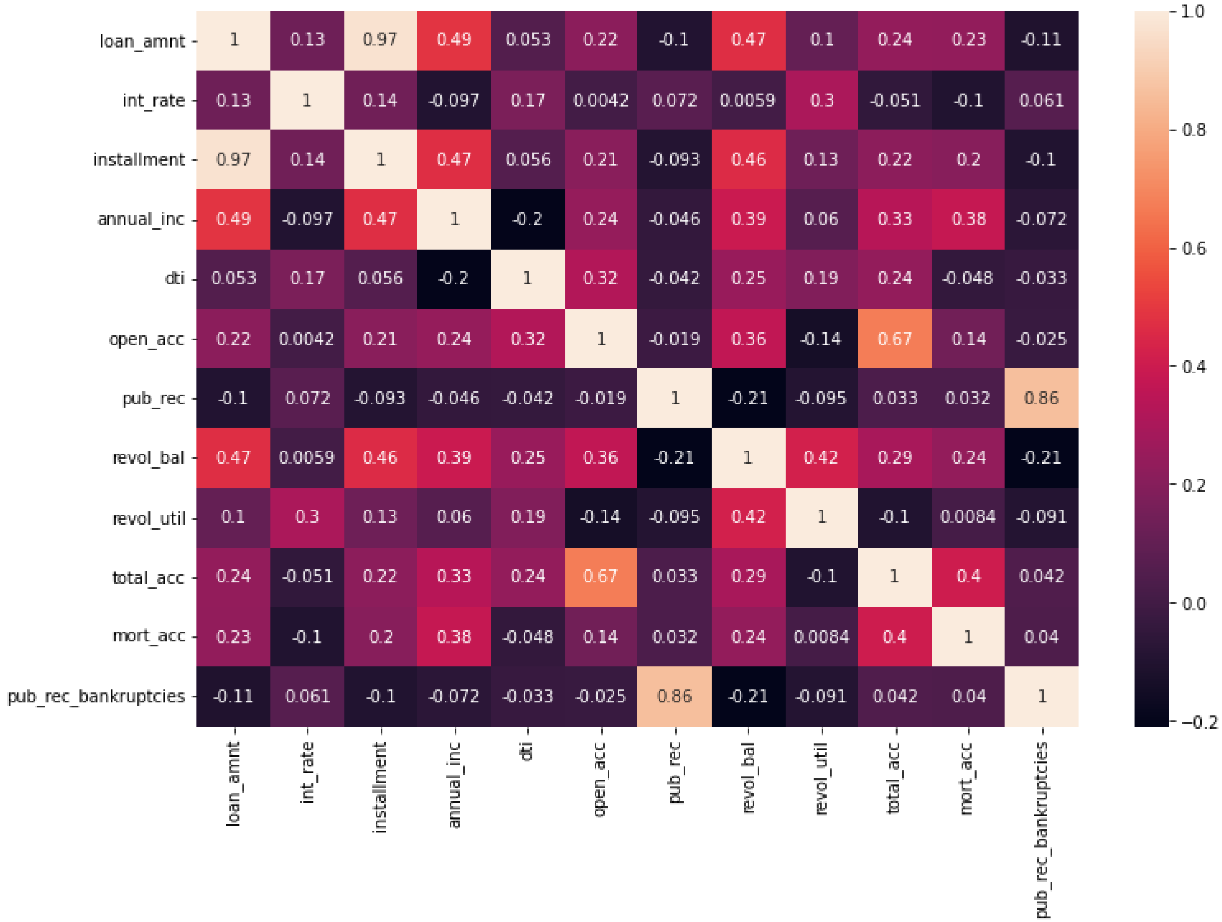
In [13]:
```
df["loan_status"].value_counts(normalize=True)*100
```

Out[13]:
```
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.

checking for very high colinearity using heatmap - correlation matrix :

In [14]:
```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(method='spearman'), annot=True)
plt.show()
```

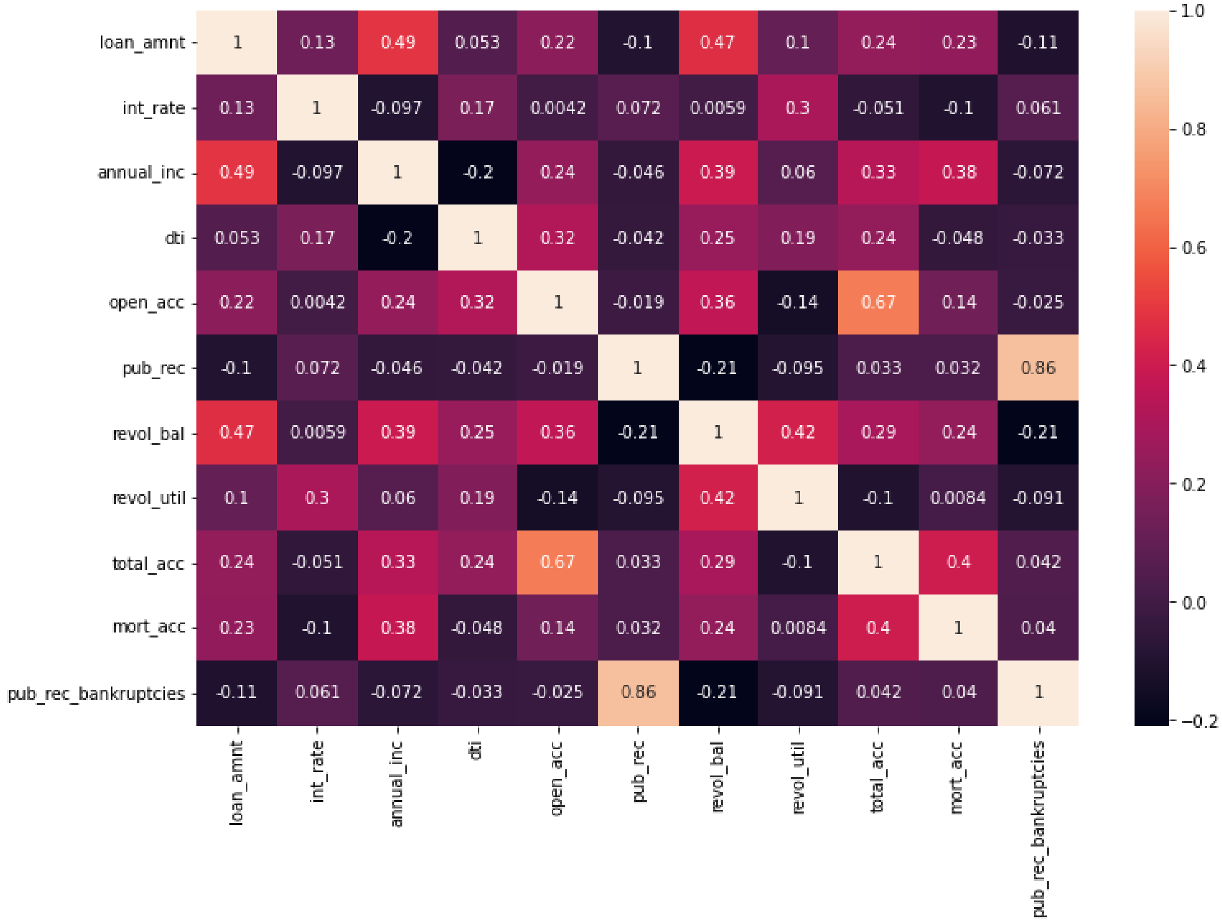


removing intallment column , since it has very high correlation with loan_amount.

basically both represent same thing

In [15]:
```
df.drop("installment",axis = 1 , inplace=True)
```

In [16]:
```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(method='spearman'), annot=True)
plt.show()
```



Data Exploration

- #### 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.

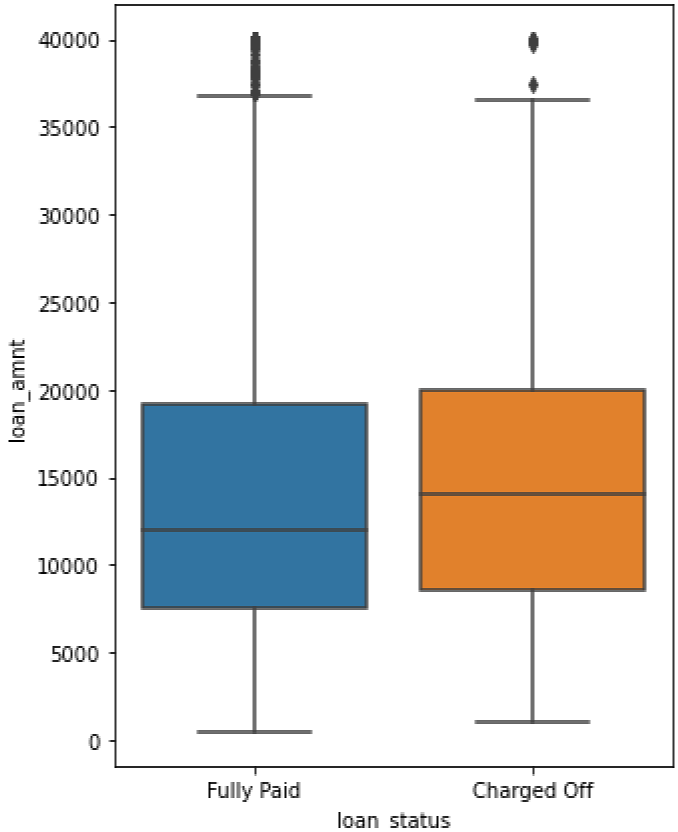
In [17]:
```
df.groupby(by = "loan_status")["loan_amnt"].describe()
```

Out[17]:

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 [18]:
```
plt.figure(figsize=(5,7))
sns.boxplot(y=df["loan_amnt"],
            x=df["loan_status"])
```

Out[18]: <AxesSubplot:xlabel='loan_status', ylabel='loan_amnt'>



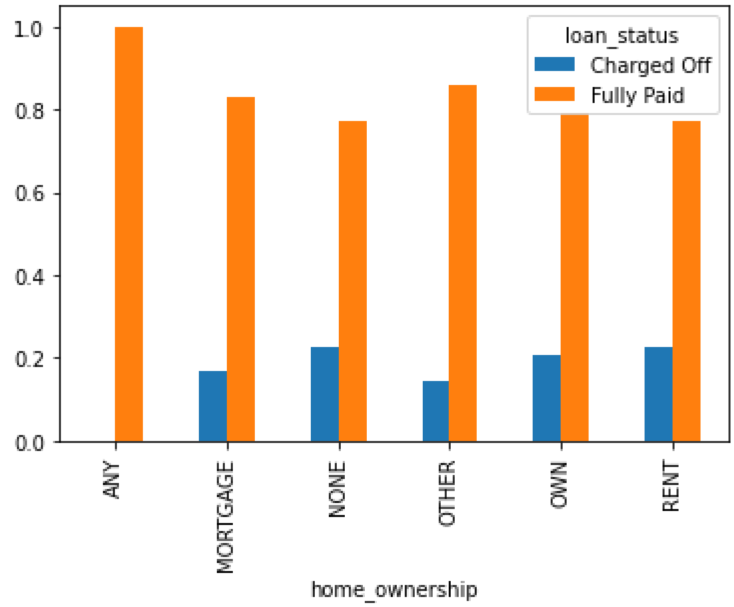
home_ownership

In [19]:
```
df["home_ownership"].value_counts()
```

Out[19]:
```
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
ANY              3
Name: home_ownership, dtype: int64
```

In [20]:
```
pd.crosstab(columns = df["loan_status"],
            index=df["home_ownership"],
            normalize="index").plot(kind="bar")
```

Out[20]:
```
<AxesSubplot:xlabel='home_ownership'>
```



In [21]:
```
df.groupby(by = "loan_status")["home_ownership"].describe()
```

Out[21]:

loan_status	count	unique	top	freq
Charged Off	77673	5	RENT	36212
Fully Paid	318357	6	MORTGAGE	164716

majority of people have home ownership as Mortgage and Rented.

Borrows having rented home, have higher conditional probability of loanStatus as Charged off.

In [22]:
```
# df.loc[(df.home_ownership == "ANY") | (df.home_ownership == "NONE")]["home_ownershir
```

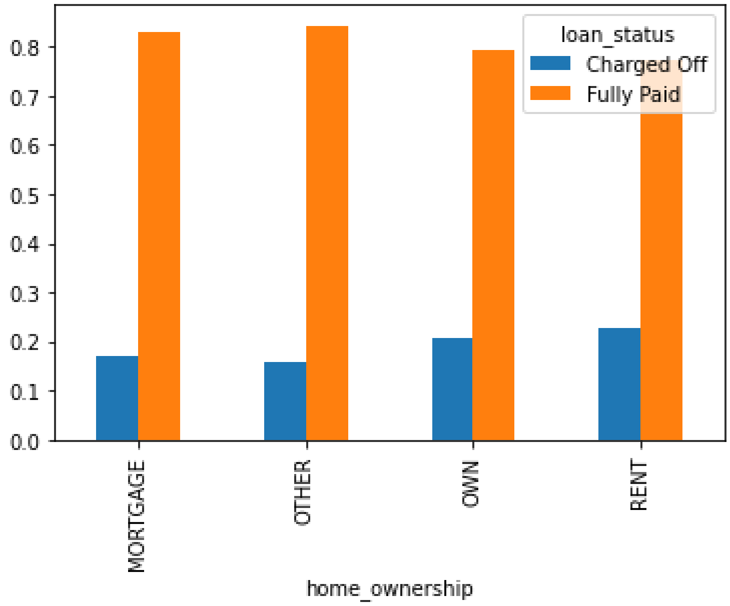
In [23]:
```
df["home_ownership"].unique()
```

Out[23]:
```
array(['RENT', 'MORTGAGE', 'OWN', 'OTHER', 'NONE', 'ANY'], dtype=object)
```

In [24]:
```
df["home_ownership"].replace({"ANY":"OTHER",
                              "NONE":"OTHER"},
                             inplace=True)
```

In [25]:
```
pd.crosstab(columns = df["loan_status"],
           index=df["home_ownership"],
           normalize="index").plot(kind="bar")
```

Out[25]:
```
<AxesSubplot:xlabel='home_ownership'>
```



In [ ]:

Interest Rate :

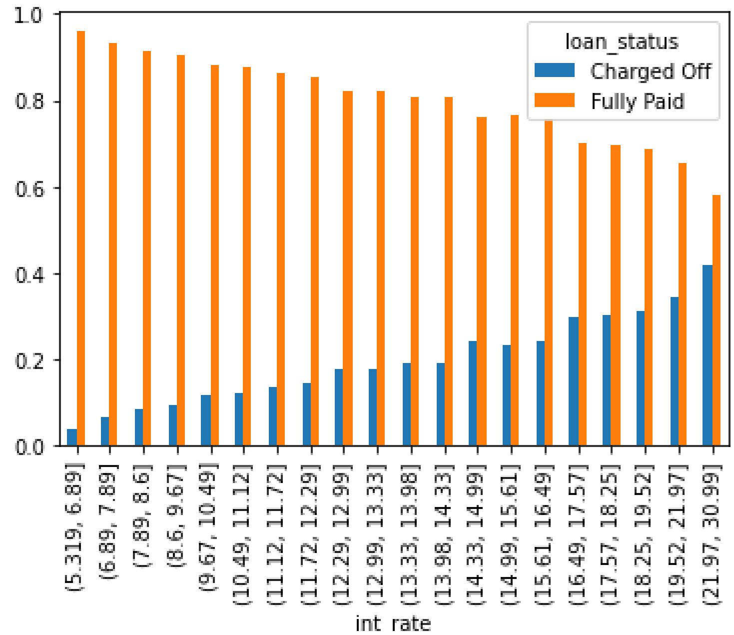
In [26]:
```
df.groupby(by = "loan_status")["int_rate"].describe()
```

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]:
```
pd.crosstab(columns = df["loan_status"],
            index=pd.qcut(df["int_rate"],20),
            normalize="index").plot(kind="bar")
```

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



In [28]:
```
# Higher the interest , probability of defailter is decreases.
```

In [ ]:

In [ ]:

Some issue with "title" having duplicat values : needs to be fixed

In [29]:
```
df["title"].value_counts()[:20]
```

Out[29]:
```
Debt consolidation          152472
Credit card refinancing      51487
Home improvement             15264
Other                        12930
Debt Consolidation           11608
Major purchase                4769
Consolidation                 3852
debt consolidation            3547
Business                      2949
Debt Consolidation Loan       2864
Medical expenses              2742
Car financing                 2139
Credit Card Consolidation     1775
Vacation                      1717
Moving and relocation         1689
consolidation                 1595
Personal Loan                 1591
Consolidation Loan            1299
Home Improvement              1268
Home buying                   1183
Name: title, dtype: int64
```

In [30]:
```
df["title"] = df["title"].str.lower()
```

In [31]:
```
df["title"].value_counts()[:20]
```

Out[31]:
```
debt consolidation            168108
credit card refinancing        51781
home improvement               17117
other                          12993
consolidation                   5583
major purchase                  4998
debt consolidation loan         3513
business                        3017
medical expenses                2820
credit card consolidation       2638
personal loan                   2460
car financing                   2160
credit card payoff              1904
consolidation loan              1887
vacation                        1866
credit card refinance           1832
moving and relocation           1693
consolidate                     1528
personal                        1465
home buying                     1196
Name: title, dtype: int64
```

In [ ]:

Changing data types of Date time columns :"

In [32]:
```
df['issue_d'] = pd.to_datetime(df['issue_d'])
df['earliest_cr_line'] = pd.to_datetime(df['earliest_cr_line'])
```

In [ ]:

Loan Grades :

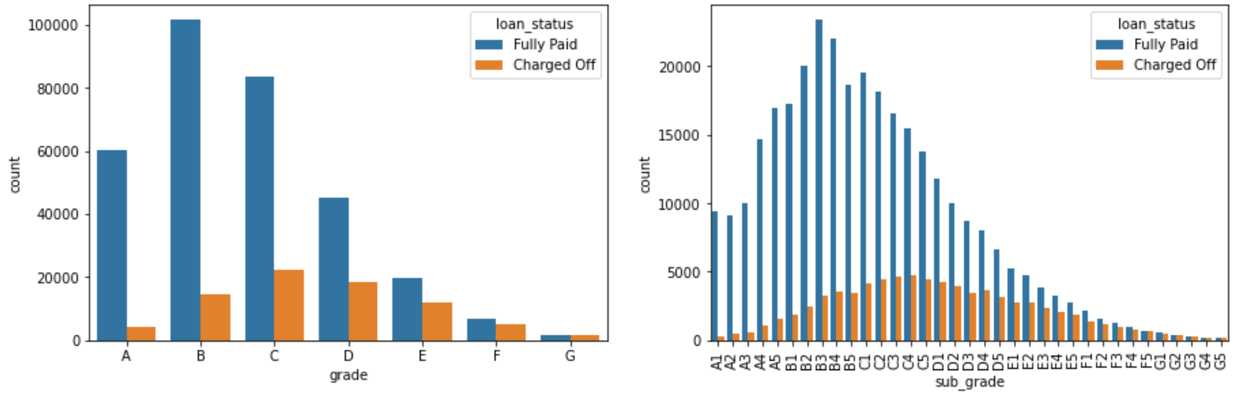
In [33]:
```
df["grade"].unique()
```

Out[33]:
```
array(['B', 'A', 'C', 'E', 'D', 'F', 'G'], dtype=object)
```

In [34]:
```
df["sub_grade"].unique()
```

Out[34]:
```
array(['B4', 'B5', 'B3', 'A2', 'C5', 'C3', 'A1', 'B2', 'C1', 'A5', 'E4',
       'A4', 'A3', 'D1', 'C2', 'B1', 'D3', 'D5', 'D2', 'E1', 'E2', 'E5',
       'F4', 'E3', 'D4', 'G1', 'F5', 'G2', 'C4', 'F1', 'F3', 'G5', 'G4',
       'F2', 'G3'], dtype=object)
```

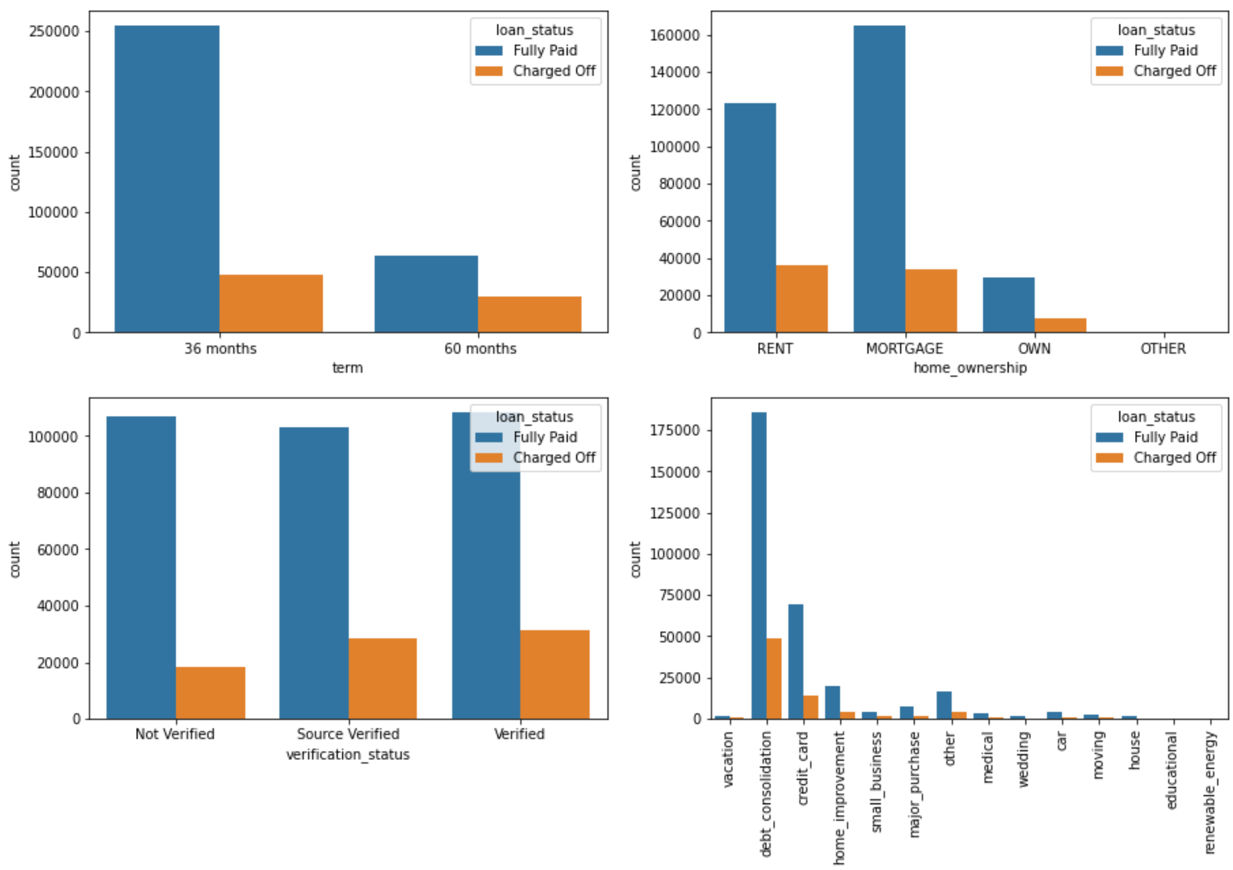
In [35]:
```
plt.figure(figsize=(15, 10))
plt.subplot(2, 2, 1)
grade = sorted(df.grade.unique().tolist())
sns.countplot(x='grade', data=df, hue='loan_status', order=grade)
plt.subplot(2, 2, 2)
sub_grade = sorted(df.sub_grade.unique().tolist())
g = sns.countplot(x='sub_grade', data=df, hue='loan_status', order=sub_grade)
g.set_xticklabels(g.get_xticklabels(), rotation=90);
```



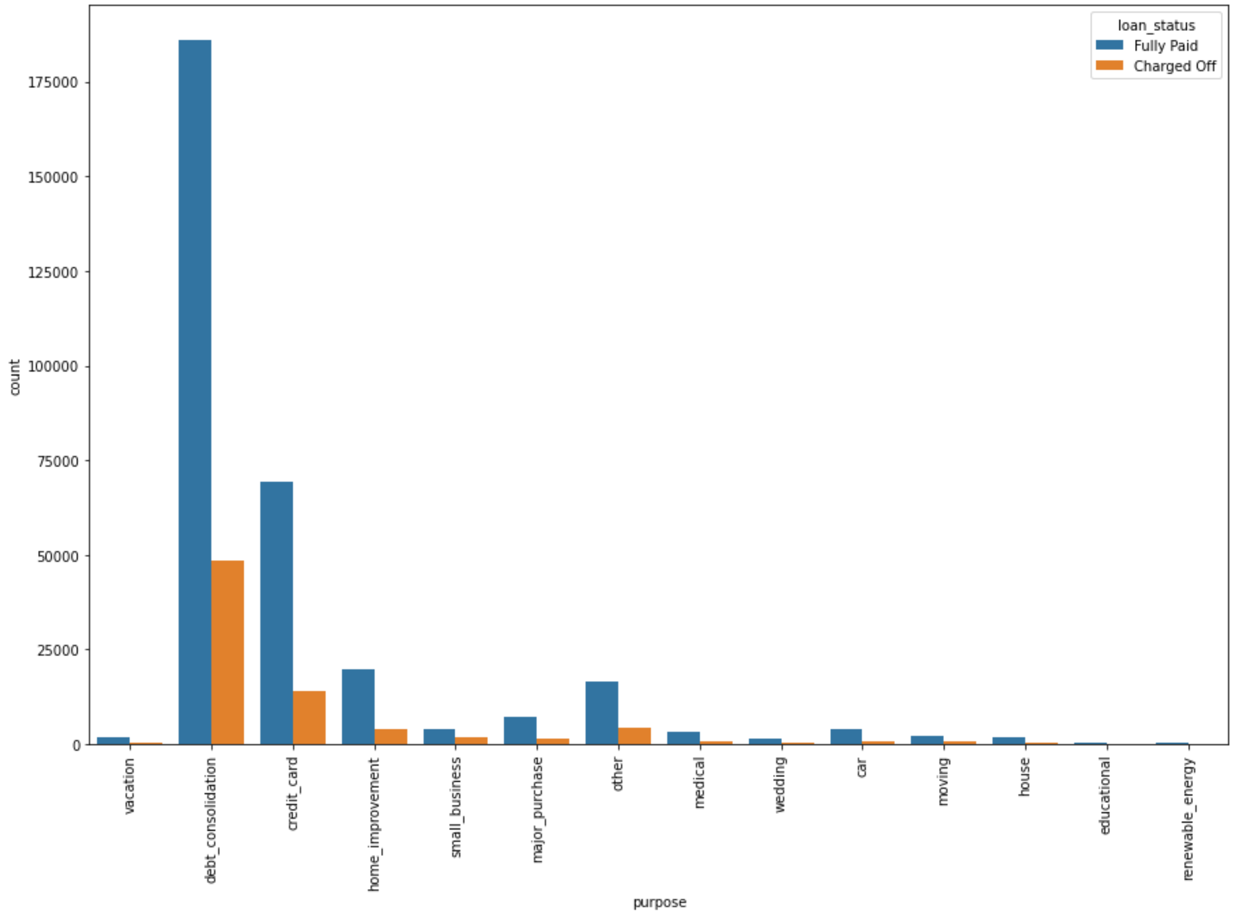

```
In [ ]:
```

Probability plots for other categorical features :

```
In [36]: plt.figure(figsize=(15, 20))
plt.subplot(4, 2, 1)
sns.countplot(x='term', data=df, hue='loan_status')
plt.subplot(4, 2, 2)
sns.countplot(x='home_ownership', data=df, hue='loan_status')
plt.subplot(4, 2, 3)
sns.countplot(x='verification_status', data=df, hue='loan_status')
plt.subplot(4, 2, 4)
g = sns.countplot(x='purpose', data=df, hue='loan_status')
g.set_xticklabels(g.get_xticklabels(), rotation=90);
```

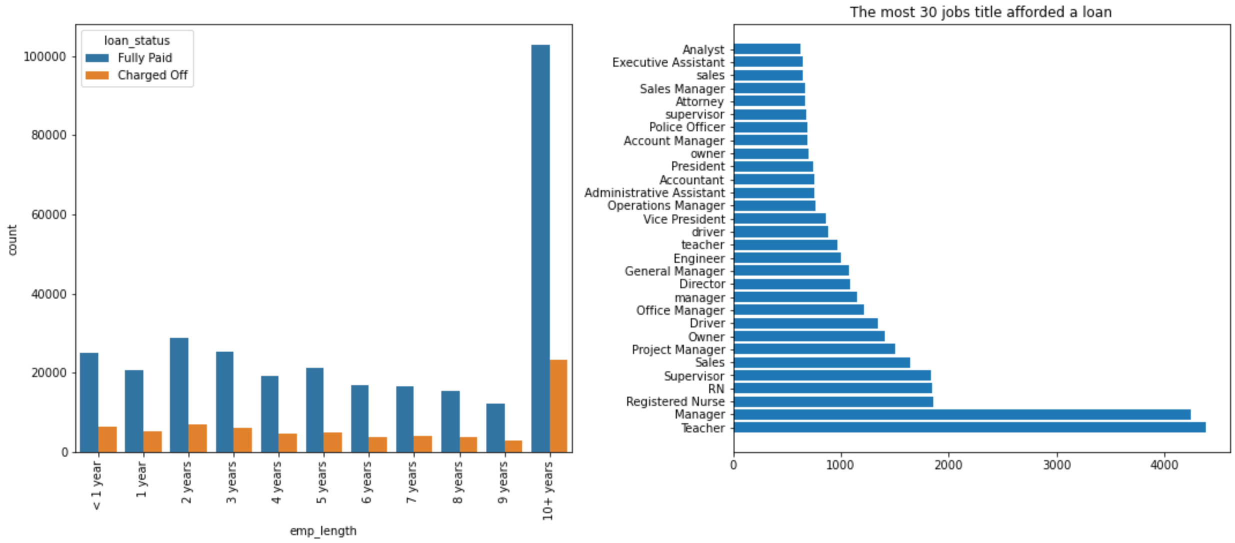


In [37]:
```
plt.figure(figsize=(15, 10))

sns.countplot(x='purpose', data=df, hue='loan_status')
plt.xticks(rotation = 90)
plt.show()
```



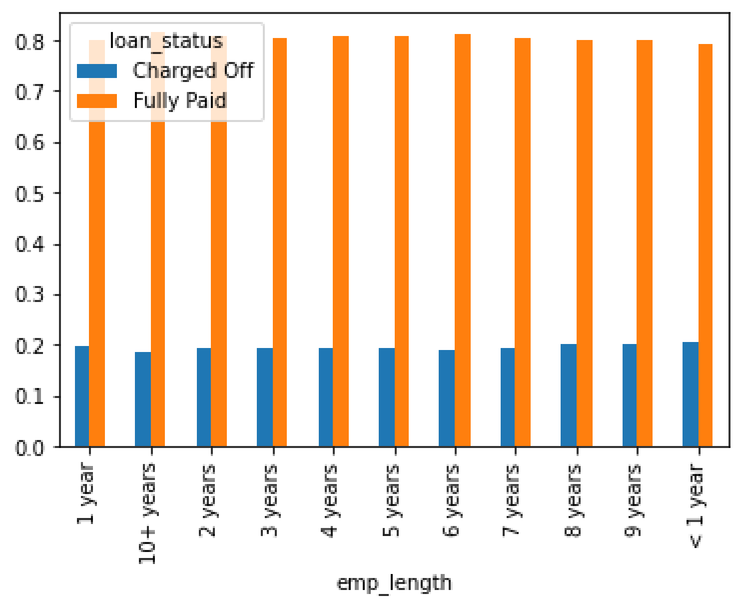
Loan taken for the porpose like dept_consolidation , credit card payments , small business investments , have high probability of borrower defaults.

In [38]:
```
plt.figure(figsize=(15, 12))

plt.subplot(2, 2, 1)
order = ['< 1 year', '1 year', '2 years', '3 years', '4 years', '5 years',
 '6 years', '7 years', '8 years', '9 years', '10+ years',]
g = sns.countplot(x='emp_length', data=df, hue='loan_status', order=order)
g.set_xticklabels(g.get_xticklabels(), rotation=90);


plt.subplot(2, 2, 2)
plt.barh(df.emp_title.value_counts()[:30].index,
         df.emp_title.value_counts()[:30])
plt.title("The most 30 jobs title afforded a loan")
plt.tight_layout()
```



In [39]:
```
pd.crosstab(index = df["emp_length"],
            columns= df["loan_status"],normalize= "index").plot(kind = "bar")
```

Out[39]: <AxesSubplot:xlabel='emp_length'>



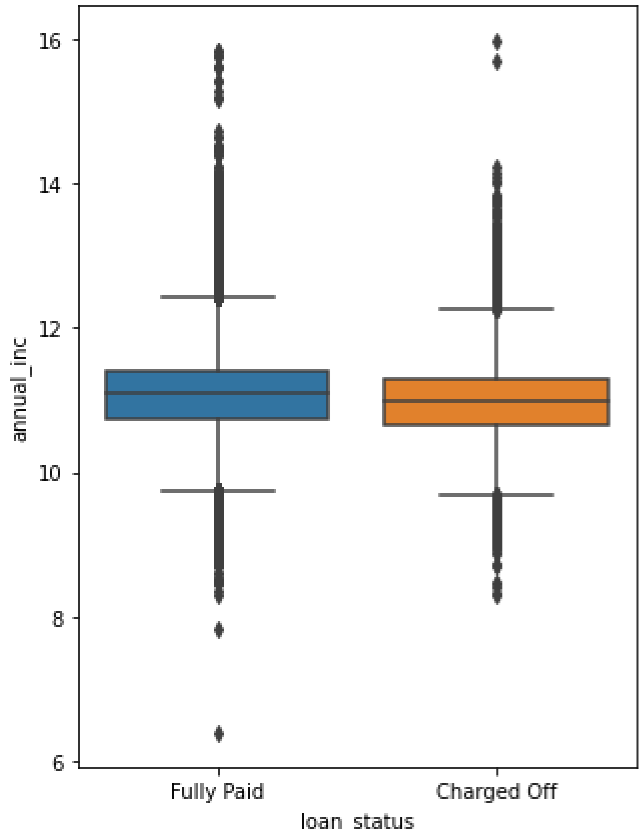
In [40]:
```
# there doent seems to be much correlation between employement length
# and loan_status.
```

In [ ]:

Annual Income :

In [41]:
```
plt.figure(figsize=(5,7))
sns.boxplot(y=np.log(df[df["annual_inc"]>0]["annual_inc"]),
            x=df["loan_status"])
```

Out[41]: <AxesSubplot:xlabel='loan_status', ylabel='annual_inc'>

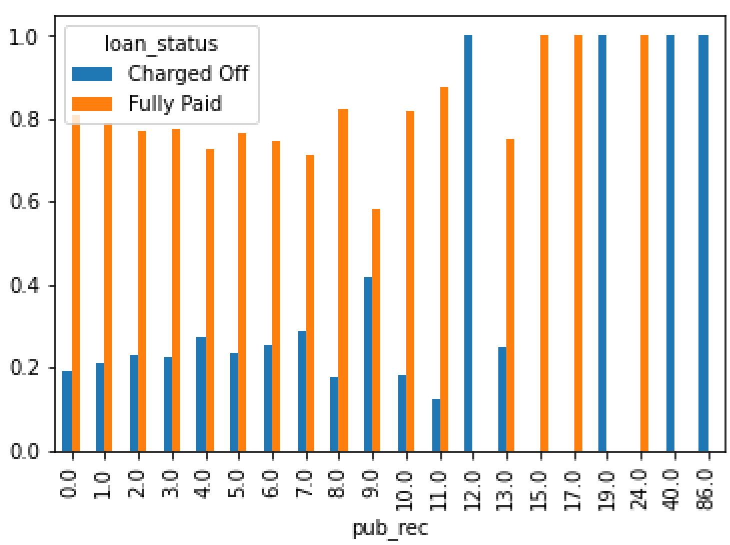


In [42]:
```
# no difference between annual income, for loan status categories
```

In [ ]:
```

```

In [43]:
```
pd.crosstab(columns = df["loan_status"],
            index=(df["pub_rec"]),
            normalize="index").plot(kind="bar")
```

Out[43]: <AxesSubplot:xlabel='pub_rec'>

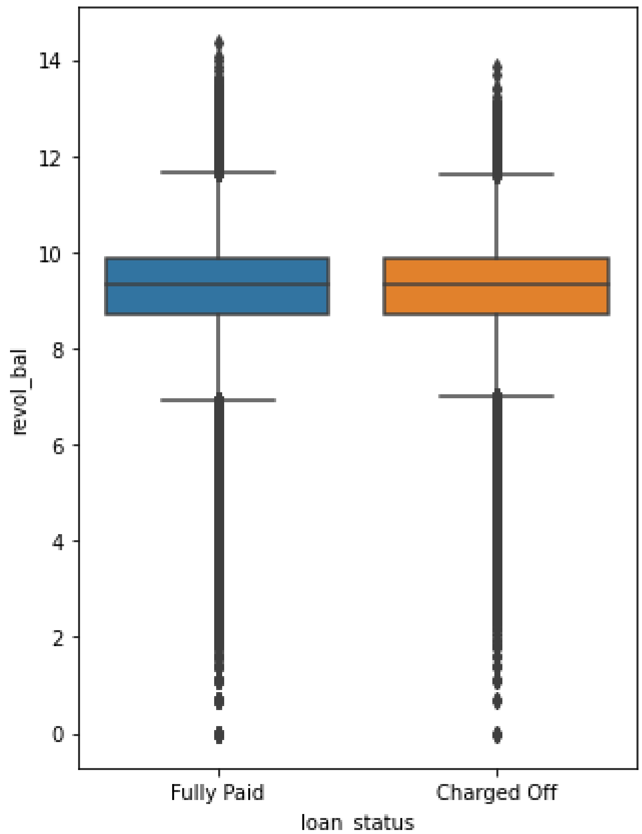


In [44]:
```
# borrowers who have more the public records in history , have higher chances of defau
```

In [45]:
```
plt.figure(figsize=(5,7))
sns.boxplot(y= np.log(df["revol_bal"]),
```

```
            x=df["loan_status"])
```

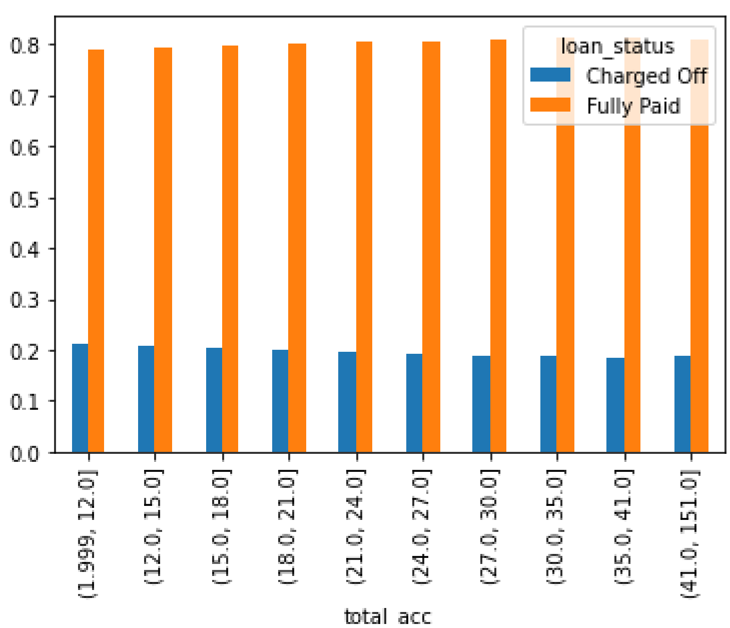
Out[45]: <AxesSubplot:xlabel='loan_status', ylabel='revol_bal'>



In [46]:
```
# revolving balance is almost similar distribution for defaulters and non-defaulter.
```

In [47]:
```
pd.crosstab(columns = df["loan_status"],
            index=pd.qcut(df["total_acc"],10),
            normalize="index").plot(kind="bar")
```

Out[47]: <AxesSubplot:xlabel='total_acc'>



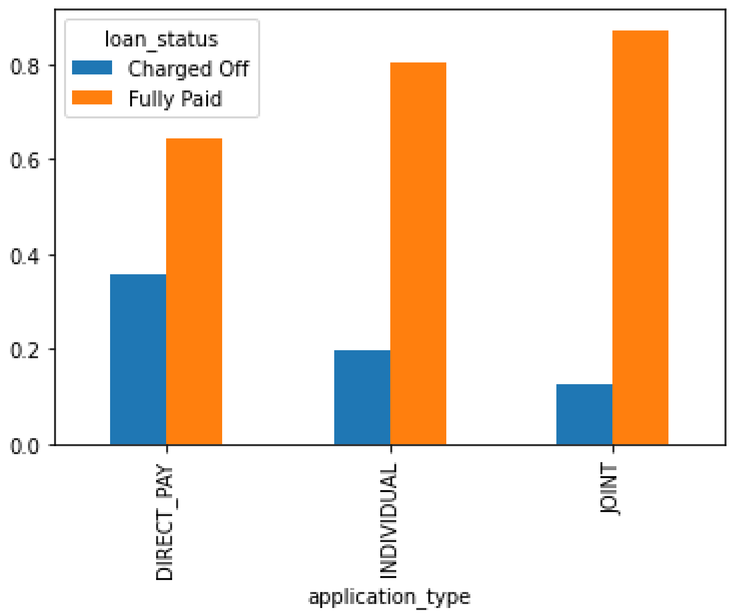
In [48]:
```
# The total number of credit lines
# currently in the borrower's credit file are same for defaulters and non defaulters.
```

Application type :

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

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

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

Out[49]:
```
<AxesSubplot:xlabel='application_type'>
```



In [ ]:

Feature Engineering :

In [ ]:

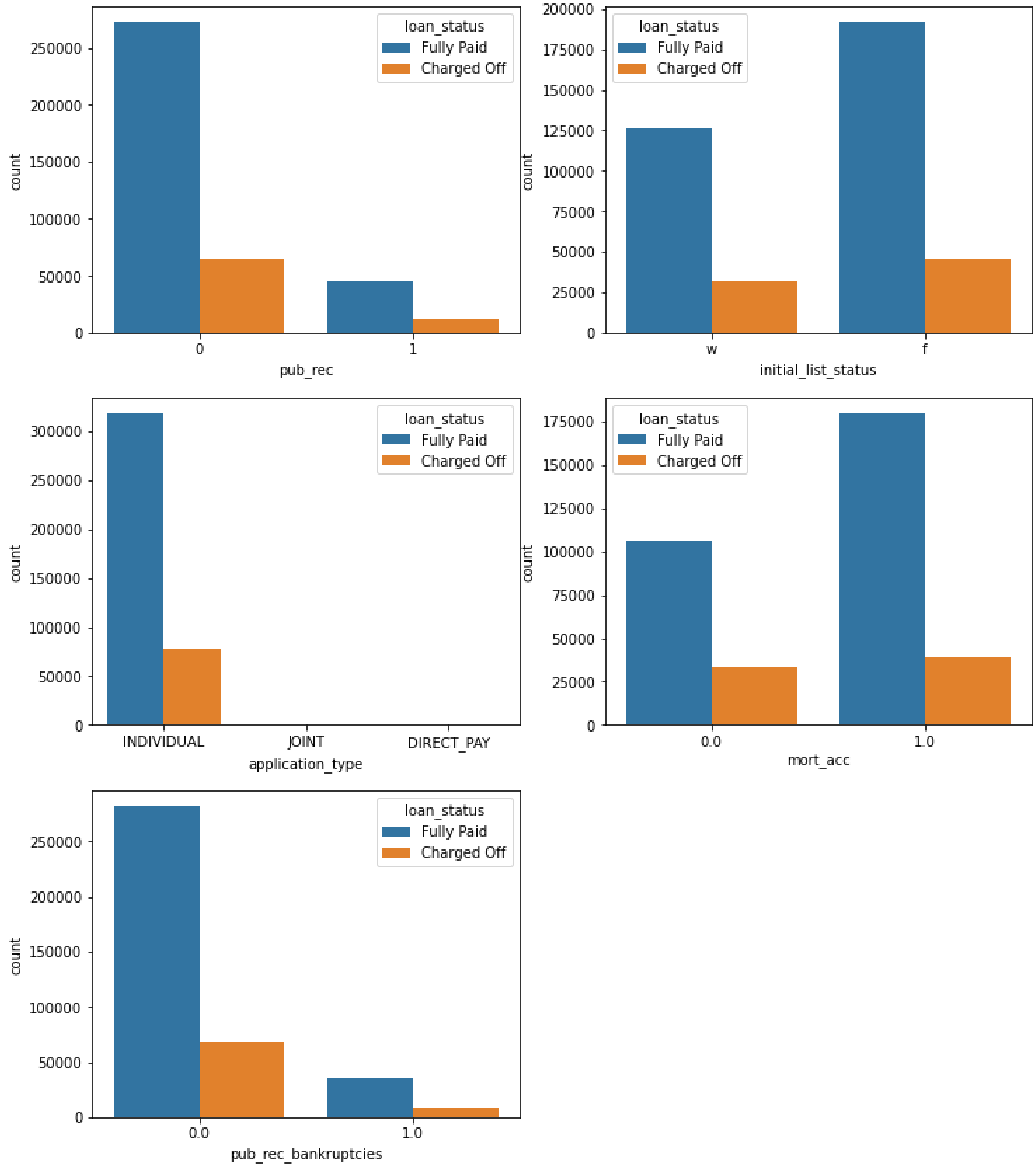
In [ ]:

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

def mort_acc(number):
    if number == 0.0:
        return 0
    elif number >= 1.0:
        return 1
```

```
    else:
        return number

def pub_rec_bankruptcies(number):
    if number == 0.0:
        return 0
    elif number >= 1.0:
        return 1
    else:
        return number
```

In [51]:
```
df['pub_rec'] = df.pub_rec.apply(pub_rec)
df['mort_acc'] = df.mort_acc.apply(mort_acc)
df['pub_rec_bankruptcies'] = df.pub_rec_bankruptcies.apply(pub_rec_bankruptcies)
```

In [52]:
```
plt.figure(figsize=(12, 30))
plt.subplot(6, 2, 1)
sns.countplot(x='pub_rec', data=df, hue='loan_status')
plt.subplot(6, 2, 2)
sns.countplot(x='initial_list_status', data=df, hue='loan_status')
plt.subplot(6, 2, 3)
sns.countplot(x='application_type', data=df, hue='loan_status')
plt.subplot(6, 2, 4)
sns.countplot(x='mort_acc', data=df, hue='loan_status')
plt.subplot(6, 2, 5)
sns.countplot(x='pub_rec_bankruptcies', data=df, hue='loan_status')
plt.show()
```



In [ ]:

Converting target values :

- ### if loanStatus is Fully Paid : 0
- ### if loanStatus is Charged Off : 1

In [53]:
```
df["loan_status"] = df["loan_status"].map({"Fully Paid":0,
                                           "Charged Off":1})
```

In [ ]:

In [ ]:

keeping an eye on missing values

In [54]:
```
# (df.isna().sum()/df.shape[0]) *100
```

In [55]:
```
missing_data[missing_data["Percent"]>0]
```

Out[55]:

	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 [ ]:

In [ ]:

Mean Value Imputation : Target Encoding :

In [56]:
```
# df.groupby("total_acc").mean()
```

In [ ]:

In [57]:
```
total_acc_avg = df.groupby(by='total_acc').mean().mort_acc
```

In [58]:
```
# morc_acc_average_as_per_total_account
```

In [59]:
```
def fill_mort_acc(total_acc, mort_acc):
    if np.isnan(mort_acc):
        return total_acc_avg[total_acc].round()
    else:
        return mort_acc
```

In [60]:
```
df['mort_acc'] = df.apply(lambda x: fill_mort_acc(x['total_acc'], x['mort_acc']),axis
```

In [ ]:

In [61]:
```
missing_data[missing_data["Percent"]>0]
```

Out[61]:

	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 [62]:
```
(df.isna().sum()/df.shape[0]) *100
```

Out[62]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
grade                   0.000000
sub_grade               0.000000
emp_title               5.789208
emp_length              4.621115
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.443148
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.069692
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                0.000000
pub_rec_bankruptcies    0.135091
address                 0.000000
dtype: float64
```

In [ ]:

In [63]:
```
df["emp_title"].nunique()
```

Out[63]:
```
173105
```

In [ ]:

Target Encoding for "pub_rec_bankruptcies":

Since there are 535 missing records of public record bankrupcies , we cannot remove them blindly or can impute with the most frequent. so , giving them some probability value of pub_rec_bankruptcy is a good idea.

replacing those missing values with target encoding

In [64]:
```
df["pub_rec_bankruptcies"].value_counts(dropna=False)
```

Out[64]:
```
0.0    350380
1.0     45115
NaN       535
Name: pub_rec_bankruptcies, dtype: int64
```

In [65]:
```
# df["pub_rec_bankruptcies"].fillna(0).value_counts(dropna=False)
```

In [ ]:

In [66]:
```
from category_encoders import TargetEncoder
TE = TargetEncoder()
```

In [67]:
```
df["pub_rec_bankruptcies"] = df["pub_rec_bankruptcies"].fillna("Record_not_cound").ast
```

In [68]:
```
df["pub_rec_bankruptcies"].value_counts(dropna=False)
```

Out[68]:
```
0.0                   350380
1.0                    45115
Record_not_cound         535
Name: pub_rec_bankruptcies, dtype: int64
```

In [69]:
```
df["pub_rec_bankruptcies"] = TE.fit_transform(df["pub_rec_bankruptcies"],df["loan_stat
```

In [70]:
```
df["pub_rec_bankruptcies"].value_counts(dropna=False)
```

Out[70]:
```
0.194991    350380
0.205364     45115
0.162617       535
Name: pub_rec_bankruptcies, dtype: int64
```

In [ ]:

In [71]:
```
(df.isna().sum()/df.shape[0]) *100
```

Out[71]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
grade                   0.000000
sub_grade               0.000000
emp_title               5.789208
emp_length              4.621115
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.443148
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.069692
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                0.000000
pub_rec_bankruptcies    0.000000
address                 0.000000
dtype: float64
```

In [ ]:

In [72]:
```
df.shape
```

Out[72]:
```
(396030, 26)
```

Dropping rest of the missing values :

In [73]:
```
df.dropna(inplace=True)
```

In [74]:
```
(df.isna().sum()/df.shape[0]) *100
```

Out[74]:
```
loan_amnt               0.0
term                    0.0
int_rate                0.0
grade                   0.0
sub_grade               0.0
emp_title               0.0
emp_length              0.0
home_ownership          0.0
annual_inc              0.0
verification_status     0.0
issue_d                 0.0
loan_status             0.0
purpose                 0.0
title                   0.0
dti                     0.0
earliest_cr_line        0.0
open_acc                0.0
pub_rec                 0.0
revol_bal               0.0
revol_util              0.0
total_acc               0.0
initial_list_status     0.0
application_type        0.0
mort_acc                0.0
pub_rec_bankruptcies    0.0
address                 0.0
dtype: float64
```

In [ ]:

Outlier Detection & Treatment :

In [75]:
```
numerical_data = df.select_dtypes(include='number')
numerical_data
```

Out[75]:

	loan_amnt	int_rate	annual_inc	loan_status	dti	open_acc	pub_rec	revol_bal	revol_util
0	10000.0	11.44	117000.0	0	26.24	16.0	0	36369.0	41.8
1	8000.0	11.99	65000.0	0	22.05	17.0	0	20131.0	53.3
2	15600.0	10.49	43057.0	0	12.79	13.0	0	11987.0	92.2
3	7200.0	6.49	54000.0	0	2.60	6.0	0	5472.0	21.5
4	24375.0	17.27	55000.0	1	33.95	13.0	0	24584.0	69.8
...	...	...	...	...	...	...	...	...	...
396025	10000.0	10.99	40000.0	0	15.63	6.0	0	1990.0	34.3
396026	21000.0	12.29	110000.0	0	21.45	6.0	0	43263.0	95.7
396027	5000.0	9.99	56500.0	0	17.56	15.0	0	32704.0	66.9
396028	21000.0	15.31	64000.0	0	15.88	9.0	0	15704.0	53.8
396029	2000.0	13.61	42996.0	0	8.32	3.0	0	4292.0	91.3

371126 rows × 12 columns

In [76]:
```
num_cols = numerical_data.columns
len(num_cols)
```

Out[76]: 12

In [77]:
```
num_cols = ['loan_amnt', 'int_rate', 'annual_inc',  'dti', 'open_acc', 'revol_bal', 'r
```

In [78]:
```
# def box_plot(col):
#     plt.figure(figsize=(8, 5))
#     sns.boxplot(x=df[col])
#     plt.title('Boxplot')
#     plt.show()

# for col in num_cols:
#     box_plot(col)
```

In [79]:
```
removed_ooutlier = df.copy()
```

In [80]:
```
for col in num_cols:
    mean = removed_ooutlier[col].mean()
    std = removed_ooutlier[col].std()
    upper_limit = mean+3*std
    lower_limit = mean-3*std
    removed_ooutlier = removed_ooutlier[(removed_ooutlier[col]<upper_limit) & (removed
removed_ooutlier.shape
```

Out[80]: (355005, 26)

In [81]:
```
removed_ooutlier
```

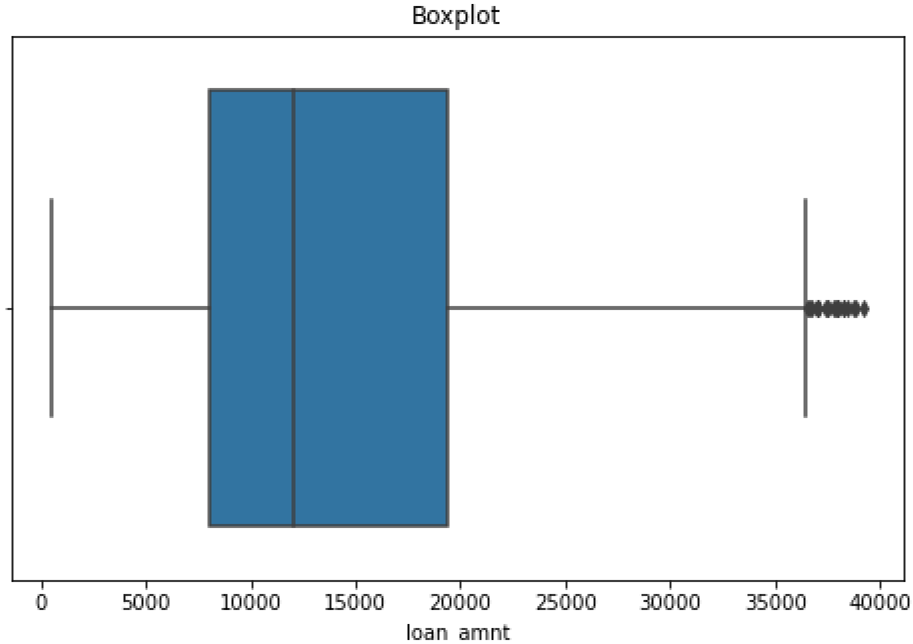
Out[81]:

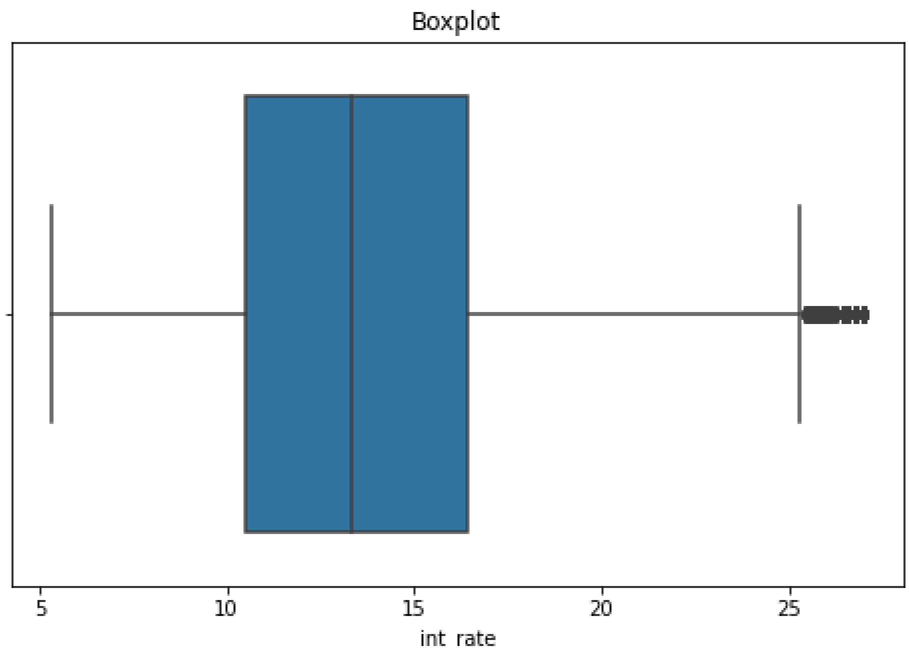
	loan_amnt	term	int_rate	grade	sub_grade	emp_title	emp_length	home_ownership
0	10000.0	36 months	11.44	B	B4	Marketing	10+ years	RENT
1	8000.0	36 months	11.99	B	B5	Credit analyst	4 years	MORTGAGE
2	15600.0	36 months	10.49	B	B3	Statistician	< 1 year	RENT
3	7200.0	36 months	6.49	A	A2	Client Advocate	6 years	RENT
4	24375.0	60 months	17.27	C	C5	Destiny Management Inc.	9 years	MORTGAGE
...	...	...	...	...	...	...	...	...
396025	10000.0	60 months	10.99	B	B4	licensed bankere	2 years	RENT
396026	21000.0	36 months	12.29	C	C1	Agent	5 years	MORTGAGE
396027	5000.0	36 months	9.99	B	B1	City Carrier	10+ years	RENT
396028	21000.0	60 months	15.31	C	C2	Gracon Services, Inc	10+ years	MORTGAGE
396029	2000.0	36 months	13.61	C	C2	Internal Revenue Service	10+ years	RENT

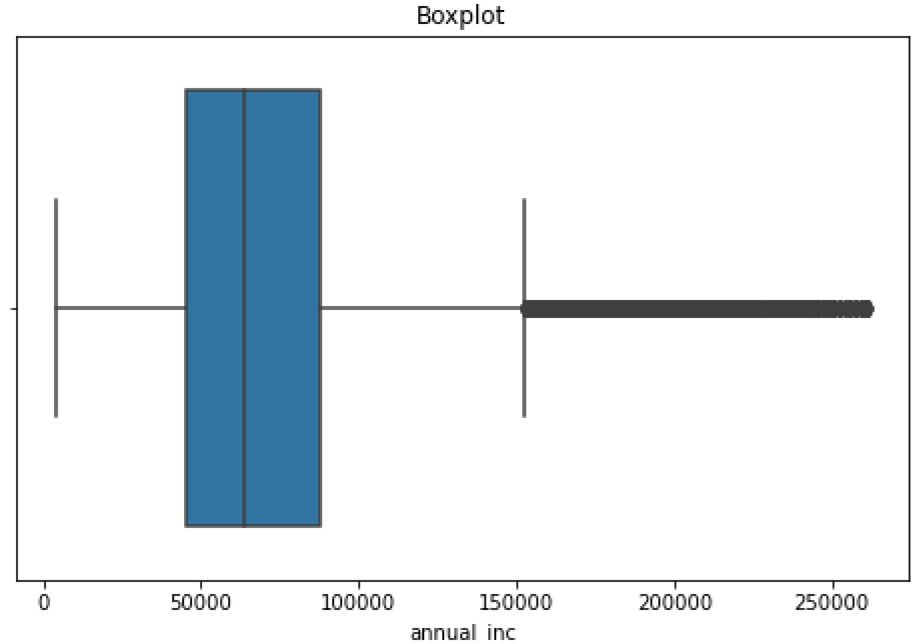
355005 rows × 26 columns

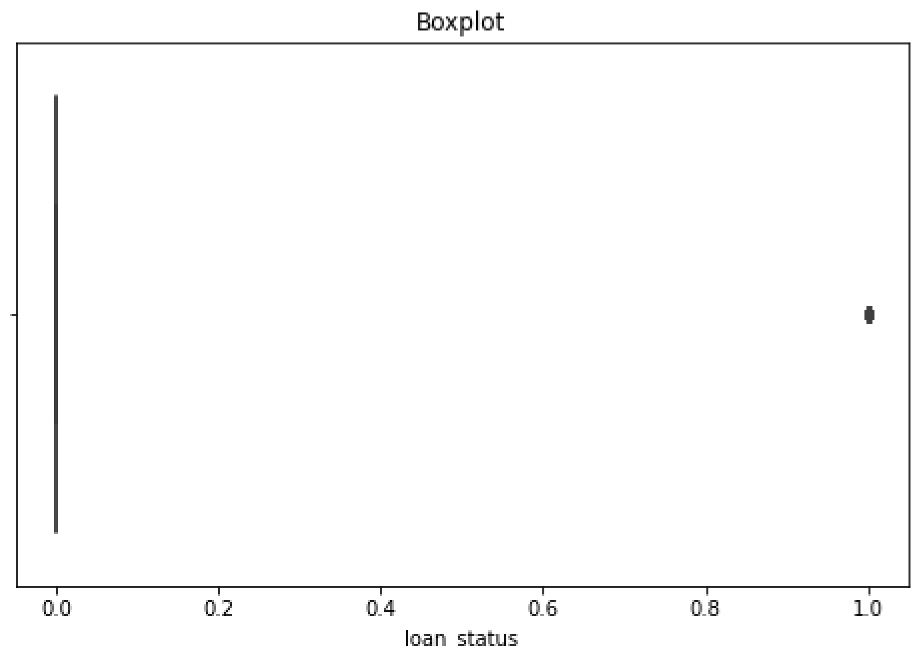
In [82]:

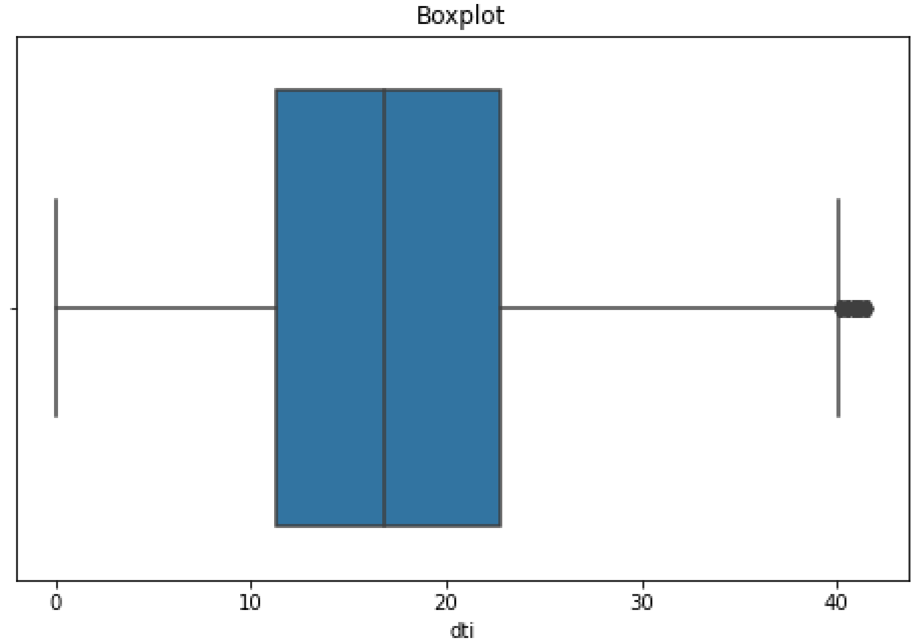
```
numerical_data = removed_ooutlier.select_dtypes(include='number')
numerical_data

num_cols = numerical_data.columns
len(num_cols)
def box_plot(col):
    plt.figure(figsize=(8, 5))
    sns.boxplot(x=removed_ooutlier[col])
    plt.title('Boxplot')
    plt.show()

for col in num_cols:
    box_plot(col)
```

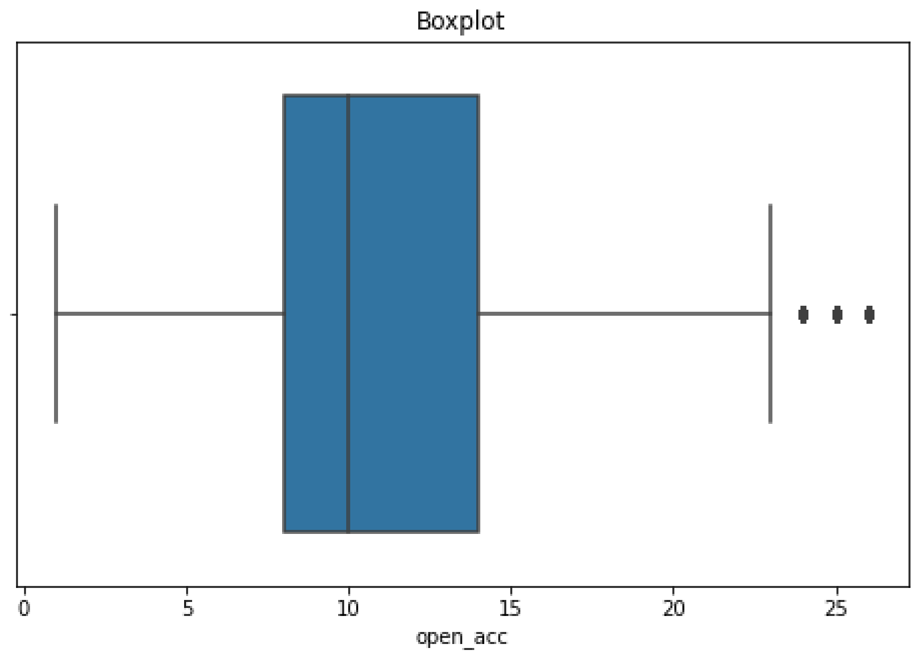

Boxplot
0
5000
10000
15000
20000
25000
30000
35000
40000
loan_amnt

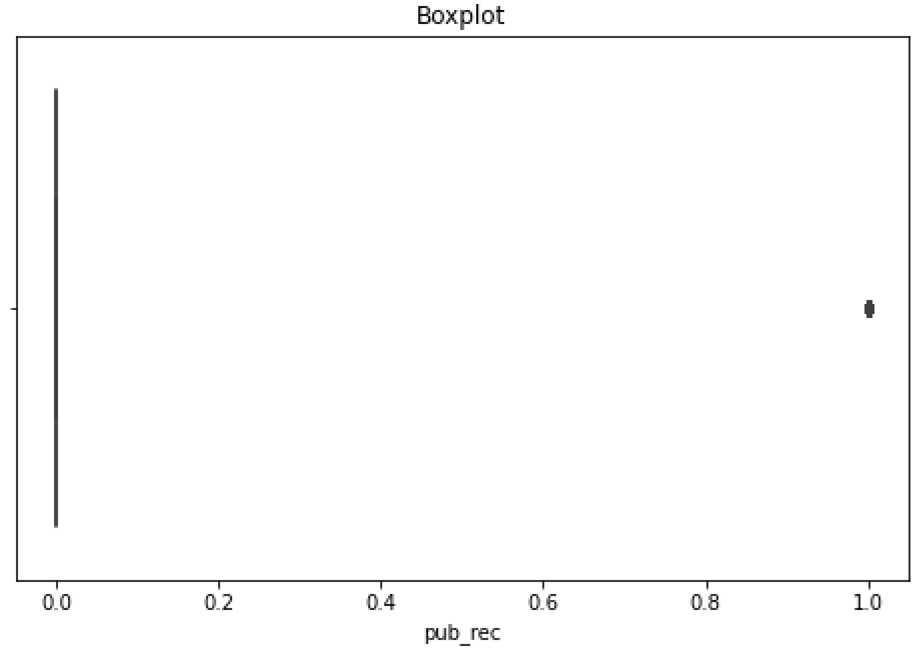

Boxplot
5
10
15
20
25
int_rate

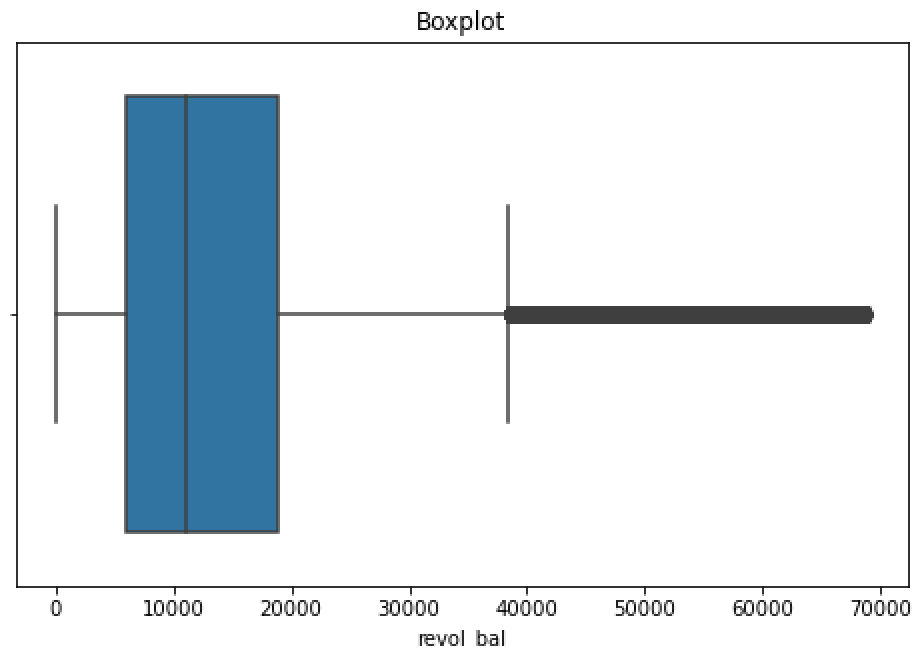

Boxplot
0
50000
100000
150000
200000
250000
annual_inc

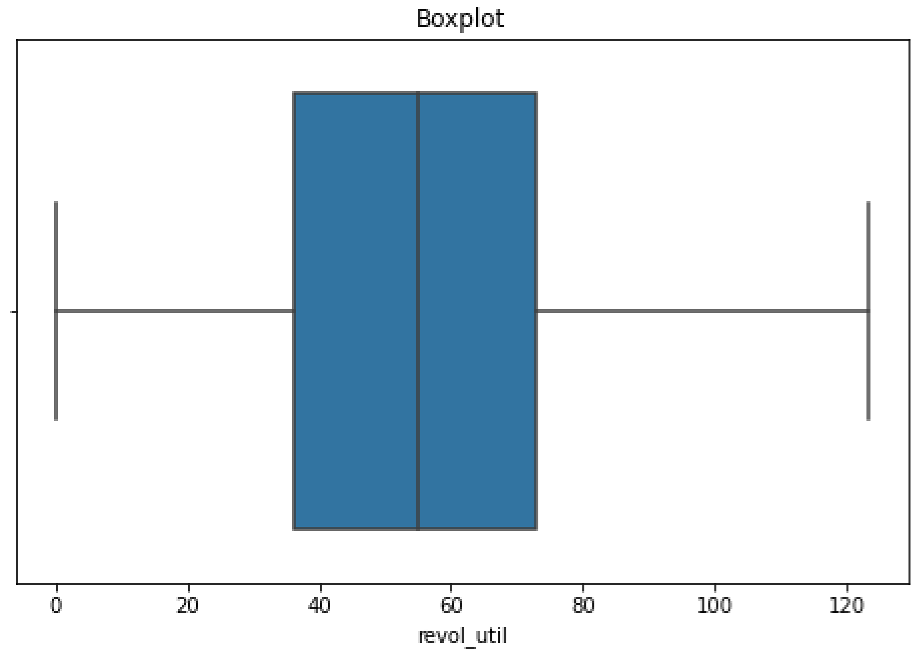

Boxplot
0.0
0.2
0.4
0.6
0.8
1.0
loan_status

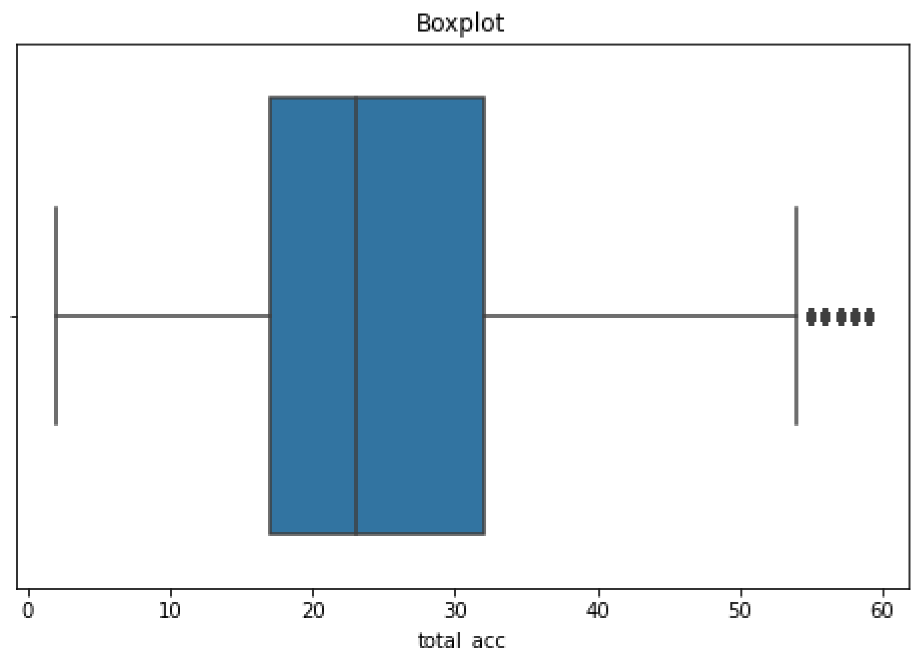

Boxplot
0
10
20
30
40
dti

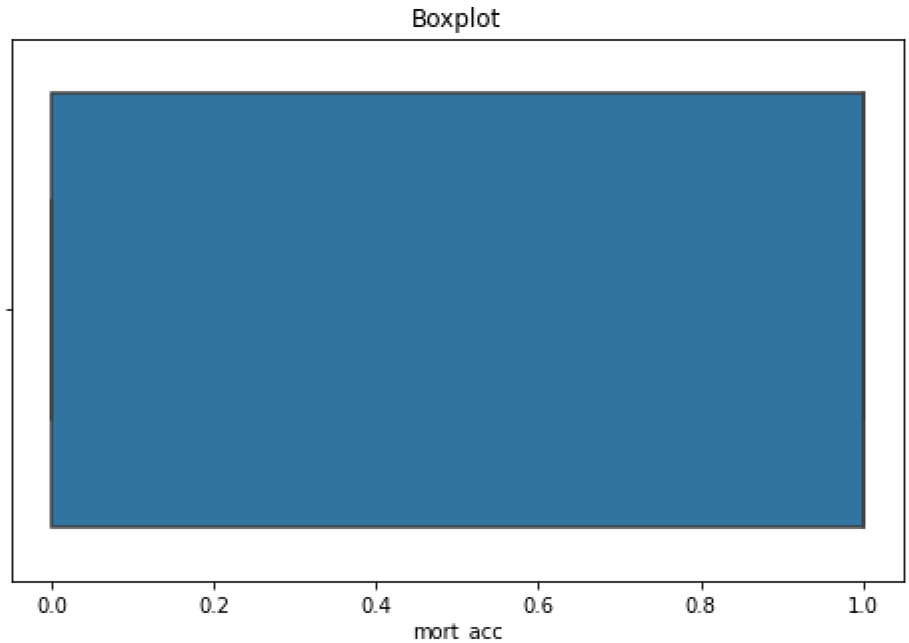

Boxplot
0
5
10
15
20
25
open_acc

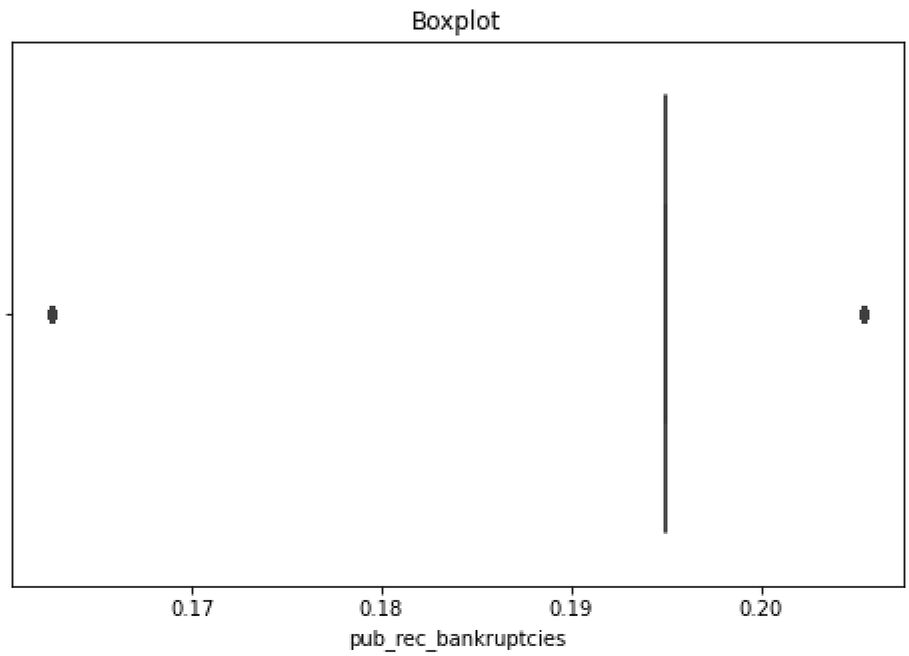

Boxplot
0.0
0.2
0.4
0.6
0.8
1.0
pub_rec


Boxplot
0
10000
20000
30000
40000
50000
60000
70000
revol_bal


Boxplot
0
20
40
60
80
100
120
revol_util


Boxplot
0
10
20
30
40
50
60
total_acc





In [83]:
```
# removed_ooutlier.pub_rec_bankruptcies.value_counts()
```

In [84]:
```
data = removed_ooutlier.copy()
```

Data Preprocessing -

In [ ]:
```

```

In [85]:
```
data['term'] = data.term.map({' 36 months': 36, ' 60 months': 60})
```

In [86]:
```
data['initial_list_status'] = data.initial_list_status.map({'w': 0, 'f': 1})
```

In [87]:
```
data.address.apply(lambda x: x[-5:])
```

Out[87]:
```
0         22690
1         05113
2         05113
3         00813
4         11650
          ...  
396025    30723
396026    05113
396027    70466
396028    29597
396029    48052
Name: address, Length: 355005, dtype: object
```

In [88]:
```
data['zip_code'] = data.address.apply(lambda x: x[-5:])
```

In [89]:
```
data['zip_code'].value_counts(normalize=True)*100
```

Out[89]:
```
70466    14.378671
30723    14.282616
22690    14.269095
48052    14.125153
00813    11.611949
29597    11.537302
05113    11.517021
93700     2.773764
11650     2.771229
86630     2.733201
Name: zip_code, dtype: float64
```

In [ ]:

In [90]:
```
# dropping iirrelevant features :
data.drop(columns=['issue_d',  'sub_grade','address', 'earliest_cr_line', 'emp_length'
```

In [91]:
```
data.drop("title",inplace=True,axis = 1)
```

In [92]:
```
data["emp_title"].nunique() # target_encoding
```

Out[92]:
```
167202
```

In [ ]:

In [93]:
```
data.columns
```

Out[93]:
```
Index(['loan_amnt', 'term', 'int_rate', 'grade', 'emp_title', 'home_ownership', 'annu
al_inc', 'verification_status', 'loan_status', 'purpose', 'dti', 'open_acc', 'pub_re
c', 'revol_bal', 'revol_util', 'total_acc', 'initial_list_status', 'application_typ
e', 'mort_acc', 'pub_rec_bankruptcies', 'zip_code'], dtype='object')
```

Target encoding for employee title :

In [94]:
```
from category_encoders import TargetEncoder
TE = TargetEncoder()

data["emp_title"] = TE.fit_transform(data["emp_title"],data["loan_status"])
```

In [ ]:

In [95]:
```
# data["emp_title"] = TE.fit_transform(data["emp_title"],data["loan_status"])
```

In [96]:
```
data
```

Out[96]:

	loan_amnt	term	int_rate	grade	emp_title	home_ownership	annual_inc	verification_status
0	10000.0	36	11.44	B	0.243902	RENT	117000.0	Not Verified
1	8000.0	36	11.99	B	0.316535	MORTGAGE	65000.0	Not Verified
2	15600.0	36	10.49	B	0.199999	RENT	43057.0	Source Verified
3	7200.0	36	6.49	A	0.192411	RENT	54000.0	Not Verified
4	24375.0	60	17.27	C	0.192411	MORTGAGE	55000.0	Verified
...	...	...	...	...	...	...	...	...
396025	10000.0	60	10.99	B	0.192411	RENT	40000.0	Source Verified
396026	21000.0	36	12.29	C	0.208092	MORTGAGE	110000.0	Source Verified
396027	5000.0	36	9.99	B	0.272727	RENT	56500.0	Verified
396028	21000.0	60	15.31	C	0.192411	MORTGAGE	64000.0	Verified
396029	2000.0	36	13.61	C	0.223881	RENT	42996.0	Verified

355005 rows × 21 columns

In [ ]:

One-hot-encoding for other categorical data :

In [97]:
```
dummies = ["grade","home_ownership","verification_status","purpose","application_type"
```

In [98]:
```
data = pd.get_dummies(data,columns=dummies,drop_first=True)
```

In [99]:
```
data.sample(5)
```

Out[99]:

	loan_amnt	term	int_rate	emp_title	annual_inc	loan_status	dti	open_acc	pub_rec	revo
160843	10500.0	36	14.65	0.192411	70000.0	1	22.10	10.0	0	14(
151293	7500.0	36	11.99	0.192411	31200.0	0	22.62	9.0	0	4(
217492	12000.0	36	12.99	0.192411	32000.0	0	12.68	9.0	0	12!
380999	14000.0	36	14.65	0.051747	105000.0	0	3.47	6.0	0	9(
10877	21000.0	36	7.90	0.192411	90000.0	0	7.85	9.0	0	14

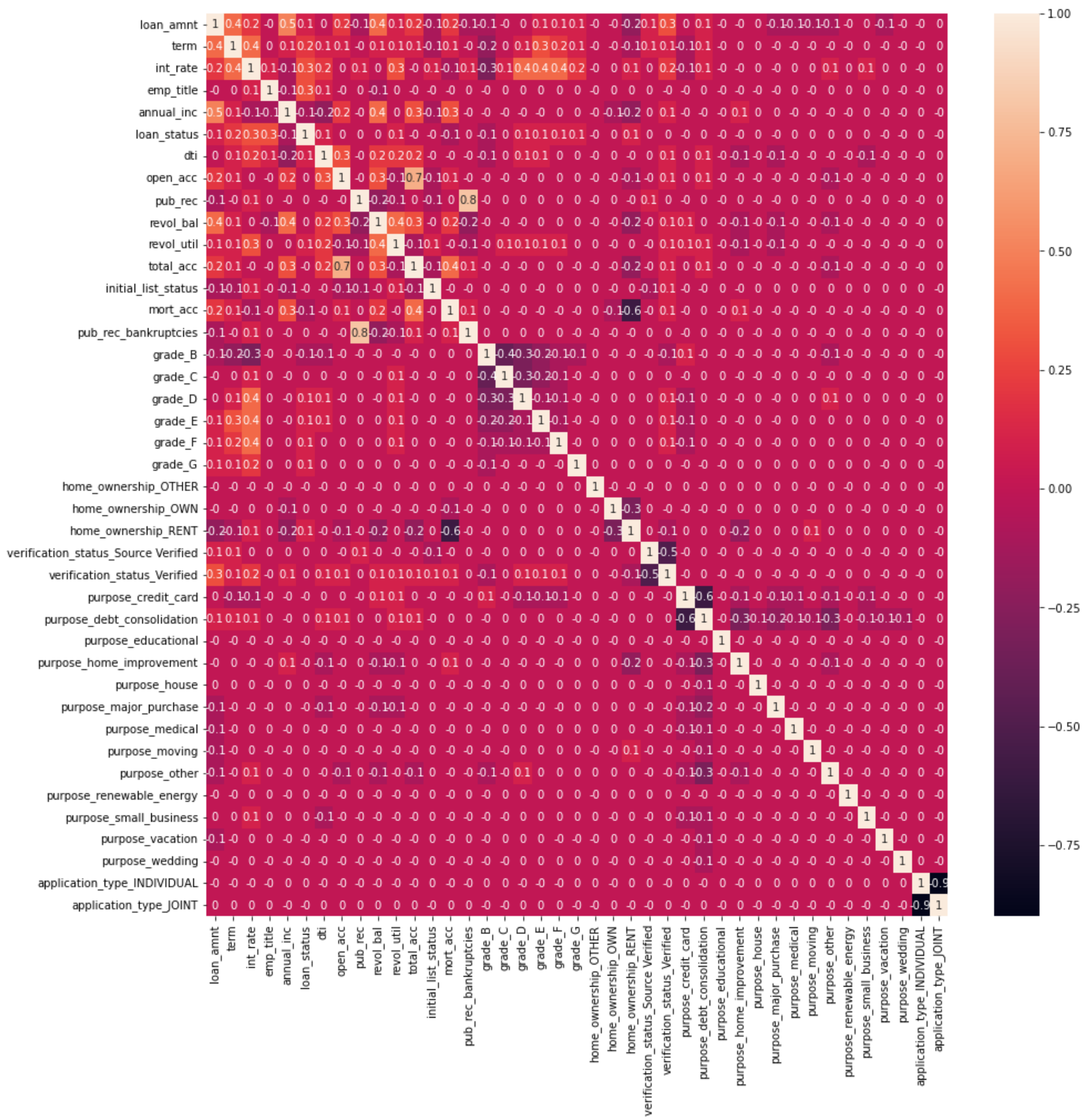
In [ ]:

Feature Importance using Heatmap and correations :

In [100…

```
plt.figure(figsize=(15,15))
sns.heatmap(data.corr().round(1),annot=True,)
```

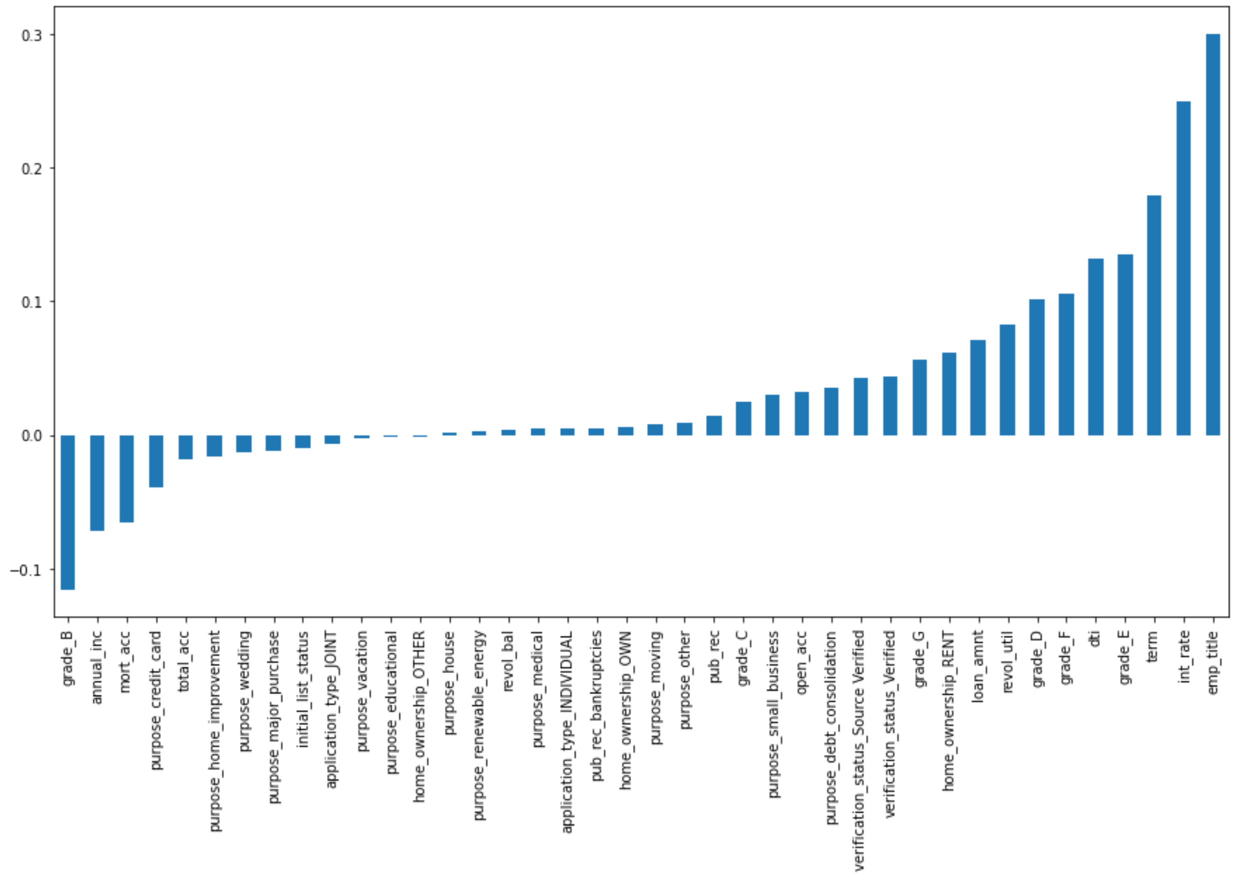
Out[100]: <AxesSubplot:>



In [101…

```
plt.figure(figsize=(15,8))

data.corr()["loan_status"].sort_values()[:-1].plot(kind = "bar")
```

Out[101]: <AxesSubplot:>



In [ ]:

Data preparation for modeling:

In [102…

```
X = data.drop("loan_status",axis = 1)
y = data["loan_status"]
```

In [103…

```
X.shape,y.shape
```

Out[103]:

```
((355005, 41), (355005,))
```

train-test-Split :

In [104…

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

In [105…

```
X_train , X_test,  y_train , y_test = train_test_split(X,y,
                                                       test_size=0.3,
                                                       random_state=42)
```

In [106…

```
X_train.shape,X_test.shape
```

Out[106]:

```
((248503, 41), (106502, 41))
```

In [ ]:

In [107…
```
X_train_non_scalled = X_train.copy()
```

In [108…
```
X_test_non_scalled = X_test.copy()
```

Scaling the data :

In [109…
```
from sklearn.preprocessing import StandardScaler
```

In [110…
```
scaler = StandardScaler()
scaler.fit(X_train)
```

Out[110]:
```
▾ StandardScaler
StandardScaler()
```

In [111…
```
X_train = scaler.transform(X_train)
```

In [112…
```
X_test = scaler.transform(X_test)
```

In [ ]:

In [ ]:

In [ ]:

Logistic Regression :

In [113…
```
from sklearn.linear_model import LogisticRegression
logistic_reg_model = LogisticRegression(
    penalty='l2',         # L2 - ridge regularisation
    dual=False,
    tol=0.0001,
    C=1.0,              # 1/lambda :
    fit_intercept=True,
    intercept_scaling=1,
    class_weight=None,
    random_state=None,
    solver='lbfgs',
    max_iter=1000,           # 1000 iterations for learning
    multi_class='auto',
    verbose=0,
    warm_start=False,
    n_jobs=None,
    l1_ratio=None,)
```

In [114…
```
logistic_reg_model.fit(X_train,y_train)
```

Out[114]:
```
▾        LogisticRegression
LogisticRegression(max_iter=1000)
```

Accsuracy results of Logistic Regression :

In [115…
```
logistic_reg_model.score(X_train ,y_train)
```
Out[115]:
```
0.8512170879224799
```

In [116…
```
logistic_reg_model.score(X_test ,y_test)
```
Out[116]:
```
0.8486319505736981
```

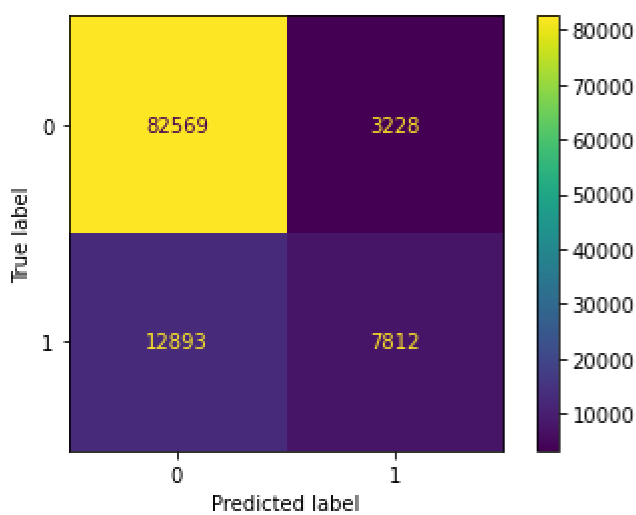
In [117…
```
# Model is predicting 84% accurate results :
```

Fetching more detailed results : using Confusion Matrix :

In [118…
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
```

In [119…
```
y_predicted = logistic_reg_model.predict(X_test)
confusion_matrix(y_test ,y_predicted )
```
Out[119]:
```
array([[82569,  3228],
       [12893,  7812]], dtype=int64)
```

In [120…
```
ConfusionMatrixDisplay(confusion_matrix(y_test ,y_predicted ),
                       display_labels=[0,1]).plot()

plt.show()
```



In [121…
```
from sklearn.metrics import f1_score,precision_score,recall_score,fbeta_score
```

In [122…
```
fbeta_score(y_true = y_test,
    y_pred = y_predicted,
    beta = 0.5)
```
Out[122]:
```
0.6021737454713635
```

In [123…
```
precision_score(y_true = y_test,
    y_pred = y_predicted)
```

Out[123]: 0.7076086956521739

In [124…
```
recall_score(y_true = y_test,
    y_pred = y_predicted)
```

Out[124]: 0.3773001690412944

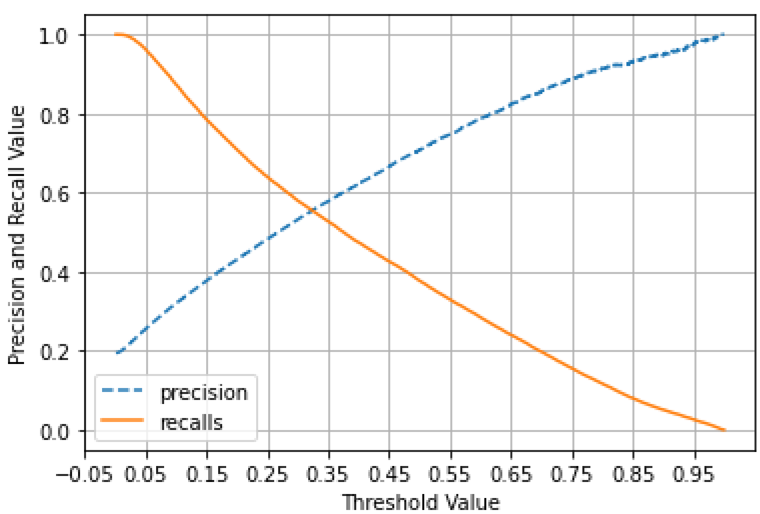
In [125…
```
from sklearn.metrics import classification_report
```

In [126…
```
print(classification_report(y_test, y_predicted))
```

```
              precision    recall  f1-score   support

           0       0.86      0.96      0.91     85797
           1       0.71      0.38      0.49     20705

    accuracy                           0.85    106502
   macro avg       0.79      0.67      0.70    106502
weighted avg       0.83      0.85      0.83    106502
```

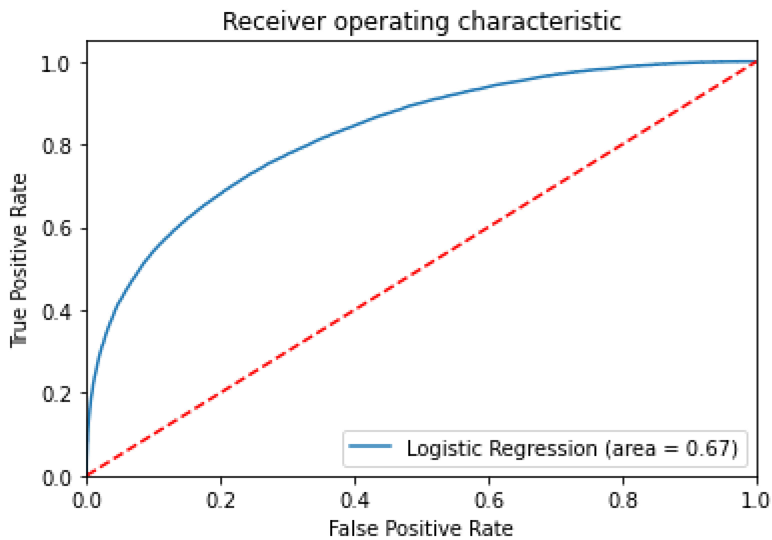
In [127…
```
from sklearn.metrics import roc_auc_score,roc_curve
from sklearn.metrics import precision_recall_curve
```

In [128…
```
def precision_recall_curve_plot(y_test, pred_proba_c1):
    precisions, recalls, thresholds = precision_recall_curve(y_test, pred_proba_c1)

    threshold_boundary = thresholds.shape[0]
    # plot precision
    plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='--', label='prec
    # plot recall
    plt.plot(thresholds, recalls[0:threshold_boundary], label='recalls')

    start, end = plt.xlim()
    plt.xticks(np.round(np.arange(start, end, 0.1), 2))

    plt.xlabel('Threshold Value'); plt.ylabel('Precision and Recall Value')
    plt.legend(); plt.grid()
    plt.show()

precision_recall_curve_plot(y_test, logistic_reg_model.predict_proba(X_test)[:,1])
```



In [129…

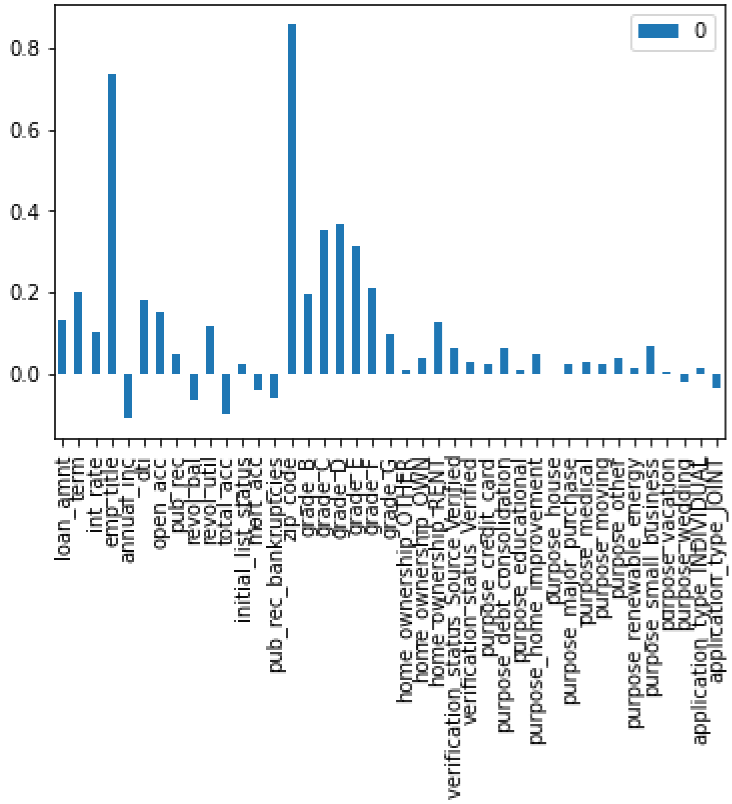
```
logit_roc_auc = roc_auc_score(y_test, logistic_reg_model.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, logistic_reg_model.predict_proba(X_test)[:,1]
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 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 [130…

```
plt.figure(figsize=(15,8))

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

<Figure size 1080x576 with 0 Axes>



Checking for Multicollinearity :

In [131…

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [132…

```
vifs = []

for i in range(X_train.shape[1]):

    vifs.append((variance_inflation_factor(exog = X_train,
                                           exog_idx=i)))
pd.DataFrame({ "coef_name : " : X.columns ,
              "vif : ": np.around(vifs,2)})
```

Out[132]:

	coef_name :	vif :
0	loan_amnt	1.89
1	term	1.55
2	int_rate	11.71
3	emp_title	1.04
4	annual_inc	1.71
5	dti	1.44
6	open_acc	2.04
7	pub_rec	3.03
8	revol_bal	1.87
9	revol_util	1.60
10	total_acc	2.06
11	initial_list_status	1.07
12	mort_acc	1.88
13	pub_rec_bankruptcies	2.99
14	zip_code	1.02
15	grade_B	3.80
16	grade_C	7.52
17	grade_D	9.70
18	grade_E	8.53
19	grade_F	5.61
20	grade_G	2.19
21	home_ownership_OTHER	1.00
22	home_ownership_OWN	1.18
23	home_ownership_RENT	1.84
24	verification_status_Source Verified	1.45
25	verification_status_Verified	1.59
26	purpose_credit_card	14.94
27	purpose_debt_consolidation	20.90
28	purpose_educational	1.05
29	purpose_home_improvement	5.55
30	purpose_house	1.47
31	purpose_major_purchase	2.79
32	purpose_medical	1.84
33	purpose_moving	1.59

	coef_name :	vif :
34	purpose_other	5.19
35	purpose_renewable_energy	1.07
36	purpose_small_business	2.08
37	purpose_vacation	1.50
38	purpose_wedding	1.40
39	application_type_INDIVIDUAL	4.94
40	application_type_JOINT	4.94

In [ ]:

In [ ]:

In [ ]:

Handling Data Imbalance :

In [133…

```
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=42)

X_smote, y_smote = sm.fit_resample(X_train, y_train.ravel())
```

In [134…

```
## After over sampling :
X_smote.shape, y_smote.shape
```

Out[134]:

```
((401802, 41), (401802,))
```

LogisticRegression

In [137…

```
def apply_logistic_regression(X_train  ,y_train):

    from sklearn.linear_model import LogisticRegression
    logistic_reg_model = LogisticRegression(
        penalty='l2',         # L2 - ridge regularisation
        dual=False,
        tol=0.0001,
        C=1.0,              # 1/lambda :
        fit_intercept=True,
        intercept_scaling=1,
        class_weight=None,
        random_state=None,
        solver='lbfgs',
        max_iter=1000,          # 1000 iterations for learning
        multi_class='auto',
        verbose=0,
        warm_start=False,
        n_jobs=None,
        l1_ratio=None,)
```

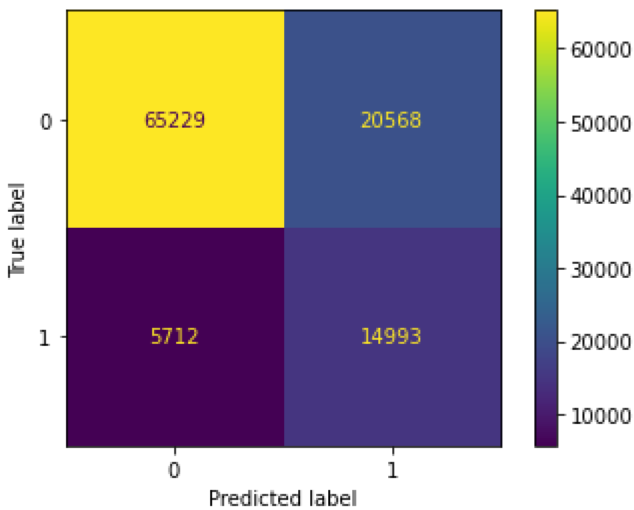
```
logistic_reg_model.fit(X_train,y_train)
print("LR train score:",logistic_reg_model.score(X_train ,y_train))
print("LR test score:",logistic_reg_model.score(X_test ,y_test))

from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

y_predicted = logistic_reg_model.predict(X_test)

print()
print("Confusion Matrix: ")
print(confusion_matrix(y_test ,y_predicted ))

ConfusionMatrixDisplay(confusion_matrix(y_test ,y_predicted ),
                      display_labels=[0,1]).plot()

plt.show()

from sklearn.metrics import f1_score,precision_score,recall_score,fbeta_score
from sklearn.metrics import classification_report

print("fbeta score : beta : 0.5")
print(fbeta_score(y_true = y_test,
    y_pred = y_predicted,
    beta = 0.5))



print(classification_report(y_test, y_predicted))


from sklearn.metrics import roc_auc_score,roc_curve
from sklearn.metrics import precision_recall_curve

print(precision_recall_curve_plot(y_test, logistic_reg_model.predict_proba(X_test)
plt.show()



def custom_predict(X, threshold):
        probs = logistic_reg_model.predict_proba(X)
        return (probs[:, 1] > threshold).astype(int)

# print(model.predict_proba(X_test))
threshold = 0.75

new_preds = custom_predict(X=X_test, threshold=threshold)


print(f"Precision at theshold {threshold} is : ",precision_score(y_test,new_preds)


print()
print()




print()
```

```
    print("fbeta score : beta : 0.5",fbeta_score(y_true = y_test, y_pred = new_preds,
                                                    beta = 0.5))
    print(classification_report(y_test, new_preds))



    print("Feature Importance : ")
    plt.figure(figsize=(15,8))

    pd.DataFrame(columns=X.columns,data= logistic_reg_model.coef_).T.plot(kind = "bar'
    plt.show()
```

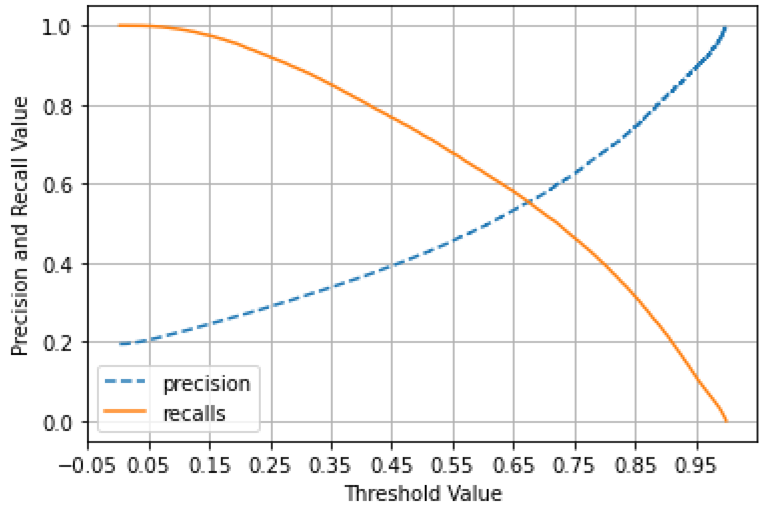
In [138…

```
apply_logistic_regression(X_smote,y_smote)
```

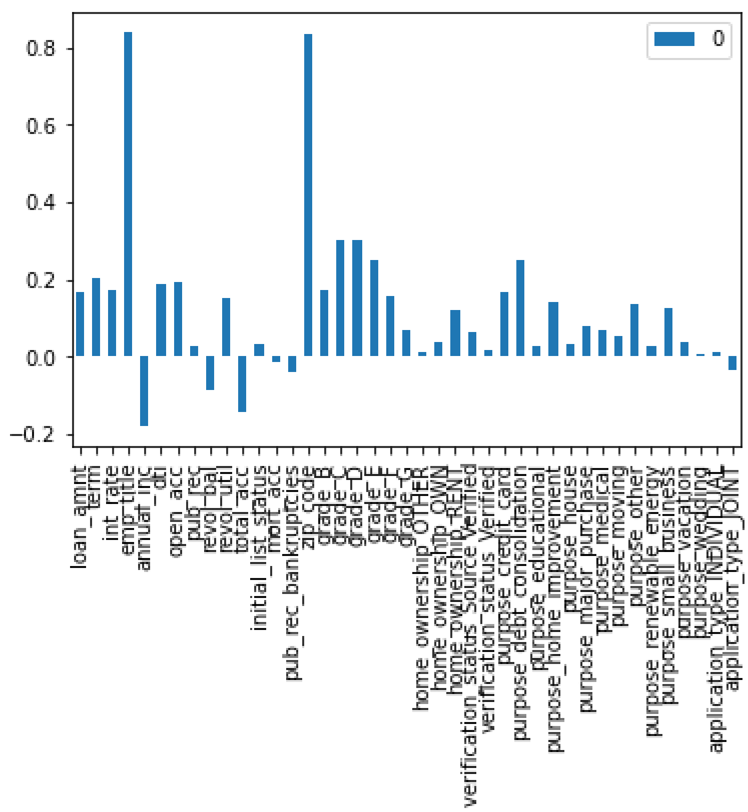
```
LR train score: 0.7491127470744297
LR test score: 0.7532440705338866

Confusion Matrix:
[[65229 20568]
 [ 5712 14993]]
```




```
fbeta score : beta : 0.5
0.46005191808479956
              precision    recall  f1-score   support

           0       0.92      0.76      0.83     85797
           1       0.42      0.72      0.53     20705

    accuracy                           0.75    106502
   macro avg       0.67      0.74      0.68    106502
weighted avg       0.82      0.75      0.77    106502
```




```
None
Precision at theshold 0.75 is :  0.624650406504065



fbeta score : beta : 0.5 0.5841493826409586
              precision    recall  f1-score   support

           0       0.88      0.93      0.90     85797
           1       0.62      0.46      0.53     20705

    accuracy                           0.84    106502
   macro avg       0.75      0.70      0.72    106502
weighted avg       0.83      0.84      0.83    106502

Feature Importance :
<Figure size 1080x576 with 0 Axes>
```



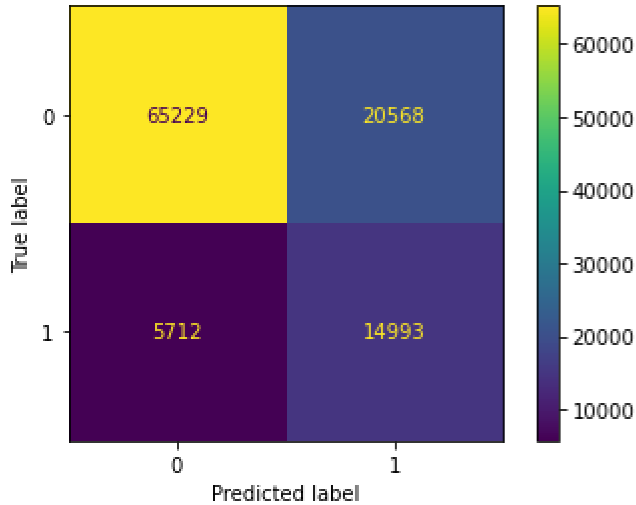
In [ ]:

In [140…

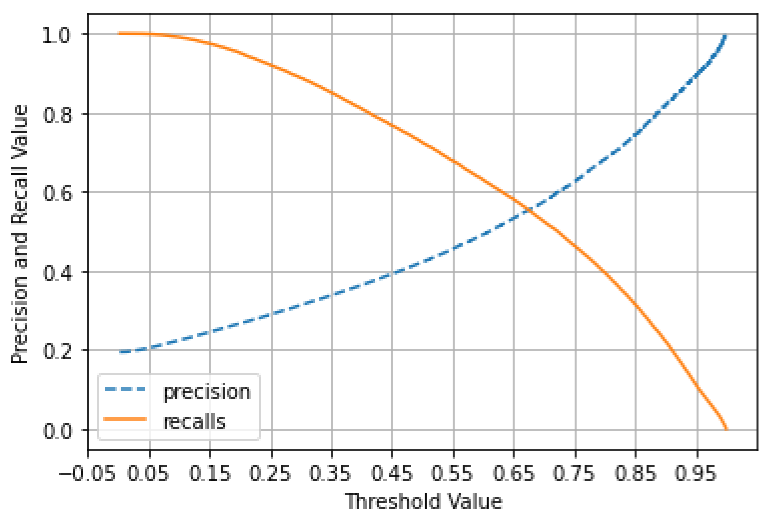
```
from sklearn.linear_model import LogisticRegression
logistic_reg_model = LogisticRegression(
    penalty='l2',         # L2 - ridge regularisation
    dual=False,
    tol=0.0001,
    C=1.0,             # 1/lambda :
    fit_intercept=True,
    intercept_scaling=1,
    class_weight=None,
    random_state=None,
    solver='lbfgs',
    max_iter=1000,          # 1000 iterations for learning
    multi_class='auto',
    verbose=0,
    warm_start=False,
    n_jobs=None,
    l1_ratio=None,)

logistic_reg_model.fit(X_smote,y_smote)
print("LR train score:",logistic_reg_model.score(X_smote,y_smote))
print("LR test score:",logistic_reg_model.score(X_test ,y_test))

from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

y_predicted = logistic_reg_model.predict(X_test)

print()
print("Confusion Matrix: ")
print(confusion_matrix(y_test ,y_predicted ))

ConfusionMatrixDisplay(confusion_matrix(y_test ,y_predicted ),
                      display_labels=[0,1]).plot()

plt.show()

from sklearn.metrics import f1_score,precision_score,recall_score,fbeta_score
from sklearn.metrics import classification_report

print("fbeta score : beta : 0.5")
print(fbeta_score(y_true = y_test,
    y_pred = y_predicted,
    beta = 0.5))



print(classification_report(y_test, y_predicted))


from sklearn.metrics import roc_auc_score,roc_curve
from sklearn.metrics import precision_recall_curve

print(precision_recall_curve_plot(y_test, logistic_reg_model.predict_proba(X_test)[:,1
plt.show()
```

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

# print(model.predict_proba(X_test))
threshold = 0.60

new_preds = custom_predict(X=X_test, threshold=threshold)


print(f"Precision at theshold {threshold} is : ",precision_score(y_test,new_preds))


print()
print()





print()
print("fbeta score : beta : 0.5",fbeta_score(y_true = y_test, y_pred = new_preds,
                                            beta = 0.5))
print(classification_report(y_test, new_preds))

logit_roc_auc = roc_auc_score(y_test, logistic_reg_model.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, logistic_reg_model.predict_proba(X_test)[:,1]
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
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()

print("Feature Importance : ")
plt.figure(figsize=(15,8))

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

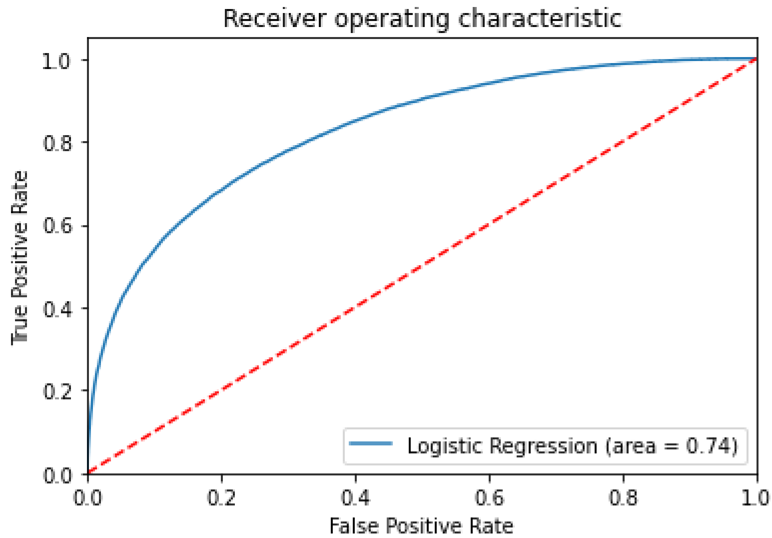
```
LR train score: 0.7491127470744297
LR test score: 0.7532440705338866

Confusion Matrix:
[[65229 20568]
 [ 5712 14993]]
```



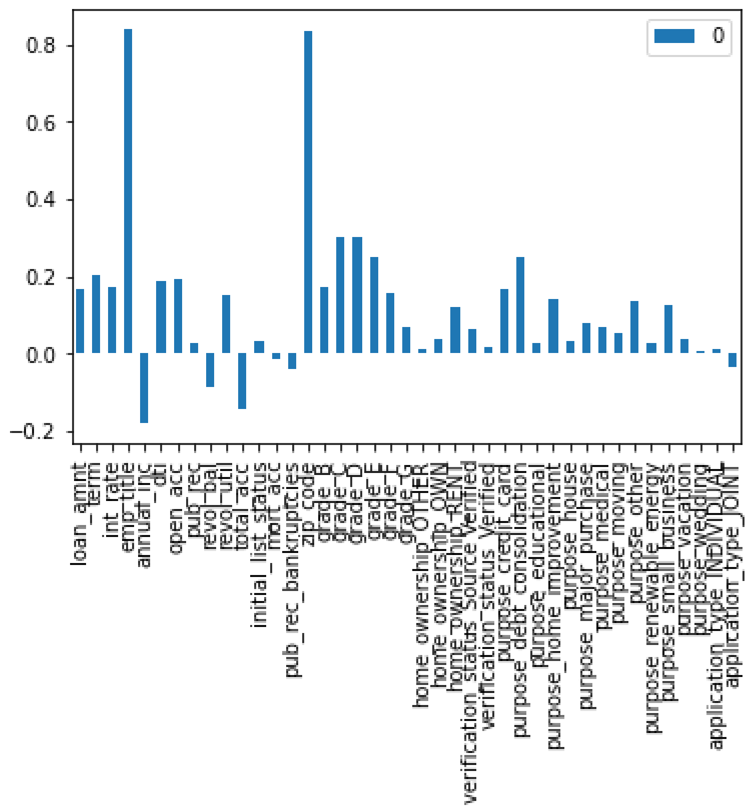

```
fbeta score : beta : 0.5
0.46005191808479956
              precision    recall  f1-score   support

           0       0.92      0.76      0.83     85797
           1       0.42      0.72      0.53     20705

    accuracy                           0.75    106502
   macro avg       0.67      0.74      0.68    106502
weighted avg       0.82      0.75      0.77    106502
```




```
None
Precision at theshold 0.6 is :  0.49157779703809773



fbeta score : beta : 0.5 0.5141778278795142
              precision    recall  f1-score   support

           0       0.90      0.84      0.87     85797
           1       0.49      0.63      0.55     20705

    accuracy                           0.80    106502
   macro avg       0.70      0.74      0.71    106502
weighted avg       0.82      0.80      0.81    106502
```




```
Feature Importance :
<Figure size 1080x576 with 0 Axes>
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

Data Analysis report :

- Interest Rate Mean and Median , Loan Amount distribution / median of Loan amount is higher for Borrowers who are more likely to be defaulter.
- Borrowers having Loan Grades E ,F, G , have more probability of default.
- G grade has the highest conditional probability of having defaulter.
- 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.
- Most of the borrowers take loans for dept-consolidation and credit card payoffs. Probability of defaulters is higher in the small_business owner borrowers.
- debt-to-income ratio is higher for defaulters.
- Number of derogatory public records increases , the probability of borrowers declared as defaulters also increases
- Application type Direct-Pay has higher probability of defaulter borrowers than individual and joint.

Report : Insights and Model Results :

- Since , NBFCs are willing to take risk giving loans to borrowers having low credit grades (who have high probability of default) , as far as they do not have bankruptcy record present in credit repost , copany can affort to give loans, and maximise their earning by receiving high interest from such borrowers.
- so this reason , we have done feature engineering steps such as ,
  - borrowers having more than 0 public records , mortgage accounts or public recorded bankruptcy present in report, have converted those feature values to 1 (means they are more likely to become a defaulter).
- Our goal in Model building was to
  - minimise , below metrics :
    - incorrectly classified as defaulter : FP
    - incorretly classified as non-defaulter : FN

      <> Minimise the False Positive,

    - means we dont want to say defaulter to a borrower who is not really a deaulter. that means we will lose the opportunity (minimise False Positive) (Dont loose opportunity)

<> Minimise the False Negatives:

- Means we dont want to declare a borrower a non-defaulter, who is actually more likely to become a defaulter. Thats a risk on company giving loans to such borrower.
- But since, as NBFCs are willing to take risk , we can be a little liberal about this point.

Hence the Precision is more important metric SO we have focused on Maximising Precision metric and also F0.5 score.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: