# Context:

A Non-Banking Finance Company like LoanTap is an online platform committed to delivering customized loanproducts to millennials. They innovate in an otherwise dull loan segment, to deliver instant, fl exible loans on consumer friendly terms tosalaried professionals and businessmen.

The data science team is building an underwriting layer to determine the creditworthiness of MSMEs as well asindividuals. Company deploys formal credit to salaried individuals and businesses 4 main fi nancial 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, whatshould 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 isimportant as we can lose out on an opportunity to fi nance 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’tdisburse 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 creditdepartment reduces the loan amount, then it will be refl ected 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 grade6. 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 oneyear and 10 means ten or more years.

9.home_ownership : The home ownership status provided by the borrower during registration or obtained fromthe credit report.

10.annual_inc : The self-reported annual income provided by the borrower during registration.

11.verifi cation_status : Indicates if income was verifi ed by Institution, not verifi ed, 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

Context: 16.dti : A ratio calculated using the borrower’s total monthly debt payments on the total debt obligations, excludingmortgage 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 availablerevolving 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

In [4]:
```
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 [5]:
```
LTDF = pd.read_csv(r"H:\Scaler\MACHINE LEARNING INTRO\Loan Tap Project\d2beiqkhq929f0.cloudfront.net_public_assets_assets_000_003
```

In [6]:
```
LTDF.head()
```

Out[6]:

| | loan_amnt | term | int_rate | installment | grade | sub_grade | emp_title | emp_length | home_ownership | annual_inc | verification_status | issue_d | loan_status |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 10000.0 | 36 months | 11.44 | 329.48 | B | B4 | Marketing | 10+ years | RENT | 117000.0 | Not Verified | Jan-2015 | Fully Paid |
| **1** | 8000.0 | 36 months | 11.99 | 265.68 | B | B5 | Credit analyst | 4 years | MORTGAGE | 65000.0 | Not Verified | Jan-2015 | Fully Paid |
| **2** | 15600.0 | 36 months | 10.49 | 506.97 | B | B3 | Statistician | < 1 year | RENT | 43057.0 | Source Verified | Jan-2015 | Fully Paid |
| **3** | 7200.0 | 36 months | 6.49 | 220.65 | A | A2 | Client Advocate | 6 years | RENT | 54000.0 | Not Verified | Nov-2014 | Fully Paid |
| **4** | 24375.0 | 60 months | 17.27 | 609.33 | C | C5 | Destiny Management Inc. | 9 years | MORTGAGE | 55000.0 | Verified | Apr-2013 | Charged Off |

In [7]:
```
LTDF.shape
```

Out[7]: (396030, 27)

In [13]:
```
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 = False)
    missingDF  = pd.concat([total_missing_df, percentage_missing_df], axis=1, keys=['Total','Percent'])
    return missingDF

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

Out[13]:

| | 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 [15]:
```
(LTDF.isna().sum()/LTDF.shape[0])*100
```

Out[15]:
```
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 [16]:
```
LTDF.describe().round(1)
```

Out[16]:

| | loan_amnt | int_rate | installment | annual_inc | dti | open_acc | pub_rec | revol_bal | revol_util | total_acc | mort_acc | pub_rec_bankruptcies |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 396030.0 | 396030.0 | 396030.0 | 396030.0 | 396030.0 | 396030.0 | 396030.0 | 396030.0 | 395754.0 | 396030.0 | 358235.0 | 395495.0 |
| **mean** | 14113.9 | 13.6 | 431.8 | 74203.2 | 17.4 | 11.3 | 0.2 | 15844.5 | 53.8 | 25.4 | 1.8 | 0.1 |
| **std** | 8357.4 | 4.5 | 250.7 | 61637.6 | 18.0 | 5.1 | 0.5 | 20591.8 | 24.5 | 11.9 | 2.1 | 0.4 |
| **min** | 500.0 | 5.3 | 16.1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 0.0 | 0.0 |
| **25%** | 8000.0 | 10.5 | 250.3 | 45000.0 | 11.3 | 8.0 | 0.0 | 6025.0 | 35.8 | 17.0 | 0.0 | 0.0 |
| **50%** | 12000.0 | 13.3 | 375.4 | 64000.0 | 16.9 | 10.0 | 0.0 | 11181.0 | 54.8 | 24.0 | 1.0 | 0.0 |
| **75%** | 20000.0 | 16.5 | 567.3 | 90000.0 | 23.0 | 14.0 | 0.0 | 19620.0 | 72.9 | 32.0 | 3.0 | 0.0 |
| **max** | 40000.0 | 31.0 | 1533.8 | 8706582.0 | 9999.0 | 90.0 | 86.0 | 1743266.0 | 892.3 | 151.0 | 34.0 | 8.0 |

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

In [17]:
```
LTDF.nunique()
```

Out[17]:
```
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 [18]:
```
LTDF.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 [19]:
```
columns_type = LTDF.dtypes
```

In [20]:
```
columns_type[columns_type=="object"]
```

Out[20]:
```
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 [21]:
```
LTDF.describe(include="object")
```

Out[21]:

| | term | grade | sub_grade | emp_title | emp_length | home_ownership | verification_status | issue_d | loan_status | purpose | title | earliest_c |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 396030 | 396030 | 396030 | 373103 | 377729 | 396030 | 396030 | 396030 | 396030 | 396030 | 394275 | 3 |
| unique | 2 | 7 | 35 | 173105 | 11 | 6 | 3 | 115 | 2 | 14 | 48817 | |
| top | 36 months | B | B3 | Teacher | 10+ years | MORTGAGE | Verified | Oct-2014 | Fully Paid | debt_consolidation | Debt consolidation | Oc |
| freq | 302005 | 116018 | 26655 | 4389 | 126041 | 198348 | 139563 | 14846 | 318357 | 234507 | 152472 | |

In [22]:
```
len(columns_type[columns_type=="object"])
```

Out[22]: 15

In [23]:
```
26-11
```

Out[23]: 15

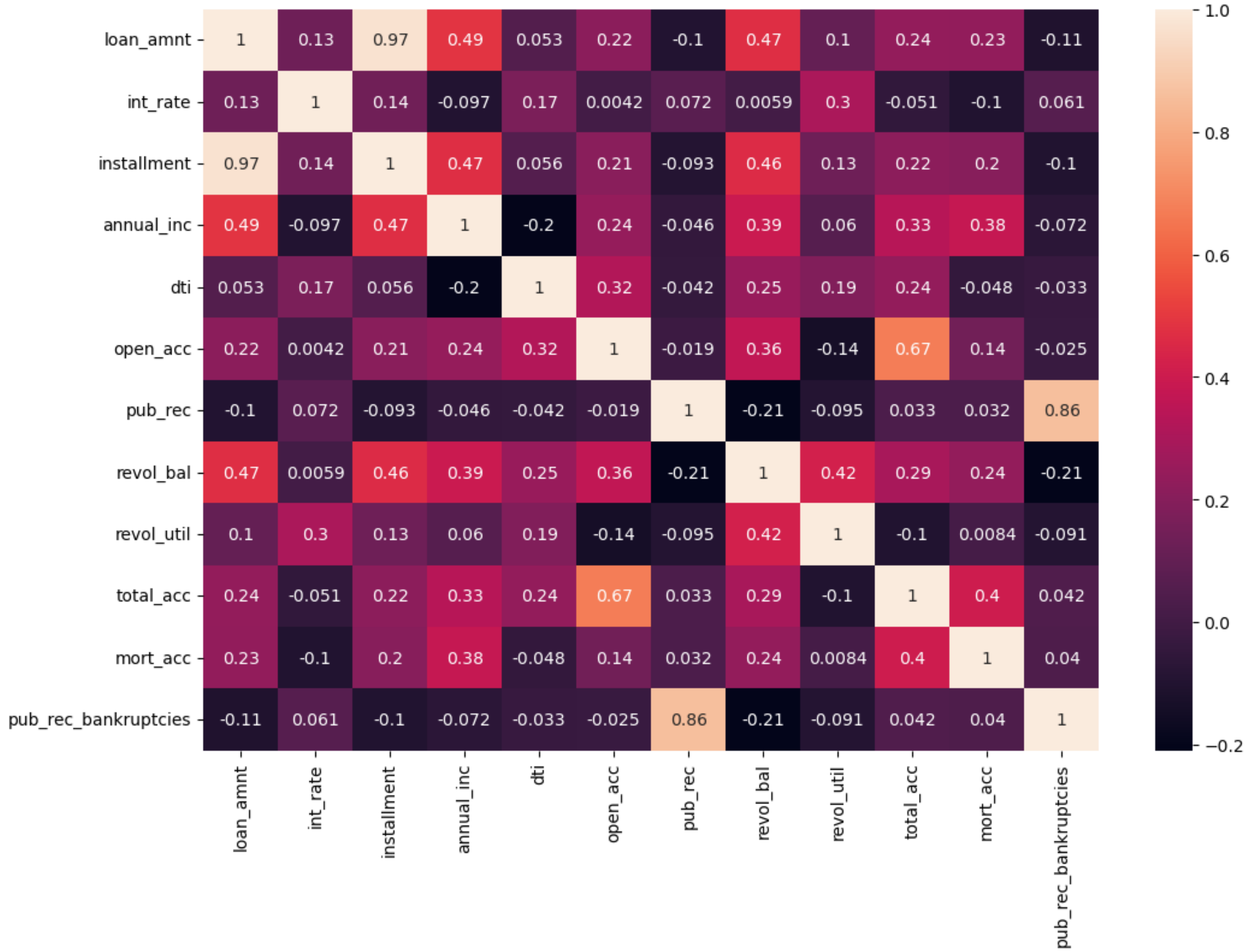
15 Non-numerical (categorical/date time) features present in the dataset.

In [25]:
```
LTDF["loan_status"].value_counts(normalize=True)*100
```

Out[25]:
```
Fully Paid     80.387092
Charged Off    19.612908
Name: loan_status, dtype: float64
```

There seems to be an imbalance in data 80% belongs to class 0 : which is loan fully paid 20% belongs to class 1 : which is charged off

In [27]:
```python
plt.figure(figsize=(12, 8))
sns.heatmap(LTDF.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 creditdepartment reduces the lo
an amount, then it will be refl ected in this value.
```

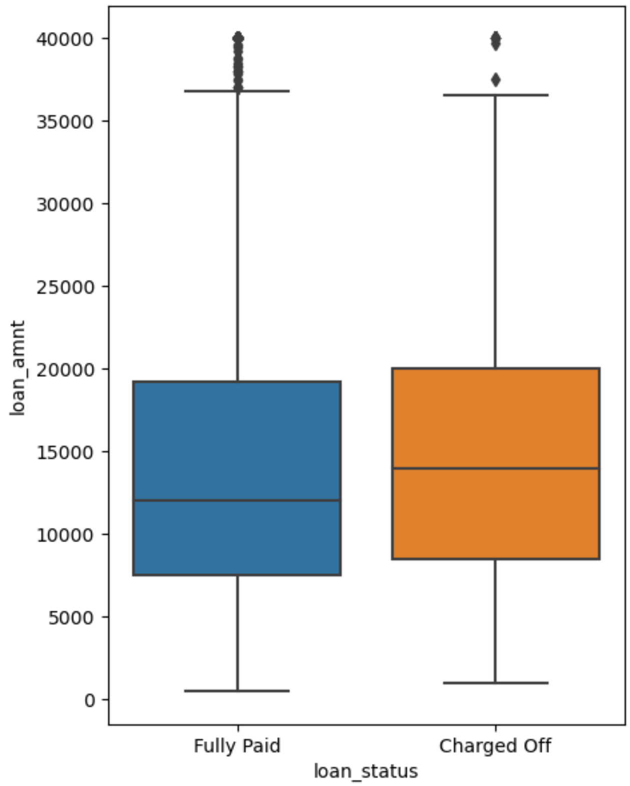
In [28]:
```python
LTDF.groupby(by ="loan_status")["loan_amnt"].describe()
```

Out[28]:

| 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 |

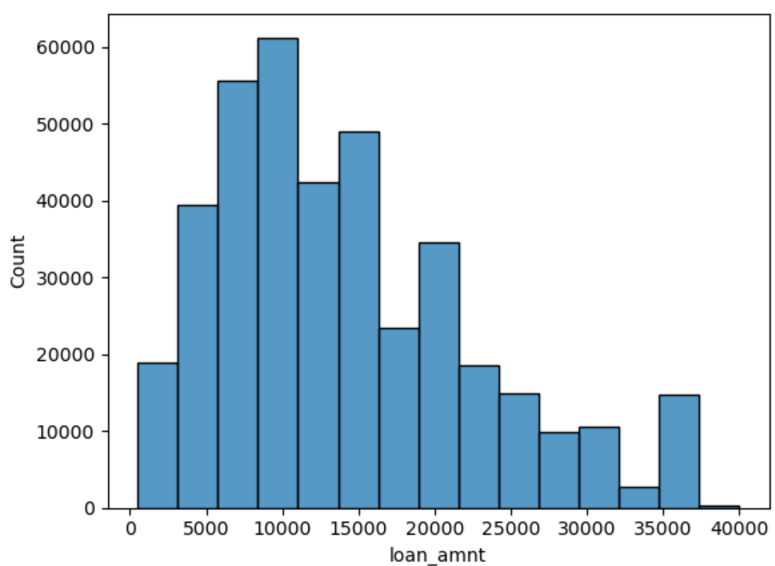
In [32]:
```
plt.figure(figsize=(5,7))
sns.boxplot(y=LTDF["loan_amnt"],x=LTDF["loan_status"])
```

Out[32]: <Axes: xlabel='loan_status', ylabel='loan_amnt'>




In [34]:
```
sns.histplot(LTDF["loan_amnt"],bins =15)
```

Out[34]: <Axes: 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 [35]:
```
LTDF["term"].value_counts(dropna=False)
```

Out[35]:
```
 36 months    302005
 60 months     94025
Name: term, dtype: int64
```

P[loan_statis | term]

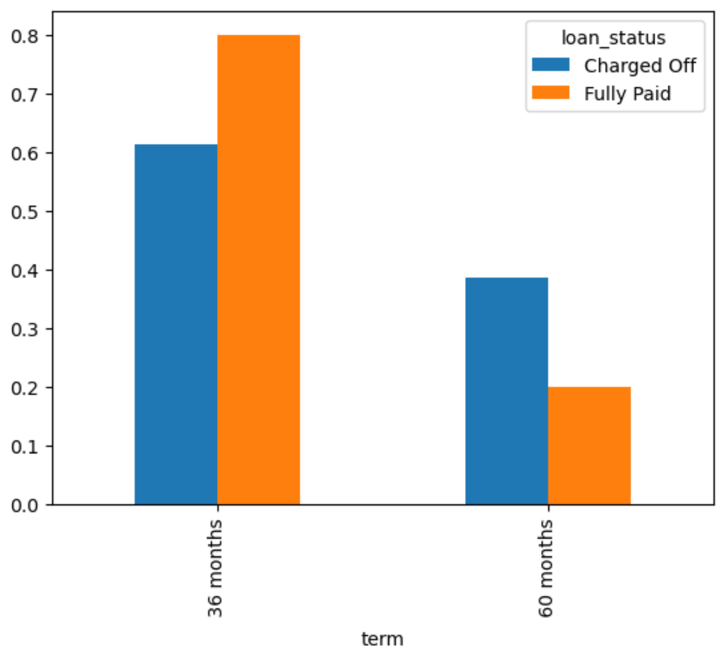
In [37]:
```
pd.crosstab(index=LTDF["term"],columns=LTDF["loan_status"], normalize="index", margins =True) *100
```

Out[37]:

| loan_status<br>term | Charged Off | Fully Paid |
|---|---|---|
| 36 months | 15.774573 | 84.225427 |
| 60 months | 31.941505 | 68.058495 |
| All | 19.612908 | 80.387092 |

In [38]:
```
pd.crosstab(index=LTDF["term"],columns =LTDF["loan_status"], normalize="columns").plot(kind ="bar")
```

Out[38]: <Axes: xlabel='term'>




We can observe that the conditional probability of loan fully paid given that its 36 month termis higher then charged off.

Loan fully paid probability when 60 month term is lower than charged off.

In [40]:
```
term_values = {' 36 months':36,' 60 months':60}
LTDF['term'] = LTDF['term'].map(term_values)
```

int_rate :

```
Interest Rate on the loan
```

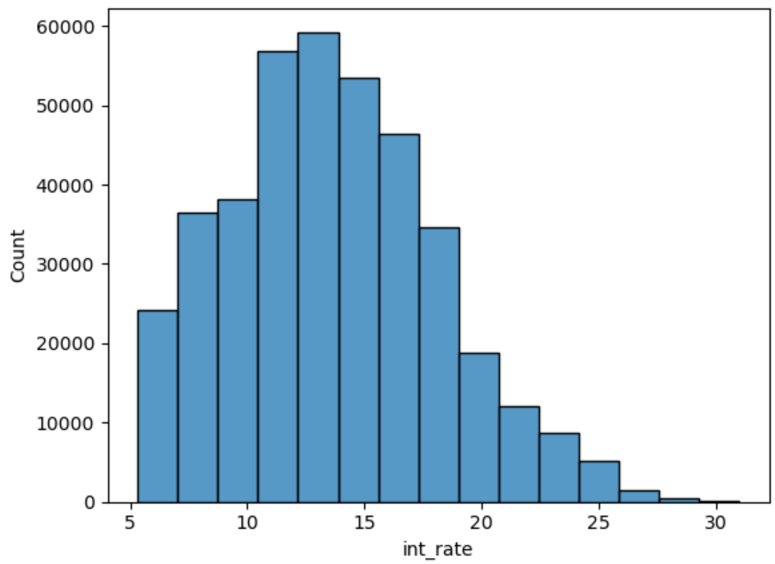
In [41]:
```
LTDF.groupby(by ="loan_status")["int_rate"].describe()
```

Out[41]:

| 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 |

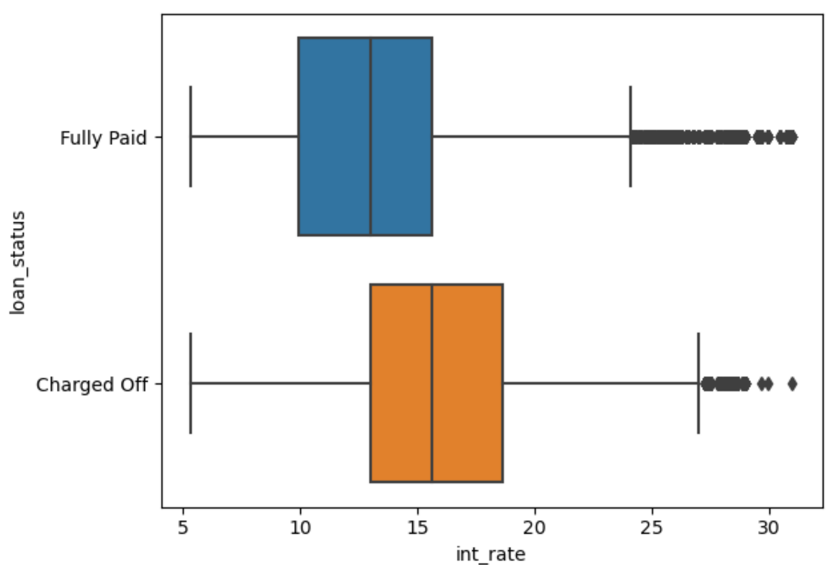
In [42]:
```
sns.histplot(LTDF["int_rate"],bins =15)
```

Out[42]: <Axes: xlabel='int_rate', ylabel='Count'>




In [43]:
```
sns.boxplot(x=LTDF["int_rate"],y=LTDF["loan_status"])
```

Out[43]: <Axes: xlabel='int_rate', ylabel='loan_status'>




In [48]:
```
LTDF[LTDF["loan_status"] =="Charged Off"]["int_rate"].median(),LTDF[LTDF["loan_status"] =="Charged Off"]["int_rate"].mean()
```

Out[48]: (15.61, 15.882587256833133)

In [49]:
```
LTDF[LTDF["loan_status"] =="Fully Paid"]["int_rate"].median(),LTDF[LTDF["loan_status"] =="Fully Paid"]["int_rate"].mean()
```

Out[49]: (12.99, 13.092105403682032)

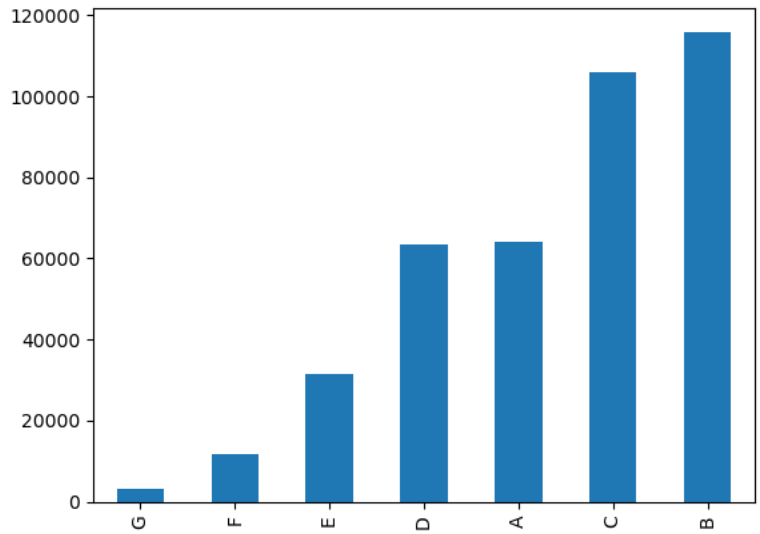
For charge_off Loan Status ,interest_rate median and mean is higher than fully paid.

Grade :

```
LoanTap assigned loan grade

Loan grades are set based on both the borrower's credit profi le and the nature of the contract.
```

In [51]:
```
LTDF["grade"].value_counts().sort_values().plot(kind ="bar")
```

Out[51]: <Axes: >




In [52]:
```
LTDF["grade"].value_counts(dropna=False)
```

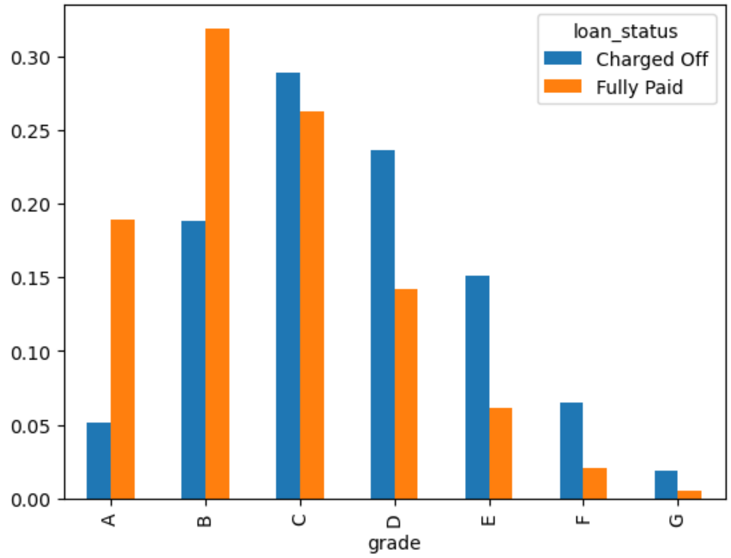
Out[52]:
```
B    116018
C    105987
A     64187
D     63524
E     31488
F     11772
G      3054
Name: grade, dtype: int64
```

In [53]:
```
pd.crosstab(index = LTDF["grade"],columns= LTDF["loan_status"],normalize="index", margins =True)
```

Out[53]:

| 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 [55]:
```
pd.crosstab(index = LTDF["grade"],columns= LTDF["loan_status"],normalize="columns").plot(kind ="bar")
plt.show()
```




Probability of loan_status as fully_paid decreases with grade is E,F,G

We can conclude the relationship exists between loan_status and LoanTap assigned loan grade.

Sub_grade :

```
LoanTap assigned loan subgrade
```

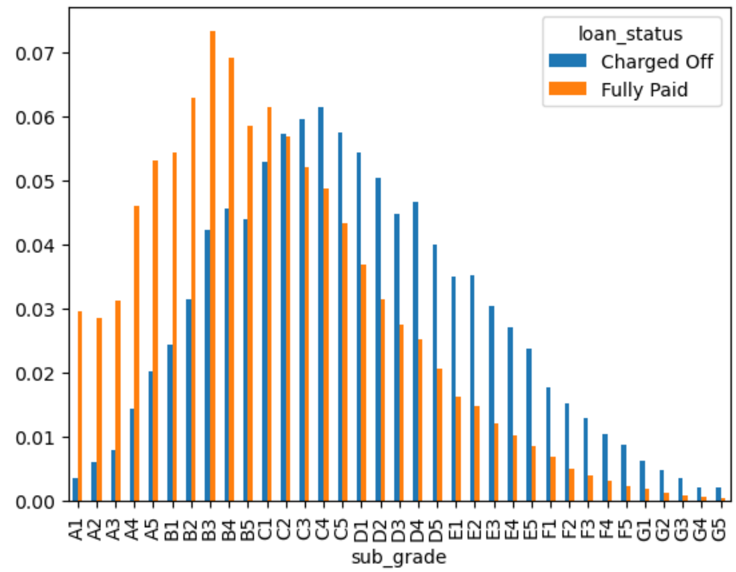
In [57]:
```
pd.crosstab(index = LTDF["sub_grade"],columns= LTDF["loan_status"],normalize= "index", margins = True)*100
```

Out[57]:

| loan_status | Charged Off | Fully Paid |
|---|---|---|
| **sub_grade** | | |
| **A1** | 2.867715 | 97.132285 |
| **A2** | 4.818647 | 95.181353 |
| **A3** | 5.805598 | 94.194402 |
| **A4** | 7.023877 | 92.976123 |
| **A5** | 8.490770 | 91.509230 |
| **B1** | 9.858200 | 90.141800 |
| **B2** | 10.851300 | 89.148700 |
| **B3** | 12.335397 | 87.664603 |
| **B4** | 13.839303 | 86.160697 |
| **B5** | 15.503736 | 84.496264 |
| **C1** | 17.369622 | 82.630378 |
| **C2** | 19.751993 | 80.248007 |
| **C3** | 21.841572 | 78.158428 |
| **C4** | 23.535503 | 76.464497 |
| **C5** | 24.506687 | 75.493313 |
| **D1** | 26.380291 | 73.619709 |
| **D2** | 28.033833 | 71.966167 |
| **D3** | 28.421828 | 71.578172 |
| **D4** | 31.131509 | 68.868491 |
| **D5** | 32.010309 | 67.989691 |
| **E1** | 34.406972 | 65.593028 |
| **E2** | 36.737990 | 63.262010 |
| **E3** | 38.037699 | 61.962301 |
| **E4** | 39.302369 | 60.697631 |
| **E5** | 40.310586 | 59.689414 |
| **F1** | 38.744344 | 61.255656 |
| **F2** | 42.480116 | 57.519884 |
| **F3** | 43.613298 | 56.386702 |
| **F4** | 45.607163 | 54.392837 |
| **F5** | 48.675734 | 51.324266 |
| **G1** | 46.124764 | 53.875236 |
| **G2** | 48.275862 | 51.724138 |
| **G3** | 51.086957 | 48.913043 |
| **G4** | 44.919786 | 55.080214 |
| **G5** | 50.316456 | 49.683544 |
| **All** | 19.612908 | 80.387092 |

In [59]:
```
pd.crosstab(index = LTDF["sub_grade"],columns= LTDF["loan_status"],normalize="columns", ).plot(kind ="bar")
```

Out[59]:
```
<Axes: xlabel='sub_grade'>
```




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 [60]:
```
LTDF["emp_title"].value_counts(dropna=False).sort_values(ascending=False).head(15)
```

Out[60]:
```
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 [61]:
```
LTDF["emp_title"].nunique()
```

Out[61]:
```
173105
```

In [62]:
```
# 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 oneyear and 10 means ten or mor
e years.
```

In [63]:
```
LTDF["emp_length"].value_counts(dropna=False)
```

Out[63]:
```
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
```

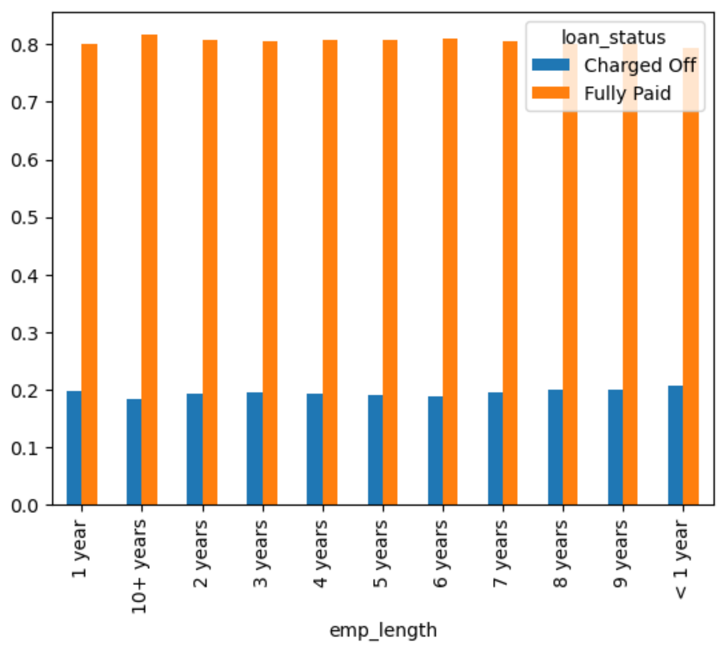
In [64]:
```
pd.crosstab(index = LTDF["emp_length"],columns= LTDF["loan_status"],normalize="index", margins =True)*100
```

Out[64]:

| loan_status<br>emp_length | Charged Off | Fully Paid |
|---|---|---|
| **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 [66]:
```
pd.crosstab(index = LTDF["emp_length"],columns= LTDF["loan_status"],normalize="index").plot(kind ="bar")
```

Out[66]: <Axes: xlabel='emp_length'>




Visually there doesn't seem to be much correlation between employement length and loan_status.

In [69]:
```
from scipy import stats
stats.chi2_contingency(pd.crosstab(index = LTDF["emp_length"],columns= LTDF["loan_status"]))
```

Out[69]:
```
Chi2ContingencyResult(statistic=122.11317384460878, pvalue=1.88404995201913e-21, dof=10, expected_freq=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 creditreport.
```

In [70]:
```
LTDF["home_ownership"].value_counts(dropna=False)
```

Out[70]:
```
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
ANY              3
Name: home_ownership, dtype: int64
```

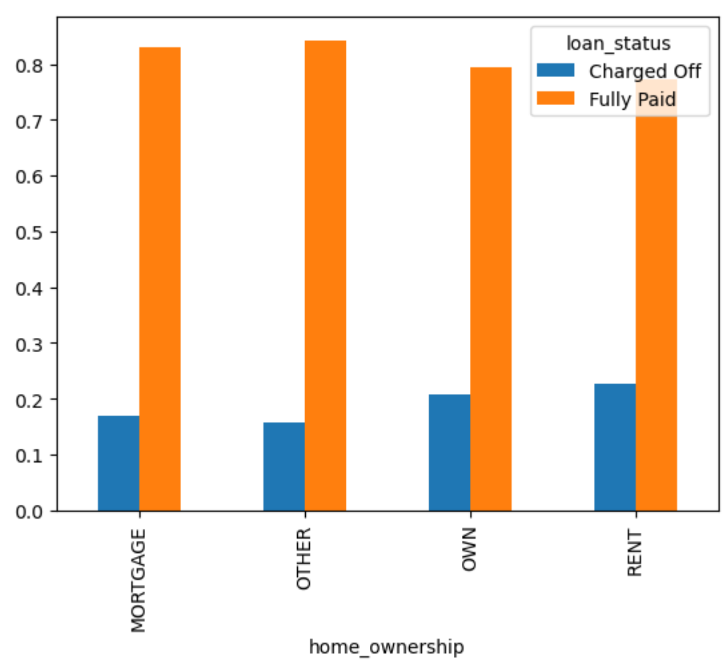
In [71]:
```
LTDF["home_ownership"] = LTDF["home_ownership"].replace({"NONE":"OTHER","ANY":"OTHER"})
```

In [72]:
```
pd.crosstab(index = LTDF["home_ownership"],columns= LTDF["loan_status"],normalize="index", margins =True)*100
```

Out[72]:

| loan_status | Charged Off | Fully Paid |
|---|---|---|
| **home_ownership** | | |
| **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 [75]:
```
pd.crosstab(index = LTDF["home_ownership"],columns= LTDF["loan_status"],normalize="index").plot(kind="bar")
plt.show()
```

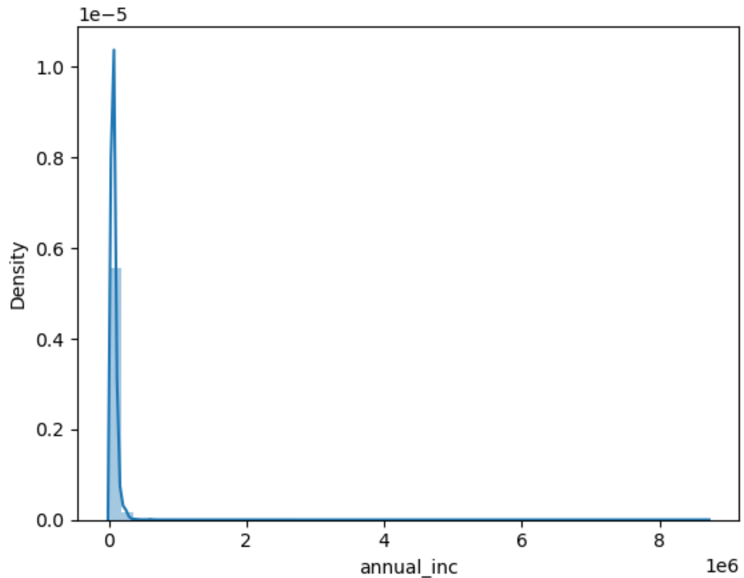



Visually there doent seems to be much correlation between home_ownership and loan_status. later target encoding or label encoding .

Annual_inc :

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

In [77]:
```
sns.distplot(LTDF["annual_inc"])
plt.show()
```

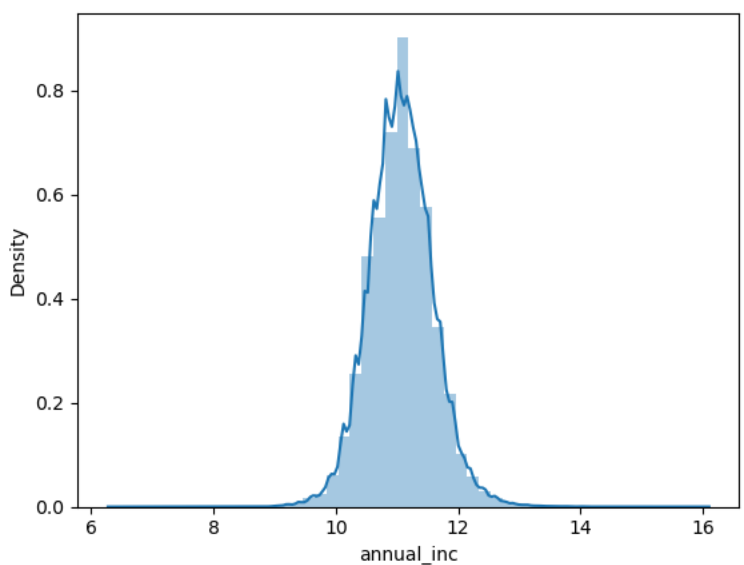



In [78]:
```
LTDF["annual_inc"].describe()
```

Out[78]:
```
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
```

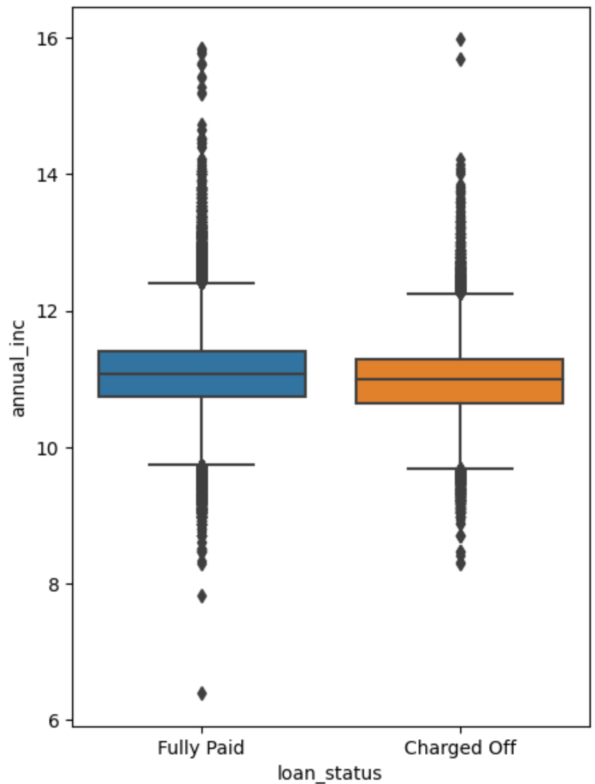
In [81]:
```
sns.distplot(np.log(LTDF[LTDF["annual_inc"]>0]["annual_inc"]))
```

Out[81]:
```
<Axes: xlabel='annual_inc', ylabel='Density'>
```



In [83]:
```
plt.figure(figsize=(5,7))
sns.boxplot(y=np.log(LTDF[LTDF["annual_inc"]>0]["annual_inc"]),x=LTDF["loan_status"])
```

Out[83]: <Axes: xlabel='loan_status', ylabel='annual_inc'>




From above boxplot, there seems to be no difference between annual income, for loan status categories

Verification_status :

```
Indicates if income was verifi ed by LoanTap, not verifi ed, or if the income source was verified
```

In [85]:
```
LTDF["verification_status"].value_counts(dropna=False)
```

Out[85]:
```
Verified           139563
Source Verified    131385
Not Verified       125082
Name: verification_status, dtype: int64
```

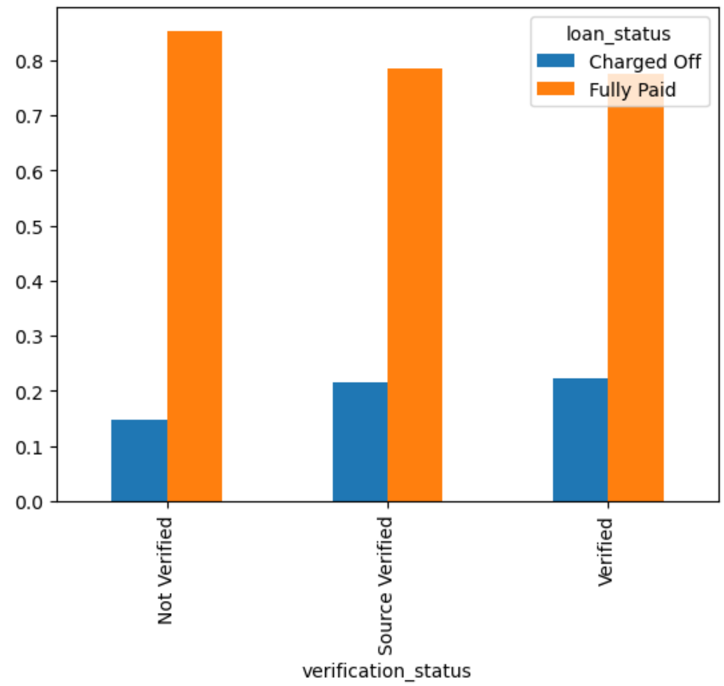
In [86]:
```
pd.crosstab(index = LTDF["verification_status"],columns= LTDF["loan_status"],normalize="index", margins =True)*100
```

Out[86]:

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

In [87]:
```
pd.crosstab(index = LTDF["verification_status"],columns= LTDF["loan_status"],normalize="index").plot(kind ="bar")
```

Out[87]: <Axes: xlabel='verification_status'>




In [88]:
```
# later label encoding
# .
# Verified 1
# Source Verified 2
# Not Verified 0
```

Purpose :

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

In [89]:
```
LTDF["purpose"].nunique()
```

Out[89]: 14

In [91]:
```
print(LTDF["purpose"].value_counts(dropna=False))
```

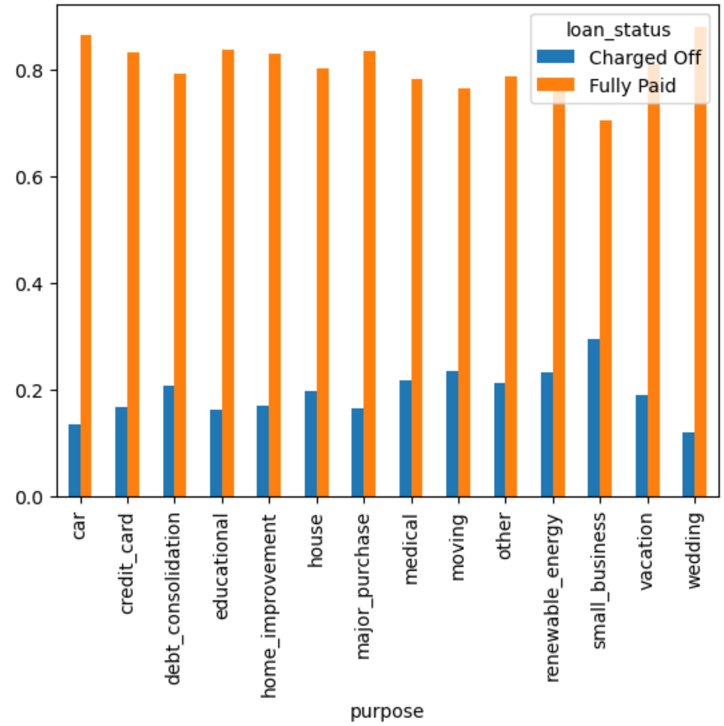
```
debt_consolidation    234507
credit_card            83019
home_improvement       24030
other                  21185
major_purchase          8790
small_business          5701
car                     4697
medical                 4196
moving                  2854
vacation                2452
house 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

In [92]:
```
pd.crosstab(index = (LTDF["purpose"],columns= (LTDF["loan_status"],normalize= "index", margins = True)*100
```

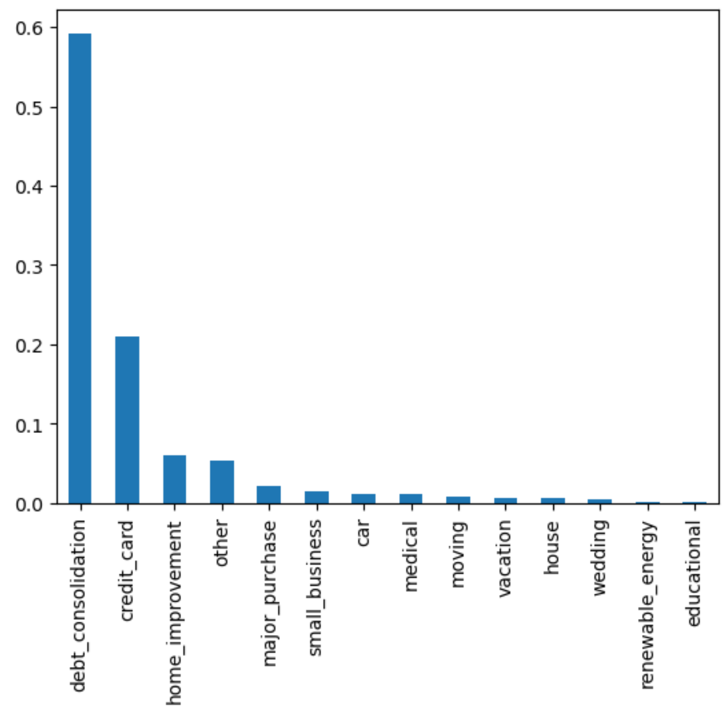
```
  Cell In[92], line 1
    pd.crosstab(index = (LTDF["purpose"],columns= (LTDF["loan_status"],normalize= "index", margins = True)*100
                                                                                 ^
SyntaxError: invalid syntax. Maybe you meant '==' or ':=' instead of '='?
```

In [93]:
```
pd.crosstab(index = LTDF["purpose"],columns= LTDF["loan_status"],normalize= "index").plot(kind = "bar")
```

Out[93]: <Axes: xlabel='purpose'>



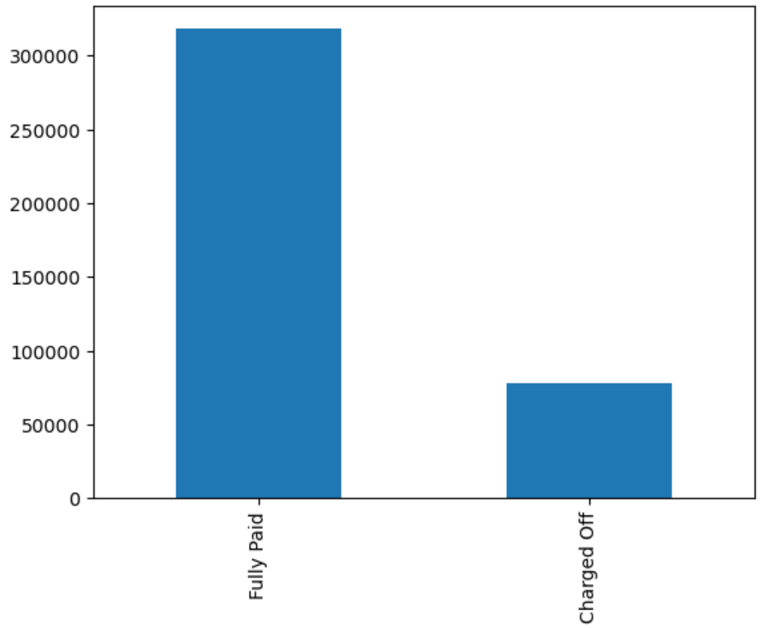

In [95]:
```
(LTDF["purpose"].value_counts(dropna=False,normalize=True)).plot(kind ="bar")
plt.show()
```




loan_status :

```
Current status of the loan - Target Variable
```

In [98]:
```
LTDF["loan_status"].value_counts(dropna=False).plot(kind ="bar")
plt.show()
```




In [99]:
```
LTDF["loan_status"].value_counts(dropna=False, normalize=True) *100
```

Out[99]:
```
Fully Paid     80.387092
Charged Off    19.612908
Name: loan_status, dtype: float64
```

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 highest in above category.

Title :

```
The loan title provided by the borrower
```

In [101]:
```
LTDF["title"].nunique()
```

Out[101]:
```
48817
```

In [102]:
```
LTDF["title"]
```

Out[102]:
```
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 [103]:
```
# Title and purpose are in a way same features.
# later needs to drop this feature.
```

Dti :

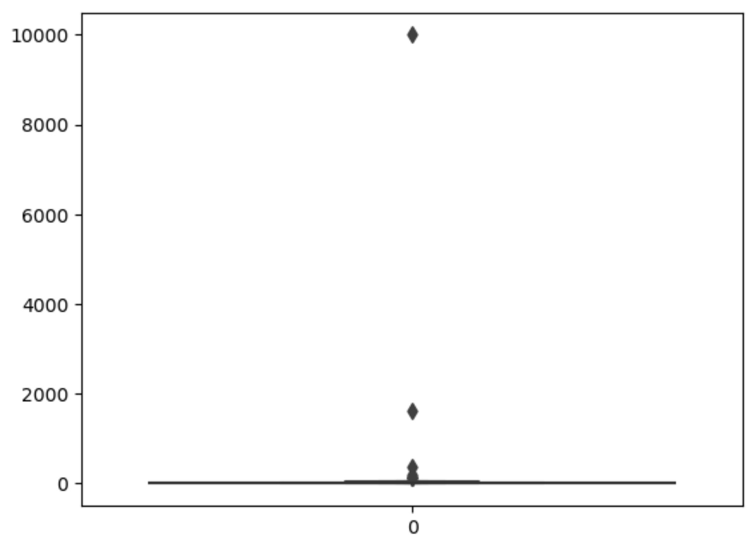
```
A ratio calculated using the borrower's total monthly debt payments on the total debt obligations,excluding mortgage and
the requested LoanTap loan, divided by the borrower's self-reportedmonthly income.
```

Dti = monthly total dept payment / monthly income excluding mortgages

In [105]:
```
LTDF["dti"].describe()
```

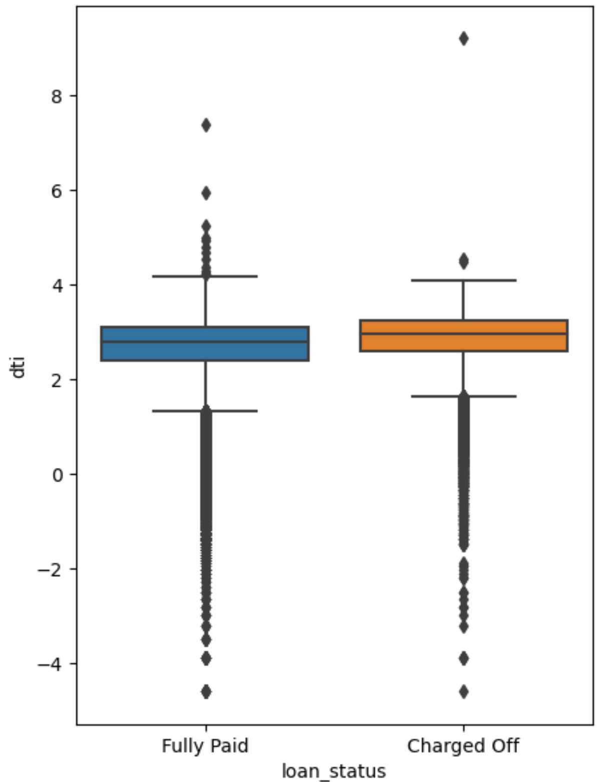
Out[105]:
```
count    396030.000000
mean         17.379514
std          18.019092
min           0.000000
25%          11.280000
50%          16.910000
75%          22.980000
max        9999.000000
Name: dti, dtype: float64
```

In [107]:
```
sns.boxenplot((LTDF["dti"]))
plt.show()
```




There are lots of outliers in dti column .

In [110]:
```
plt.figure(figsize=(5,7))

sns.boxplot(y=np.log(LTDF[LTDF["dti"]>0]["dti"]),x=LTDF["loan_status"])
plt.show()
```




issue_d :

```
The month which the loan was funded.
```

In [112]:
```
# 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
```

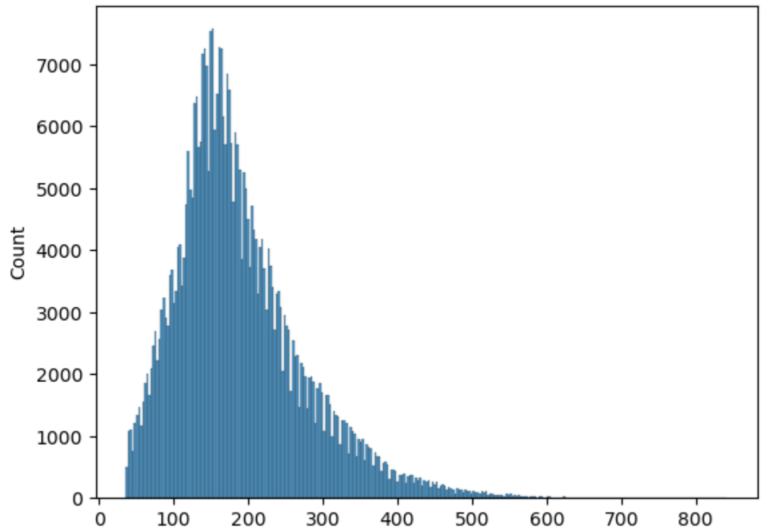
In [122]:
```
LTDF["Loan_Tenure"] = ((pd.to_datetime(LTDF["issue_d"]) -pd.to_datetime(LTDF["earliest_cr_line"]))/np.timedelta64(1,'M'))
```

In [123]:
```
# pd.to_datetime(df["earliest_cr_line"])
```

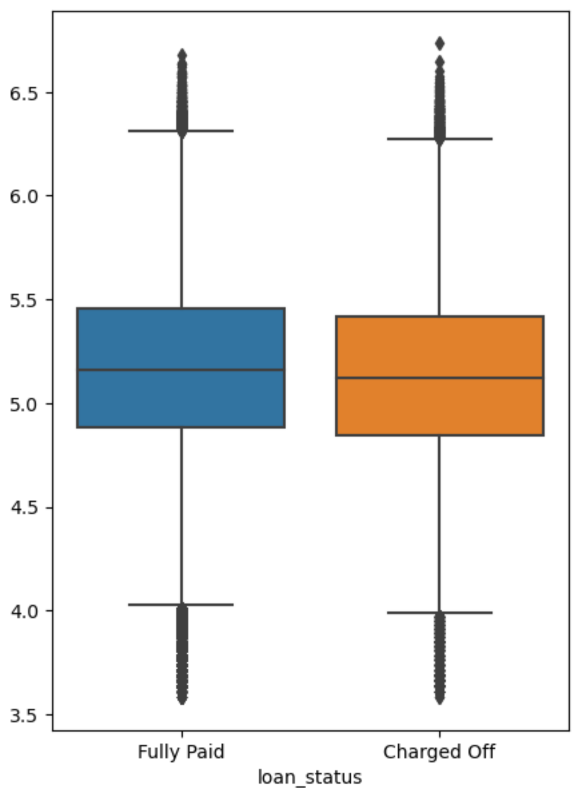
In [124]:
```
# The month which the loan was funded
```

In [125]:
```
# pd.to_datetime(df["issue_d"])
```

In [128]:
```
sns.histplot(((pd.to_datetime(LTDF["issue_d"]) -pd.to_datetime(LTDF["earliest_cr_line"]))/np.timedelta64(1,'M')))
plt.show()
```




In [139]:
```
plt.figure(figsize=(5,7))

sns.boxplot(y=np.log(((pd.to_datetime(LTDF["issue_d"]) -pd.to_datetime(LTDF["earliest_cr_line"]))/np.timedelta64(1,'M'))),x=LTDF[
plt.show()
```




open_acc :

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

In [141]:
```
LTDF.groupby("loan_status")["open_acc"].describe()
```

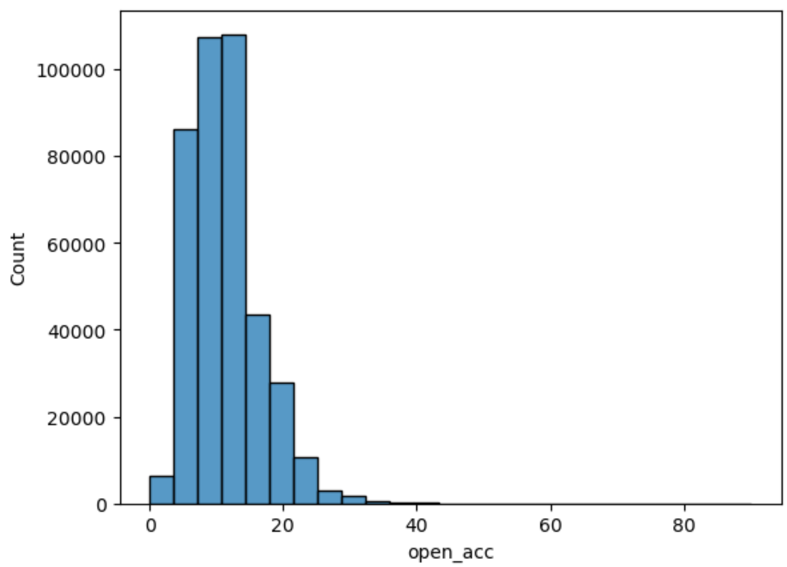
Out[141]:

| 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 |

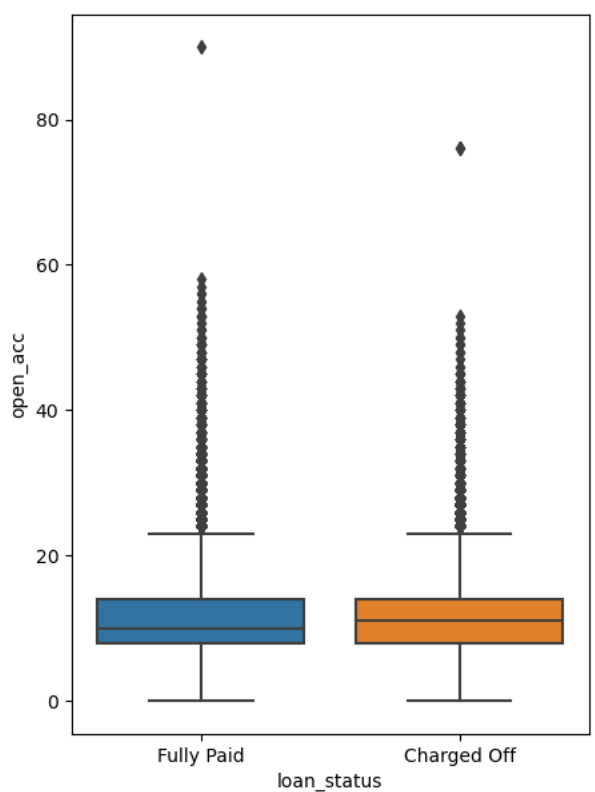
In [142]:
```
LTDF["open_acc"].nunique()
```

Out[142]: 61

In [144]:
```
sns.histplot(LTDF["open_acc"],bins =25)
plt.show()
```




In [148]:
```
plt.figure(figsize=(5,7))
sns.boxplot(y= LTDF["open_acc"],x=LTDF["loan_status"])
plt.show()
```



pub_rec :

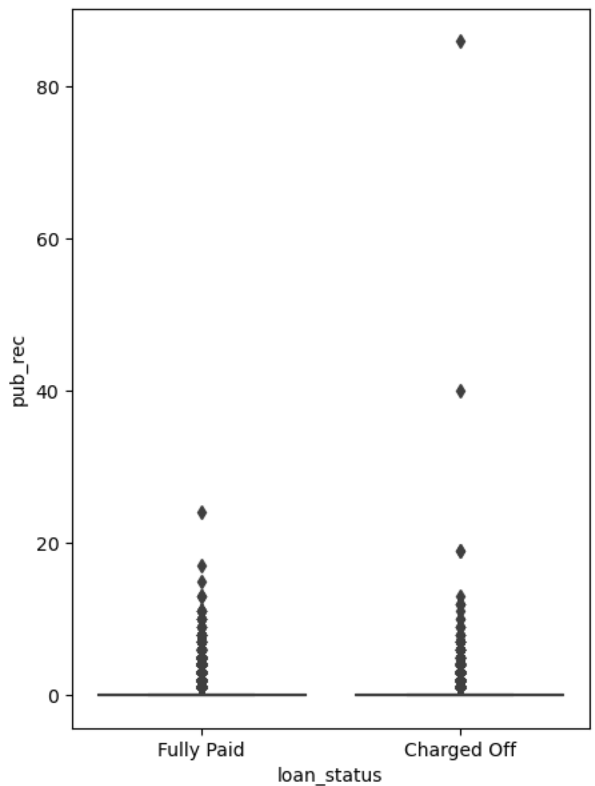
```
Number of derogatory public records

“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 [149]:
```
LTDF.groupby("loan_status")["pub_rec"].describe()
```

Out[149]:

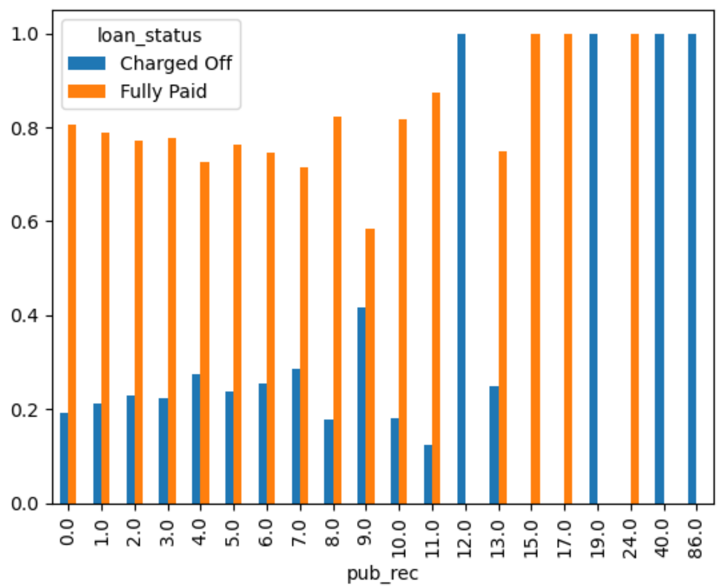
| loan_status | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **Charged Off** | 77673.0 | 0.199606 | 0.648283 | 0.0 | 0.0 | 0.0 | 0.0 | 86.0 |
| **Fully Paid** | 318357.0 | 0.172966 | 0.497637 | 0.0 | 0.0 | 0.0 | 0.0 | 24.0 |

In [151]:
```
plt.figure(figsize=(5,7))
sns.boxplot(y= LTDF["pub_rec"],x=LTDF["loan_status"])
plt.show()
```



In [153]:
```
print(LTDF["pub_rec"].value_counts(dropna=False))

pd.crosstab(index = LTDF["pub_rec"],columns= LTDF["loan_status"],normalize="index", margins =True)*100

pd.crosstab(index = LTDF["pub_rec"],columns= LTDF["loan_status"],normalize="index").plot(kind ="bar")
plt.show()
```

```
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
```




revol_bal :

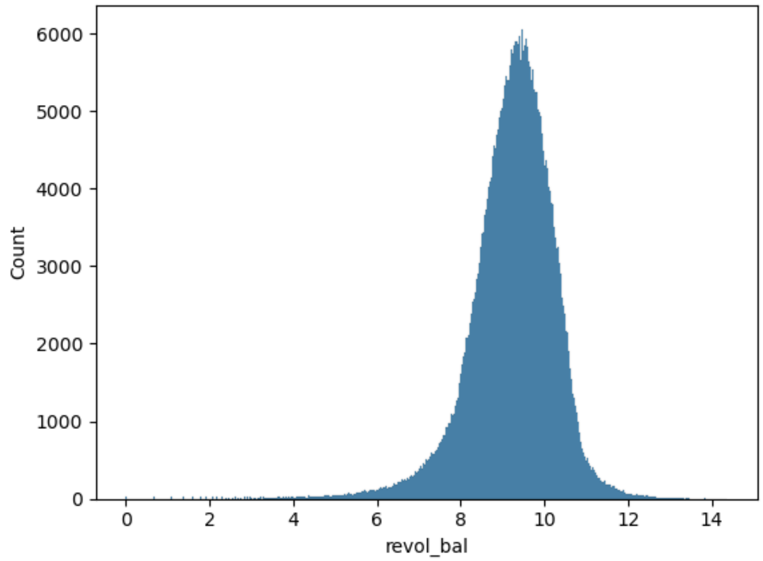
```
Total credit revolving balance

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

In [155]:
```
LTDF.groupby("loan_status")["revol_bal"].describe()
```

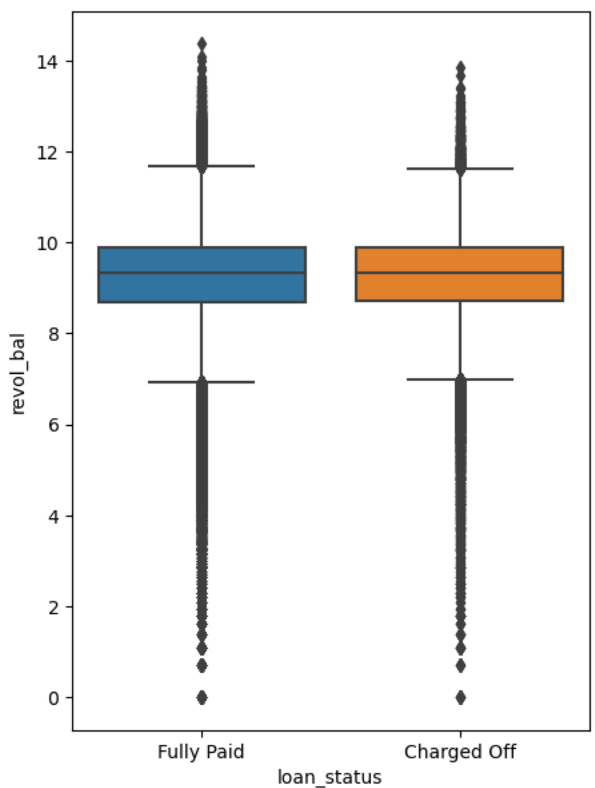
Out[155]:

| 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 [157]:
```
sns.histplot(np.log(LTDF["revol_bal"]))
plt.show()
```




In [159]:
```
plt.figure(figsize=(5,7))

sns.boxplot(y= np.log(LTDF["revol_bal"]),x=LTDF["loan_status"])
plt.show()
```




revol_util :

```
Revolving line utilization rate, or the amount of credit the borrower is using relative to all availablerevolving credi
t.
```

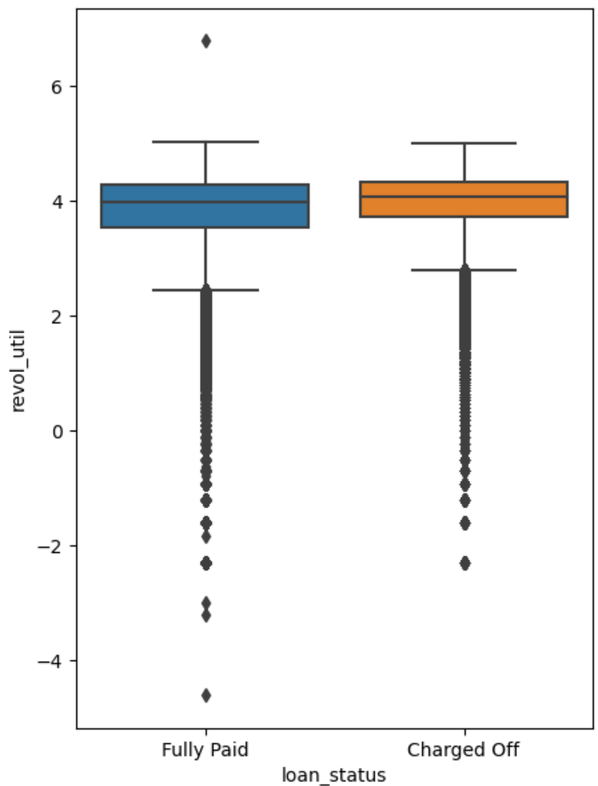
Your credit utilization rate, sometimes called your credit utilization ratio, is the amount of revolving credit you'recurrently using divided by the total amount of revolving credit you have available. In other words, it's how much youcurrently owe divided by your credit limit. It is generally expressed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

In [160]:
```
LTDF.groupby("loan_status")["revol_util"].describe()
```

Out[160]:

| 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 [162]:
```
plt.figure(figsize=(5,7))

sns.boxplot(y= np.log(LTDF["revol_util"]),x=LTDF["loan_status"])
plt.show()
```




total_acc :

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

In [163]:
```
1# df["total_acc"].value_counts()
```

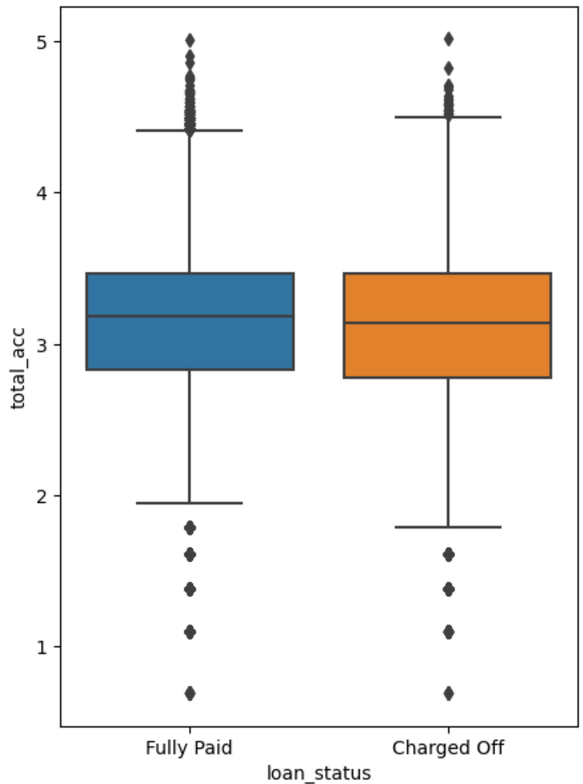
Out[163]: 1

In [164]:
```
LTDF.groupby("loan_status")["total_acc"].describe()
```

Out[164]:

| 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 [165]:
```
plt.figure(figsize=(5,7))

sns.boxplot(y= np.log(LTDF["total_acc"]),x=LTDF["loan_status"])

plt.show()
```




initial_list_status :

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

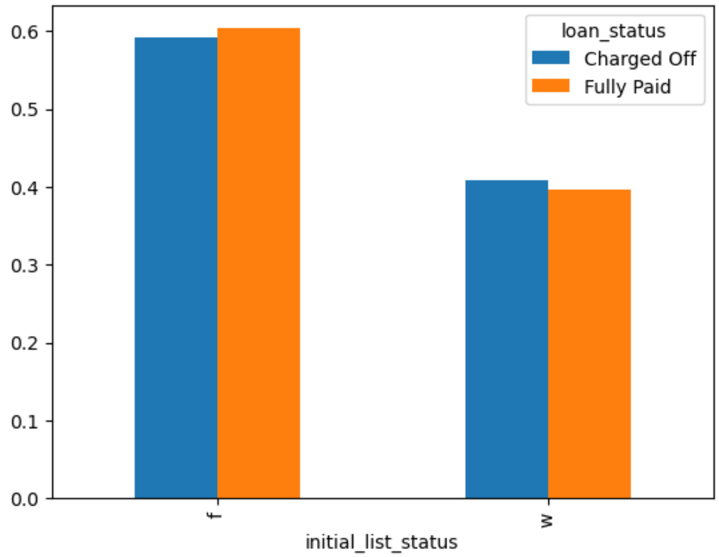
In [166]:
```
LTDF["initial_list_status"].value_counts()
```

Out[166]:
```
f    238066
w    157964
Name: initial_list_status, dtype: int64
```

In [167]:
```
print(LTDF["initial_list_status"].value_counts(dropna=False))
```

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

In [169]:
```
pd.crosstab(index = LTDF["initial_list_status"],columns= LTDF["loan_status"],normalize="columns").plot(kind ="bar")
plt.show()
```




application_type :

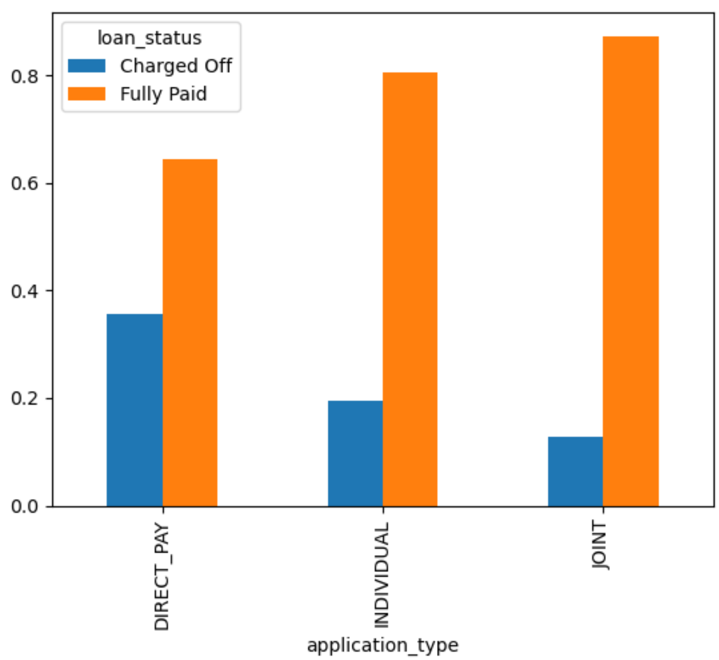
```
Indicates whether the loan is an individual application or a joint application with two co-borrowers
```

In [170]:
```
LTDF["application_type"].value_counts()
```

Out[170]:
```
INDIVIDUAL    395319
JOINT            425
DIRECT_PAY       286
Name: application_type, dtype: int64
```

In [172]:
```
print(LTDF["application_type"].value_counts(dropna=False))

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

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




mort_acc :

Number of mortgage accounts.

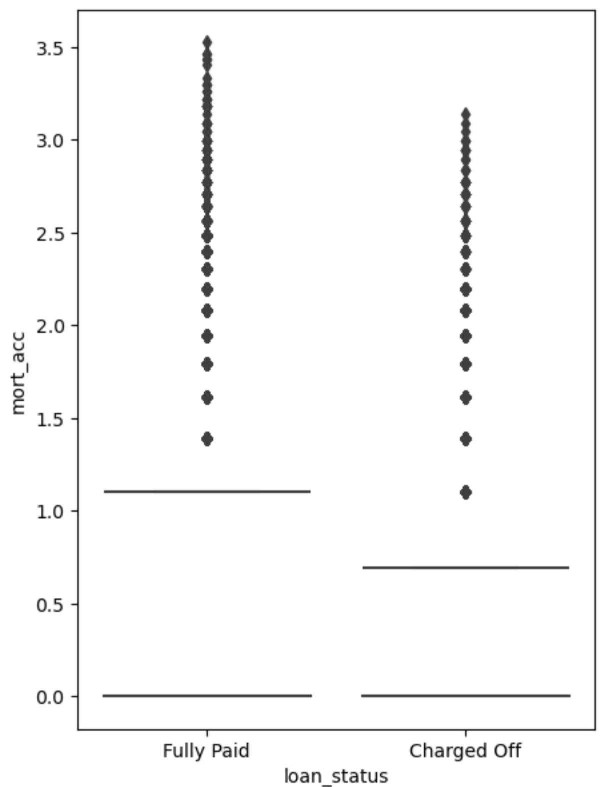
In [174]:
```
# df["mort_acc"].value_counts(dropna=False)
```

In [175]:
```
LTDF.groupby("loan_status")["mort_acc"].describe()
```

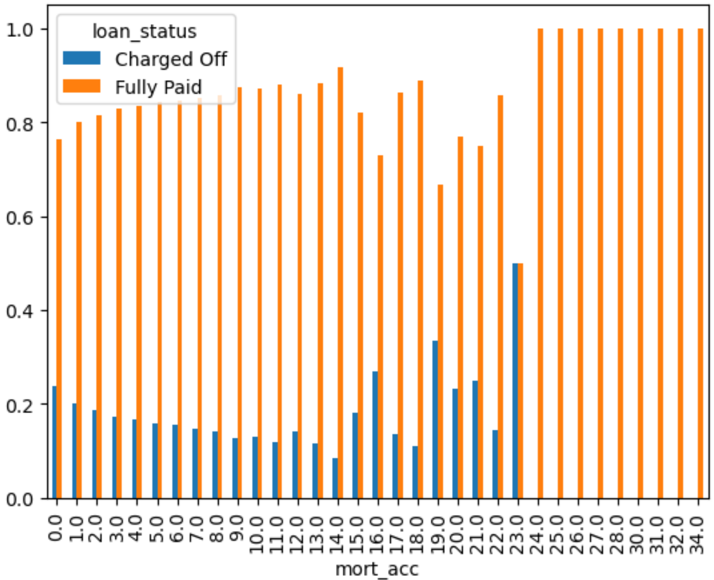
Out[175]:

| 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 [177]:
```
plt.figure(figsize=(5,7))

sns.boxplot(y= np.log(LTDF["mort_acc"]),x=LTDF["loan_status"])
plt.show()
```



In [179]:
```
pd.crosstab(index = LTDF["mort_acc"],columns= LTDF["loan_status"],normalize="index").plot(kind ="bar")
plt.show()
```




pub_rec_bankruptcies :

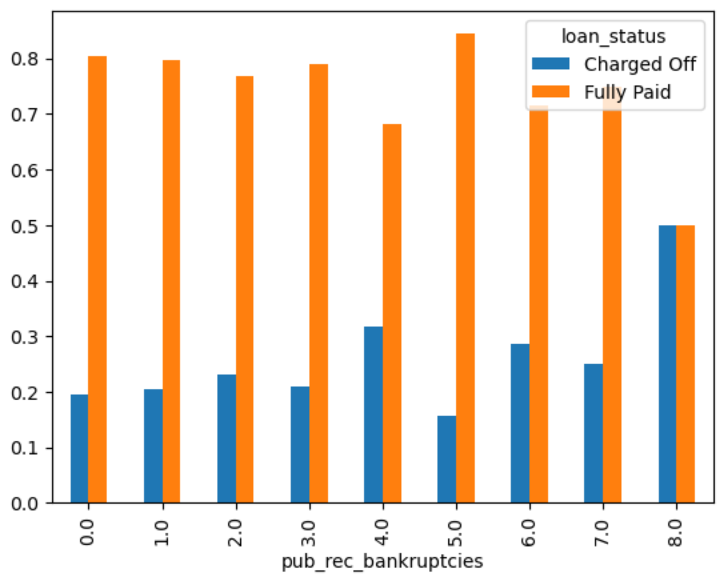
```
Number of public record bankruptcies
```

In [180]:
```
LTDF["pub_rec_bankruptcies"].value_counts()
```

Out[180]:
```
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 [181]:
```
print(LTDF["pub_rec_bankruptcies"].value_counts(dropna=False))

print(pd.crosstab(index = LTDF["pub_rec_bankruptcies"],columns= LTDF["loan_status"],normalize="index", margins =True)*100)

pd.crosstab(index = LTDF["pub_rec_bankruptcies"],columns= LTDF["loan_status"],normalize="index").plot(kind ="bar")

plt.show()
```

```
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
```




Address :

    Address of the individual

In [182]:
```
LTDF["address"][10]
```

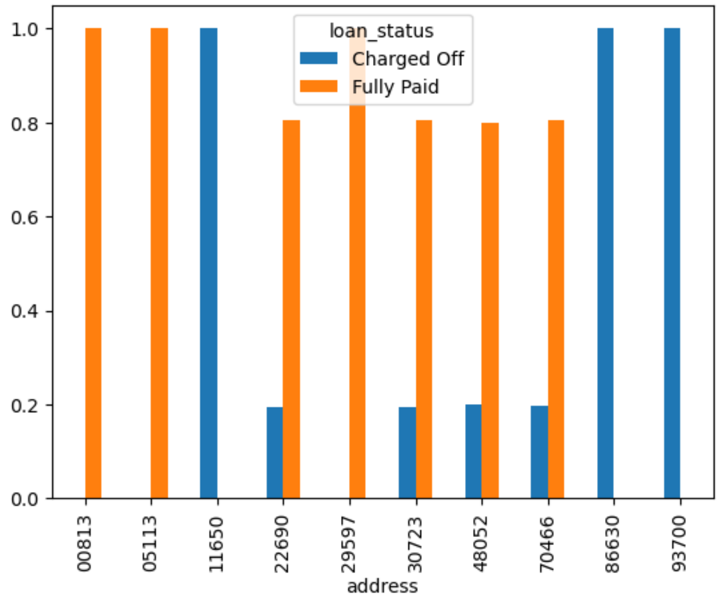
Out[182]: '40245 Cody Drives\r\nBartlettfort, NM 00813'

In [184]:
```
LTDF["address"] = LTDF["address"].str.split().apply(lambda x:x[-1])
```

In [185]:
```
LTDF["address"].value_counts()
```

Out[185]:
```
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 [187]:
```
pd.crosstab(index = LTDF["address"],columns= LTDF["loan_status"],normalize="index").plot(kind ="bar")
plt.show()
```




In [188]:
```
LTDF["pin_code"] = LTDF["address"]

LTDF.drop(["address"],axis =1,inplace=True)
```

Dropping unimportant columns

In [190]:
```
LTDF.drop(["title","issue_d","earliest_cr_line","initial_list_status"],axis =1, inplace=True)
```

In [191]:
```
LTDF.drop(["pin_code"],axis=1,inplace=True)
```

In [192]:
```
LTDF.drop(["Loan_Tenure"],axis=1,inplace=True)
```

Missing value treatment

In [193]:
```
missing_data[missing_data["Percent"]>0]
```

Out[193]:

| | 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 [197]:
```
from sklearn.impute import SimpleImputer
Imputer = SimpleImputer(strategy="most_frequent")

LTDF["mort_acc"] = Imputer.fit_transform(LTDF["mort_acc"].values.reshape(-1,1))
```

In [198]:
```
LTDF.dropna(inplace=True)
```

In [199]:
```
missing_df(LTDF)
```

Out[199]:

| | 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 [200]:
```
from category_encoders import TargetEncoder
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[200], line 1
----> 1 from category_encoders import TargetEncoder

ModuleNotFoundError: No module named 'category_encoders'
```

In [201]:
```
pip install category_encoders
```

```
Collecting category_encoders
  Downloading category_encoders-2.6.1-py2.py3-none-any.whl (81 kB)
                                              0.0/81.9 kB ? eta -:--:--
     ------------------                    41.0/81.9 kB 667.8 kB/s eta 0:00:01
     ---------------------------------------- 81.9/81.9 kB 1.2 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.14.0 in d:\users\india\anaconda3\lib\site-packages (from category_encoders) (1.23.5)
Requirement already satisfied: scikit-learn>=0.20.0 in d:\users\india\anaconda3\lib\site-packages (from category_encoders) (1.2.1)
Requirement already satisfied: scipy>=1.0.0 in d:\users\india\anaconda3\lib\site-packages (from category_encoders) (1.10.0)
Requirement already satisfied: statsmodels>=0.9.0 in d:\users\india\anaconda3\lib\site-packages (from category_encoders) (0.13.5)
Requirement already satisfied: pandas>=1.0.5 in d:\users\india\anaconda3\lib\site-packages (from category_encoders) (1.5.3)
Requirement already satisfied: patsy>=0.5.1 in d:\users\india\anaconda3\lib\site-packages (from category_encoders) (0.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in d:\users\india\anaconda3\lib\site-packages (from pandas>=1.0.5->category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in d:\users\india\anaconda3\lib\site-packages (from pandas>=1.0.5->category_encoders) (2022.7)
Requirement already satisfied: six in d:\users\india\anaconda3\lib\site-packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in d:\users\india\anaconda3\lib\site-packages (from scikit-learn>=0.20.0->category_encoders) (1.1.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in d:\users\india\anaconda3\lib\site-packages (from scikit-learn>=0.20.0->category_encoders) (2.2.0)
Requirement already satisfied: packaging>=21.3 in d:\users\india\anaconda3\lib\site-packages (from statsmodels>=0.9.0->category_encoders) (22.0)
Installing collected packages: category_encoders
Successfully installed category_encoders-2.6.1
Note: you may need to restart the kernel to use updated packages.
```

In [202]:
```
from category_encoders import TargetEncoder
```

In [203]:
```
TE = TargetEncoder()
```

In [204]:
```
LTDF["loan_status"].replace({"Fully Paid":0,"Charged Off" : 1},inplace=True)
```

In [205]:
```
LTDF.sample(3)
```

Out[205]:

| | loan_amnt | term | int_rate | installment | grade | sub_grade | emp_title | emp_length | home_ownership | annual_inc | verification_status | loan_status | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **60648** | 3000.0 | 36 | 19.52 | 110.76 | E | E2 | sales | 4 years | RENT | 65000.0 | Not Verified | 0 | c |
| **209558** | 9600.0 | 36 | 13.65 | 326.48 | C | C1 | welder | 3 years | MORTGAGE | 103000.0 | Not Verified | 1 | c |
| **71752** | 5000.0 | 36 | 14.47 | 172.04 | C | C2 | Assistant Public Relations Officer | 10+ years | RENT | 55000.0 | Source Verified | 0 | debt_cor |

In [206]:
```
LTDF.columns
```

Out[206]:
```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grade', '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_type', 'mort_acc', 'pub_rec_bankruptcies'], dtype='object')
```

In [207]:
```
LTDF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 372161 entries, 0 to 396029
Data columns (total 22 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   loan_amnt             372161 non-null  float64
 1   term                  372161 non-null  int64
 2   int_rate              372161 non-null  float64
 3   installment           372161 non-null  float64
 4   grade                 372161 non-null  object
 5   sub_grade             372161 non-null  object
 6   emp_title             372161 non-null  object
 7   emp_length            372161 non-null  object
 8   home_ownership        372161 non-null  object
 9   annual_inc            372161 non-null  float64
 10  verification_status   372161 non-null  object
 11  loan_status           372161 non-null  int64
 12  purpose               372161 non-null  object
 13  dti                   372161 non-null  float64
 14  open_acc              372161 non-null  float64
 15  pub_rec               372161 non-null  float64
 16  revol_bal             372161 non-null  float64
 17  revol_util            372161 non-null  float64
 18  total_acc             372161 non-null  float64
 19  application_type      372161 non-null  object
 20  mort_acc              372161 non-null  float64
 21  pub_rec_bankruptcies  372161 non-null  float64
dtypes: float64(12), int64(2), object(8)
memory usage: 65.3+ MB
```

In [211]:
```
t_enc = ["sub_grade","grade",'term','emp_title','emp_length','home_ownership','verification_status','purpose','application_type']
```

In [212]:
```
for col in target_enc:
    from category_encoders import TargetEncoder
    TEncoder = TargetEncoder()

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

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

In [213]:
```
LTDF
```

Out[213]:

| | loan_amnt | term | int_rate | installment | grade | sub_grade | emp_title | emp_length | home_ownership | annual_inc | verification_status | loan_status | purp |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 10000.0 | 36 | 11.44 | 329.48 | 0.121856 | 0.134935 | 0.247136 | 0.184208 | 0.222392 | 117000.0 | 0.144925 | 0 | 0.183 |
| **1** | 8000.0 | 36 | 11.99 | 265.68 | 0.121856 | 0.150496 | 0.214018 | 0.191896 | 0.166495 | 65000.0 | 0.144925 | 0 | 0.203 |
| **2** | 15600.0 | 36 | 10.49 | 506.97 | 0.121856 | 0.119644 | 0.189214 | 0.206840 | 0.222392 | 43057.0 | 0.214123 | 0 | 0.162 |
| **3** | 7200.0 | 36 | 6.49 | 220.65 | 0.059785 | 0.044741 | 0.167211 | 0.189319 | 0.222392 | 54000.0 | 0.144925 | 0 | 0.162 |
| **4** | 24375.0 | 60 | 17.27 | 609.33 | 0.207325 | 0.239437 | 0.297320 | 0.200951 | 0.166495 | 55000.0 | 0.216398 | 1 | 0.162 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **396025** | 10000.0 | 60 | 10.99 | 217.38 | 0.121856 | 0.134935 | 0.167211 | 0.193219 | 0.222392 | 40000.0 | 0.214123 | 0 | 0.203 |
| **396026** | 21000.0 | 36 | 12.29 | 700.42 | 0.207325 | 0.168489 | 0.220430 | 0.191915 | 0.166495 | 110000.0 | 0.214123 | 0 | 0.203 |
| **396027** | 5000.0 | 36 | 9.99 | 161.32 | 0.121856 | 0.094672 | 0.267968 | 0.184208 | 0.222392 | 56500.0 | 0.216398 | 0 | 0.203 |
| **396028** | 21000.0 | 60 | 15.31 | 503.02 | 0.207325 | 0.192642 | 0.167211 | 0.184208 | 0.166495 | 64000.0 | 0.216398 | 0 | 0.203 |
| **396029** | 2000.0 | 36 | 13.61 | 67.98 | 0.207325 | 0.192642 | 0.217205 | 0.184208 | 0.222392 | 42996.0 | 0.216398 | 0 | 0.203 |

372161 rows × 22 columns

# Outlier treatment:

In [214]:
```
def outlier_remover(a,LTDF):
    q1 = a.quantile(.25)
    q3 = a.quantile(.75)
    iqr = q3 - q1
    maxx = q3 + 1.5 * iqr
    minn = q1 - 1.5 * iqr
    return LTDF.loc[(a>=minn) & (a<=maxx)]
```

In [215]:
```
floats = ['loan_amnt','int_rate','annual_inc','dti','open_acc','revol_bal','revol_util','total_acc']
```

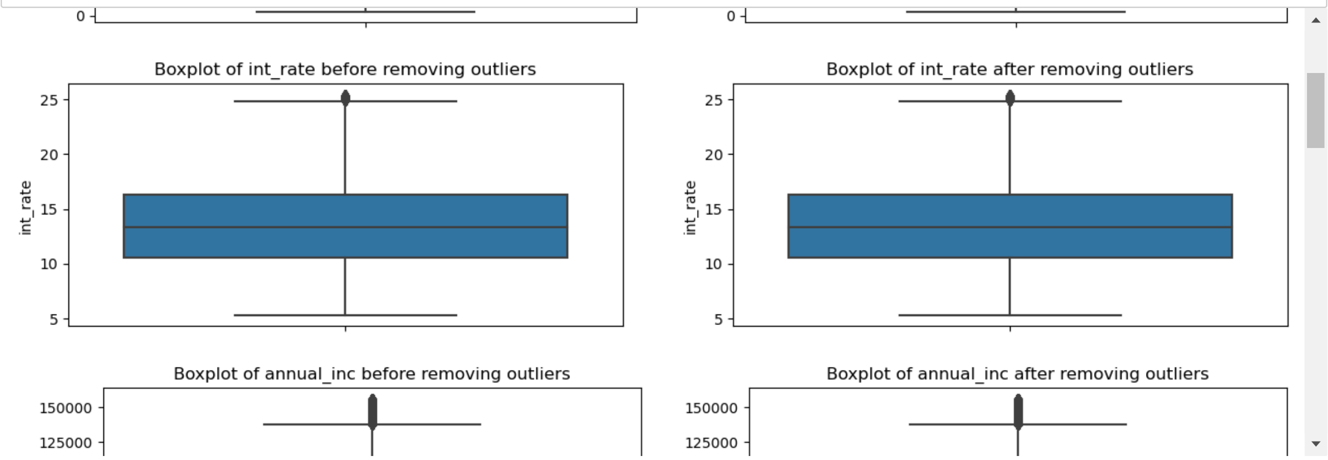
In [216]:
```
LTDF.sample(3)
```

Out[216]:

| | loan_amnt | term | int_rate | installment | grade | sub_grade | emp_title | emp_length | home_ownership | annual_inc | verification_status | loan_status | purp |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **20389** | 10000.0 | 36 | 12.29 | 333.53 | 0.207325 | 0.168489 | 0.214018 | 0.189319 | 0.222392 | 52000.0 | 0.214123 | 1 | 0.203 |
| **32978** | 9000.0 | 60 | 16.99 | 223.63 | 0.283818 | 0.257257 | 0.214854 | 0.184208 | 0.222392 | 47600.0 | 0.216398 | 0 | 0.203 |
| **319218** | 18000.0 | 60 | 15.31 | 431.16 | 0.207325 | 0.192642 | 0.297320 | 0.195177 | 0.222392 | 82000.0 | 0.216398 | 1 | 0.203 |

In [218]:
```
for i in floats:
    LTDF = outlier_remover(LTDF[i],LTDF)
```

In [220]:
```
for i in floats:
    plt.figure(figsize=(15,3))
    plt.subplot(121)
    sns.boxplot(y=LTDF[i])
    plt.title(f"Boxplot of {i} before removing outliers")
    plt.subplot(122)
    sns.boxplot(y=LTDF[i])
    plt.title(f"Boxplot of {i} after removing outliers")
    plt.show()
```




## Missing value check :

In [222]:
```
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 =False)
    missingDF = pd.concat([total_missing_df, percentage_missing_df],axis =1, keys=['Total','Percent'])
    return missingDF

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

Out[222]:

| Total | Percent |
|---|---|

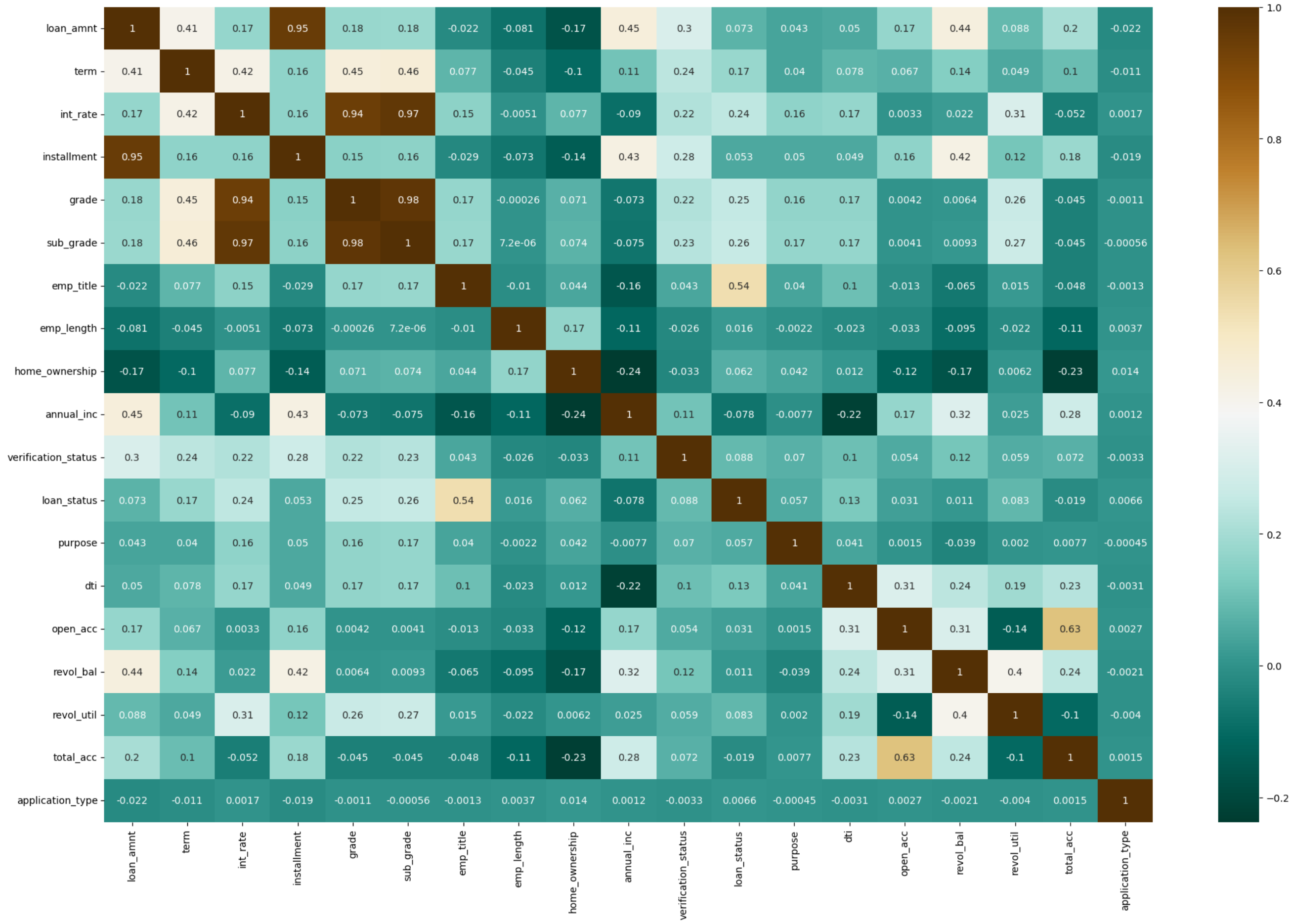
In [223]:
```
LTDF.columns
```

Out[223]:
```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grade', 'emp_title', 'emp_length', 'home_ownership', 'annu
al_inc', 'verification_status', 'loan_status', 'purpose', 'dti', 'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc',
'application_type', 'mort_acc', 'pub_rec_bankruptcies'], dtype='object')
```

In [224]:
```
LTDF.drop(["mort_acc","pub_rec_bankruptcies"],axis =1, inplace=True)
```

In [225]:
```
LTDF.drop(["pub_rec"],axis =1, inplace=True)
```

In [226]:
```
plt.figure(figsize=(24,15))

sns.heatmap(LTDF.corr(),annot=True,cmap='BrBG_r')
plt.show()
```




## Train-test split :

In [234]:
```
X = LTDF.drop(["loan_status"],axis = 1)
y = LTDF["loan_status"]
```

In [235]:
```
from sklearn.model_selection import train_test_split
```

In [242]:
```
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 [261]:
```
from sklearn.linear_model import LogisticRegression
LR1st = LogisticRegression(class_weight='balanced')
```

In [262]:
```
LR1st.fit(X_train,y_train)
```

Out[262]:
```
▸ LogisticRegression
```

In [263]:
```
LR1st.score(X_test,y_test)
```

Out[263]:
```
0.5601821629519103
```

In [257]:
```
from sklearn.metrics import f1_score,recall_score,precision_score
```

In [264]:
```
f1_score(y_test,LR1st.predict(X_test))
```
Out[264]:
```
0.37774601429356786
```

In [265]:
```
recall_score(y_test,LR1st.predict(X_test))
```
Out[265]:
```
0.6946219167003639
```

In [266]:
```
precision_score(y_test,LR1st.predict(X_test))
```
Out[266]:
```
0.25940803382663846
```

## Standardizing - preprocessing

In [268]:
```
from sklearn.preprocessing import StandardScaler
StandardScaler = StandardScaler()
```

In [269]:
```
StandardScaler.fit(X_train)
```
Out[269]:
```
▸ StandardScaler
```

In [270]:
```
X_train = StandardScaler.transform(X_train)
X_test = StandardScaler.transform(X_test)
```

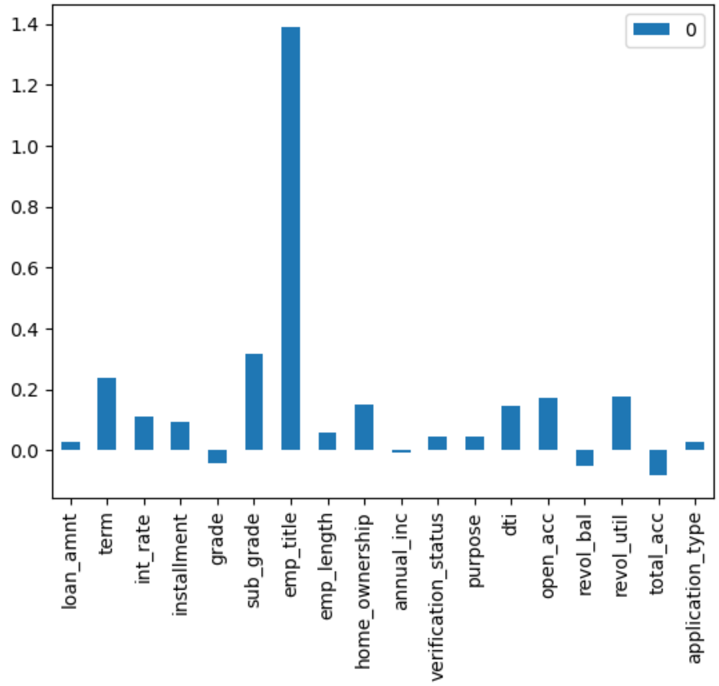
In [271]:
```
from sklearn.linear_model import LogisticRegression
LR_Std = LogisticRegression(C=1.0)
LR_Std.fit(X_train,y_train)
print("Accuracy: ",LR_Std.score(X_test,y_test))
print("f1_score: ",f1_score(y_test,LR_Std.predict(X_test)))
print("recall_score: ",recall_score(y_test,LR_Std.predict(X_test)))
print("precision_score: ",precision_score(y_test,LR_Std.predict(X_test)))
```
```
Accuracy:  0.8677764307252324
f1_score:  0.6057010428736963
recall_score:  0.5284270117266477
precision_score:  0.7094462540716613
```

In [272]:
```
pd.DataFrame(data=LR_Std.coef_,columns=X.columns).T
```
Out[272]:

| | 0 |
|---|---|
| loan_amnt | 0.029073 |
| term | 0.237479 |
| int_rate | 0.111919 |
| installment | 0.094617 |
| grade | -0.041210 |
| sub_grade | 0.318576 |
| emp_title | 1.391447 |
| emp_length | 0.060450 |
| home_ownership | 0.149212 |
| annual_inc | -0.009224 |
| verification_status | 0.044304 |
| purpose | 0.044598 |
| dti | 0.146776 |
| open_acc | 0.171955 |
| revol_bal | -0.050087 |
| revol_util | 0.178362 |
| total_acc | -0.080461 |
| application_type | 0.027765 |

In [273]:
```
pd.DataFrame(data=LR_Std.coef_,columns=X.columns).T.plot(kind ="bar")
```

Out[273]: <Axes: >




## Data Balancing :

In [274]:
```
from imblearn.over_sampling import SMOTE
```

In [275]:
```
SmoteBL = SMOTE(k_neighbors=7)
```

In [276]:
```
X_smote , y_smote = SmoteBL.fit_resample(X_train,y_train)
```

In [277]:
```
X_smote.shape, y_smote.shape
```

Out[277]: ((416188, 18), (416188,))

In [278]:
```
# y_smote.value_counts()
```

In [279]:
```
from sklearn.linear_model import LogisticRegression
```

In [280]:
```
LogReg = LogisticRegression(max_iter= 1000,class_weight="balanced")
```

In [281]:
```
from sklearn.model_selection import cross_val_score
```

In [282]:
```
cross_val_score(estimator = LogReg,
cv=5,
X = X_smote,
y = y_smote,
scoring= "f1"
)
```

Out[282]: array([0.81035184, 0.81622338, 0.81952938, 0.82030926, 0.81703501])

In [283]:
```
cross_val_score(estimator = LogReg,
cv=5,
X = X_smote,
y = y_smote,
scoring= "precision"
)
```

Out[283]: array([0.83180691, 0.83388063, 0.83457467, 0.83418621, 0.83157083])

In [284]:
```
cross_val_score(estimator = LogReg,
cv=5,
X = X_smote,
y = y_smote,
scoring= "accuracy"
)
```

Out[284]:
```
array([0.8151205 , 0.82003412, 0.8227252 , 0.82325168, 0.82017612])
```

In [285]:
```
cross_val_score(estimator = LogReg,
cv=5,
X = X_train,
y = y_train,
scoring= "precision"
)
```

Out[285]:
```
array([0.53069755, 0.53775322, 0.53556687, 0.53524751, 0.52940374])
```

In [286]:
```
from sklearn.linear_model import LogisticRegression
LogReg = LogisticRegression(max_iter=1000,class_weight="balanced")
```

In [287]:
```
LogReg.fit(X= X_train ,y = y_train)
```

Out[287]:
```
LogisticRegression
LogisticRegression(class_weight='balanced', max_iter=1000)
```

In [288]:
```
LogReg.score(X_test,y_test)
```

Out[288]:
```
0.826307936211881
```

In [289]:
```
LogReg.coef_.round(2)
```

Out[289]:
```
array([[ 0.08,  0.22, -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 [290]:
```
from sklearn.metrics import confusion_matrix, f1_score, precision_score,recall_score
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)))
```

```
[[43363  8610]
 [ 2565  9800]]
0.532319391634981
0.7925596441568945
0.6368805848903332
```

In [291]:
```
LogReg.coef_
```

Out[291]:
```
array([[ 0.07752502,  0.21645772, -0.07393633,  0.05412738, -0.04786132,
         0.53236815,  1.46811872,  0.05193312,  0.13760179,  0.01138659,
         0.05617464,  0.04580286,  0.16649176,  0.16001178, -0.05509122,
         0.15538849, -0.06998371,  0.03507524]])
```

In [292]:
```
LTDF.drop(["loan_status"], axis =1).columns
```

Out[292]:
```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grade', 'emp_title', 'emp_length', 'home_ownership', 'annu
al_inc', 'verification_status', 'purpose', 'dti', 'open_acc', 'revol_bal', 'revol_util', 'total_acc', 'application_type'], dtyp
e='object')
```

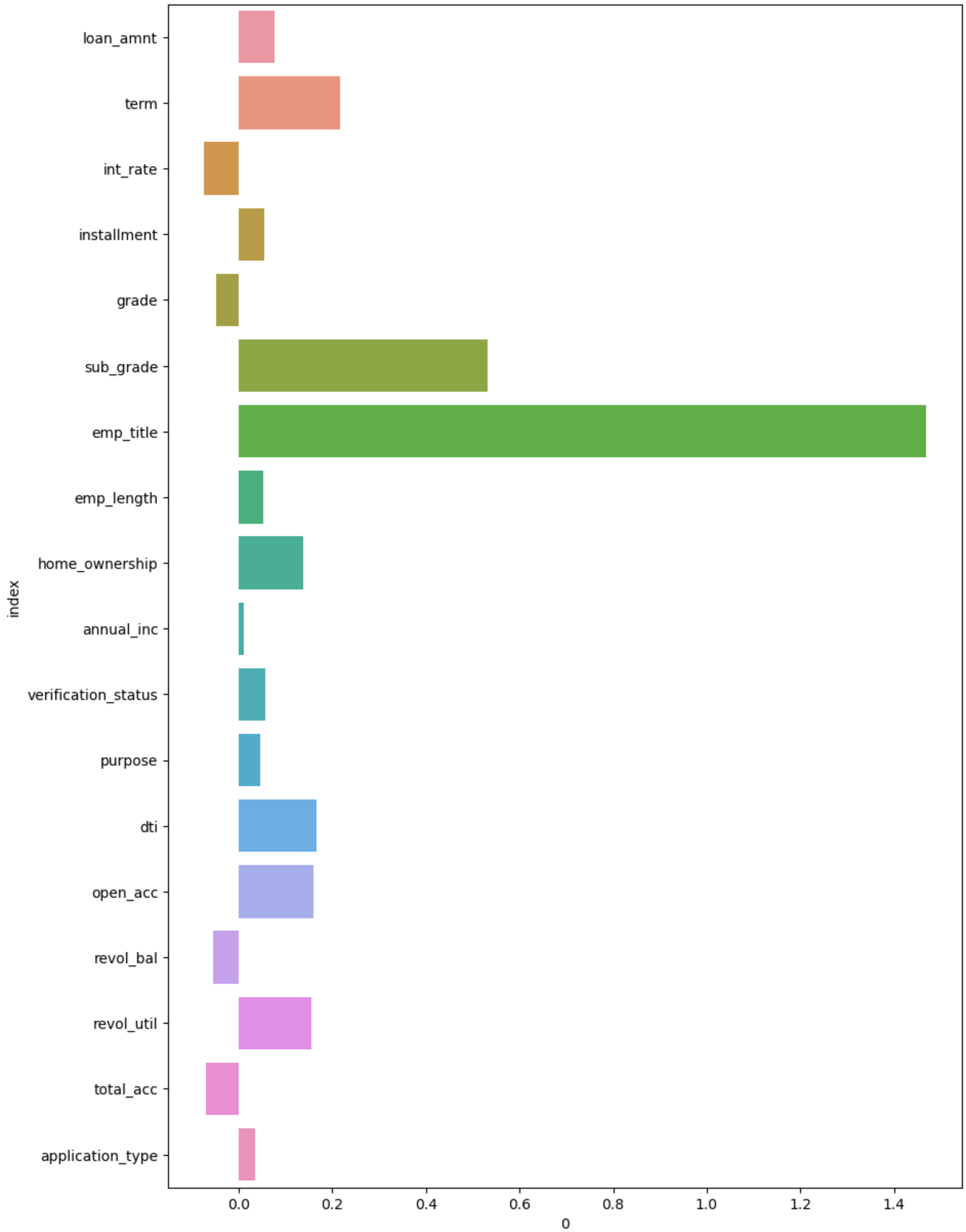
In [294]:
```
feature_importance = pd.DataFrame(index = LTDF.drop(["loan_status"],axis =1).columns,data = LogReg.coef_.ravel()).reset_index()
```

In [295]:
```
feature_importance
```

Out[295]:

| | index | 0 |
|---|---|---|
| 0 | loan_amnt | 0.077525 |
| 1 | term | 0.216458 |
| 2 | int_rate | -0.073936 |
| 3 | installment | 0.054127 |
| 4 | grade | -0.047861 |
| 5 | sub_grade | 0.532368 |
| 6 | emp_title | 1.468119 |
| 7 | emp_length | 0.051933 |
| 8 | home_ownership | 0.137602 |
| 9 | annual_inc | 0.011387 |
| 10 | verification_status | 0.056175 |
| 11 | purpose | 0.045803 |
| 12 | dti | 0.166492 |
| 13 | open_acc | 0.160012 |
| 14 | revol_bal | -0.055091 |
| 15 | revol_util | 0.155388 |
| 16 | total_acc | -0.069984 |
| 17 | application_type | 0.035075 |

In [298]:
```
plt.figure(figsize=(10,15))

sns.barplot(y = feature_importance["index"],x = feature_importance[0])
plt.show()
```

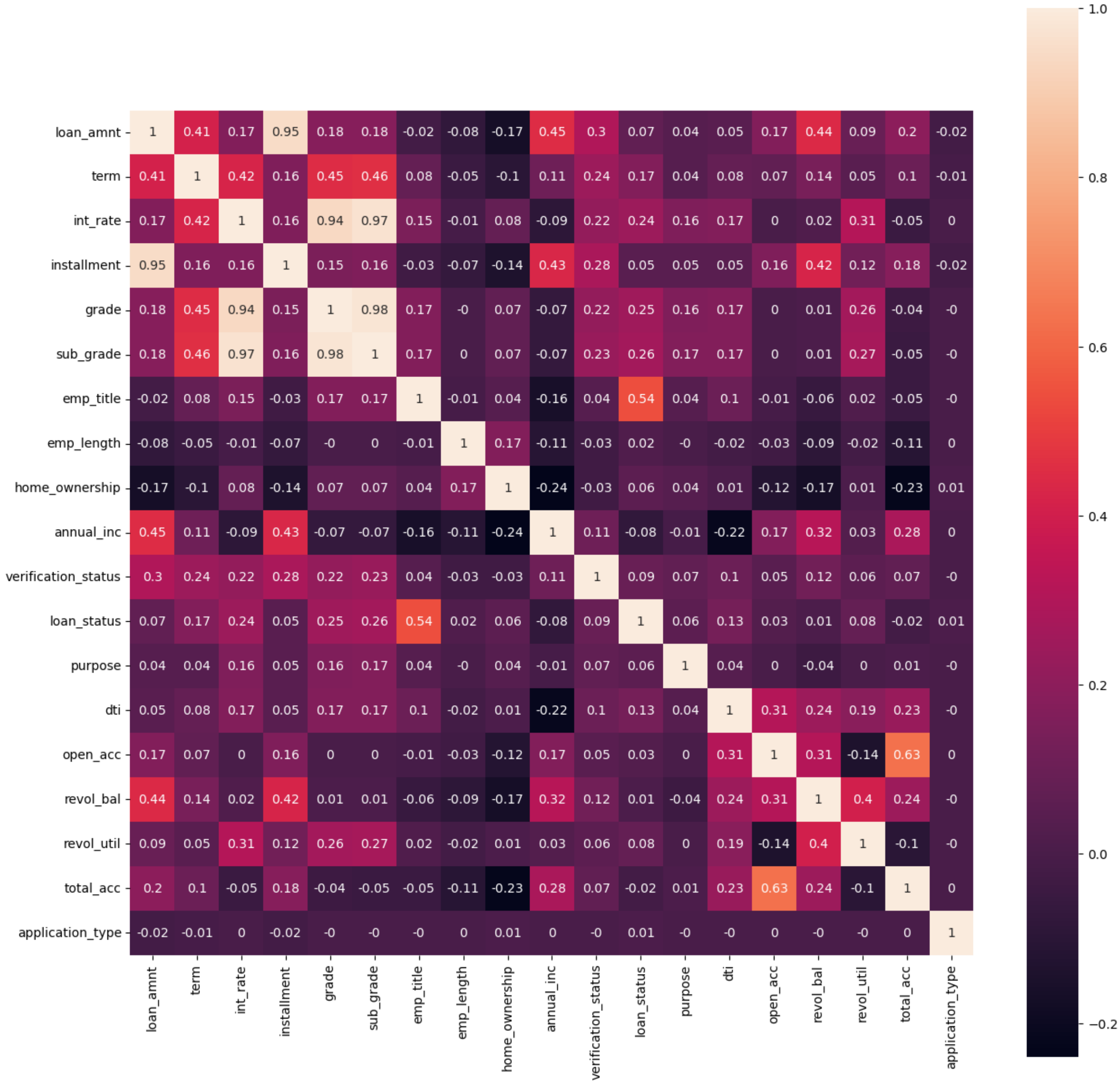



In [299]:
```
LogReg.score(X_train,y_train)
```

Out[299]: 0.8278608898387411

In [300]:
```
LogReg.score(X_test,y_test)
```
Out[300]:
```
0.826307936211881
```

In [303]:
```
plt.figure(figsize=(15,15))

sns.heatmap(LTDF.corr().round(2),annot=True,square=True)
plt.show()
```




## Metrics :

In [304]:
```
from sklearn.metrics import confusion_matrix, f1_score, precision_score,recall_score
```

In [305]:
```
confusion_matrix(y_test, LogReg.predict(X_test))
```
Out[305]:
```
array([[43363,  8610],
       [ 2565,  9800]], dtype=int64)
```

In [306]:
```
precision_score(y_test ,LogReg.predict(X_test))
```
Out[306]:
```
0.532319391634981
```

In [307]:
```
recall_score(y_test ,LogReg.predict(X_test))
```

Out[307]: 0.7925596441568945

In [308]:
```
pd.crosstab(y_test ,LogReg.predict(X_test))
```

Out[308]:

| col_0<br>loan_status | 0 | 1 |
|---|---|---|
| 0 | 43363 | 8610 |
| 1 | 2565 | 9800 |

In [309]:
```
recall_score(y_train ,LogReg.predict(X_train))
```

Out[309]: 0.7932028585350008

In [310]:
```
recall_score(y_test ,LogReg.predict(X_test))
```

Out[310]: 0.7925596441568945

In [311]:
```
f1_score(y_test ,LogReg.predict(X_test))
```

Out[311]: 0.6368805848903332

In [312]:
```
f1_score(y_train ,LogReg.predict(X_train))
```

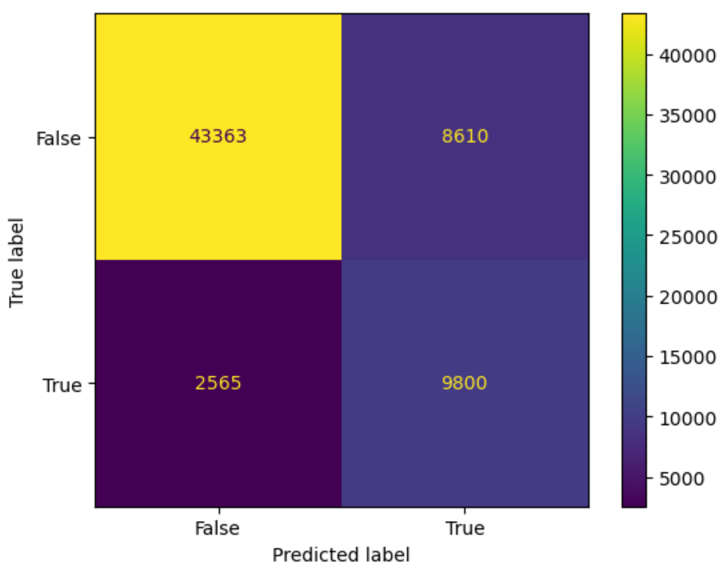
Out[312]: 0.6381901339431559

In [314]:
```
from sklearn.metrics import ConfusionMatrixDisplay
```

In [315]:
```
from sklearn.metrics import fbeta_score
```

In [317]:
```
m_display = ConfusionMatrixDisplay(confusion_matrix= confusion_matrix(y_test,LogReg.predict(X_test)),display_labels=[False,True])
```

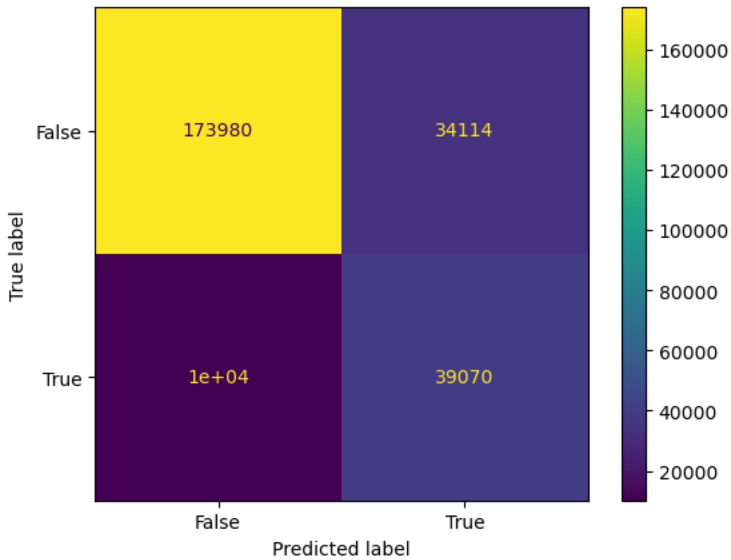
In [319]:
```
cm_display.plot()
plt.show()
```




In [320]:
```
# fbeta_score
```

In [321]:
```
cm_display = ConfusionMatrixDisplay(confusion_matrix= confusion_matrix(y_train,LogReg.predict(X_train)),display_labels=[False,Tru
```

In [323]:
```
cm_display.plot()
plt.show()
```




In [324]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [325]:
```
DecisionTreeClassifier = DecisionTreeClassifier(max_depth=5, splitter="best",criterion="entropy",class_weight ="balanced")
```

In [326]:
```
DecisionTreeClassifier.fit(X_train,y_train)
```

Out[326]:
```
DecisionTreeClassifier
DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
                       max_depth=5)
```

In [327]:
```
DecisionTreeClassifier.score(X_test,y_test)
```

Out[327]: 0.7946936491653456

In [328]:
```
# DecisionTreeClassifier.score(X_smote,y_smote)
```

In [329]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [330]:
```
RF = RandomForestClassifier(n_estimators=30,max_depth=10,class_weight="balanced")
```

In [331]:
```
RF.fit(X_train,y_train)
```

Out[331]:
```
RandomForestClassifier
RandomForestClassifier(class_weight='balanced', max_depth=10, n_estimators=30)
```

In [332]:
```
RF.score(X_test,y_test)
```

Out[332]: 0.8115266250116572

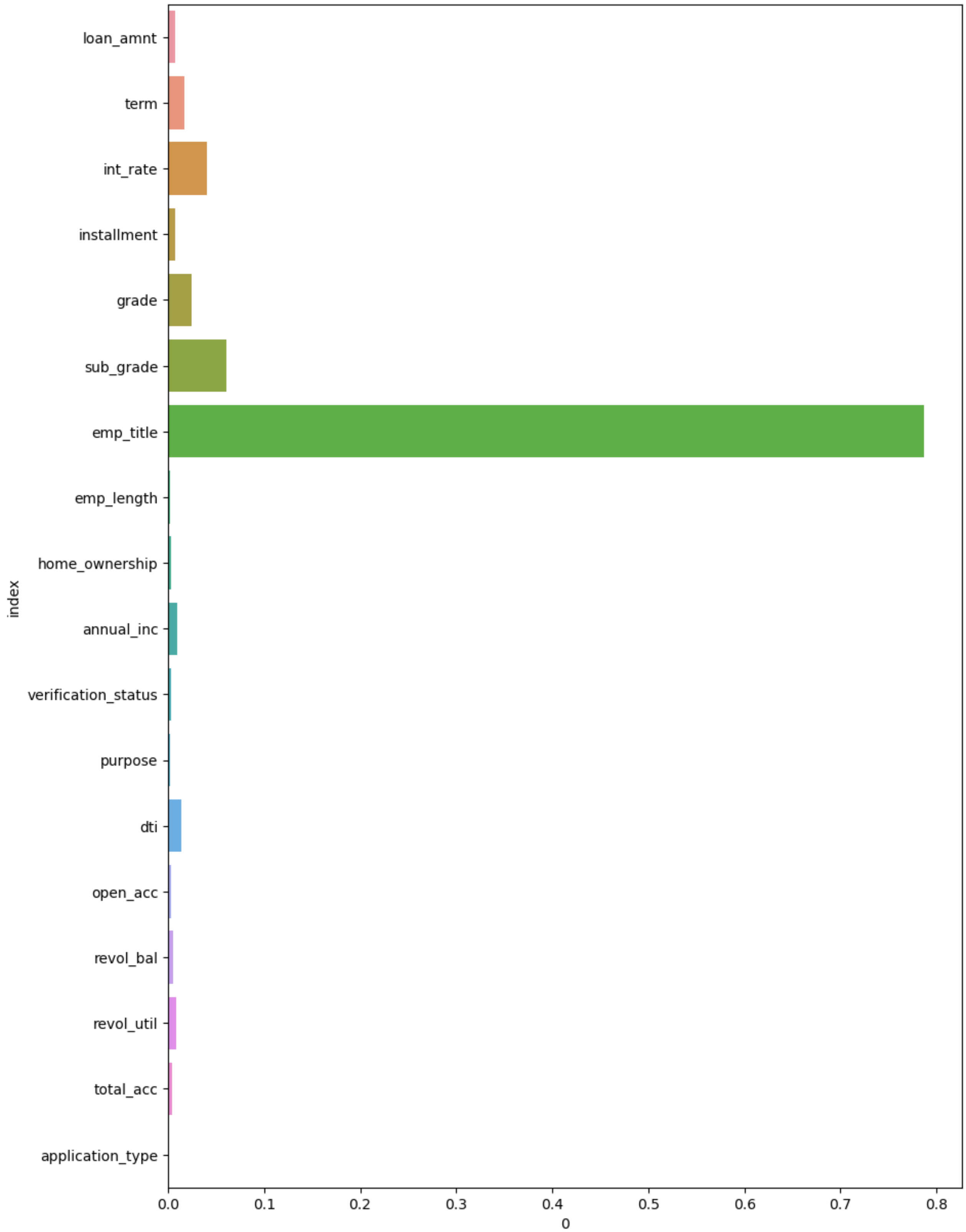
In [334]:
```
feature_importance = pd.DataFrame(index = LTDF.drop(["loan_status"],axis = 1).columns,data = RF.feature_importances_.ravel()).res
```

In [335]:
```
feature_importance
```

Out[335]:

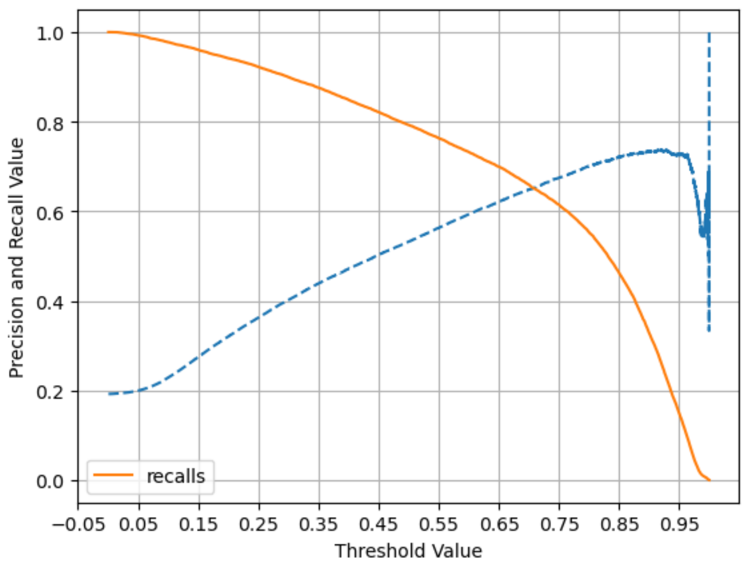
| | index | 0 |
|---|---|---|
| 0 | loan_amnt | 0.007324 |
| 1 | term | 0.017251 |
| 2 | int_rate | 0.040163 |
| 3 | installment | 0.007328 |
| 4 | grade | 0.024697 |
| 5 | sub_grade | 0.061272 |
| 6 | emp_title | 0.787078 |
| 7 | emp_length | 0.002030 |
| 8 | home_ownership | 0.003052 |
| 9 | annual_inc | 0.009833 |
| 10 | verification_status | 0.003085 |
| 11 | purpose | 0.001755 |
| 12 | dti | 0.013996 |
| 13 | open_acc | 0.003635 |
| 14 | revol_bal | 0.005292 |
| 15 | revol_util | 0.008277 |
| 16 | total_acc | 0.003844 |
| 17 | application_type | 0.000087 |

In [337]:
```
plt.figure(figsize=(10,15))

sns.barplot(y = feature_importance["index"],x = feature_importance[0])
plt.show()
```

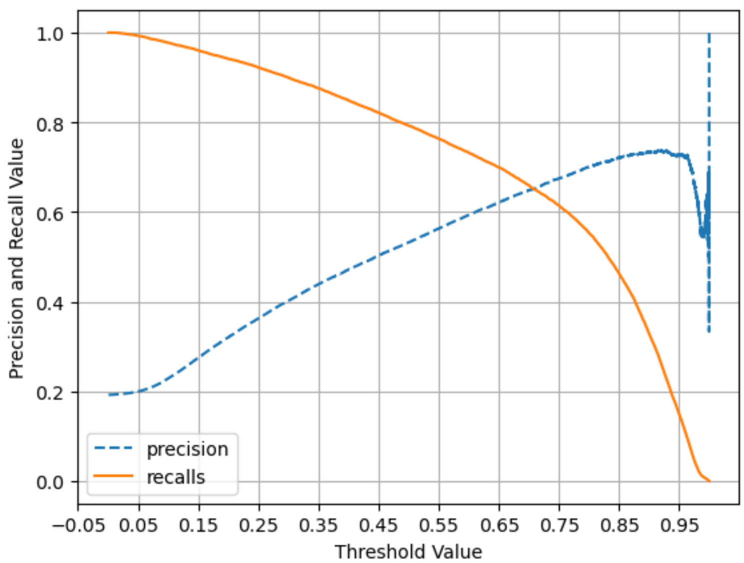



In [338]:
```
from sklearn.metrics import precision_recall_curve
```

In [340]:
```
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='--')

    # 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, LogReg.predict_proba(X_test)[:,1])
```

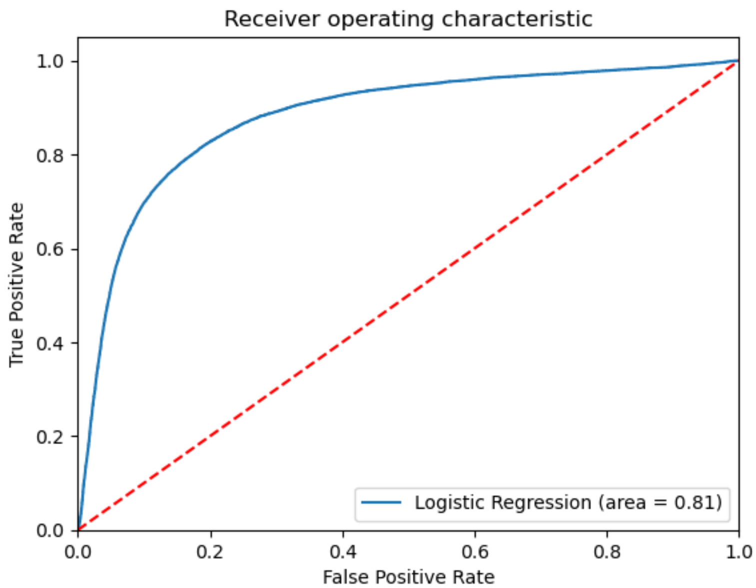


In [341]:
```
def precision_recall_curve_plot(y_test,pred_proba_c1):

    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='precision')

    # 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, LogReg.predict_proba(X_test)[:,1])
```




In [342]:
```
from sklearn.metrics import roc_auc_score,roc_curve
```

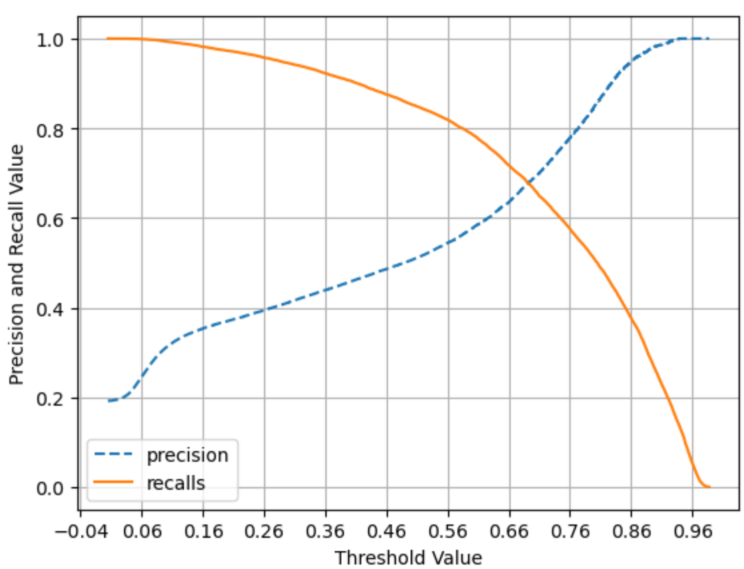
In [343]:
```
logit_roc_auc = roc_auc_score(y_test, LogReg.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, LogReg.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)'% logit_roc_auc)
plt.plot([0,1], [0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
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()
```



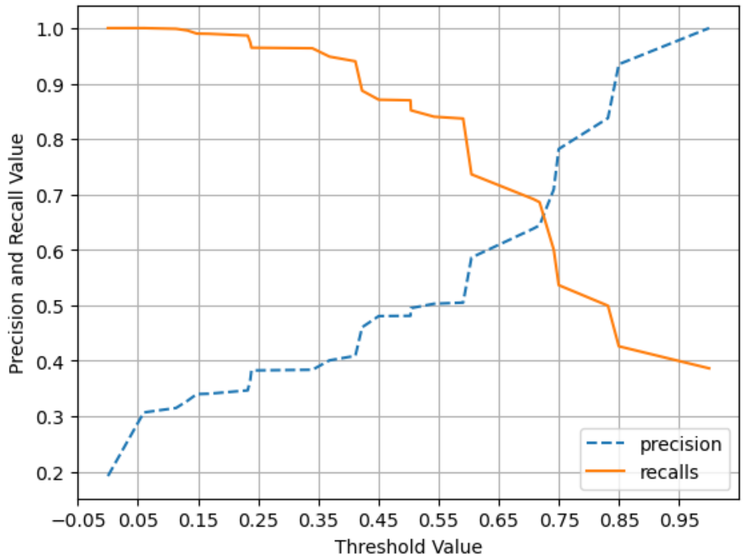

In [344]:
```
LogReg.predict_proba(X_test)
```

Out[344]:
```
array([[0.73857178, 0.26142822],
       [0.85552479, 0.14447521],
       [0.53519655, 0.46480345],
       ...,
       [0.61316038, 0.38683962],
       [0.80044351, 0.19955649],
       [0.72890824, 0.27109176]])
```

In [345]:
```
precision_recall_curve_plot(y_test, RF.predict_proba(X_test)[:,1])
```



In [346]:
```
precision_recall_curve_plot(y_test, DecisionTreeClassifier.predict_proba(X_test)[:,1])
```




In [347]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(class_weight="balanced")
model.fit(X_train, y_train)
```

Out[347]:
```
LogisticRegression
LogisticRegression(class_weight='balanced')
```

In [348]:
```
def custom_predict(X, threshold):
    probs = model.predict_proba(X)
    return (probs[:, 1] > threshold).astype(int)
```

In [349]:
```
new_preds = custom_predict(X=X_test, threshold=0.75)
```

In [350]:
```
model.score(X_test,y_test)
```

Out[350]: 0.826307936211881

In [351]:
```
precision_score(y_test,new_preds)
```

Out[351]: 0.6748379074518164

# 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 , verifi ed income borrowers probability of defaulter is higher than those who are not verifi ed by loantap.

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 fi le is same as for loan status as fully paid and defaulters.

Number of derogatory public records increases , the probability of borrowers declared as defaulters alsoincreases 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 classifi er and RandomForest model are : Employee Title, Loan Grade and Sub-Grade, Interest rate and dept-to-income ratio.

# 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 fi nance more individuals and earn interest on it. This wecan achieve by setting up the higher threshold.

Giving loans to those even having slightly higher probability of defaulter, we can maximise the earning , by thisrisk taking method.

and Since NPA is a real problem in the industry , Company should more investigate and check for the proof ofassets. Since it was observed in probability plot, verifi ed borrowers had higher probability of defaulters thannon-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 [ ]: