Problem Statement

LoanTap is an online platform committed to delivering customized loan products to millennials. They innovate in an otherwise dull loan segment, to deliver instant, flexible loans on consumer friendly terms to salaried professionals and businessmen.

LoanTap deploys formal credit to salaried individuals and businesses by 4 main financial instruments:

- Personal Loan
- EMI Free Loan
- Personal Overdraft
- Advance Salary Loan

LoanTap wants to build an underwriting layer to determine the creditworthiness of MSMEs as well as individuals. This case study will focus on the underwriting process behind Personal Loan only. 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?

Data dictionary:

- 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.
- term : The number of payments on the loan. Values are in months and can be either 36 or 60.
- int_rate : Interest Rate on the loan
- installment : The monthly payment owed by the borrower if the loan originates.
- grade : LoanTap assigned loan grade
- sub_grade : LoanTap assigned loan subgrade
- emp_title :The job title supplied by the Borrower when applying for the loan.*
- 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.
- home_ownership : The home ownership status provided by the borrower during registration or obtained from the credit report.
- annual_inc : The self-reported annual income provided by the borrower during registration.
- verification_status : Indicates if income was verified by LoanTap, not verified, or if the income source was verified
- issue_d : The month which the loan was funded
- loan_status : Current status of the loan - Target Variable
- purpose : A category provided by the borrower for the loan request.
- title : The loan title provided by the borrower
- 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-reported monthly income.
- earliest_cr_line :The month the borrower's earliest reported credit line was opened
- open_acc : The number of open credit lines in the borrower's credit file.
- pub_rec : Number of derogatory public records
- revol_bal : Total credit revolving balance
- revol_util : Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.
- total_acc : The total number of credit lines currently in the borrower's credit file
- initial_list_status : The initial listing status of the loan. Possible values are – W, F
- application_type : Indicates whether the loan is an individual application or a joint application with two co-borrowers
- mort_acc : Number of mortgage accounts.
- pub_rec_bankruptcies : Number of public record bankruptcies
- Address: Address of the individual

Loading dependencies and dataset

In [1]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

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

pd.set_option('display.max_columns', 500)

# from scipy.stats import levene, f_oneway, kruskal
# from scipy.stats import ttest_ind
# from scipy.stats import chi2_contingency
# from statsmodels.graphics.gofplots import qqplot

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from category_encoders import TargetEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.impute import KNNImputer

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

from sklearn.linear_model import LogisticRegression
from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import roc_curve
```

In [2]:
```
df = pd.read_csv('./data/loantap.csv')
df.head()
```

Out[2]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	117000.0
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000.0
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RENT	43057.0
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	RENT	54000.0
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000.0

Basic checks on data

In [3]:
```
df.shape
```

Out[3]:
```
(396030, 27)
```

In [4]:
```
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
```

Missing values

In [5]:
```python
(100*df.isna().sum()/df.shape[0]).sort_values(ascending=False)
```

Out[5]:
```
mort_acc                9.543469
emp_title               5.789208
emp_length              4.621115
title                   0.443148
pub_rec_bankruptcies    0.135091
revol_util              0.069692
loan_amnt               0.000000
dti                     0.000000
application_type        0.000000
initial_list_status     0.000000
total_acc               0.000000
revol_bal               0.000000
pub_rec                 0.000000
open_acc                0.000000
earliest_cr_line        0.000000
purpose                 0.000000
term                    0.000000
loan_status             0.000000
issue_d                 0.000000
verification_status     0.000000
annual_inc              0.000000
home_ownership          0.000000
sub_grade               0.000000
grade                   0.000000
installment             0.000000
int_rate                0.000000
address                 0.000000
dtype: float64
```

Descriptive Metrics across each Feature

In [6]:
```python
df.describe(include='O')
```

Out[6]:

	term	grade	sub_grade	emp_title	emp_length	home_ownership	verification_status	issue_d	loan_status
count	396030	396030	396030	373103	377729	396030	396030	396030	396030
unique	2	7	35	173105	11	6	3	115	2
top	36 months	B	B3	Teacher	10+ years	MORTGAGE	Verified	Oct-2014	Fully Paid
freq	302005	116018	26655	4389	126041	198348	139563	14846	318357

In [7]:

```
df.describe().round(1)
```

Out[7]:

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	m
count	396030.0	396030.0	396030.0	396030.0	396030.0	396030.0	396030.0	396030.0	395754.0	396030.0	3
mean	14113.9	13.6	431.8	74203.2	17.4	11.3	0.2	15844.5	53.8	25.4	
std	8357.4	4.5	250.7	61637.6	18.0	5.1	0.5	20591.8	24.5	11.9	
min	500.0	5.3	16.1	0.0	0.0	0.0	0.0	0.0	0.0	2.0	
25%	8000.0	10.5	250.3	45000.0	11.3	8.0	0.0	6025.0	35.8	17.0	
50%	12000.0	13.3	375.4	64000.0	16.9	10.0	0.0	11181.0	54.8	24.0	
75%	20000.0	16.5	567.3	90000.0	23.0	14.0	0.0	19620.0	72.9	32.0	
max	40000.0	31.0	1533.8	8706582.0	9999.0	90.0	86.0	1743266.0	892.3	151.0	

EDA: Target Column & identifying Similar Independent Features

Target Column

In [8]:

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

Out[8]:

```
Fully Paid     80.387092
Charged Off    19.612908
Name: loan_status, dtype: float64
```

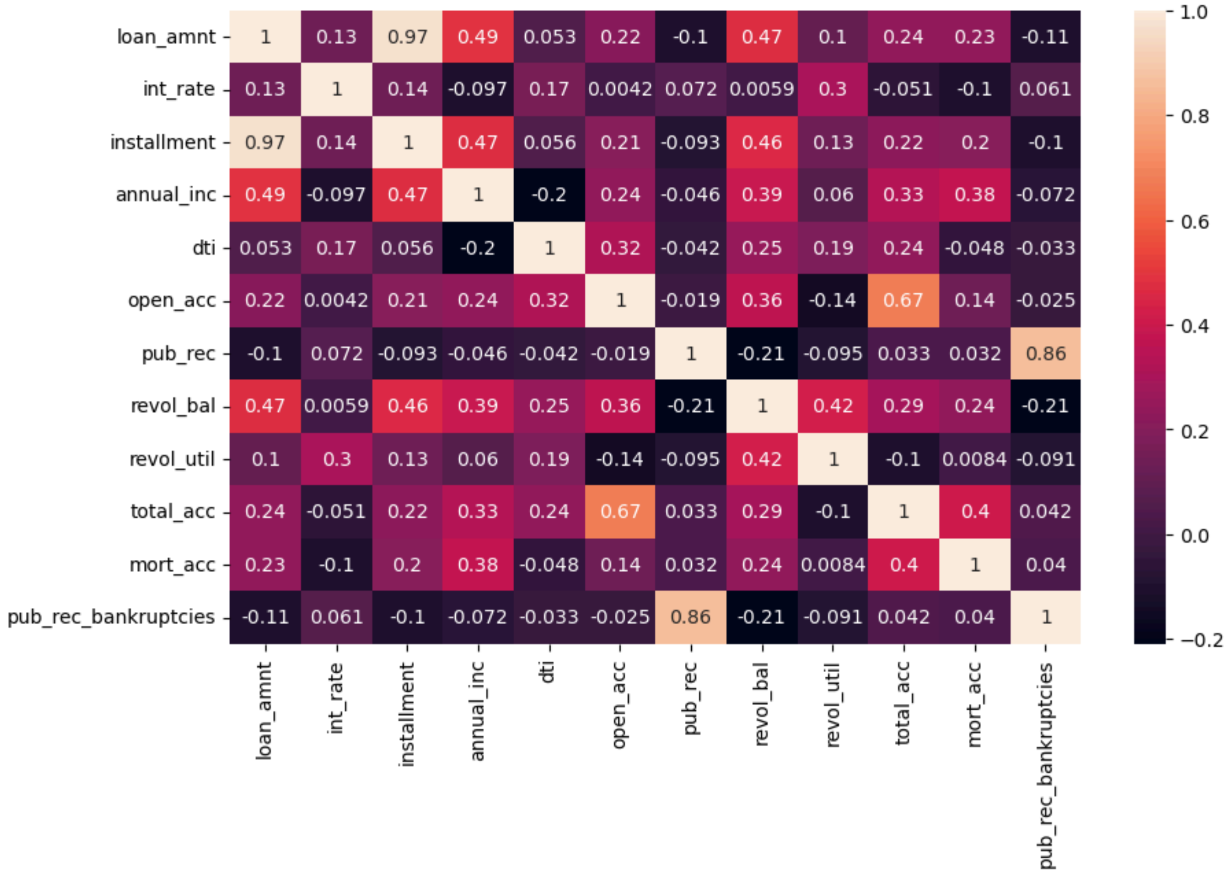
As we can see, there is an imbalance in the data.

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

Similar Independent Features

In [9]:

```
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(method='spearman'), annot=True)
plt.show()
```



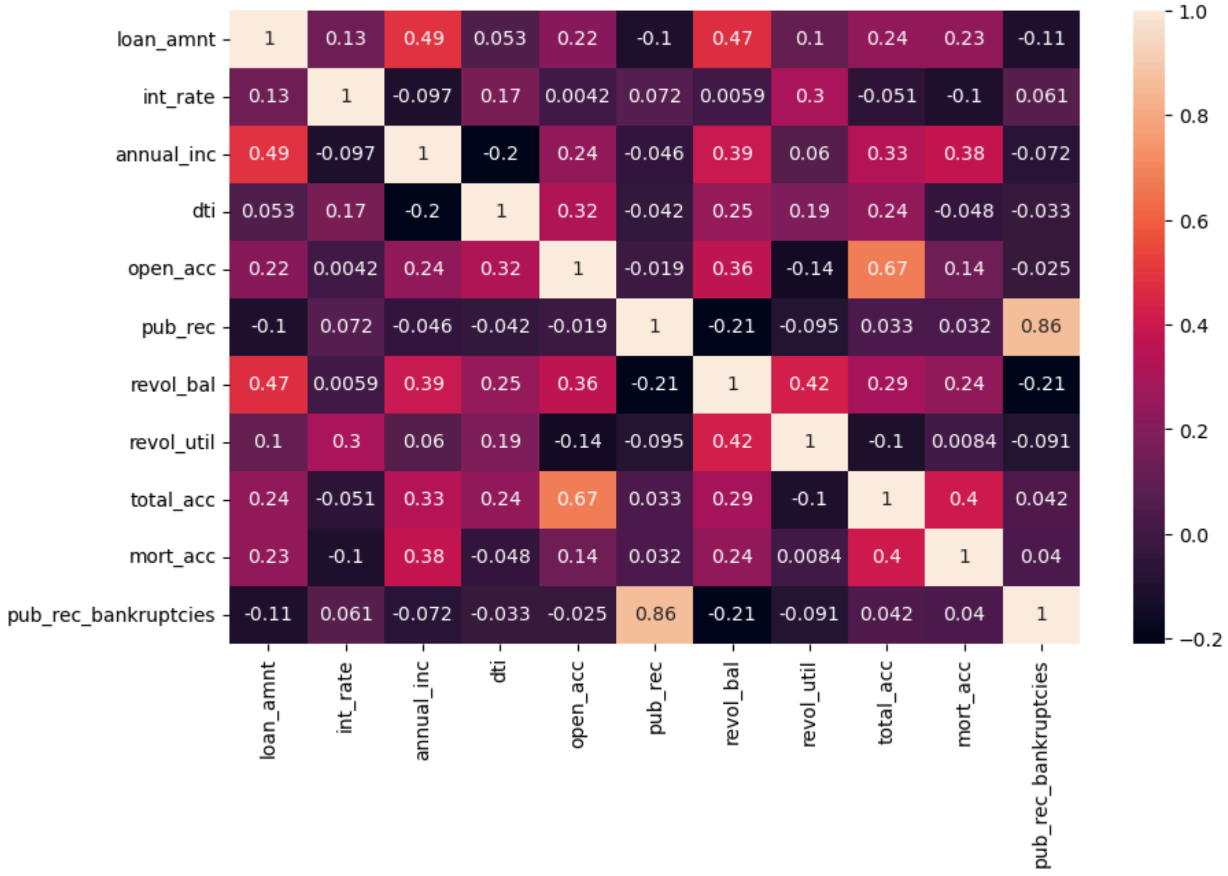
Observations:

- Loan Amount and Installment are very similar to each other --> Thus dropping the feature Installment.

In [10]:
```
df = df.drop('installment', axis=1).copy()
df.shape
```

Out[10]:
```
(396030, 26)
```

In [11]:
```
# Correlation matrix after dropping Installment
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(method='spearman'), annot=True)
plt.show()
```



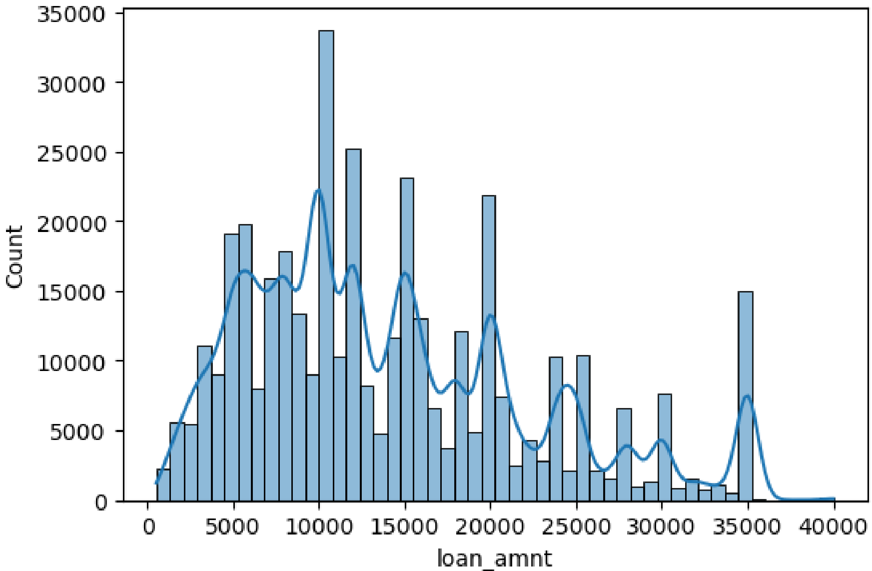
In [12]:
```
# Categorical vs Numerical Columns
cat_cols = df.dtypes.loc[df.dtypes=='object'].index
num_cols = df.columns[~df.columns.isin(cat_cols)]
print('-'*50)
print('Total categorical columns:', cat_cols.shape[0])
print('Total numerical columns:', num_cols.shape[0])
print('-'*50)
```

```
--------------------------------------------------
Total categorical columns: 15
Total numerical columns: 11
--------------------------------------------------
```

EDA: Numerical Columns

Loan Amount

In [13]:
```
plt.figure(figsize=(6, 4))
sns.histplot(df['loan_amnt'], bins=50, kde=True)
plt.show()
```

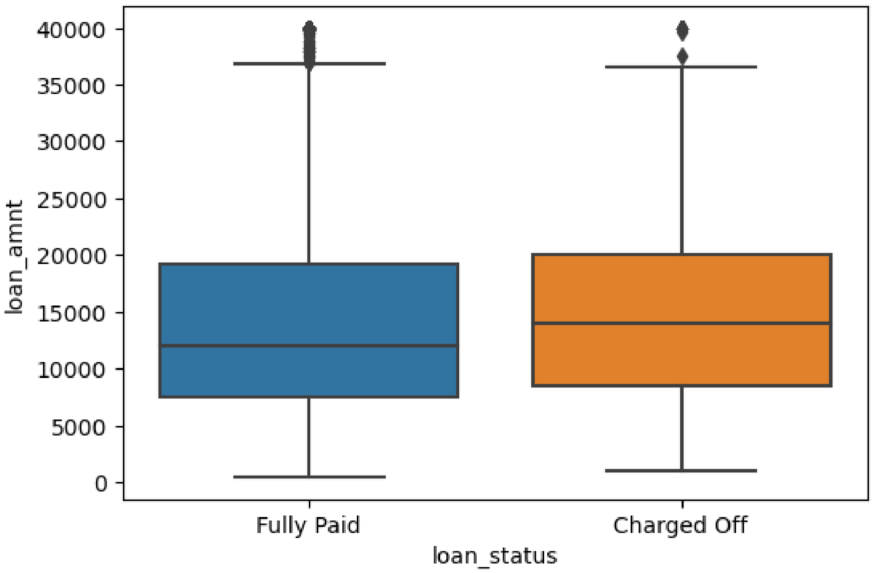



```
In [14]: df.groupby('loan_status')['loan_amnt'].describe()
```

Out[14]:

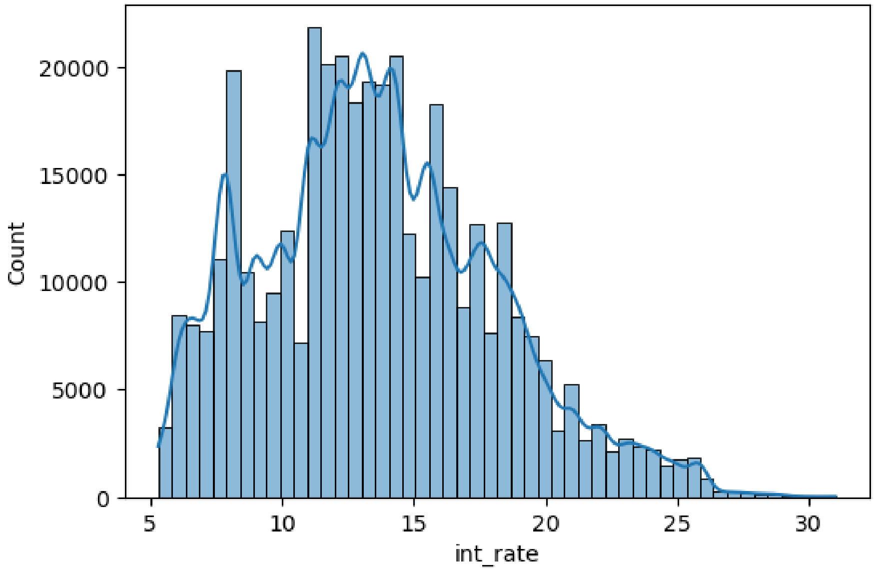
loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	15126.300967	8505.090557	1000.0	8525.0	14000.0	20000.0	40000.0
Fully Paid	318357.0	13866.878771	8302.319699	500.0	7500.0	12000.0	19225.0	40000.0

```
In [15]: plt.figure(figsize=(6, 4))
sns.boxplot(x=df['loan_status'], y=df['loan_amnt'])
plt.show()
```



Interest Rate

```
In [16]: plt.figure(figsize=(6, 4))
sns.histplot(df['int_rate'], bins=50, kde=True)
plt.show()
```

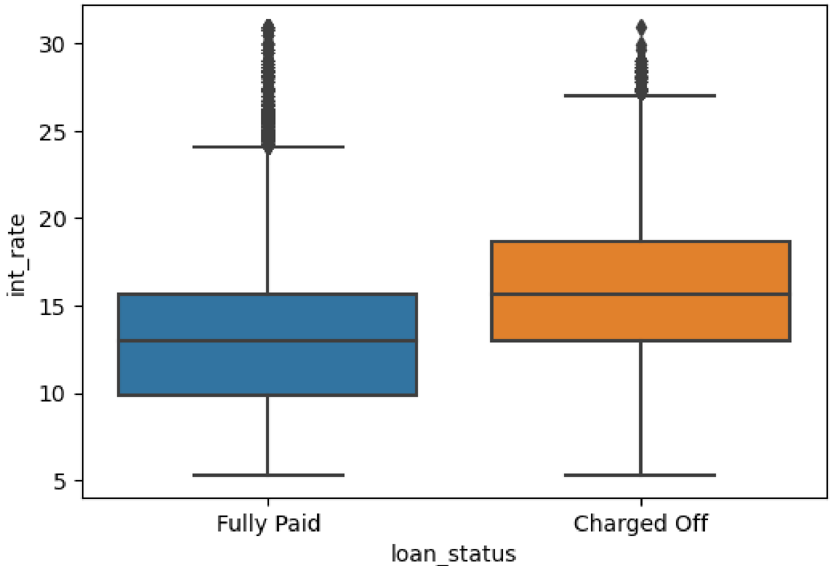


In [17]:
```
df.groupby('loan_status')["int_rate"].describe()
```

Out[17]:

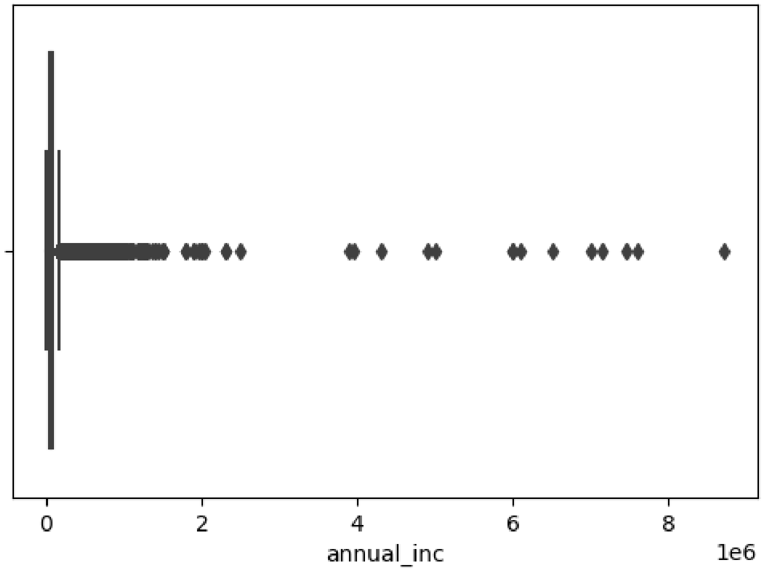
loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	15.882587	4.388135	5.32	12.99	15.61	18.64	30.99
Fully Paid	318357.0	13.092105	4.319105	5.32	9.91	12.99	15.61	30.99

In [18]:
```
plt.figure(figsize=(6, 4))
sns.boxplot(x=df['loan_status'], y=df['int_rate'])
plt.show()
```

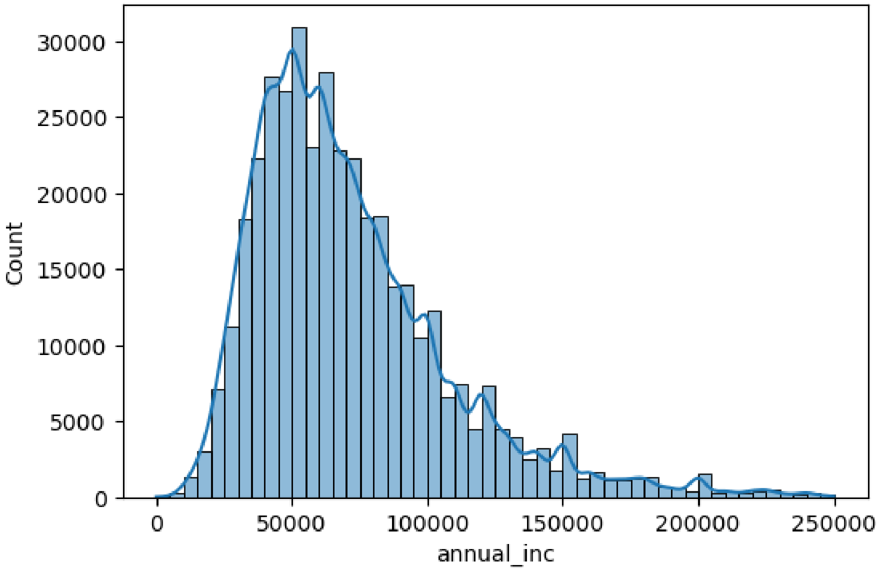


Annual Income

In [19]:
```
plt.figure(figsize=(6, 4))
sns.boxplot(x=df['annual_inc'])
plt.show()
```



In [20]:
```
plt.figure(figsize=(6, 4))
sns.histplot(df['annual_inc'].loc[df['annual_inc']<df['annual_inc'].quantile(0.99)], bins=50, kde=
plt.show()
```

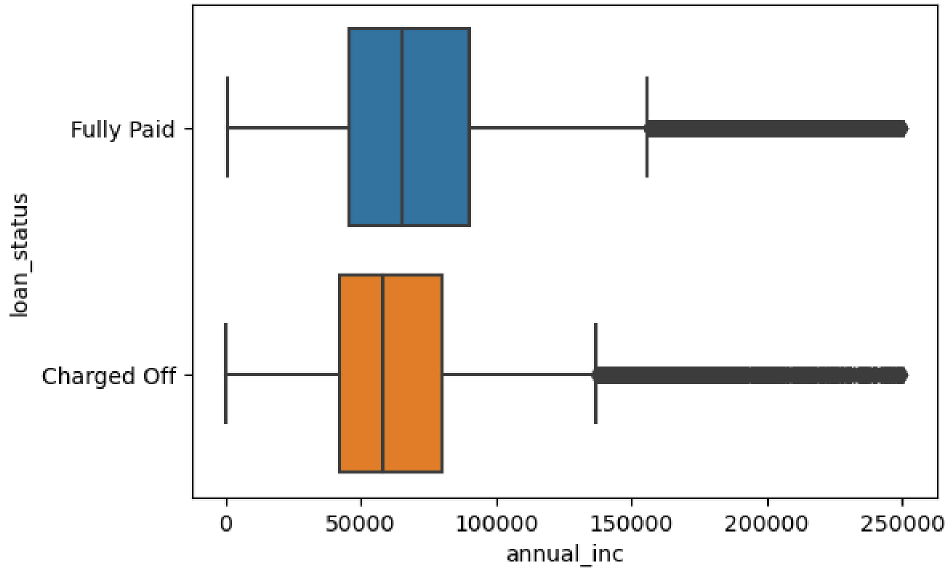


In [21]:
```
df.groupby('loan_status')["annual_inc"].describe()
```

Out[21]:

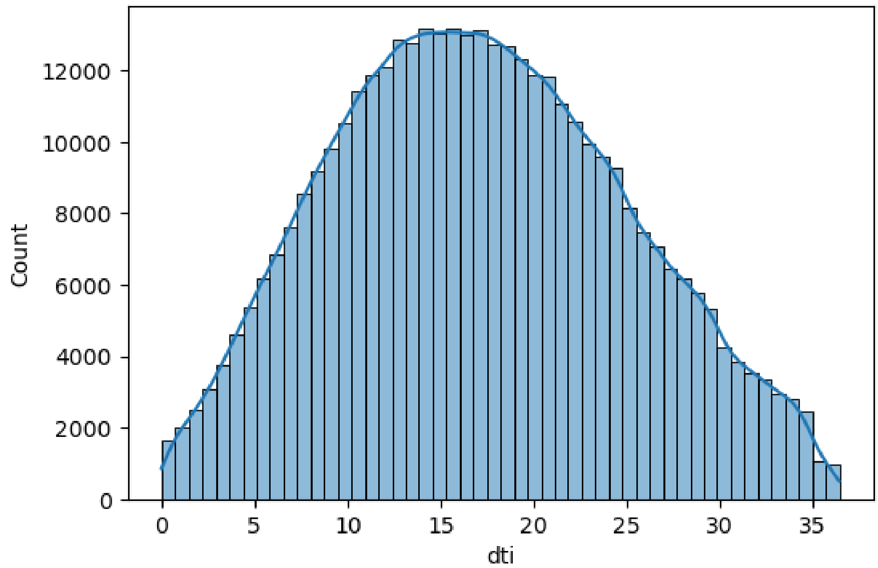
loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	67535.537710	58303.457136	0.0	42000.00	59000.0	80000.0	8706582.0
Fully Paid	318357.0	75829.951566	62315.991907	600.0	46050.53	65000.0	90000.0	7600000.0

In [22]:
```
plt.figure(figsize=(6,4))
sns.boxplot(x=df['annual_inc'].loc[df['annual_inc']<df['annual_inc'].quantile(0.99)], y=df["loan_s
plt.show()
```



Debt to Income Ratio

In [23]:
```
plt.figure(figsize=(6, 4))
sns.histplot(df['dti'].loc[df['dti']<df['dti'].quantile(0.99)], bins=50, kde=True)
plt.show()
```

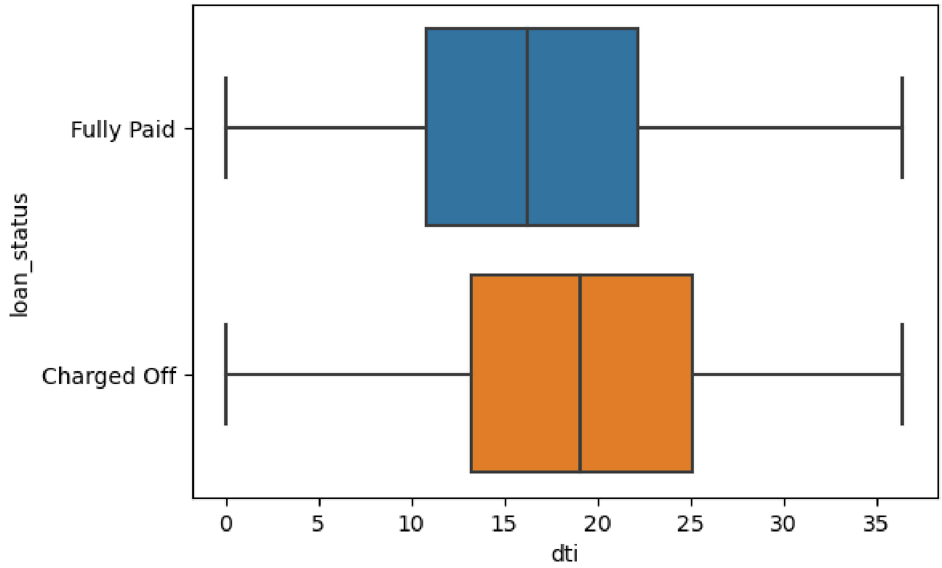


In [24]:
```
df.groupby('loan_status')["dti"].describe()
```

Out[24]:

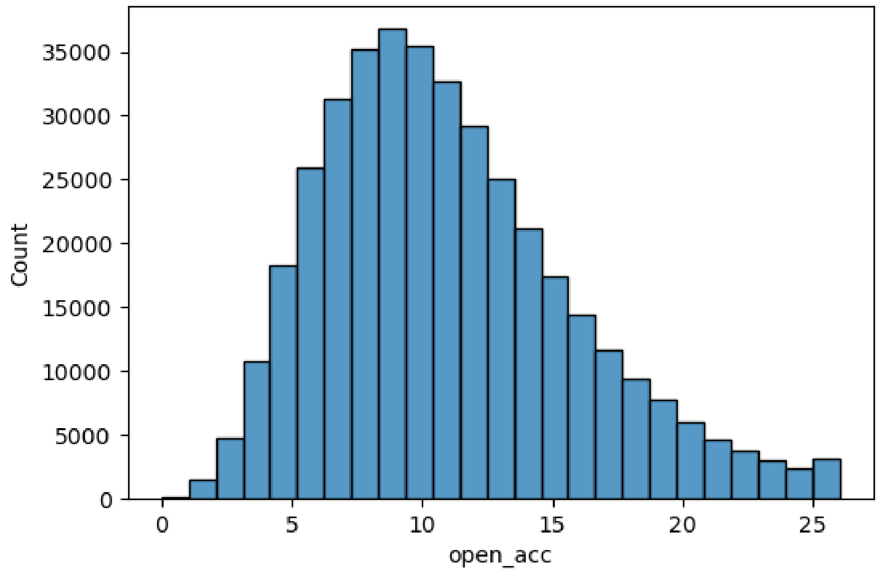
loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	19.656346	36.781068	0.0	13.33	19.34	25.55	9999.0
Fully Paid	318357.0	16.824010	8.500979	0.0	10.87	16.34	22.29	1622.0

In [25]:
```
plt.figure(figsize=(6,4))
sns.boxplot(x=df['dti'].loc[df['dti']<df['dti'].quantile(0.99)], y=df["loan_status"])
plt.show()
```



Total Open Credit Lines

In [26]:
```
plt.figure(figsize=(6, 4))
sns.histplot(df['open_acc'].loc[df['open_acc']<df['open_acc'].quantile(0.99)], bins=25)
plt.show()
```

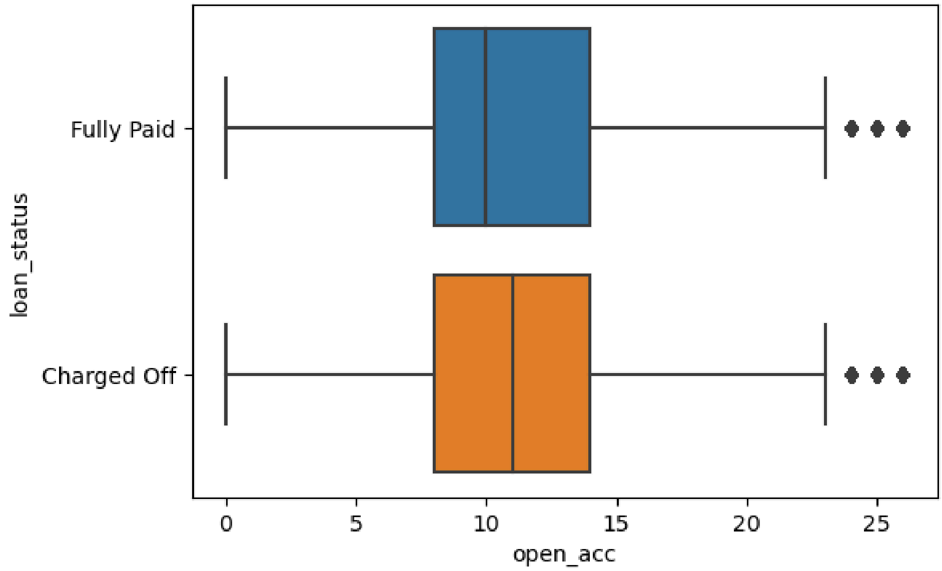


In [27]:
```
df.groupby('loan_status')["open_acc"].describe()
```

Out[27]:

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 [28]:
```
plt.figure(figsize=(6,4))
sns.boxplot(x=df['open_acc'].loc[df['open_acc']<df['open_acc'].quantile(0.99)], y=df["loan_status"]
plt.show()
```

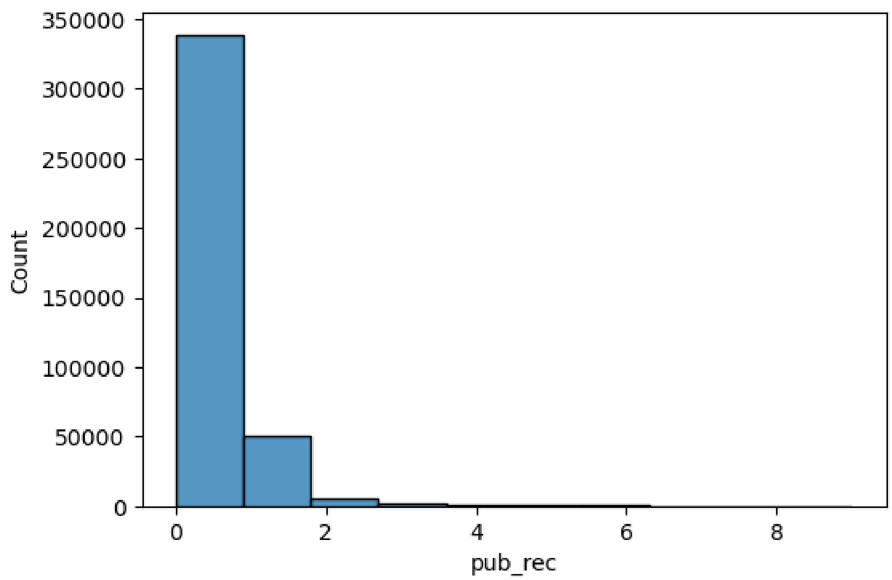


Derogatory Public Records

In [29]:
```
df['pub_rec'].value_counts().sort_index()
```

Out[29]:
```
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
12.0          4
13.0          4
15.0          1
17.0          1
19.0          2
24.0          1
40.0          1
86.0          1
Name: pub_rec, dtype: int64
```

In [30]:
```
plt.figure(figsize=(6, 4))
sns.histplot(df['pub_rec'].loc[df['pub_rec']<10], bins=10)
plt.show()
```



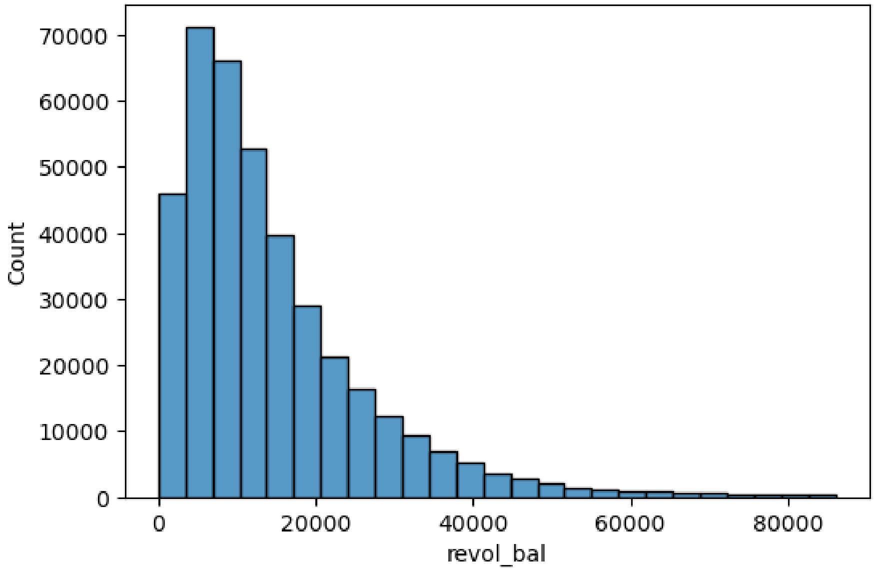
In [31]:
```
df.groupby('loan_status')["pub_rec"].describe()
```

Out[31]:

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

Revolving Balance

In [32]:
```
plt.figure(figsize=(6, 4))
sns.histplot(df['revol_bal'].loc[df['revol_bal']<df['revol_bal'].quantile(0.99)], bins=25)
plt.show()
```

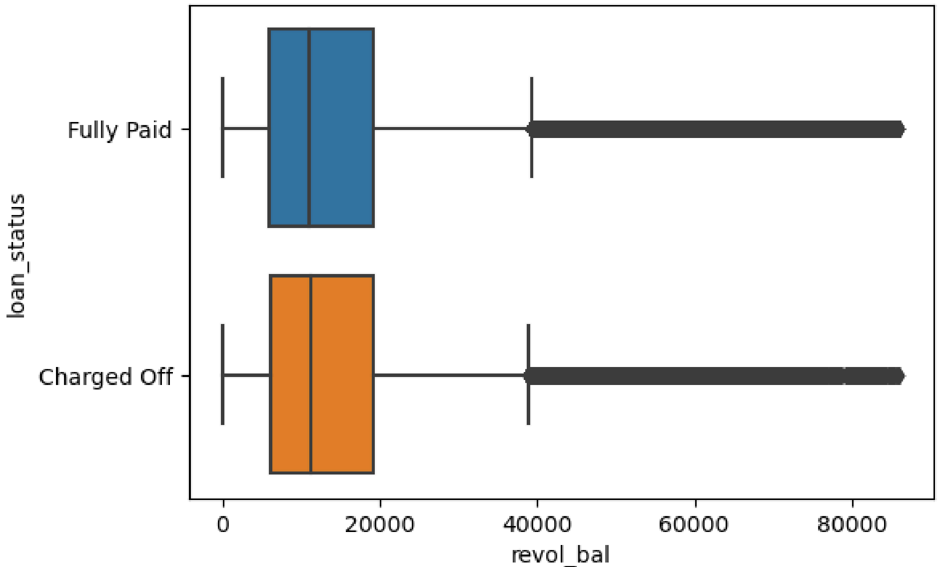


In [33]:
```
df.groupby('loan_status')["revol_bal"].describe()
```

Out[33]:

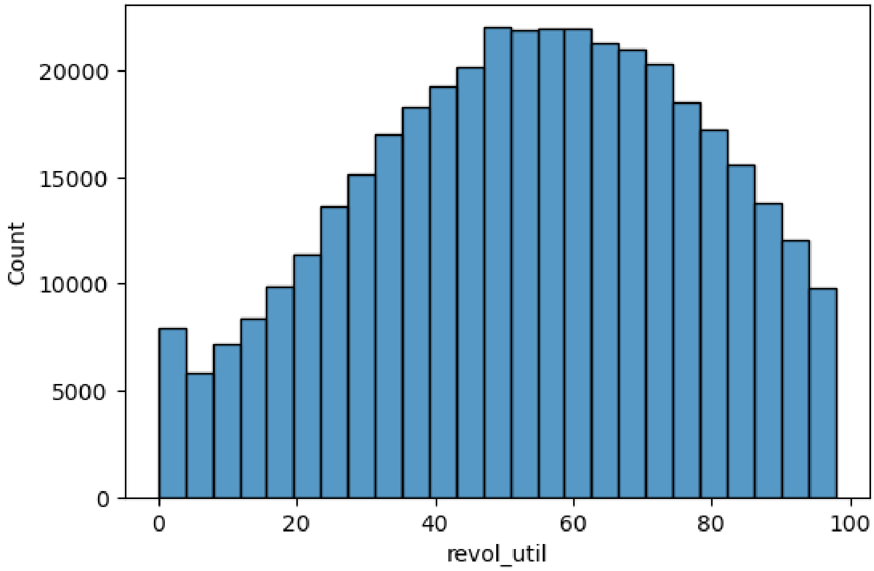
loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	15390.454701	18203.387930	0.0	6150.0	11277.0	19485.0	1030826.0
Fully Paid	318357.0	15955.327918	21132.193457	0.0	5992.0	11158.0	19657.0	1743266.0

In [34]:
```
plt.figure(figsize=(6,4))
sns.boxplot(x=df['revol_bal'].loc[df['revol_bal']<df['revol_bal'].quantile(0.99)], y=df["loan_statu
plt.show()
```



Revolving line Utilization rate

In [35]:
```
plt.figure(figsize=(6, 4))
sns.histplot(df['revol_util'].loc[df['revol_util']<df['revol_util'].quantile(0.99)], bins=25)
plt.show()
```

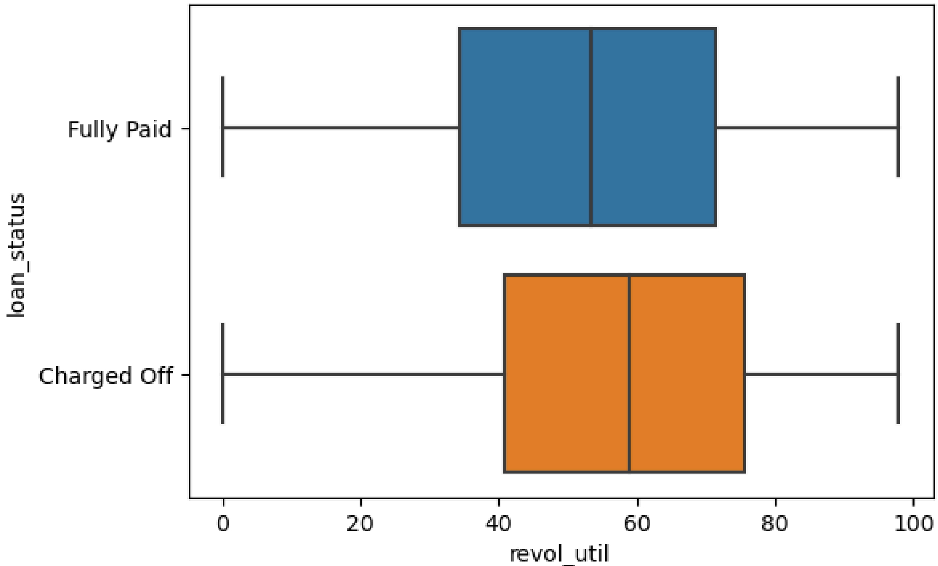


In [36]:
```
df.groupby('loan_status')["revol_util"].describe()
```

Out[36]:

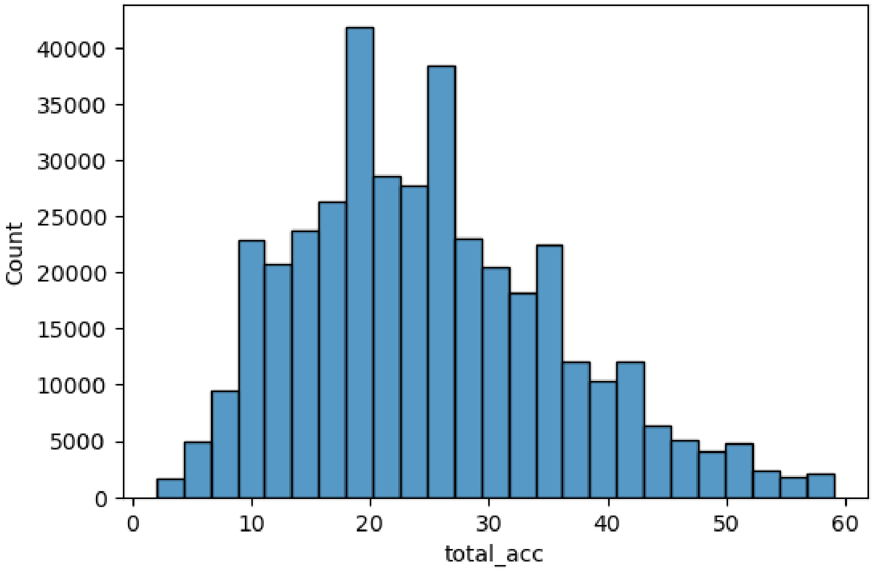
loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77610.0	57.869824	23.492176	0.0	41.2	59.3	76.2	148.0
Fully Paid	318144.0	52.796918	24.578304	0.0	34.6	53.7	72.0	892.3

In [37]:
```
plt.figure(figsize=(6,4))
sns.boxplot(x=df['revol_util'].loc[df['revol_util']<df['revol_util'].quantile(0.99)], y=df["loan_s
plt.show()
```



Total Credit Lines

In [38]:
```
plt.figure(figsize=(6, 4))
sns.histplot(df['total_acc'].loc[df['total_acc']<df['total_acc'].quantile(0.99)], bins=25)
plt.show()
```

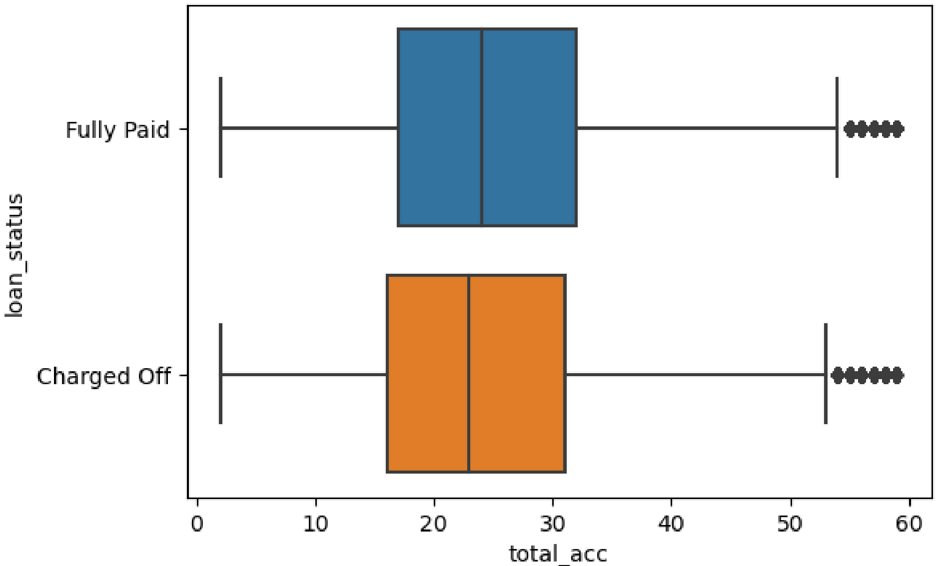


In [39]:
```
df.groupby('loan_status')["total_acc"].describe()
```

Out[39]:

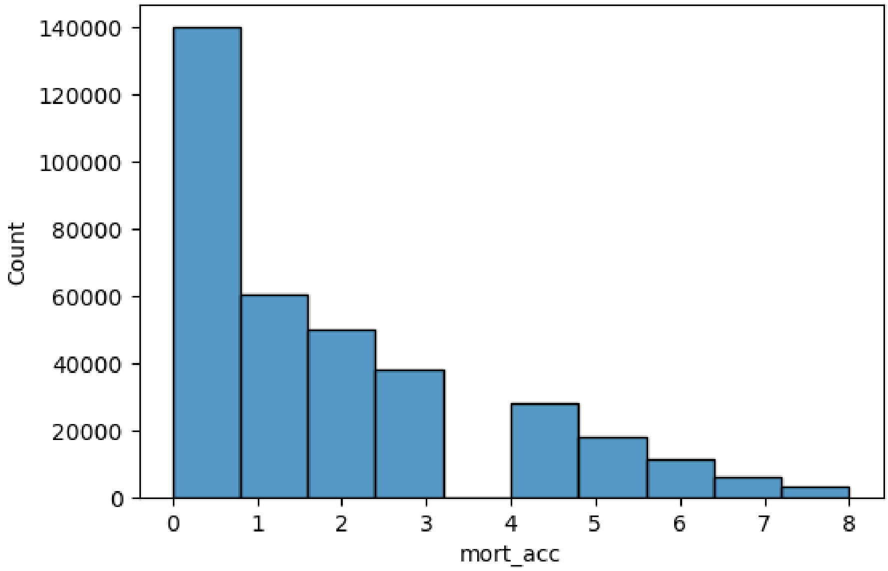
	count	mean	std	min	25%	50%	75%	max
loan_status								
Charged Off	77673.0	24.984152	11.913692	2.0	16.0	23.0	32.0	151.0
Fully Paid	318357.0	25.519800	11.878117	2.0	17.0	24.0	32.0	150.0

In [40]:
```
plt.figure(figsize=(6,4))
sns.boxplot(x=df['total_acc'].loc[df['total_acc']<df['total_acc'].quantile(0.99)], y=df["loan_statu
plt.show()
```



Total Mortgage A/Cs

In [41]:
```
plt.figure(figsize=(6, 4))
sns.histplot(df['mort_acc'].loc[df['mort_acc']<df['mort_acc'].quantile(0.99)], bins=10)
plt.show()
```

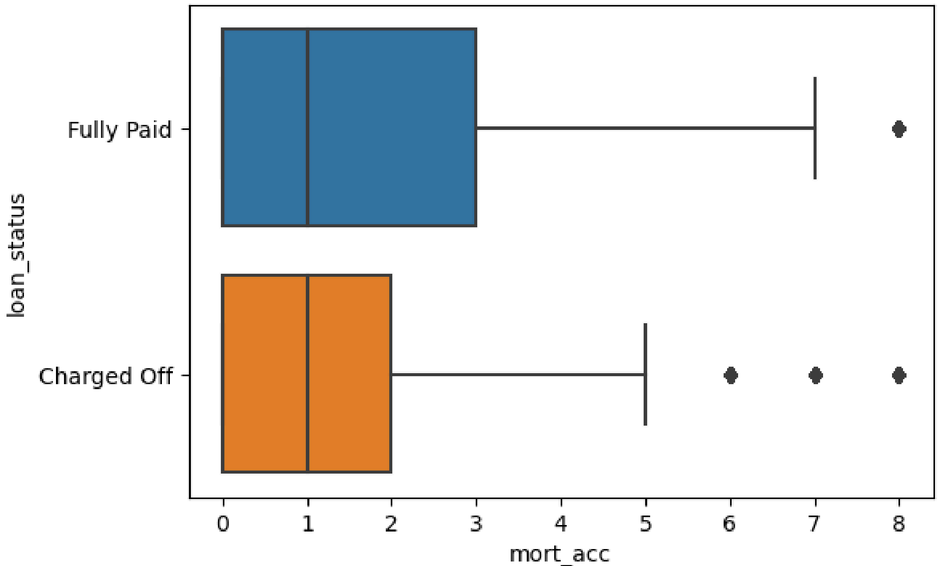


In [42]:
```
df.groupby('loan_status')["mort_acc"].describe()
```

Out[42]:

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 [43]:
```
plt.figure(figsize=(6,4))
sns.boxplot(x=df['mort_acc'].loc[df['mort_acc']<df['mort_acc'].quantile(0.99)], y=df["loan_status"]
plt.show()
```



Number of public record bankruptcies

In [44]:
```
df['pub_rec_bankruptcies'].value_counts()
```

Out[44]:
```
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 [45]:
```
df.groupby('loan_status')["pub_rec_bankruptcies"].describe()
```

Out[45]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77586.0	0.128412	0.368853	0.0	0.0	0.0	0.0	8.0
Fully Paid	317909.0	0.119997	0.352992	0.0	0.0	0.0	0.0	8.0

EDA: Categorical Columns

Loan Term

In [46]:
```
df["term"].value_counts()
```

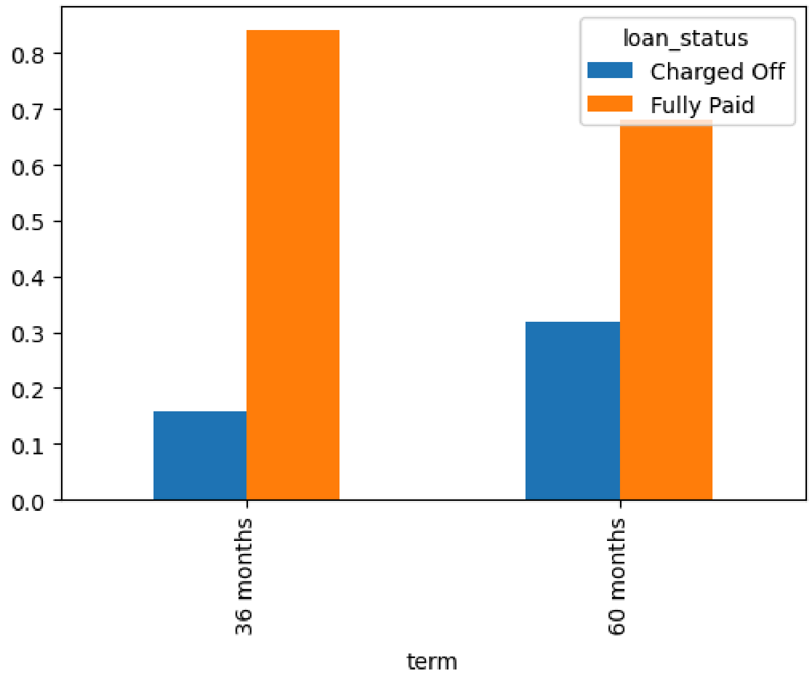
Out[46]:
```
 36 months    302005
 60 months     94025
Name: term, dtype: int64
```

In [47]:
```
df.groupby('loan_status')['term'].describe()
```

Out[47]:

loan_status	count	unique	top	freq
Charged Off	77673	2	36 months	47640
Fully Paid	318357	2	36 months	254365

In [48]:
```
pd.crosstab(columns=df["loan_status"], index=df["term"], normalize="index").plot(kind="bar", figsi
plt.show()
```



Home Ownership

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

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

In [50]:
```
df["home_ownership"].replace({"ANY":"OTHER", "NONE":"OTHER"}, inplace=True)
df["home_ownership"].value_counts()
```

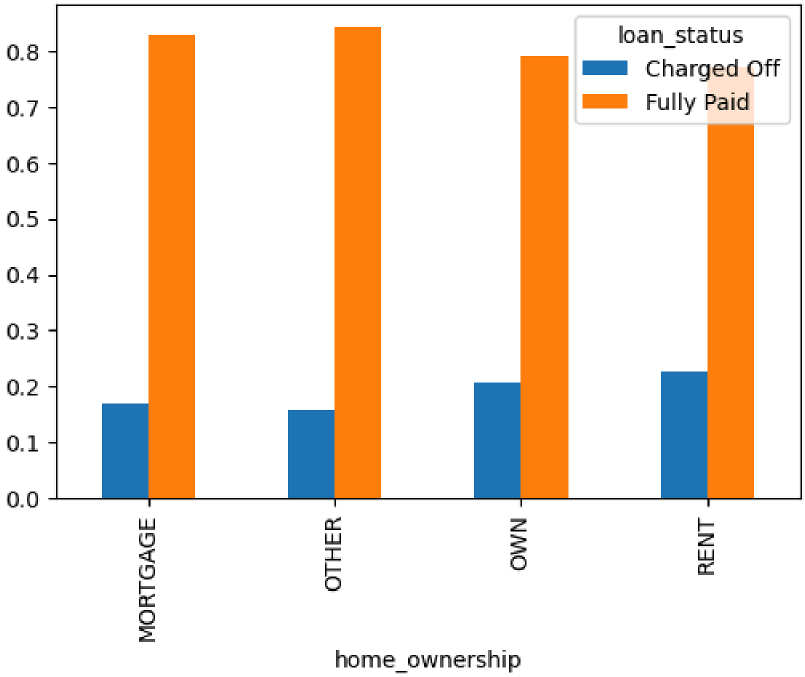
Out[50]:
```
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          146
Name: home_ownership, dtype: int64
```

In [51]:
```
df.groupby('loan_status')['home_ownership'].describe()
```

Out[51]:

loan_status	count	unique	top	freq
Charged Off	77673	4	RENT	36212
Fully Paid	318357	4	MORTGAGE	164716

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



Loan Grade/Sub-Grade

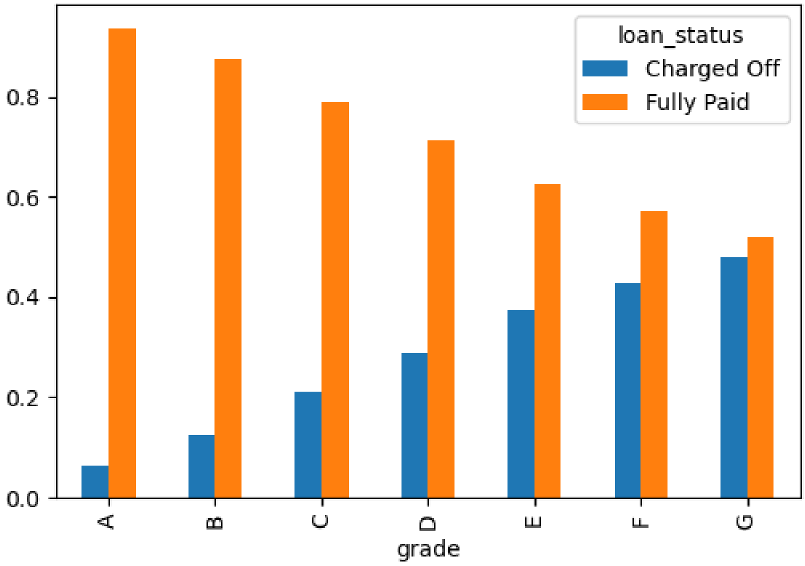
In [53]:
```
df["grade"].value_counts()
```

Out[53]:
```
B    116018
C    105987
A     64187
D     63524
E     31488
F     11772
G      3054
Name: grade, dtype: int64
```

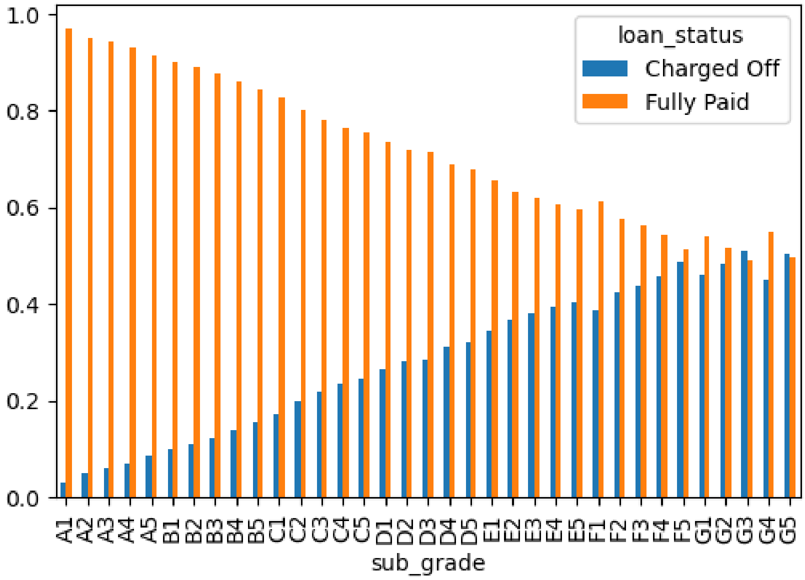
In [54]:
```
df["sub_grade"].value_counts()
```

Out[54]:
```
B3     26655
B4     25601
C1     23662
C2     22580
B2     22495
B5     22085
C3     21221
C4     20280
B1     19182
A5     18526
C5     18244
D1     15993
A4     15789
D2     13951
D3     12223
D4     11657
A3     10576
A1      9729
D5      9700
A2      9567
E1      7917
E2      7431
E3      6207
E4      5361
E5      4572
F1      3536
F2      2766
F3      2286
F4      1787
F5      1397
G1      1058
G2       754
G3       552
G4       374
G5       316
Name: sub_grade, dtype: int64
```

In [55]:
```
pd.crosstab(columns=df['loan_status'], index=df['grade'], normalize='index').plot(kind='bar', figsi
plt.show()
```



In [56]:
```
pd.crosstab(columns=df['loan_status'], index=df['sub_grade'], normalize='index').plot(kind='bar', f
plt.show()
```



Verification Status

In [57]:
```
df["verification_status"].value_counts()
```

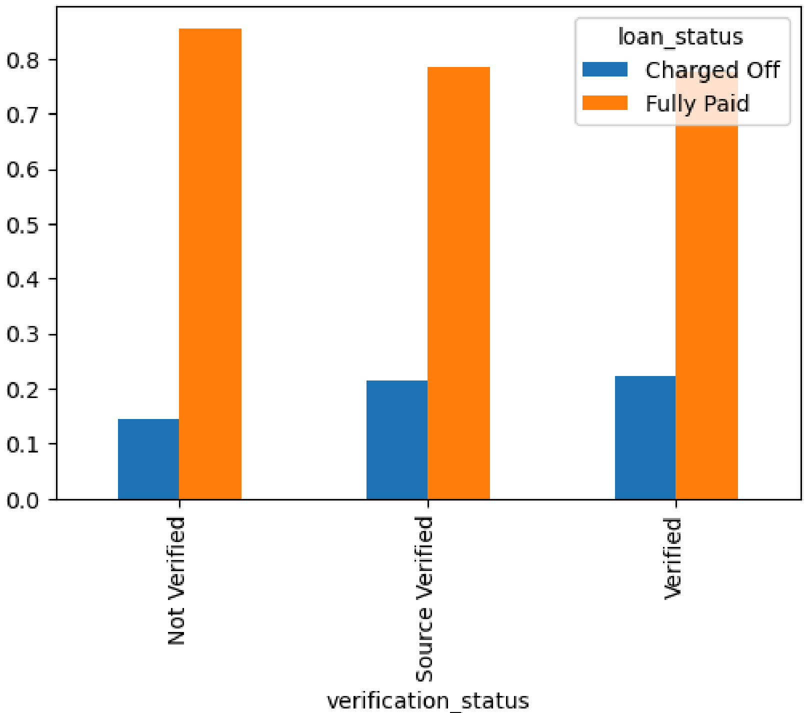
Out[57]:
```
Verified           139563
Source Verified    131385
Not Verified       125082
Name: verification_status, dtype: int64
```

In [58]:
```
df.groupby('loan_status')['verification_status'].describe()
```

Out[58]:

	count	unique	top	freq
loan_status				
Charged Off	77673	3	Verified	31152
Fully Paid	318357	3	Verified	108411

In [59]:
```
pd.crosstab(columns=df["loan_status"], index=df["verification_status"], normalize="index").plot(kin
plt.show()
```



Employee Employment Length & Profession

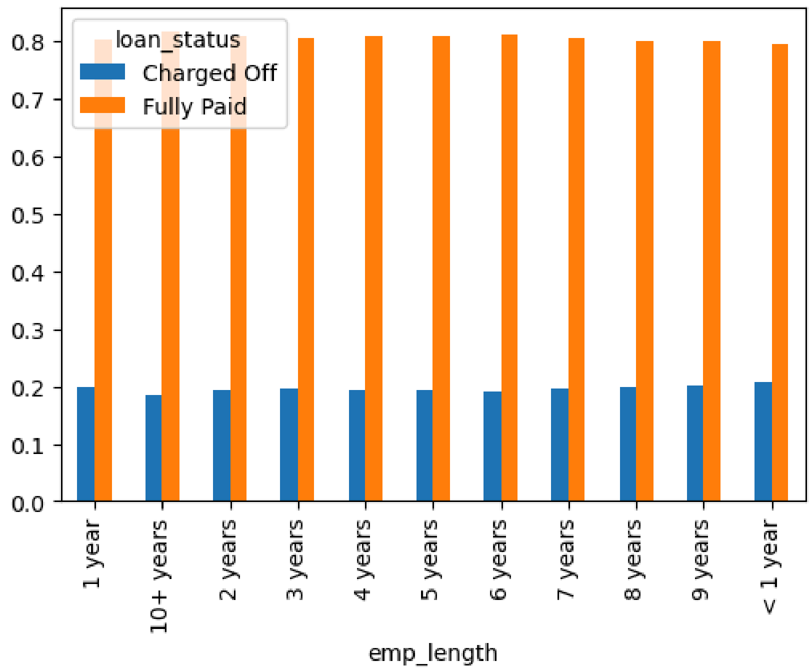
In [60]:
```
df["emp_length"].value_counts()
```

Out[60]:
```
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
9 years       15314
Name: emp_length, dtype: int64
```

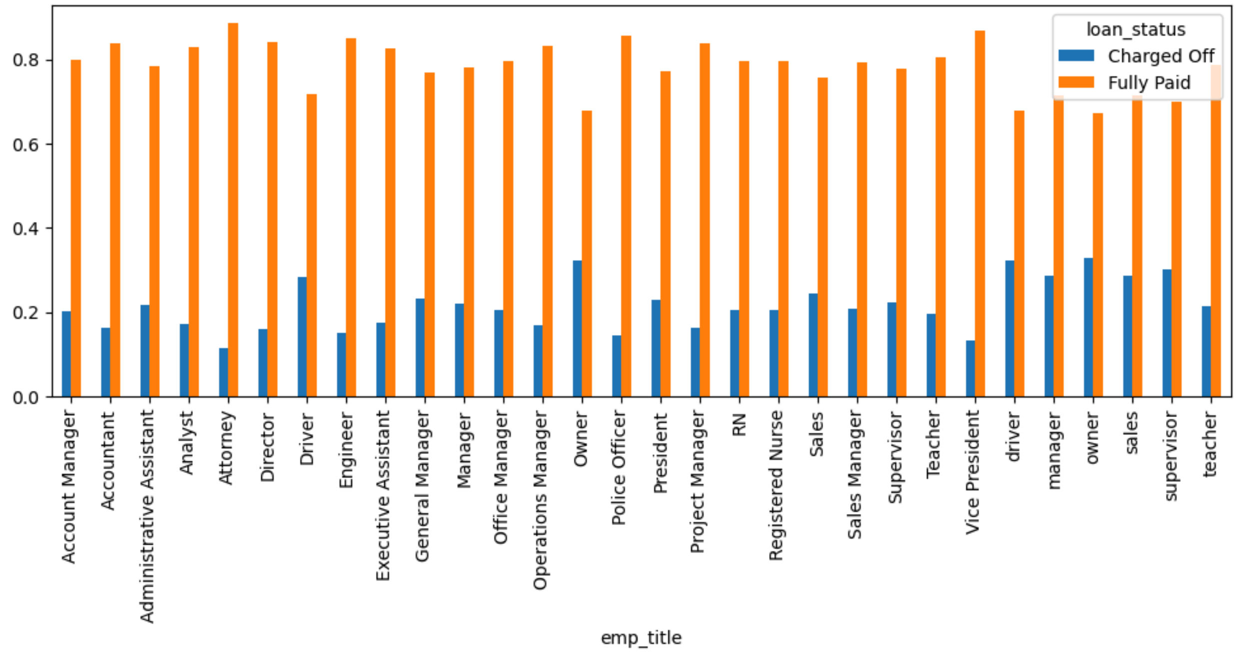
In [61]:
```
df["emp_title"].value_counts()[:30]
```

Out[61]:
```
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
teacher                      962
driver                       882
Vice President               857
Operations Manager           763
Administrative Assistant     756
Accountant                   748
President                    742
owner                        697
Account Manager              692
Police Officer               686
supervisor                   673
Attorney                     667
Sales Manager                665
sales                        645
Executive Assistant          642
Analyst                      623
Name: emp_title, dtype: int64
```

In [62]:
```
pd.crosstab(columns=df["loan_status"], index=df["emp_length"], normalize="index").plot(kind="bar",
plt.show()
```



In [63]:
```
top30_popular_emp_titles = df["emp_title"].value_counts()[:30].index
df_top30_popular_emp_titles = df.loc[df['emp_title'].isin(top30_popular_emp_titles)].copy()
pd.crosstab(columns=df_top30_popular_emp_titles["loan_status"], index=df_top30_popular_emp_titles['
plt.show()
```



Purpose

In [64]:
```
df["purpose"].value_counts()
```

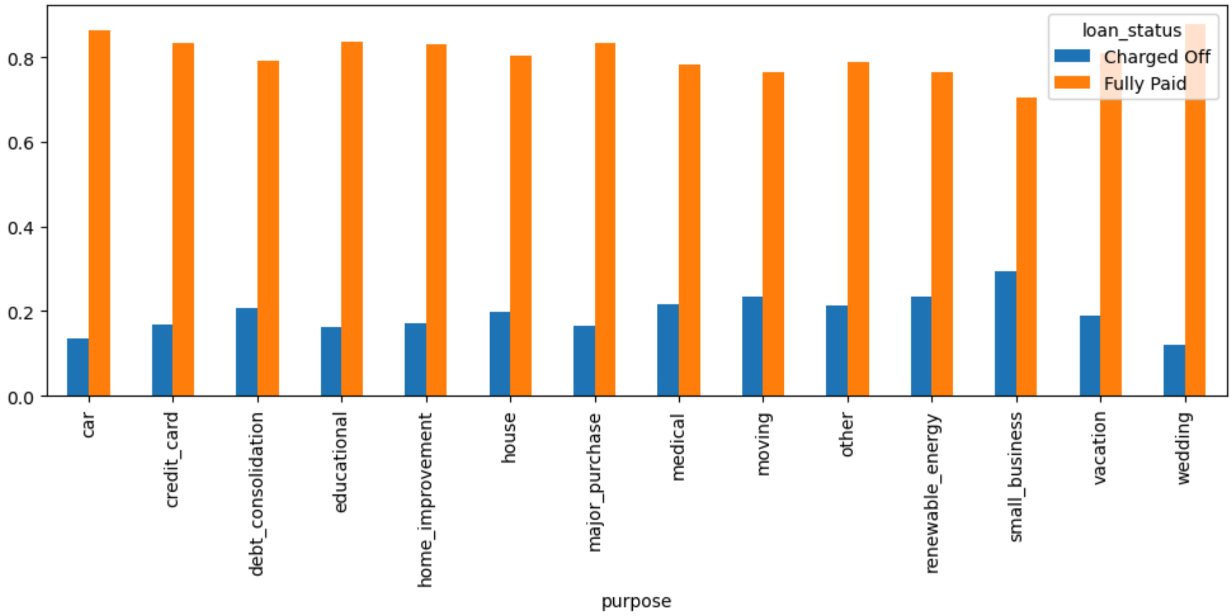
Out[64]:
```
debt_consolidation    234507
credit_card            83019
home_improvement       24030
other                  21185
major_purchase          8790
small_business          5701
car                     4697
medical                 4196
moving                  2854
vacation                2452
house                   2201
wedding                 1812
renewable_energy         329
educational              257
Name: purpose, dtype: int64
```

In [65]:
```
df.groupby('loan_status')['purpose'].describe()
```

Out[65]:

loan_status	count	unique	top	freq
Charged Off	77673	14	debt_consolidation	48640
Fully Paid	318357	14	debt_consolidation	185867

In [66]:
```
pd.crosstab(columns=df["loan_status"], index=df["purpose"], normalize="index").plot(kind="bar", fi
plt.show()
```



Application Type

In [67]:
```
df["application_type"].value_counts()
```

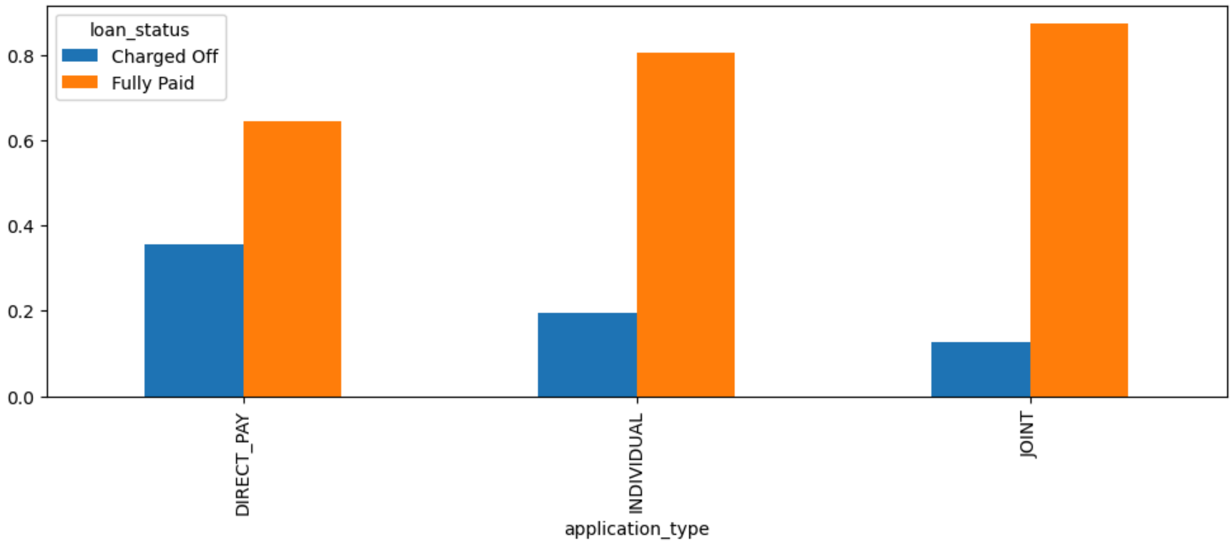
Out[67]:
```
INDIVIDUAL    395319
JOINT            425
DIRECT_PAY       286
Name: application_type, dtype: int64
```

In [68]:
```
df.groupby('loan_status')['application_type'].describe()
```

Out[68]:

loan_status	count	unique	top	freq
Charged Off	77673	3	INDIVIDUAL	77517
Fully Paid	318357	3	INDIVIDUAL	317802

In [69]:
```
pd.crosstab(columns=df["loan_status"], index=df["application_type"], normalize="index").plot(kind='
plt.show()
```



Date columns: Fixing dtypes

In [70]:
```
df['issue_d'] = pd.to_datetime(df['issue_d'])
df['earliest_cr_line'] = pd.to_datetime(df['earliest_cr_line'])
```

Data Cleaning

- We saw that most of the values in the 2 features (pub_rec, pub_rec_bankruptcies) are 0.
- Hence we will redefine the values of these 2 features
  - If original valiue is 0, keep it as it is
  - Else it should be set to 1

In [71]:
```
def pub_rec_cln(number):
    if number == 0.0:
        return 0
    else:
        return 1


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

df['pub_rec'] = df.pub_rec.apply(pub_rec_cln)
df['pub_rec_bankruptcies'] = df.pub_rec_bankruptcies.apply(pub_rec_bankruptcies_cln)
```

Handling Missing Values

Dropping unrelevant columns

In [72]:
```
df_final1 = df.drop(['emp_length', 'title', 'initial_list_status', 'address', 'issue_d', 'earliest_
df_final1.head()
```

Out[72]:

	loan_amnt	term	int_rate	grade	sub_grade	emp_title	home_ownership	annual_inc	verification_status	loan_
0	10000.0	36 months	11.44	B	B4	Marketing	RENT	117000.0	Not Verified	Ful
1	8000.0	36 months	11.99	B	B5	Credit analyst	MORTGAGE	65000.0	Not Verified	Ful
2	15600.0	36 months	10.49	B	B3	Statistician	RENT	43057.0	Source Verified	Ful
3	7200.0	36 months	6.49	A	A2	Client Advocate	RENT	54000.0	Not Verified	Ful
4	24375.0	60 months	17.27	C	C5	Destiny Management Inc.	MORTGAGE	55000.0	Verified	Charg

In [73]:
```
df_final1.shape
```

Out[73]:
```
(396030, 20)
```

In [74]:
```
# Rechecking missing values
100*df_final1.isna().sum().sort_values(ascending=False)/df_final1.shape[0]
```

Out[74]:
```
mort_acc                9.543469
emp_title               5.789208
pub_rec_bankruptcies    0.135091
revol_util              0.069692
dti                     0.000000
application_type        0.000000
total_acc               0.000000
revol_bal               0.000000
pub_rec                 0.000000
open_acc                0.000000
loan_amnt               0.000000
term                    0.000000
loan_status             0.000000
verification_status     0.000000
annual_inc              0.000000
home_ownership          0.000000
sub_grade               0.000000
grade                   0.000000
int_rate                0.000000
purpose                 0.000000
dtype: float64
```

Filling missing values: Total Mortgage A/Cs

In [75]:
```
# Median mortgage A/Cs across total A/Cs
total_acc_mort_acc_50p = df_final1.groupby('total_acc')['mort_acc'].median()
total_acc_mort_acc_50p
```

Out[75]:
```
total_acc
2.0      0.0
3.0      0.0
4.0      0.0
5.0      0.0
6.0      0.0
        ...
124.0    1.0
129.0    1.0
135.0    3.0
150.0    2.0
151.0    0.0
Name: mort_acc, Length: 118, dtype: float64
```

In [76]:
```
def fill_mort_acc(total_acc, mort_acc):
    if np.isnan(mort_acc):
        return total_acc_mort_acc_50p[total_acc].round()
    else:
        return mort_acc

df_final1['mort_acc'] = df_final1[['total_acc', 'mort_acc']].apply(lambda x: fill_mort_acc(x[0], x
```

In [77]:
```
df_final1['mort_acc'].isna().sum()
```

Out[77]:
```
0
```

Filling missing values: Employee Title

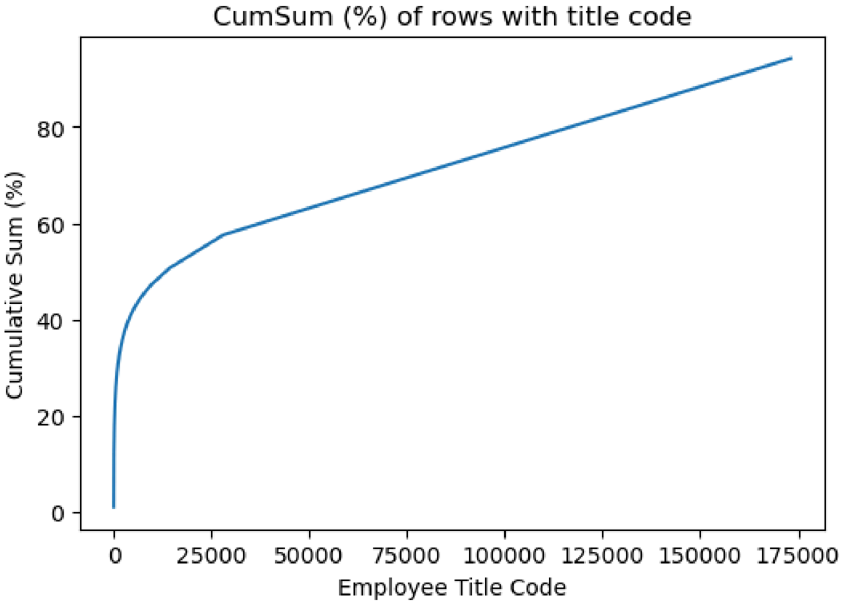
In [78]:
```
df_final1['emp_title'].nunique()
```

Out[78]:
```
173105
```

In [79]:
```
ser_emp_title_cumsum = 100*df_final1['emp_title'].value_counts().cumsum()/df_final1.shape[0]
ser_emp_title_cumsum
```

Out[79]:
```
Teacher                    1.108249
Manager                    2.181400
Registered Nurse           2.650052
RN                         3.116178
Supervisor                 3.578264
                            ...
Postman                   94.209782
McCarthy & Holthus, LLC   94.210035
jp flooring               94.210287
Histology Technologist    94.210540
Gracon Services, Inc      94.210792
Name: emp_title, Length: 173105, dtype: float64
```

In [80]:
```
plt.figure(figsize=(6, 4))
plt.plot(np.arange(0, len(ser_emp_title_cumsum)), ser_emp_title_cumsum.values)
plt.xlabel('Employee Title Code')
plt.ylabel('Cumulative Sum (%)')
plt.title('CumSum (%) of rows with title code')
plt.show()
```



In [81]:
```
# Since there are many titles and each title has significant amount of rows, it is better to impute
# So we will impute missing values in emp_title with 'Others'
df_final1['emp_title'].fillna('Others', inplace=True)
```

In [82]:
```
df_final1['emp_title'].isna().sum()
```

Out[82]:
```
0
```

Dealing with missing values of other columns:

- Revolving line Utilization rate
- Number of public record bankruptcies

In [83]:
```
# Rechecking missing values again
100*df_final1.isna().sum().sort_values(ascending=False)/df_final1.shape[0]
```

Out[83]:
```
pub_rec_bankruptcies    0.135091
revol_util              0.069692
term                    0.000000
mort_acc                0.000000
application_type        0.000000
total_acc               0.000000
revol_bal               0.000000
pub_rec                 0.000000
open_acc                0.000000
dti                     0.000000
loan_amnt               0.000000
loan_status             0.000000
verification_status     0.000000
annual_inc              0.000000
home_ownership          0.000000
emp_title               0.000000
sub_grade               0.000000
grade                   0.000000
int_rate                0.000000
purpose                 0.000000
dtype: float64
```

In [84]:
```
# Since there are only a few rows with nan values across the 2 features (pub_rec_bankruptcies, revo
# There are at max 0.2% rows that will get dropped --> Hence this would make sense
df_final2 = df_final1.dropna(how='any', axis=0).copy()
df_final2.shape
```

Out[84]:
```
(395219, 20)
```

In [85]:
```
# Final check: missing values
df_final2.isna().sum()
```

Out[85]:
```
loan_amnt               0
term                    0
int_rate                0
grade                   0
sub_grade               0
emp_title               0
home_ownership          0
annual_inc              0
verification_status     0
loan_status             0
purpose                 0
dti                     0
open_acc                0
pub_rec                 0
revol_bal               0
revol_util              0
total_acc               0
application_type        0
mort_acc                0
pub_rec_bankruptcies    0
dtype: int64
```

Handling Outliers

In [86]:
```
# Categorical vs Numerical Columns
cat_cols = df_final2.dtypes.loc[df_final2.dtypes=='object'].index
num_cols = df_final2.columns[~df_final2.columns.isin(cat_cols)]
print('-'*50)
print('Total categorical columns:', cat_cols.shape[0])
print('Total numerical columns:', num_cols.shape[0])
print('-'*50)
```

```
--------------------------------------------------
Total categorical columns: 9
Total numerical columns: 11
--------------------------------------------------
```

In [87]:
```
num_cols
```

Out[87]:
```
Index(['loan_amnt', 'int_rate', 'annual_inc', 'dti', 'open_acc', 'pub_rec',
       'revol_bal', 'revol_util', 'total_acc', 'mort_acc',
       'pub_rec_bankruptcies'],
      dtype='object')
```

In [88]:
```
num_cols_filtered = num_cols[~num_cols.isin(['pub_rec', 'pub_rec_bankruptcies'])]
num_cols_filtered
```

Out[88]:
```
Index(['loan_amnt', 'int_rate', 'annual_inc', 'dti', 'open_acc', 'revol_bal',
       'revol_util', 'total_acc', 'mort_acc'],
      dtype='object')
```

In [89]:
```
df_final2[num_cols_filtered].describe().round(1)
```

Out[89]:

	loan_amnt	int_rate	annual_inc	dti	open_acc	revol_bal	revol_util	total_acc	mort_acc
count	395219.0	395219.0	395219.0	395219.0	395219.0	395219.0	395219.0	395219.0	395219.0
mean	14122.1	13.6	74199.4	17.4	11.3	15851.7	53.8	25.4	1.7
std	8357.1	4.5	61557.3	18.0	5.1	20584.3	24.4	11.9	2.1
min	500.0	5.3	0.0	0.0	1.0	0.0	0.0	2.0	0.0
25%	8000.0	10.5	45000.0	11.3	8.0	6038.0	35.9	17.0	0.0
50%	12000.0	13.3	64000.0	16.9	10.0	11190.0	54.8	24.0	1.0
75%	20000.0	16.6	90000.0	23.0	14.0	19626.0	72.9	32.0	3.0
max	40000.0	31.0	8706582.0	9999.0	90.0	1743266.0	892.3	151.0	34.0

Handling Outliers using IQR Method

In [90]:
```
def outlier_detect_cap(col, df):
    q25 = np.quantile(df[col], 0.25)
```

```
    q75 = np.quantile(df[col], 0.75)
    iqr = q75-q25
    upp_whis = q75 + 1.5*iqr
    low_whis = q25 - 1.5*iqr

    df[col] = np.where((df[col] < low_whis), low_whis, df[col])
    df[col] = np.where((df[col] > upp_whis), upp_whis, df[col])

    return df

df_final3 = df_final2.copy()
for col in num_cols_filtered:
    outlier_detect_cap(col, df_final3)
```

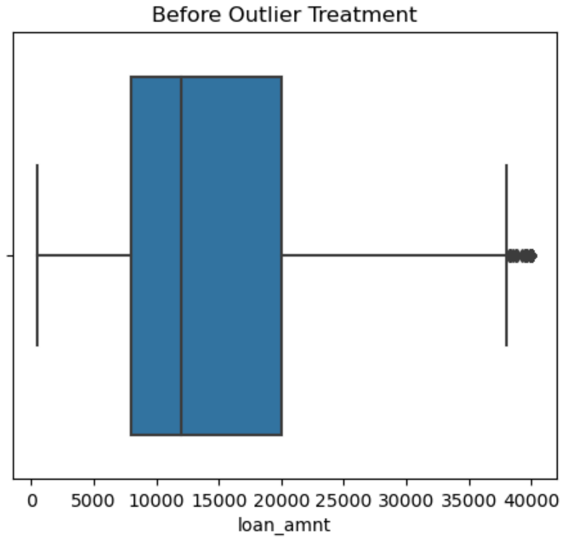
In [91]:
```
df_final3[num_cols_filtered].describe().round(1)
```

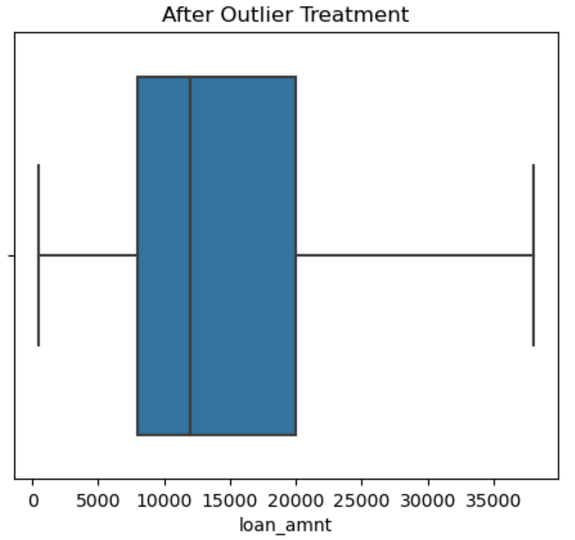
Out[91]:

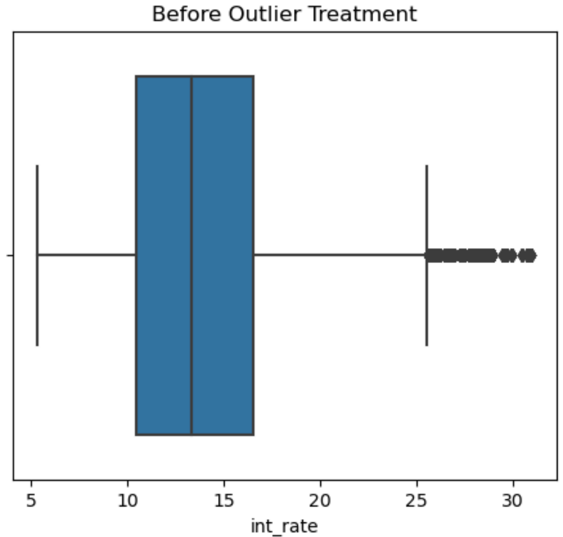
	loan_amnt	int_rate	annual_inc	dti	open_acc	revol_bal	revol_util	total_acc	mort_acc
count	395219.0	395219.0	395219.0	395219.0	395219.0	395219.0	395219.0	395219.0	395219.0
mean	14121.1	13.6	70995.3	17.4	11.2	14177.6	53.8	25.3	1.7
std	8354.3	4.5	34309.1	8.1	4.7	10702.8	24.4	11.4	1.9
min	500.0	5.3	0.0	0.0	1.0	0.0	0.0	2.0	0.0
25%	8000.0	10.5	45000.0	11.3	8.0	6038.0	35.9	17.0	0.0
50%	12000.0	13.3	64000.0	16.9	10.0	11190.0	54.8	24.0	1.0
75%	20000.0	16.6	90000.0	23.0	14.0	19626.0	72.9	32.0	3.0
max	38000.0	25.6	157500.0	40.5	23.0	40008.0	128.4	54.5	7.5

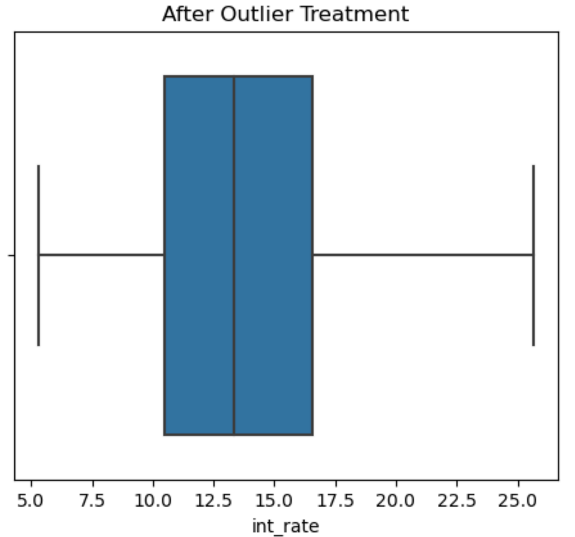
Visualizing Boxplots: Before & After Outlier Treatment

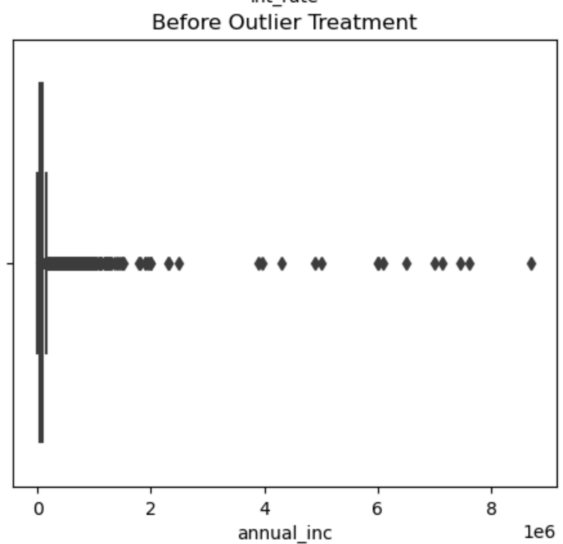
In [92]:
```
plt.figure(figsize=(12, 48))
i = 1
for col in num_cols_filtered:
    plt.subplot(9, 2, i)
    sns.boxplot(x=df_final2[col])
    plt.title('Before Outlier Treatment')
    plt.subplot(9, 2, i+1)
    sns.boxplot(x=df_final3[col])
    plt.title('After Outlier Treatment')
    i+=2

plt.show()
```

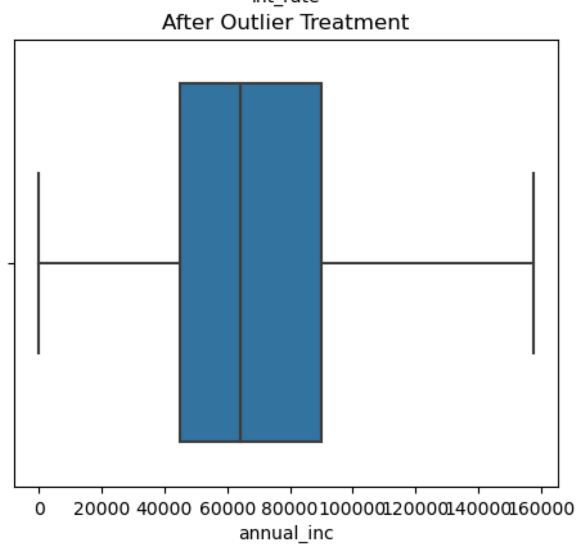

Before Outlier Treatment
0 5000 10000 15000 20000 25000 30000 35000 40000
loan_amnt

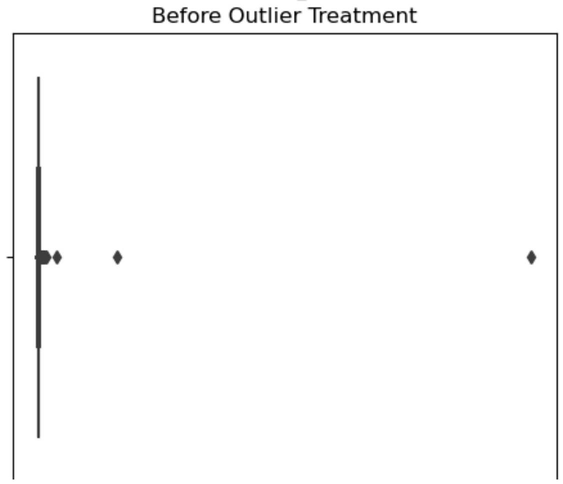

After Outlier Treatment
0 5000 10000 15000 20000 25000 30000 35000
loan_amnt

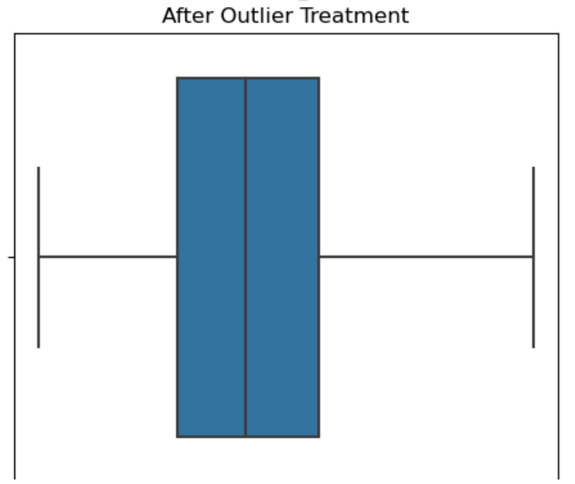

Before Outlier Treatment
5 10 15 20 25 30
int_rate

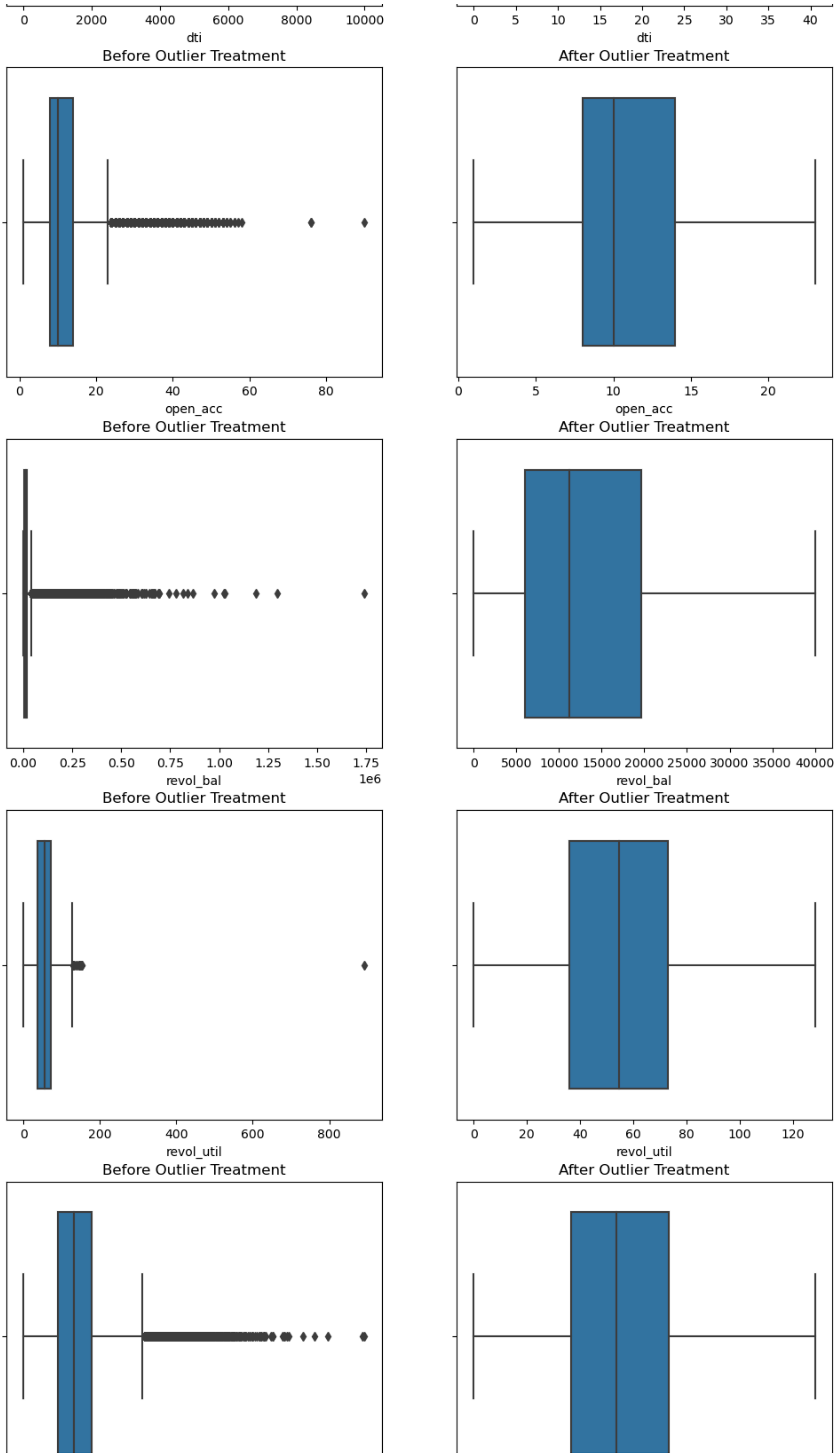

After Outlier Treatment
5.0 7.5 10.0 12.5 15.0 17.5 20.0 22.5 25.0
int_rate


Before Outlier Treatment
0 2 4 6 8
1e6
annual_inc

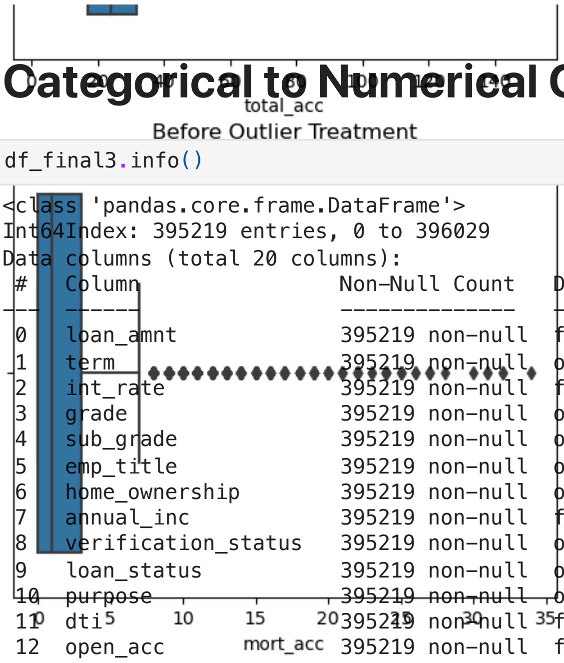

After Outlier Treatment
0 20000 40000 60000 80000 100000 120000 140000 160000
annual_inc

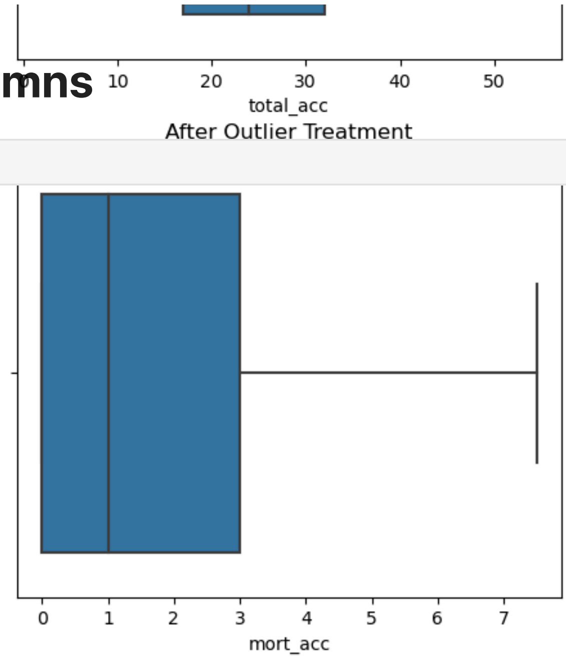

Before Outlier Treatment


After Outlier Treatment


0 2000 4000 6000 8000 10000
dti
0 5 10 15 20 25 30 35 40
dti
Before Outlier Treatment
After Outlier Treatment
0 20 40 60 80
open_acc
0 5 10 15 20
open_acc
Before Outlier Treatment
After Outlier Treatment
0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75
1e6
revol_bal
0 5000 10000 15000 20000 25000 30000 35000 40000
revol_bal
Before Outlier Treatment
After Outlier Treatment
0 200 400 600 800
revol_util
0 20 40 60 80 100 120
revol_util
Before Outlier Treatment
After Outlier Treatment

Categorical to Numerical Columns





In [93]:
```
df_final3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 395219 entries, 0 to 396029
Data columns (total 20 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             395219 non-null  float64
 1   term                  395219 non-null  object 
 2   int_rate              395219 non-null  float64
 3   grade                 395219 non-null  object 
 4   sub_grade             395219 non-null  object 
 5   emp_title             395219 non-null  object 
 6   home_ownership        395219 non-null  object 
 7   annual_inc            395219 non-null  float64
 8   verification_status   395219 non-null  object 
 9   loan_status           395219 non-null  object 
 10  purpose               395219 non-null  object 
 11  dti                   395219 non-null  float64
 12  open_acc              395219 non-null  float64
 13  pub_rec               395219 non-null  int64  
 14  revol_bal             395219 non-null  float64
 15  revol_util            395219 non-null  float64
 16  total_acc             395219 non-null  float64
 17  application_type      395219 non-null  object 
 18  mort_acc              395219 non-null  float64
 19  pub_rec_bankruptcies  395219 non-null  float64
dtypes: float64(10), int64(1), object(9)
memory usage: 63.3+ MB
```

Unique values per categorical columns

In [94]:
```
cat_cols
```

Out[94]:
```
Index(['term', 'grade', 'sub_grade', 'emp_title', 'home_ownership',
       'verification_status', 'loan_status', 'purpose', 'application_type'],
      dtype='object')
```

In [95]:
```
for col in cat_cols:
    print(f'Unique values in {col}: {df_final3[col].nunique()}')
    print('-'*50)
```

```
Unique values in term: 2
--------------------------------------------------
Unique values in grade: 7
--------------------------------------------------
Unique values in sub_grade: 35
--------------------------------------------------
Unique values in emp_title: 172651
--------------------------------------------------
Unique values in home_ownership: 4
--------------------------------------------------
Unique values in verification_status: 3
--------------------------------------------------
Unique values in loan_status: 2
--------------------------------------------------
Unique values in purpose: 14
--------------------------------------------------
Unique values in application_type: 3
--------------------------------------------------
```

Encoding Categorical Columns to Numeric Columns

In [96]:
```
df_encoded = df_final3.copy()
```

Simple Encoding

In [97]:
```
df_encoded['loan_status'] = df_encoded['loan_status'].map({'Fully Paid':0, 'Charged Off':1})
df_encoded['term'] = df_encoded['term'].map({' 36 months':36, ' 60 months':60})
```

We will apply the following encoding of categorical features:

- Target Encoding: grade, sub_grade, emp_title, purpose
- One Hot Encoding: home_ownership, verification_status, application_type

Target Encoding

In [98]:
```
TE = TargetEncoder()

df_encoded["grade"] = TE.fit_transform(df_encoded["grade"],df_encoded["loan_status"])
df_encoded["sub_grade"] = TE.fit_transform(df_encoded["sub_grade"],df_encoded["loan_status"])
df_encoded["emp_title"] = TE.fit_transform(df_encoded["emp_title"],df_encoded["loan_status"])
df_encoded["purpose"] = TE.fit_transform(df_encoded["purpose"],df_encoded["loan_status"])
df_encoded["home_ownership"] = TE.fit_transform(df_encoded["home_ownership"],df_encoded["loan_statu
df_encoded["verification_status"] = TE.fit_transform(df_encoded["verification_status"],df_encoded["
df_encoded["application_type"] = TE.fit_transform(df_encoded["application_type"],df_encoded["loan_s
```

One Hot Encoding

In [99]:
```
# ho_OHE = OneHotEncoder()
# df_home_ownership_ohe = pd.DataFrame(ho_OHE.fit_transform(df_encoded[['home_ownership']]).toarray
#                                     columns=ho_OHC.get_feature_names_out(),
#                                     index=df_encoded['home_ownership'].index)
# df_home_ownership_ohe.head()
```

In [100…
```
# vs_OHE = OneHotEncoder()
# df_verification_status_ohe = pd.DataFrame(vs_OHE.fit_transform(df_encoded[['verification_status']
#                                     columns=vs_OHE.get_feature_names_out(),
#                                     index=df_encoded['verification_status'].index)
# df_verification_status_ohe.head()
```

In [101…
```
# at_OHE = OneHotEncoder()
# df_application_type_ohe = pd.DataFrame(at_OHE.fit_transform(df_encoded[['application_type']]).toa
#                                     columns=at_OHE.get_feature_names_out(),
#                                     index=df_encoded['application_type'].index)
# df_application_type_ohe.head()
```

Final Encoded Dataset

In [102…
```
# df_encoded_f = pd.concat([df_encoded, df_home_ownership_ohe, df_verification_status_ohe, df_appli
# df_encoded_f.drop(['home_ownership', 'verification_status', 'application_type'], axis=1, inplace=
# df_encoded_f
```

In [103…
```
df_encoded_f = df_encoded.copy()
df_encoded_f
```

Out[103]:

	loan_amnt	term	int_rate	grade	sub_grade	emp_title	home_ownership	annual_inc	verification_status
0	10000.0	36	11.44	0.125721	0.138469	0.247140	0.226701	117000.0	0.146248
1	8000.0	36	11.99	0.125721	0.155039	0.217342	0.169572	65000.0	0.146248
2	15600.0	36	10.49	0.125721	0.123332	0.192009	0.226701	43057.0	0.214758
3	7200.0	36	6.49	0.062929	0.048223	0.170631	0.226701	54000.0	0.146248
4	24375.0	60	17.27	0.211865	0.245018	0.300739	0.169572	55000.0	0.223175
...	...	...	...	...	...	...	...	...	...
396025	10000.0	60	10.99	0.125721	0.138469	0.170631	0.226701	40000.0	0.214758
396026	21000.0	36	12.29	0.211865	0.173790	0.220430	0.169572	110000.0	0.214758
396027	5000.0	36	9.99	0.125721	0.098537	0.268003	0.226701	56500.0	0.223175
396028	21000.0	60	15.31	0.211865	0.197462	0.170631	0.169572	64000.0	0.223175
396029	2000.0	36	13.61	0.211865	0.197462	0.217234	0.226701	42996.0	0.223175

395219 rows × 20 columns

In [104…
```
df_encoded_f.shape
```

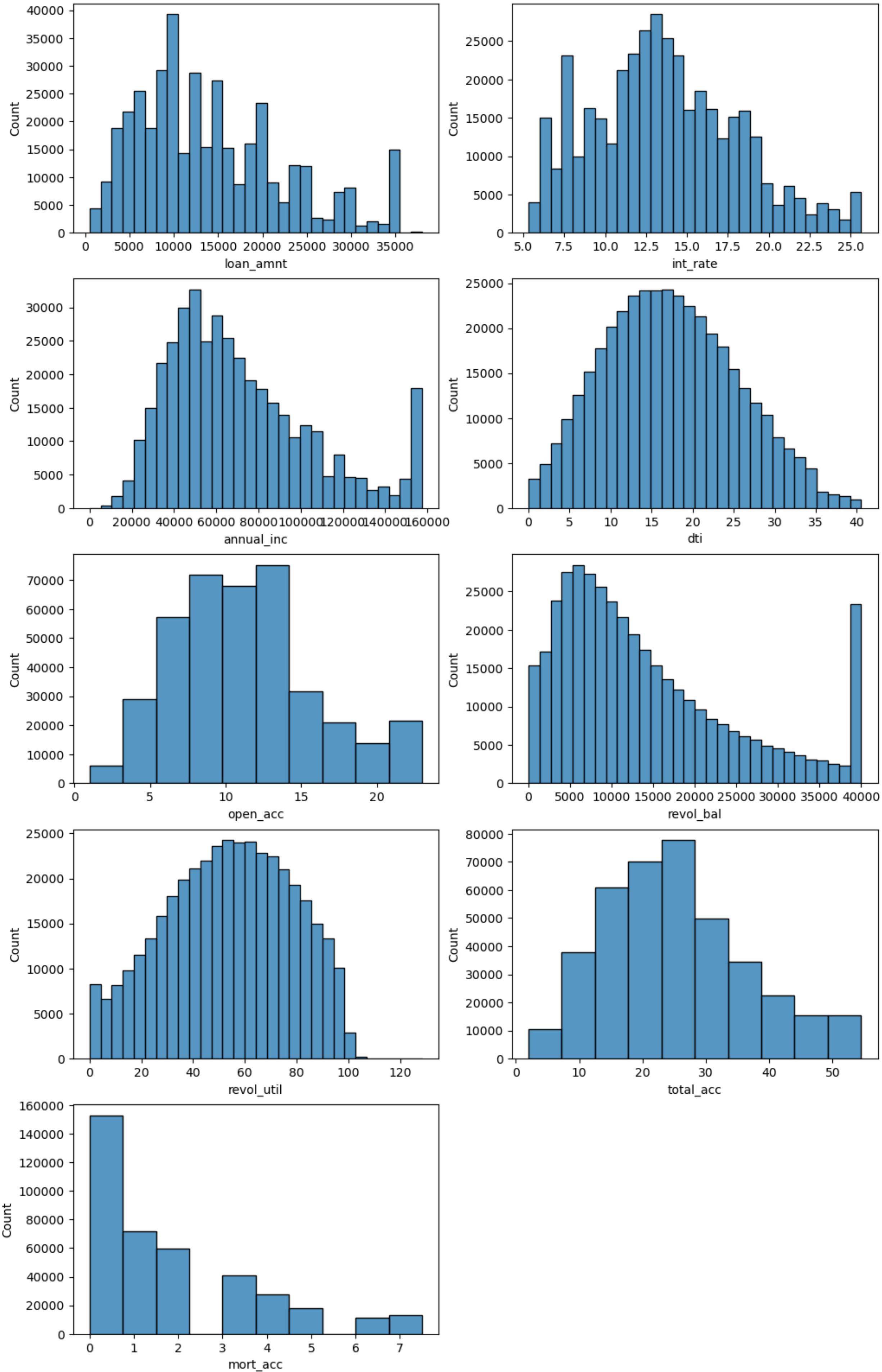
Out[104]:
```
(395219, 20)
```

In [105…
```
df_encoded_f.dtypes
```

Out[105]:
```
loan_amnt               float64
term                      int64
int_rate                float64
grade                   float64
sub_grade               float64
emp_title               float64
home_ownership          float64
annual_inc              float64
verification_status     float64
loan_status               int64
purpose                 float64
dti                     float64
open_acc                float64
pub_rec                   int64
revol_bal               float64
revol_util              float64
total_acc               float64
application_type        float64
mort_acc                float64
pub_rec_bankruptcies    float64
dtype: object
```

Column Transformation: Log Tranformation

In [106…
```python
plt.figure(figsize=(12, 20))
i = 1
for col in num_cols_filtered:
    plt.subplot(5, 2, i)
    if col in ['open_acc', 'mort_acc', 'total_acc']:
        sns.histplot(x = df_encoded_f[col], bins=10)
    else:
        sns.histplot(x = df_encoded_f[col], bins=30)
    i += 1

plt.show()
```



Fixing zero values in those features which wll undergo log transformation

In [107…

```
def check_lower_percentile_vals(col, df):
    for i in range(10):
```

```
        print(f'{round(0.01*i, 2)} percentile of {col}: {round(df[col].quantile(0.01*i), 1)}')
```

In [108…

```
log_transform_cols = ['loan_amnt', 'annual_inc', 'revol_bal']

for col in log_transform_cols:
    print(f'Column name: {col}')
    check_lower_percentile_vals(col, df_encoded_f)
    print('-'*50)
```

```
Column name: loan_amnt
0.0 percentile of loan_amnt: 500.0
0.01 percentile of loan_amnt: 1600.0
0.02 percentile of loan_amnt: 2100.0
0.03 percentile of loan_amnt: 2575.0
0.04 percentile of loan_amnt: 3000.0
0.05 percentile of loan_amnt: 3250.0
0.06 percentile of loan_amnt: 3600.0
0.07 percentile of loan_amnt: 4000.0
0.08 percentile of loan_amnt: 4200.0
0.09 percentile of loan_amnt: 4750.0
--------------------------------------------------
Column name: annual_inc
0.0 percentile of annual_inc: 0.0
0.01 percentile of annual_inc: 19000.0
0.02 percentile of annual_inc: 22000.0
0.03 percentile of annual_inc: 25000.0
0.04 percentile of annual_inc: 26000.0
0.05 percentile of annual_inc: 28000.0
0.06 percentile of annual_inc: 30000.0
0.07 percentile of annual_inc: 30000.0
0.08 percentile of annual_inc: 31680.0
0.09 percentile of annual_inc: 32742.9
--------------------------------------------------
Column name: revol_bal
0.0 percentile of revol_bal: 0.0
0.01 percentile of revol_bal: 184.0
0.02 percentile of revol_bal: 613.0
0.03 percentile of revol_bal: 1017.0
0.04 percentile of revol_bal: 1376.0
0.05 percentile of revol_bal: 1708.0
0.06 percentile of revol_bal: 2023.1
0.07 percentile of revol_bal: 2321.0
0.08 percentile of revol_bal: 2604.0
0.09 percentile of revol_bal: 2875.0
--------------------------------------------------
```

Performing Log Transformation

In [109…

```
df_col_trnsfrm = df_encoded_f.copy()

df_col_trnsfrm['annual_inc'] = np.where(df_col_trnsfrm['annual_inc']<19000, 19000, df_col_trnsfrm[
df_col_trnsfrm['revol_bal'] = np.where(df_col_trnsfrm['revol_bal']<180, 180, df_col_trnsfrm['revol_
```

In [110…

```
for col in log_transform_cols:
    df_col_trnsfrm[col] = np.log(df_col_trnsfrm[col])
```

In [111…

```
# plt.figure(figsize=(12, 4))
# i = 1
# for col in log_transform_cols:
#     plt.subplot(1, 3, i)
#     sns.histplot(x = df_col_trnsfrm[col], bins=30)
#     i += 1

# plt.show()
```

Train-Test Split & Feature Scaling

In [112…

```
X = df_col_trnsfrm.drop(['loan_status'], axis=1)
y = df_col_trnsfrm['loan_status']
X.shape, y.shape
```

Out[112]:

```
((395219, 19), (395219,))
```

In [113…

```
X_train , X_test,  y_train , y_test = train_test_split(X, y, test_size=0.25, random_state=42)
print('X_train Shape:', X_train.shape)
print('y_train Shape:', y_train.shape)
```

```
print('X_test Shape:', X_test.shape)
print('y_test Shape:', y_test.shape)
```

```
X_train Shape: (296414, 19)
y_train Shape: (296414,)
X_test Shape: (98805, 19)
y_test Shape: (98805,)
```

In [114…

```
std_scaler = StandardScaler()
X_train_scl = std_scaler.fit_transform(X_train)
X_test_scl = std_scaler.transform(X_test)
```

In [ ]:

Modelling & Metrics

Logistic Regression Model & Cross-Validation Accuracy

In [115…

```
log_reg = 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=42,
                             solver='lbfgs',
                             max_iter=1000,       # 1000 iterations for learning
                             multi_class='auto',
                             verbose=0,
                             warm_start=False,
                             n_jobs=None,
                             l1_ratio=None
                            )

cross_val_log_reg = cross_val_score(log_reg, X_train_scl, y_train, cv=10, scoring='accuracy')
print(cross_val_log_reg)
print('-'*50)
pd.Series(cross_val_log_reg).describe()
```

```
[0.85905135 0.85641994 0.8602321  0.8619189  0.86080092 0.85989002
 0.86130697 0.85884417 0.85874296 0.85685368]
--------------------------------------------------
```

Out[115]:

```
count    10.000000
mean      0.859406
std       0.001798
min       0.856420
25%       0.858768
50%       0.859471
75%       0.860659
max       0.861919
dtype: float64
```

Confusion Matrix

In [116…

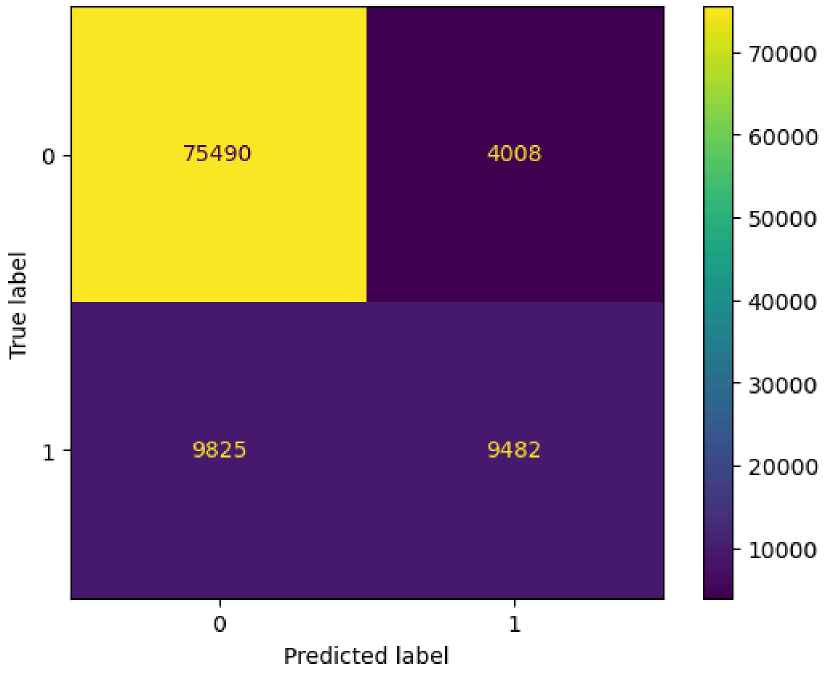
```
log_reg.fit(X_train_scl, y_train)
```

Out[116]:

```
LogisticRegression
LogisticRegression(max_iter=1000, multi_class='auto', random_state=42)
```

In [117…

```
y_pred = log_reg.predict(X_test_scl)
ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred), display_labels=[0,1]).plot()
plt.show()
```



Precision, Recall, F1 score

In [118…

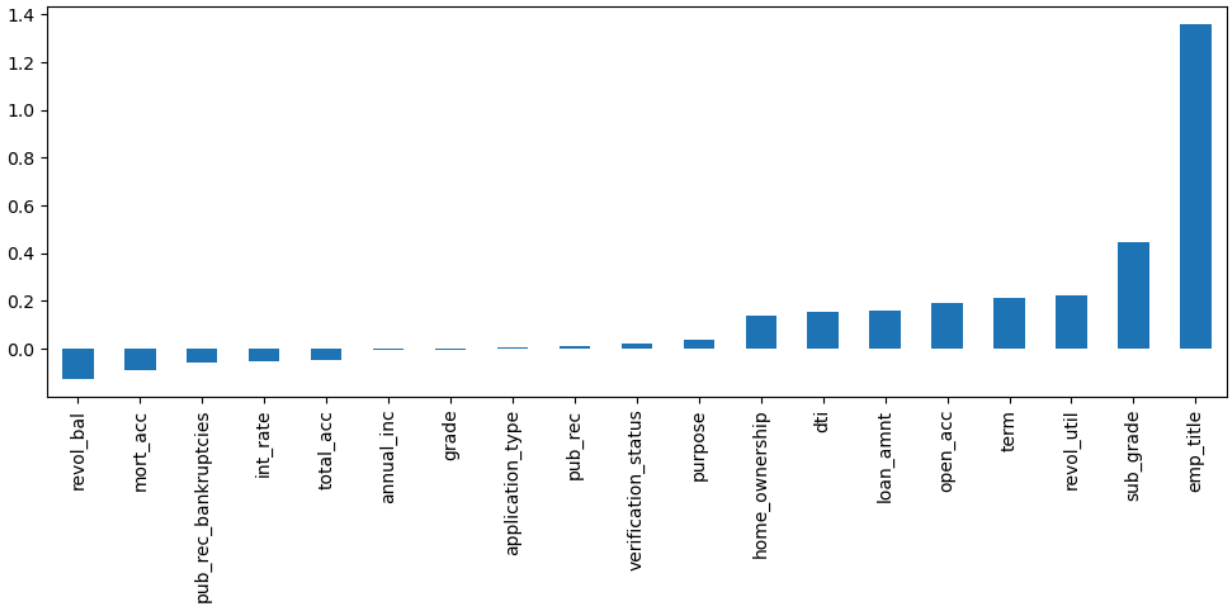
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.95      0.92     79498
           1       0.70      0.49      0.58     19307

    accuracy                           0.86     98805
   macro avg       0.79      0.72      0.75     98805
weighted avg       0.85      0.86      0.85     98805
```

Feature Importances

In [119…

```
pd.Series(log_reg.coef_[0], index=X_train.columns).sort_values().plot(kind='bar', figsize=(12, 4))
plt.show()
```



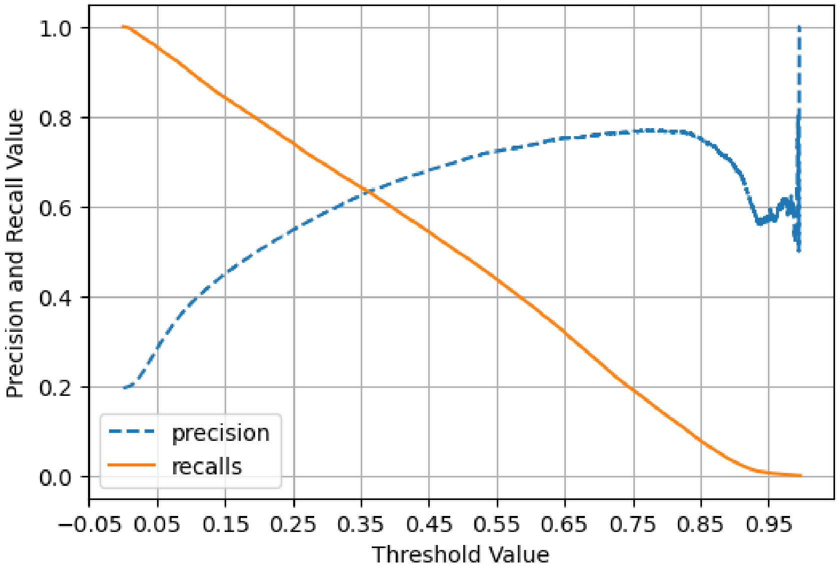
Precision Recall TradeOff

- In the context of this problem, False Negatives are more critical
- We would like to reduce the number of samples which belongs to Class1 but were predicted as Class0 by model

- Thus from a critically standpoint, we should aim to increase our Recall metric
- From a money-making standpoint, we should aim to reduce False Positives or increase Precision Metric

In [120…

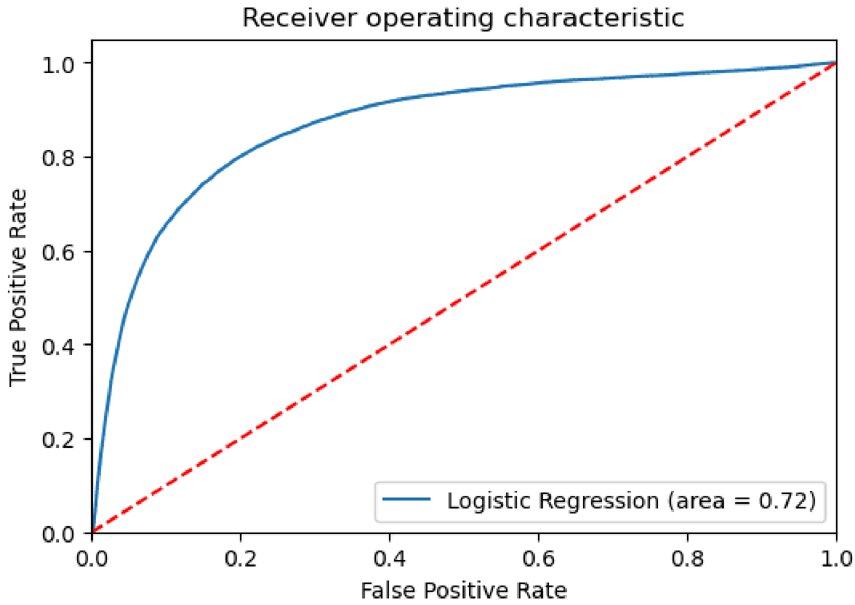
```
def precision_recall_tradeoff(y_test, pred_proba):
    precisions, recalls, thresholds = precision_recall_curve(y_test, pred_proba)

    plt.figure(figsize=(6, 4))
    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_tradeoff(y_test, log_reg.predict_proba(X_test_scl)[:,1])
```



ROC Curve and ROC-AUC

In [121…

```
logit_roc_auc = roc_auc_score(y_test, log_reg.predict(X_test_scl))
fpr, tpr, thresholds = roc_curve(y_test, log_reg.predict_proba(X_test_scl)[:,1])
plt.figure(figsize=(6, 4))
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()
```



Multicollinearity Check

Possible Plan of Action:

- We can iteratively drop features one by one (depending on high vif values) and get rid of multicollinear features.
- Post this exercise we can again fit a model and see if the metrics have improved or not

In [122…
```
def calc_vif(X):
    # Calculating the VIF
     vif = pd.DataFrame()
     vif['Feature'] = X.columns
     vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(len(X.columns))]
     vif['VIF'] = round(vif['VIF'], 2)
     vif = vif.sort_values(by='VIF', ascending = False)
     return vif

X_vif = X.copy()
```

In [123…
```
calc_vif(X_vif)[:5]
```

Out[123]:

	Feature	VIF
7	annual_inc	742.17
16	application_type	639.74
0	loan_amnt	325.36
13	revol_bal	190.05
2	int_rate	181.08

In [124…
```
X_vif.drop('annual_inc', axis=1, inplace=True)
calc_vif(X_vif)[:5]
```

Out[124]:

	Feature	VIF
15	application_type	351.07
0	loan_amnt	271.75
12	revol_bal	185.15
2	int_rate	180.94
4	sub_grade	174.81

In [125…
```
X_vif.drop('application_type', axis=1, inplace=True)
calc_vif(X_vif)[:5]
```

Out[125]:

	Feature	VIF
0	loan_amnt	218.98
12	revol_bal	172.72
2	int_rate	171.30
4	sub_grade	169.12
3	grade	105.27

In [126…

```
X_vif.drop('loan_amnt', axis=1, inplace=True)
calc_vif(X_vif)[:5]
```

Out[126]:

	Feature	VIF
1	int_rate	167.63
3	sub_grade	167.06
11	revol_bal	120.68
2	grade	105.26
7	purpose	59.22

In [127…

```
X_vif.drop('int_rate', axis=1, inplace=True)
calc_vif(X_vif)[:5]
```

Out[127]:

	Feature	VIF
10	revol_bal	117.34
1	grade	105.16
2	sub_grade	104.78
6	purpose	57.88
4	home_ownership	50.53

In [128…

```
X_vif.drop('revol_bal', axis=1, inplace=True)
calc_vif(X_vif)[:5]
```

Out[128]:

	Feature	VIF
1	grade	105.10
2	sub_grade	103.42
6	purpose	51.34
4	home_ownership	44.03
5	verification_status	33.82

In [129…

```
X_vif.drop('grade', axis=1, inplace=True)
calc_vif(X_vif)[:5]
```

Out[129]:

	Feature	VIF
5	purpose	51.31
3	home_ownership	44.00
4	verification_status	33.81
0	term	22.10
2	emp_title	14.48

In [130…

```
X_vif.drop('purpose', axis=1, inplace=True)
calc_vif(X_vif)[:5]
```

Out[130]:

	Feature	VIF
4	verification_status	31.63
3	home_ownership	30.69
0	term	21.72
2	emp_title	14.09
9	total_acc	13.80

In [131…

```
X_vif.drop('verification_status', axis=1, inplace=True)
calc_vif(X_vif)[:5]
```

Out[131]:

	Feature	VIF
3	home_ownership	23.58
0	term	20.09
8	total_acc	13.79
2	emp_title	13.78
5	open_acc	13.26

In [132…

```
X_vif.drop('home_ownership', axis=1, inplace=True)
calc_vif(X_vif)[:5]
```

Out[132]:

	Feature	VIF
0	term	16.28
7	total_acc	13.65
4	open_acc	13.09
2	emp_title	10.48
3	dti	6.84

In [133…

```
X_vif.drop('term', axis=1, inplace=True)
calc_vif(X_vif)[:5]
```

Out[133]:

	Feature	VIF
6	total_acc	13.42
3	open_acc	12.87
1	emp_title	8.44
2	dti	6.84
5	revol_util	5.79

In [134…

```
X_vif.drop('total_acc', axis=1, inplace=True)
calc_vif(X_vif)[:5]
```

Out[134]:

	Feature	VIF
1	emp_title	8.38
2	dti	6.77
3	open_acc	6.43
5	revol_util	5.78
0	sub_grade	5.09

Fitting Logistic Regression model after dropping multicollinear features

In [135…

```
X_vif_train , X_vif_test,  y_vif_train , y_vif_test = train_test_split(X_vif, y, test_size=0.25, ra
print('X_train Shape:', X_train.shape)
print('y_train Shape:', y_train.shape)
print('X_test Shape:', X_test.shape)
print('y_test Shape:', y_test.shape)
```

```
X_vif_train_scl = std_scaler.fit_transform(X_vif_train)
X_vif_test_scl = std_scaler.transform(X_vif_test)
```

```
X_train Shape: (296414, 19)
y_train Shape: (296414,)
X_test Shape: (98805, 19)
y_test Shape: (98805,)
```

In [136…

```
# Accuracy score : Cross validation
log_reg2 = LogisticRegression(penalty='l2', C=1.0, max_iter=1000)
cross_val_log_reg2 = cross_val_score(log_reg2, X_vif_train_scl, y_vif_train, cv=10, scoring='accura
print(cross_val_log_reg2)
print('-'*50)
pd.Series(cross_val_log_reg2).describe()
```

```
[0.85736455 0.85520545 0.8589164  0.85972606 0.85688742 0.85972133
 0.85891164 0.85513309 0.85634763 0.85398603]
--------------------------------------------------
```

Out[136]:

```
count    10.000000
mean      0.857220
std       0.002056
min       0.853986
25%       0.855491
50%       0.857126
75%       0.858915
max       0.859726
dtype: float64
```

In [137…

```
# Fitting model
log_reg2.fit(X_vif_train_scl, y_vif_train)
```

Out[137]:

```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

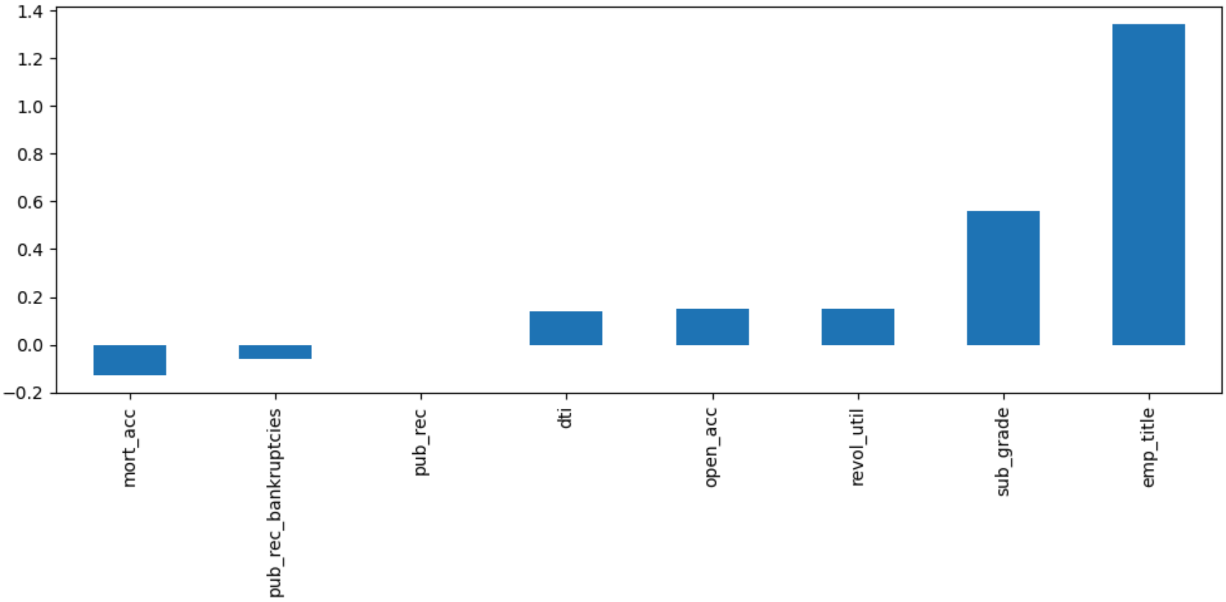
In [138…

```
# Classification Report
y_vif_pred = log_reg2.predict(X_vif_test_scl)
print(classification_report(y_vif_test, y_vif_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.95      0.91     79498
           1       0.69      0.49      0.57     19307

    accuracy                           0.86     98805
   macro avg       0.79      0.72      0.74     98805
weighted avg       0.85      0.86      0.85     98805
```

In [139…

```
# Feature Importances
pd.Series(log_reg2.coef_[0], index=X_vif_train.columns).sort_values().plot(kind='bar', figsize=(12,
plt.show()
```



Handling Imbalanced data using SMOTE

In [140…
```
# # !pip install -U threadpoolctl
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=42)
X_vif_train_scl_smote, y_vif_train_smote = sm.fit_resample(X_vif_train_scl, y_vif_train)
```

In [141…
```
# Before oversampling:
y_vif_train.value_counts(normalize=True)
```

Out[141]:
```
0    0.803599
1    0.196401
Name: loan_status, dtype: float64
```

In [143…
```
# After oversampling:
y_vif_train_smote.value_counts(normalize=True)
```

Out[143]:
```
0    0.5
1    0.5
Name: loan_status, dtype: float64
```

Fitting Model & Cross validation Accuracy

In [144…
```
# Accuracy score : Cross validation
log_reg3 = LogisticRegression(penalty='l2', C=1.0, max_iter=1000)
cross_val_log_reg3 = cross_val_score(log_reg3, X_vif_train_scl_smote, y_vif_train_smote, cv=10, sc
print(cross_val_log_reg3)
print('-'*50)
pd.Series(cross_val_log_reg3).describe()
```

```
[0.79531906 0.79584383 0.79922334 0.80140638 0.8        0.79848866
 0.79968093 0.79829551 0.79812758 0.79898822]
--------------------------------------------------
```

Out[144]:
```
count    10.000000
mean      0.798537
std       0.001832
min       0.795319
25%       0.798170
50%       0.798738
75%       0.799567
max       0.801406
dtype: float64
```

Precision Recall, F1 score

In [145…
```
# Fitting model
log_reg3.fit(X_vif_train_scl_smote, y_vif_train_smote)
```

Out[145]:
```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

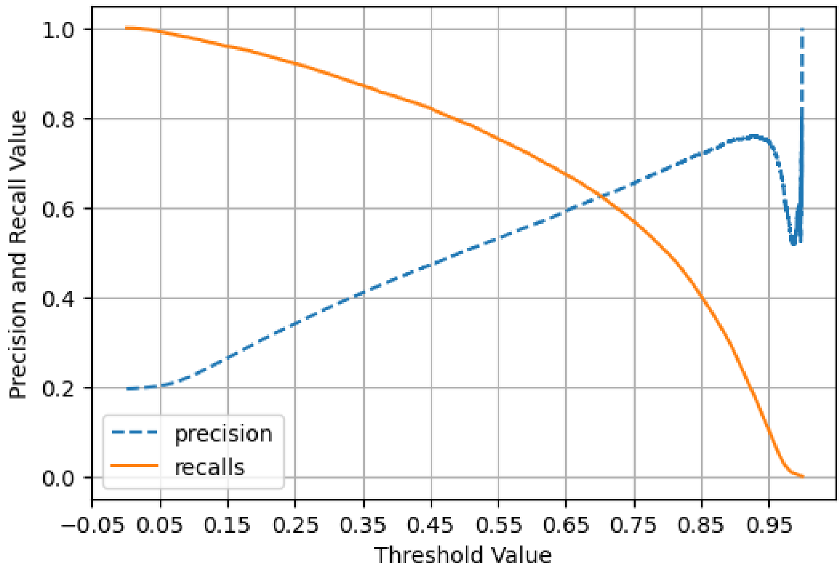
In [146…
```
# Classification Report
y_vif_smote_pred = log_reg2.predict(X_vif_test_scl)
print(classification_report(y_vif_test, y_vif_smote_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.95      0.91     79498
           1       0.69      0.49      0.57     19307

    accuracy                           0.86     98805
   macro avg       0.79      0.72      0.74     98805
weighted avg       0.85      0.86      0.85     98805
```

Precision Recall TradeOff

In [147…
```
def precision_recall_tradeoff(y_test, pred_proba):
    precisions, recalls, thresholds = precision_recall_curve(y_test, pred_proba)
```

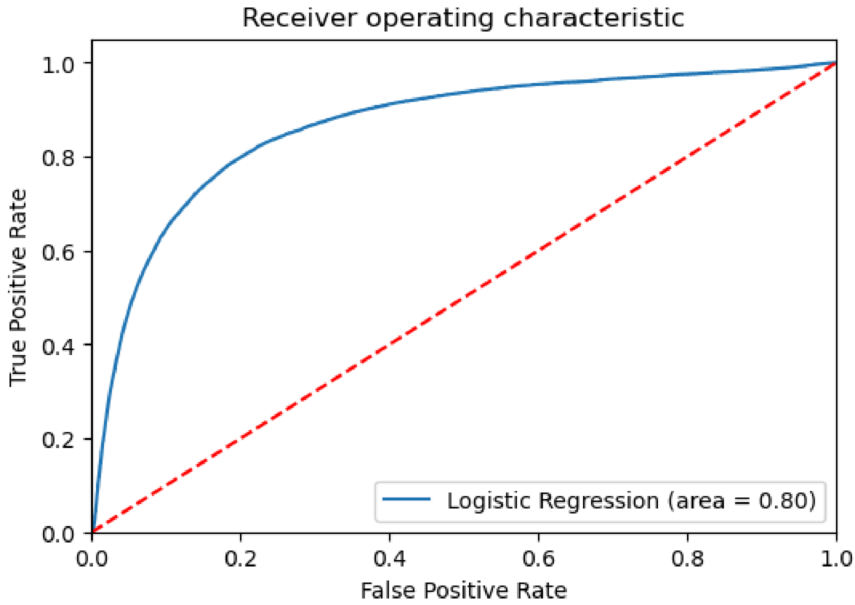
```
    plt.figure(figsize=(6, 4))
    threshold_boundary = thresholds.shape[0]
    # plot precision
    plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='--', label='precision')
    # plot recall
    plt.plot(thresholds, recalls[0:threshold_boundary], label='recalls')

    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_tradeoff(y_vif_test, log_reg3.predict_proba(X_vif_test_scl)[:,1])
```



ROC Curve and ROC-AUC (ROC-AUC has improved)

In [148…

```
logit_roc_auc = roc_auc_score(y_vif_test, log_reg3.predict(X_vif_test_scl))
fpr, tpr, thresholds = roc_curve(y_vif_test, log_reg3.predict_proba(X_vif_test_scl)[:,1])
plt.figure(figsize=(6, 4))
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()
```



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 slightly higher for those who's loan status is as fully paid.
- Most of the borrowers take loans for debt-consolidation and credit card payoffs. Probability of defaulters is higher in the small_business owner borrowers.
- debt-to-income ratio is higher for defaulters.
- As number of derogatory public records increases, the probability of borrowers declared as defaulters also increases
- Application type Direct-Pay has higher probability of default than individual and joint.

Insights & Recommendations:

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 report, company 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:
- For borrowers having more than 0 derogatory public records, public recorded bankruptcy present in past history, we 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.

One complexity we can include in future modelling is to tune precision/recall based on loan amount.

- If loan amount is above a certain threshold, we can play conservative and prioritize recall.
- If loan amount is below a certain threshold, we can prioritize precision.