```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import math as m
from statsmodels.stats import weightstats as stests
from scipy import stats
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
import statsmodels.api as sm
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import precision_recall_curve,precision_score,recall_score
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import (
    accuracy_score, confusion_matrix, classification_report,
    roc_auc_score, roc_curve, auc,
    ConfusionMatrixDisplay, RocCurveDisplay
)
from statsmodels.stats.outliers_influence import variance_inflation_factor
from imblearn.over_sampling import SMOTE
```

```
df=pd.read_csv('/content/logistic_regression.csv')
df.head()
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	...	open_acc	pub_re
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	117000.0	...	16.0	0.(
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	...	17.0	0.(
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RENT	43057.0	...	13.0	0.(
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	RENT	54000.0	...	6.0	0.(
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000.0	...	13.0	0.(

5 rows × 27 columns

```
df.shape
```

```
(396030, 27)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153108 entries, 0 to 153107
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   loan_amnt             153108 non-null  float64
 1   term                  153108 non-null  object
 2   int_rate              153108 non-null  float64
 3   installment           153108 non-null  float64
 4   grade                 153108 non-null  object
 5   sub_grade             153108 non-null  object
 6   emp_title             144251 non-null  object
 7   emp_length            146003 non-null  object
 8   home_ownership        153108 non-null  object
 9   annual_inc            153108 non-null  float64
```

```
 10  verification_status   153108 non-null  object
 11  issue_d               153108 non-null  object
 12  loan_status           153108 non-null  object
 13  purpose               153108 non-null  object
 14  title                 152436 non-null  object
 15  dti                   153108 non-null  float64
 16  earliest_cr_line      153108 non-null  object
 17  open_acc              153108 non-null  float64
 18  pub_rec               153108 non-null  float64
 19  revol_bal             153108 non-null  float64
 20  revol_util            153009 non-null  float64
 21  total_acc             153107 non-null  float64
 22  initial_list_status   153107 non-null  object
 23  application_type      153107 non-null  object
 24  mort_acc              138532 non-null  float64
 25  pub_rec_bankruptcies  152900 non-null  float64
 26  address               153107 non-null  object
dtypes: float64(12), object(15)
memory usage: 31.5+ MB
```

```
df.describe()
```

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_bal	revol_util
count	153108.000000	153108.000000	153108.000000	1.531080e+05	153108.000000	153108.000000	153108.000000	1.531080e+05	153009.000000
mean	14109.079212	13.642356	431.699796	7.427598e+04	17.323505	11.299468	0.178848	1.581667e+04	53.789170
std	8366.358660	4.461013	250.947422	6.047275e+04	8.130035	5.134781	0.521013	2.059875e+04	24.532567
min	500.000000	5.320000	16.250000	2.500000e+03	0.000000	0.000000	0.000000	0.000000e+00	0.000000
25%	8000.000000	10.490000	250.330000	4.500000e+04	11.257500	8.000000	0.000000	6.034000e+03	35.800000
50%	12000.000000	13.330000	375.380000	6.400000e+04	16.880000	10.000000	0.000000	1.116900e+04	54.900000
75%	20000.000000	16.490000	568.040000	9.000000e+04	22.950000	14.000000	0.000000	1.962600e+04	72.900000
max	40000.000000	30.990000	1533.810000	7.446395e+06	189.900000	90.000000	40.000000	1.743266e+06	892.300000

```
df.columns
```

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

```
df_num=df.select_dtypes(include='number')
df_cat=df.select_dtypes(include='object')
```

```
df_num.corr()
```

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	mort_
loan_amnt	1.000000	0.166946	0.954097	0.348422	0.038128	0.197100	-0.079874	0.327194	0.099845	0.221939	0.22
int_rate	0.166946	1.000000	0.160552	-0.060417	0.175971	0.007656	0.063532	-0.010659	0.296522	-0.039757	-0.08
installment	0.954097	0.160552	1.000000	0.342433	0.035774	0.187414	-0.069317	0.316280	0.123915	0.201167	0.19
annual_inc	0.348422	-0.060417	0.342433	1.000000	-0.180989	0.139216	-0.010967	0.288734	0.027514	0.198131	0.23
dti	0.038128	0.175971	0.035774	-0.180989	1.000000	0.302754	-0.039417	0.142528	0.191705	0.227638	-0.05
open_acc	0.197100	0.007656	0.187414	0.139216	0.302754	1.000000	-0.016834	0.222224	-0.133404	0.680634	0.11
pub_rec	-0.079874	0.063532	-0.069317	-0.010967	-0.039417	-0.016834	1.000000	-0.102451	-0.077147	0.020573	0.01
revol_bal	0.327194	-0.010659	0.316280	0.288734	0.142528	0.222224	-0.102451	1.000000	0.225033	0.193074	0.19
revol_util	0.099845	0.296522	0.123915	0.027514	0.191705	-0.133404	-0.077147	0.225033	1.000000	-0.102758	0.01
total_acc	0.221939	-0.039757	0.201167	0.198131	0.227638	0.680634	0.020573	0.193074	-0.102758	1.000000	0.38
mort_acc	0.224667	-0.085317	0.196060	0.238239	-0.055543	0.113817	0.012185	0.195786	0.010181	0.382764	1.00
pub_rec_bankruptcies	-0.108374	0.058053	-0.100313	-0.051659	-0.029090	-0.026784	0.715249	-0.123977	-0.087028	0.041136	0.02

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

We can see installment and loan amount are positivly correlated hence we can drop anyone of the column

```
df.drop('installment',axis=1,inplace=True)
```

Missing value Detection

```
(df.isna().sum()/len(df)*100).sort_values(ascending=False)
```

	0
mort_acc	9.543469
emp_title	5.789208
emp_length	4.621115
title	0.443401
pub_rec_bankruptcies	0.135091
revol_util	0.069692
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
loan_amnt	0.000000
term	0.000000
purpose	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
int_rate	0.000000
address	0.000000

Handlig Missing values

```
df['mort_acc'] = SimpleImputer(strategy = "median").fit_transform(df[['mort_acc']])
```

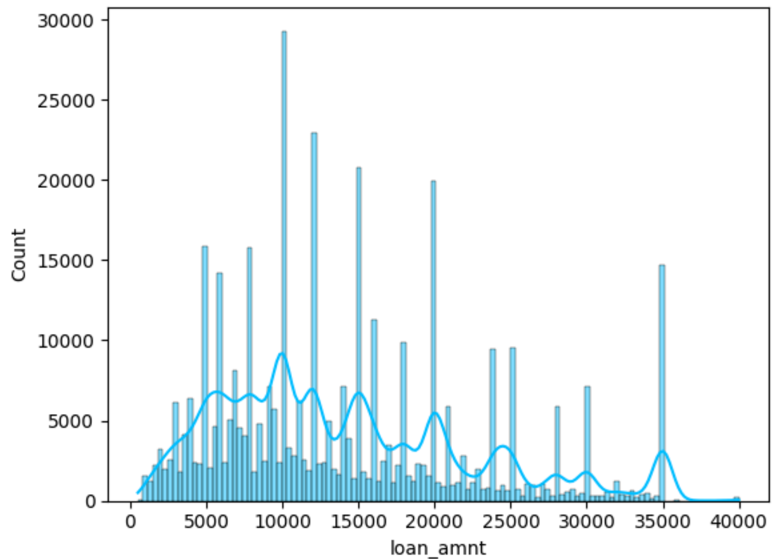
```
df['mort_acc'].isna().sum()
```

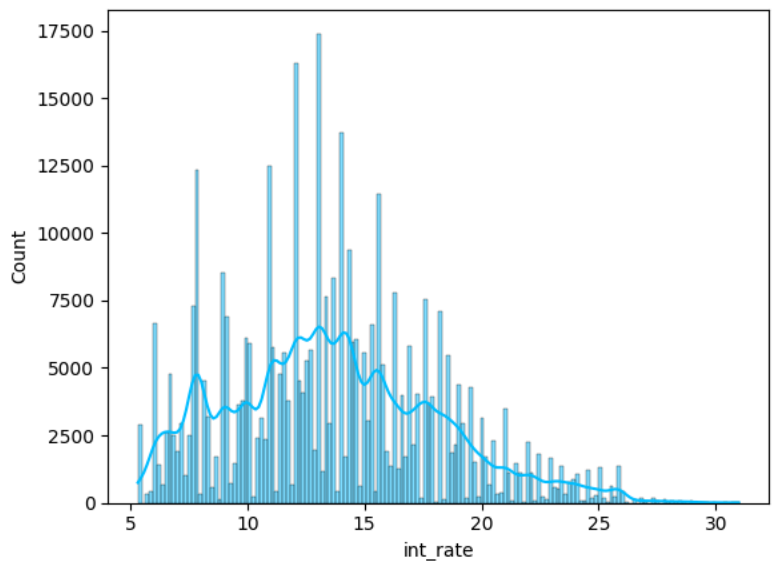
```
0
```

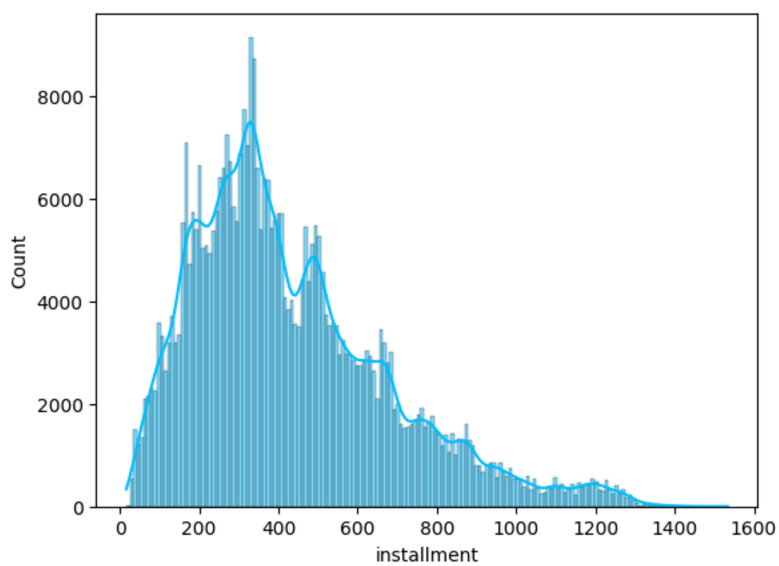
```
df.dropna(inplace=True)
```

Univariate analysis

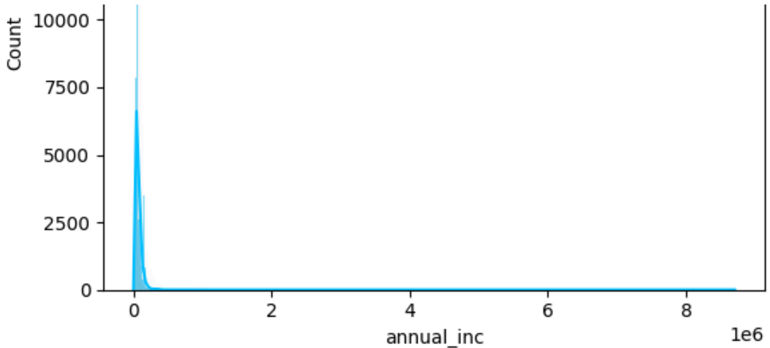
```
for col in enumerate(df_num):
  sns.histplot(df_num[col[1]],kde=True,color='deepskyblue')
  plt.show()
```

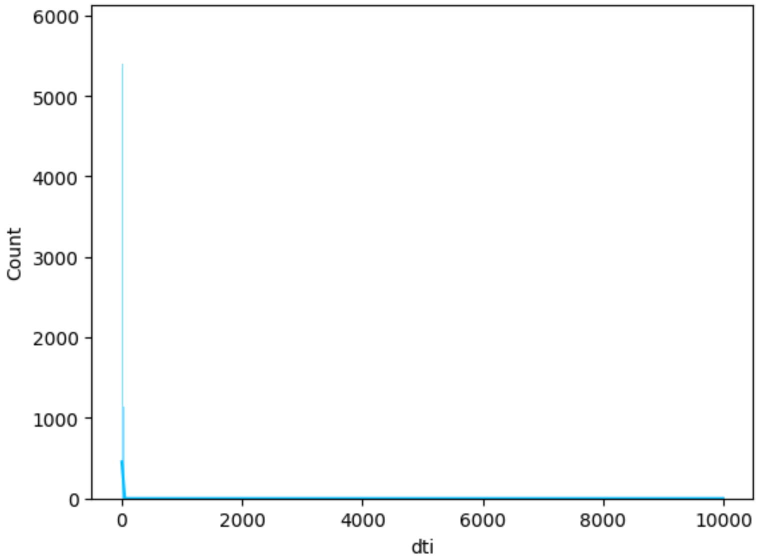

Count
loan_amnt
0
5000
10000
15000
20000
25000
30000
35000
40000

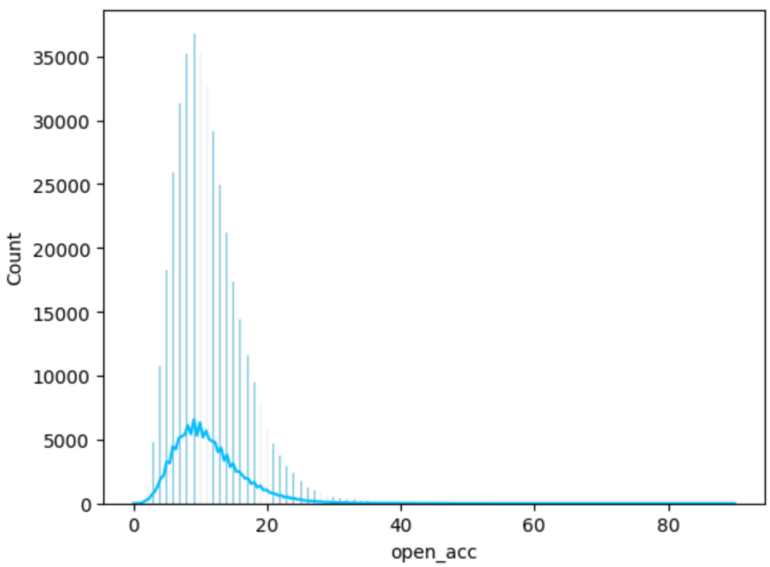

Count
int_rate
0
2500
5000
7500
10000
12500
15000
17500
5
10
15
20
25
30


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

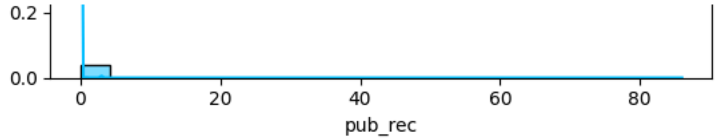

12500
15000
17500

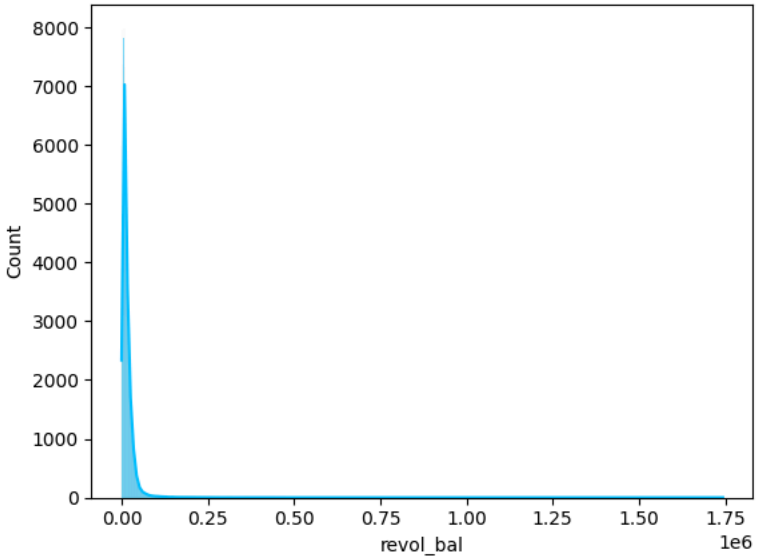

Count
10000
7500
5000
2500
0
0
2
4
6
8
annual_inc
1e6

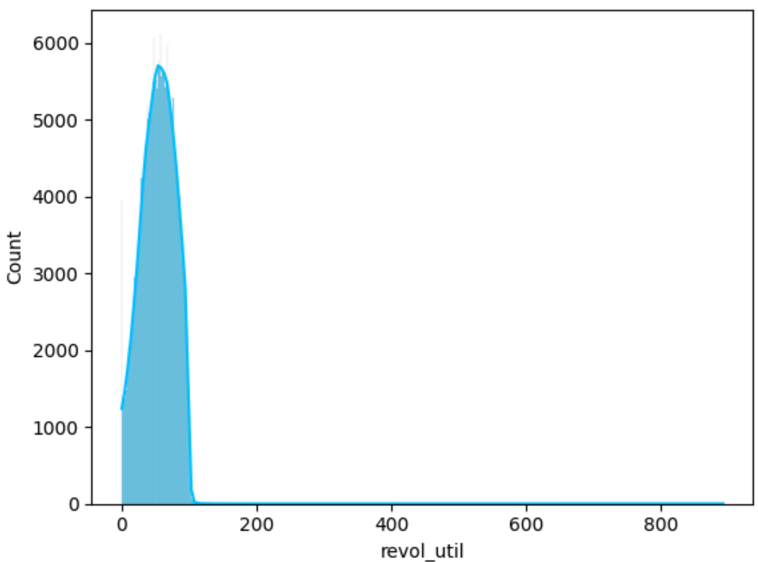

6000
5000
4000
Count
3000
2000
1000
0
0
2000
4000
6000
8000
10000
dti

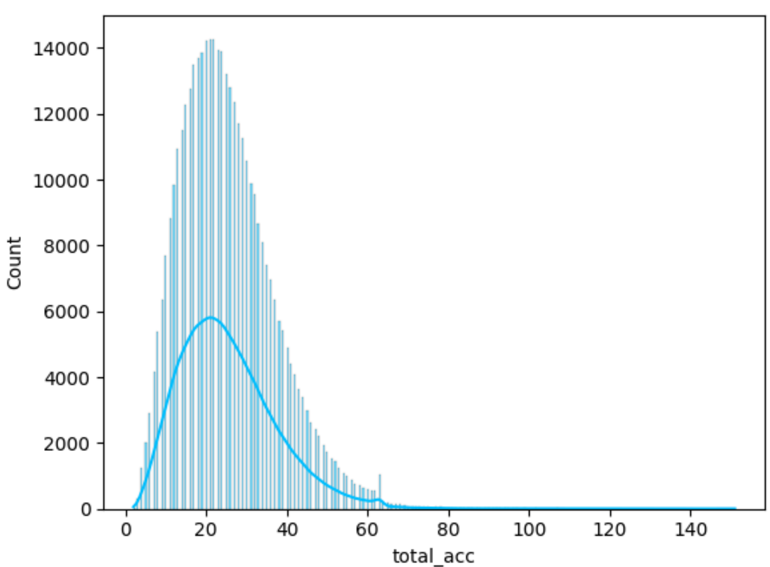

35000
30000
25000
Count
20000
15000
10000
5000
0
0
20
40
60
80
open_acc

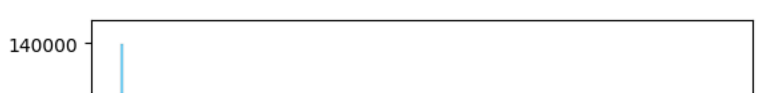

1e7
1.4
1.2
1.0
Count
0.8
0.6
0.4

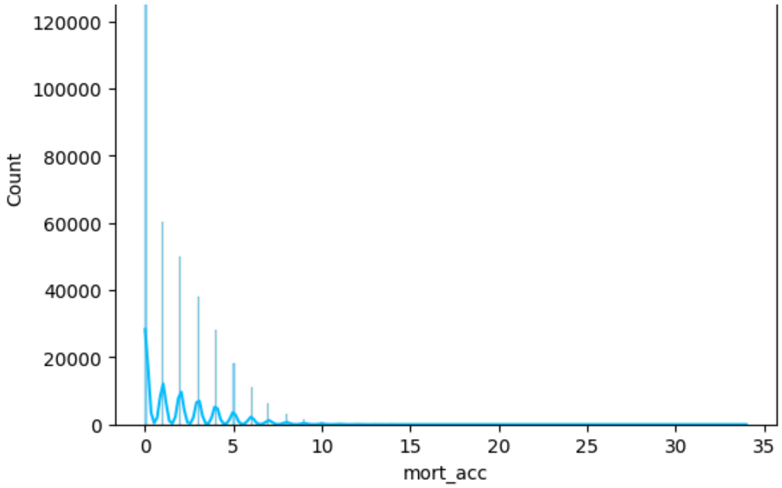

0.2
0.0
0
20
40
60
80
pub_rec

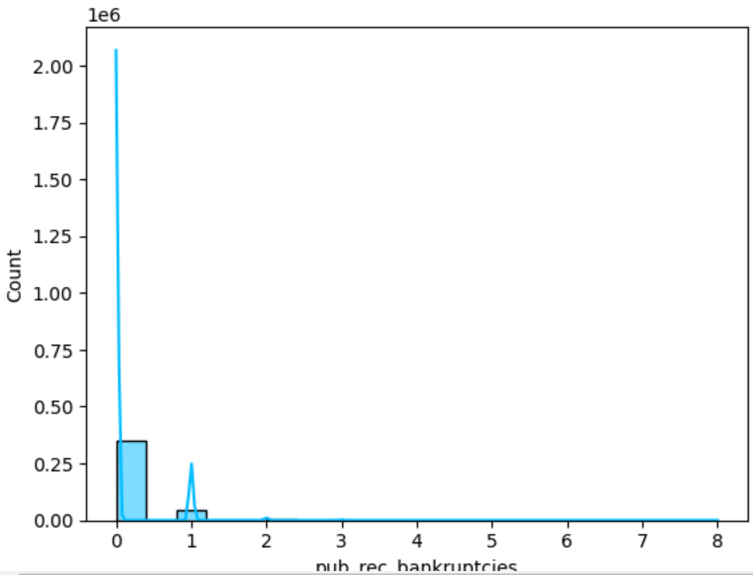

8000
7000
6000
5000
4000
3000
2000
1000
0
Count
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1e6
revol_bal


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


14000
12000
10000
8000
6000
4000
2000
0
Count
0
20
40
60
80
100
120
140
total_acc


140000


Count
120000
100000
80000
60000
40000
20000
0
0
5
10
15
20
25
30
35
mort_acc


1e6
Count
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
1
2
3
4
5
6
7
8

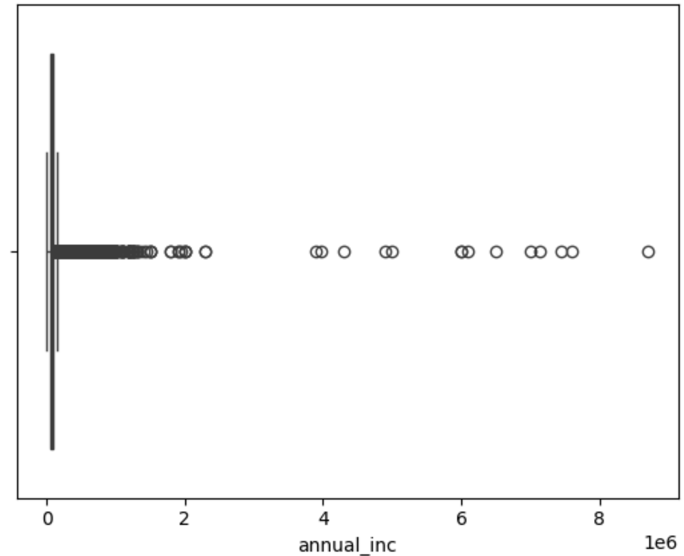
We can see most of the variable distribution is right skewed and

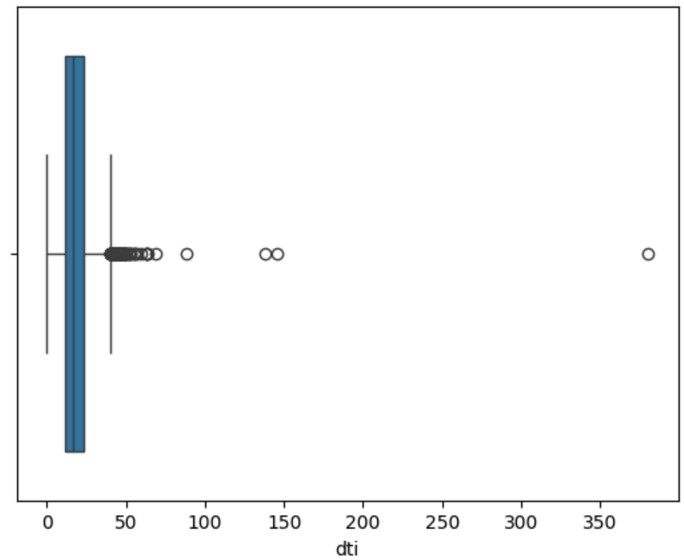
outlier are present in 'annual_inc', 'dti', 'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'mort_acc', 'pub_rec_bankruptcies'

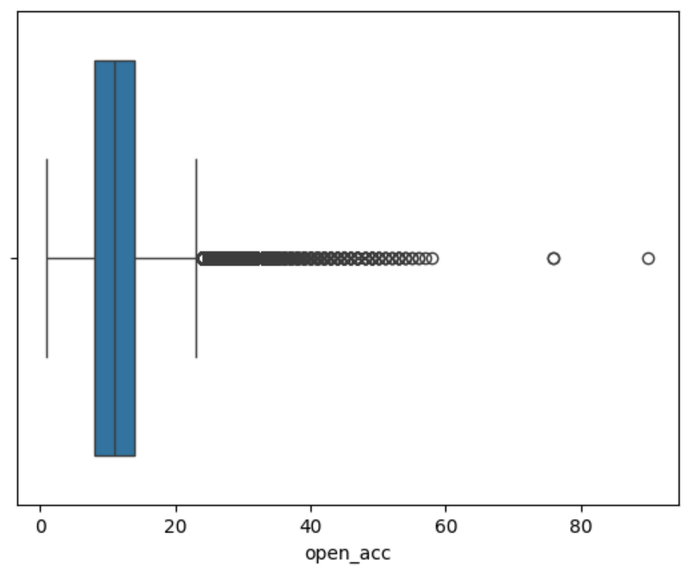
Outlier Detection

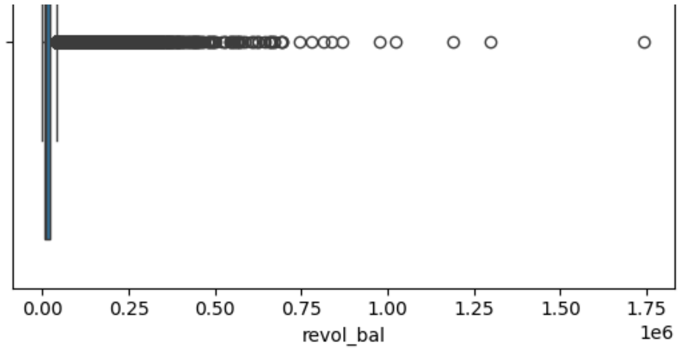
```
outlier=['annual_inc', 'dti', 'open_acc', 'revol_bal', 'revol_util', 'mort_acc', 'pub_rec_bankruptcies']
```

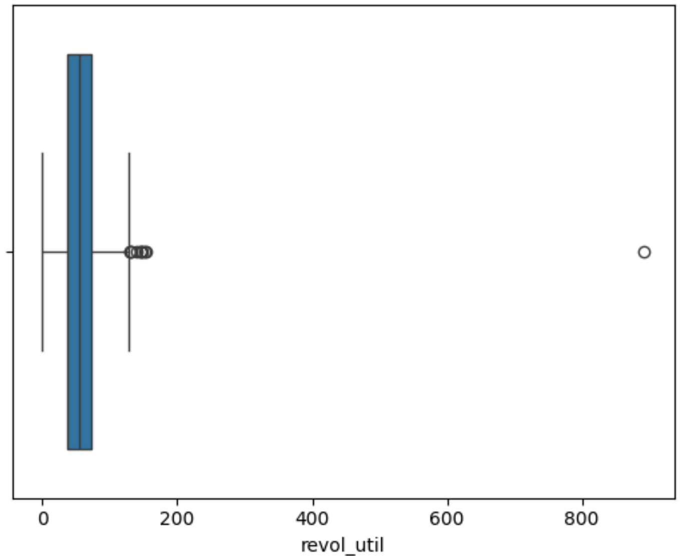
```
for col in enumerate(outlier):
  sns.boxplot(x=df[col[1]])
  plt.show()
```

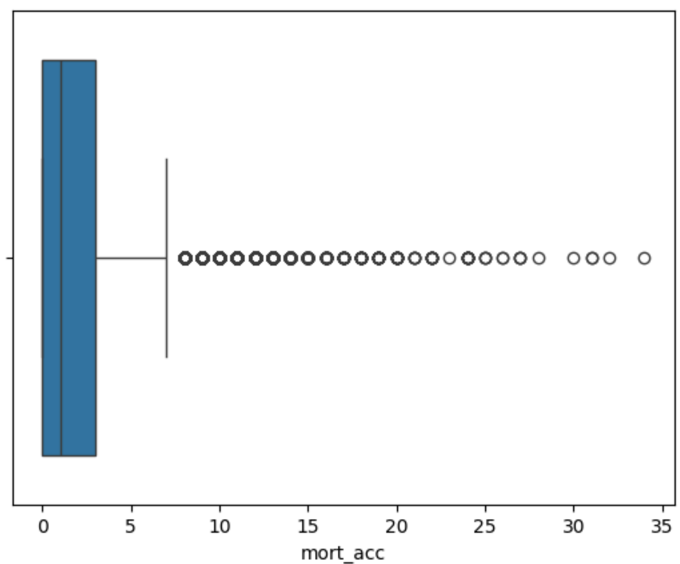

0
2
4
6
8
1e6
annual_inc

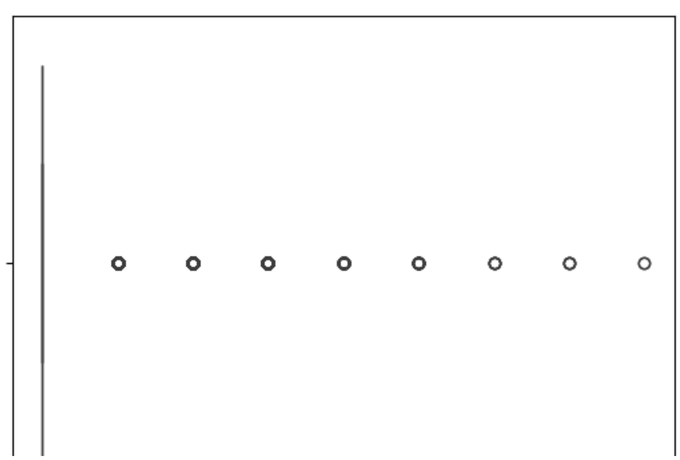

0
50
100
150
200
250
300
350
dti

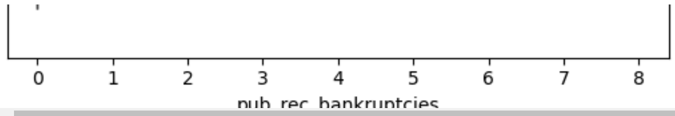

0
20
40
60
80
open_acc


0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1e6
revol_bal


0
200
400
600
800
revol_util


0
5
10
15
20
25
30
35
mort_acc




0
1
2
3
4
5
6
7
8
pub_rec_bankruptcies

Outlier Treatment

```
x_outlier=df[['open_acc','revol_util']]
```

```
q1=x_outlier.quantile(0.25)
q3=x_outlier.quantile(0.75)
iqr=q3-q1
print(iqr)
```

```
open_acc       6.0
revol_util    36.9
dtype: float64
```

```
df = df[~(x_outlier[(x_outlier < (q1 - 1.5*iqr)) | (x_outlier > (q3 + 1.5*iqr))]).any(axis=1)]
df.head()
```

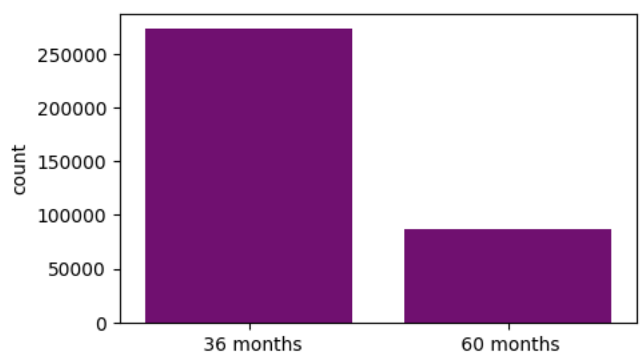
	loan_amnt	term	int_rate	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verification_status	...	open_acc
0	10000.0	36 months	11.44	B	B4	Marketing	10+ years	RENT	117000.0	Not Verified	...	16.0
1	8000.0	36 months	11.99	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	Not Verified	...	17.0
2	15600.0	36 months	10.49	B	B3	Statistician	< 1 year	RENT	43057.0	Source Verified	...	13.0
3	7200.0	36 months	6.49	A	A2	Client Advocate	6 years	RENT	54000.0	Not Verified	...	6.0
4	24375.0	60 months	17.27	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000.0	Verified	...	13.0

5 rows × 26 columns

Univariate Analysis of categorical values

```
fig = plt.figure(figsize=(5,3))
plt.xlabel('term')
sns.countplot(data=df, x='term', order = df['term'].value_counts().index,color='purple')
```

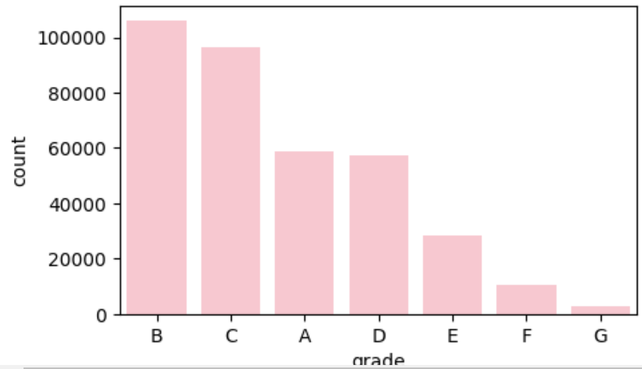
```
<Axes: xlabel='term', ylabel='count'>
```



From the graph,we can see most of the borrower chose 36 months as loan term

```
fig = plt.figure(figsize=(5,3))
plt.xlabel('grade')
sns.countplot(data=df, x='grade', order = df['grade'].value_counts().index,color='pink')
```

```
<Axes: xlabel='grade', ylabel='count'>
```



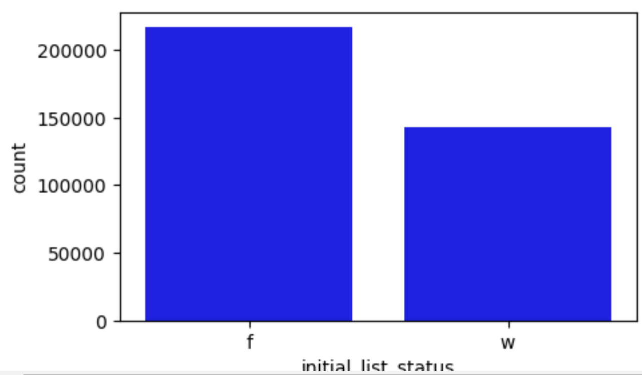
From the graph,B grade is the highest risk grade given to most of the borrower

```
fig = plt.figure(figsize=(10,8))
plt.xlabel('sub_grade')
plt.xticks(rotation = 75)
sns.countplot(data=df, x='sub_grade', order = df['sub_grade'].value_counts().index,color='grey')d
```

```
  File "<ipython-input-120-2738a771aaab>", line 4
    sns.countplot(data=df, x='sub_grade', order = df['sub_grade'].value_counts().index,color='grey')d
                                                                                                    ^
SyntaxError: invalid syntax
```

```
fig = plt.figure(figsize=(5,3))
plt.xlabel('initial_list_status')
sns.countplot(data=df, x='initial_list_status', order = df['initial_list_status'].value_counts().index,color='blue')
```

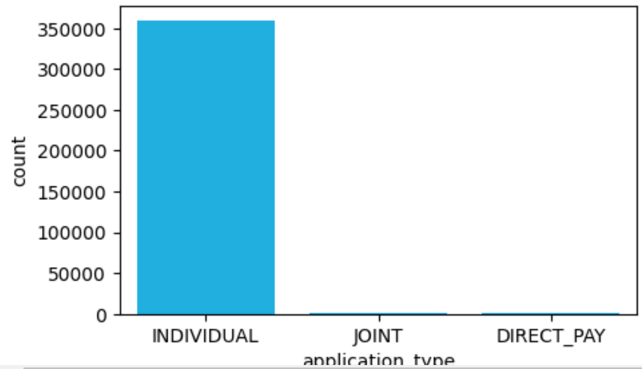
```
<Axes: xlabel='initial_list_status', ylabel='count'>
```



Most of the listed loan status is F(fractional)

```
fig = plt.figure(figsize=(5,3))
plt.xlabel('application_type')
sns.countplot(data=df, x='application_type', order = df['application_type'].value_counts().index,color='deepskyblue')
```

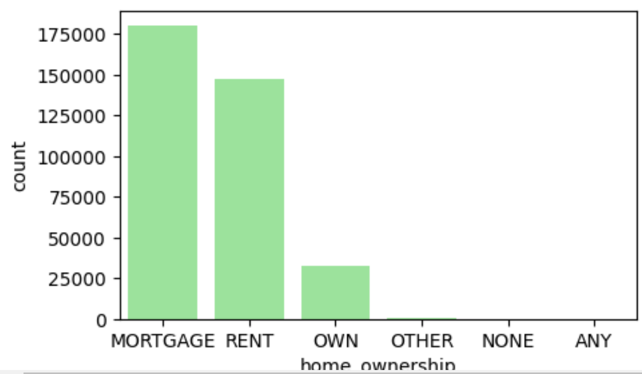
```
<Axes: xlabel='application_type', ylabel='count'>
```



We can see almost all account are indivdual accounts

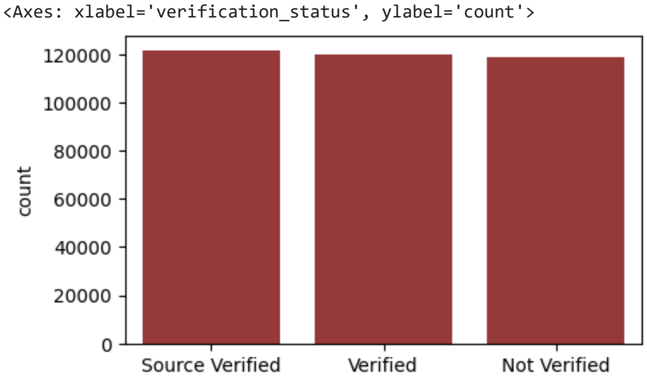
```
fig = plt.figure(figsize=(5,3))
plt.xlabel('home_ownership')
sns.countplot(data=df, x='home_ownership', order = df['home_ownership'].value_counts().index,color='lightgreen')
```

```
<Axes: xlabel='home_ownership', ylabel='count'>
```




```
fig = plt.figure(figsize=(5,3))
plt.xlabel('verification_status')
sns.countplot(data=df, x='verification_status', order = df['verification_status'].value_counts().index,color='brown')
```

```
<Axes: xlabel='verification_status', ylabel='count'>
```



So from that we can infer that people with grade 'B' and subgrade 'B3' are more likely to fully pay the loan.

Feature Engineering

Checking Duplicate values in dataframe

```
df.duplicated().sum()
```

```
0
```

Modify the existing variables

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

pub_rec_bankruptcies	count
0.0	320836
1.0	37803
2.0	1675
3.0	321
4.0	66
5.0	30
6.0	5
7.0	4
8.0	2

```
def mortfunc(x):
  if x==0:
    return 0
  else:
    return 1
def pub_rec_bankrfunc(x):
  if x==0:
    return 0
  else:
    return 1
def pub_rec_func(x):
  if x==0:
    return 0
  else:
    return 1
```

```
df['mort_acc']=df['mort_acc'].apply(mortfunc)
df['pub_rec']=df['pub_rec'].apply(pub_rec_func)
df['pub_rec_bankruptcies']=df['pub_rec_bankruptcies'].apply(pub_rec_bankrfunc)
```

```
df['issue_d'].head(2)
```

	issue_d
0	Jan-2015
1	Jan-2015

```
df['earliest_cr_line'].head(2)
```

	earliest_cr_line
0	Jun-1990
1	Jul-2004

Modify Date variables

```
import warnings
warnings.filterwarnings("ignore")
```

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

```
df['issue_d_month']=df['issue_d'].dt.month
df['issue_d_year']=df['issue_d'].dt.year
df['earliest_cr_line_mon']=df['earliest_cr_line'].dt.month
df['earliest_cr_line_year']=df['earliest_cr_line'].dt.year
df.drop(['issue_d','earliest_cr_line'],axis=1,inplace=True)
```

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

```
def emp_length_func(x):
  if x=='< 1 year':
    return 0.5
  elif x=='1 year':
    return 1
  elif x=='2 years':
    return 2
  elif x=='3 years':
    return 3
  elif x=='4 years':
    return 4
  elif x=='5 years':
    return 5
  elif x=='6 years':
    return 6
  elif x=='7 years':
    return 7
  elif x=='8 years':
    return 9
  elif x=='9 years':
    return 9
  elif x=='10+ years':
    return 10
```

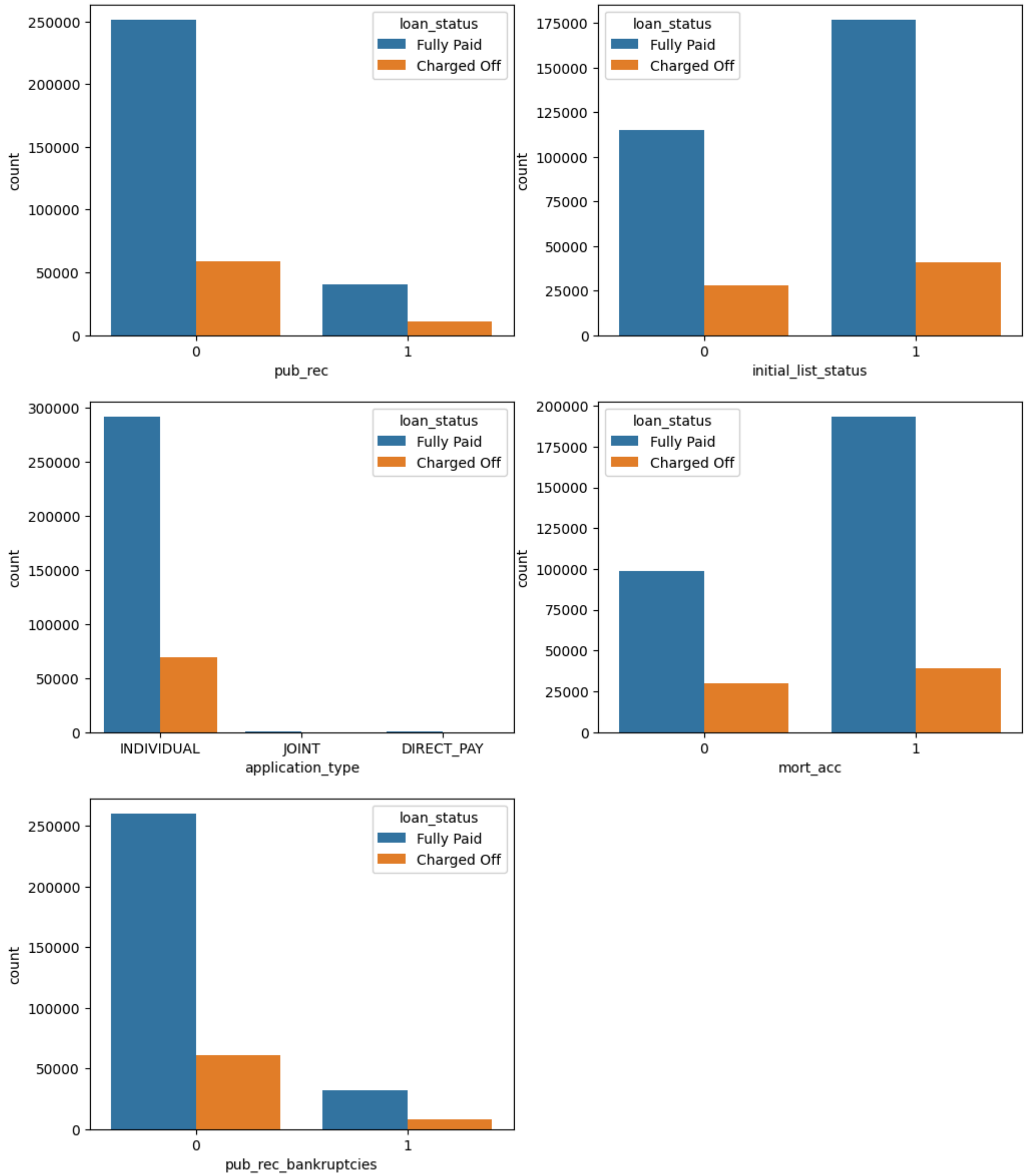
```
df['emp_length']=df['emp_length'].apply(emp_length_func)
```

```
#Modify the initial_list_status
df['initial_list_status']=df['initial_list_status'].map({'w':0,'f':1})
```

Bivariate Analysis

```
plt.figure(figsize=(12,30))

plt.subplot(6,2,1)
sns.countplot(x='pub_rec',data=df,hue='loan_status')

plt.subplot(6,2,2)
sns.countplot(x='initial_list_status',data=df,hue='loan_status')

plt.subplot(6,2,3)
sns.countplot(x='application_type',data=df,hue='loan_status')

plt.subplot(6,2,4)
sns.countplot(x='mort_acc',data=df,hue='loan_status')

plt.subplot(6,2,5)
sns.countplot(x='pub_rec_bankruptcies',data=df,hue='loan_status')
```

<Axes: xlabel='pub_rec_bankruptcies', ylabel='count'>

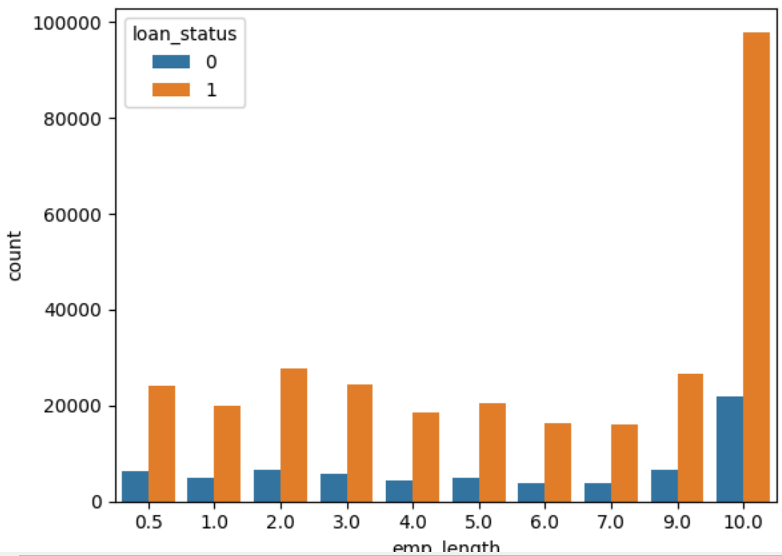


Indivdual bank account holders are likely to pay the loan amount fully

```
# Let's fetch ZIP from address and then drop the remaining details -
df['zip_code'] = df.address.apply(lambda x: x[-5:])
df['zip_code'].value_counts(normalize=True)*100
```

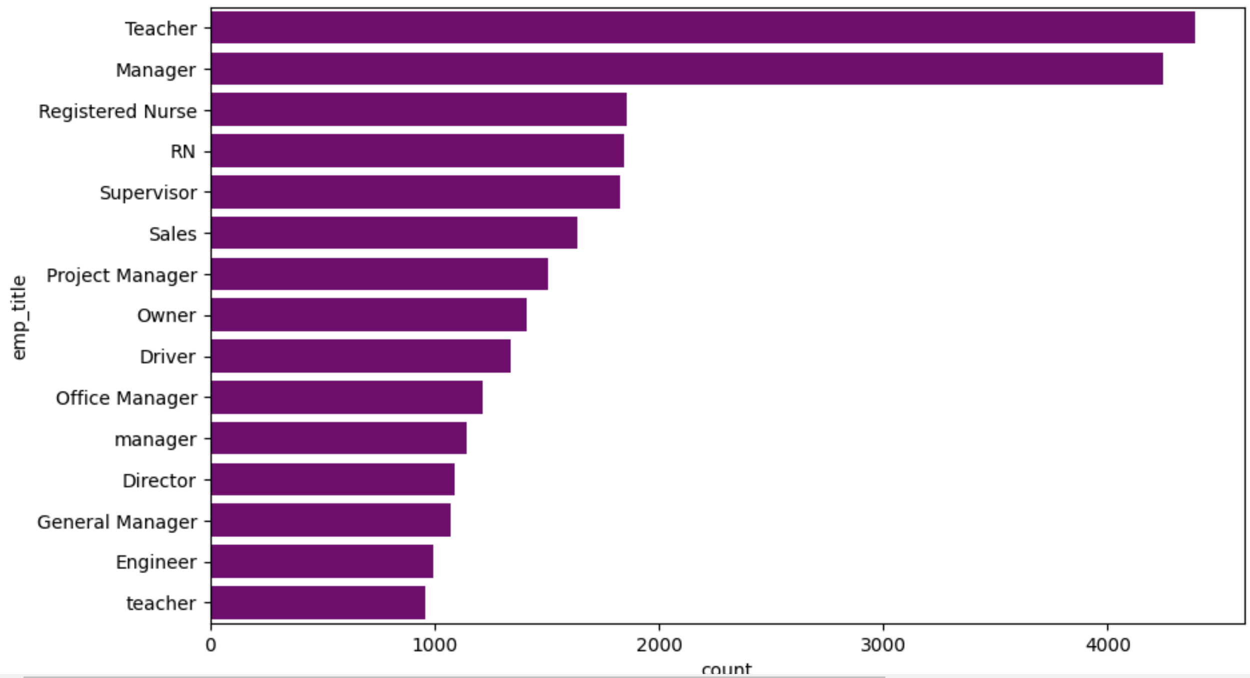
zip_code	proportion
70466	14.381192
30723	14.262825
22690	14.258112
48052	14.135310
00813	11.638789
29597	11.559508
05113	11.531510
93700	2.759590
11650	2.752937
86630	2.720227

```
sns.countplot(x='emp_length',data=df,hue='loan_status')
plt.show()
```



Person who employed for more than 10 years has successfully paid of the loan

```
plt.figure(figsize=(10,6))
sns.countplot(y='emp_title',data=df,order=pd.value_counts(df['emp_title']).iloc[:15].index,color='purple')
plt.show()
```



Teacher and Manager are the most employement title can afford load payment.

```
# Dropping some variables which we can let go for now
df.drop(columns=['emp_title', 'title', 'sub_grade',
                   'address'],
                   axis=1, inplace=True)
```

One hot encoding for remaing categorical columns

```
df.select_dtypes(include='object').head(3)
```

	grade	home_ownership	verification_status	loan_status	purpose	application_type	zip_code
0	B	RENT	Not Verified	Fully Paid	vacation	INDIVIDUAL	22690
1	B	MORTGAGE	Not Verified	Fully Paid	debt_consolidation	INDIVIDUAL	05113
2	B	RENT	Source Verified	Fully Paid	credit_card	INDIVIDUAL	05113

```
dummies=['purpose', 'zip_code', 'grade', 'verification_status', 'application_type', 'home_ownership']
df=pd.get_dummies(df,columns=dummies,drop_first=True)
pd.set_option('display.max_columns',None)
pd.set_option('display.max_rows',None)
```

```
conv=['purpose_credit_card', 'purpose_debt_consolidation',
       'purpose_educational', 'purpose_home_improvement', 'purpose_house',
       'purpose_major_purchase', 'purpose_medical', 'purpose_moving',
       'purpose_other', 'purpose_renewable_energy', 'purpose_small_business',
       'purpose_vacation', 'purpose_wedding', 'zip_code_05113',
       'zip_code_11650', 'zip_code_22690', 'zip_code_29597', 'zip_code_30723',
       'zip_code_48052', 'zip_code_70466', 'zip_code_86630', 'zip_code_93700',
       'grade_B', 'grade_C', 'grade_D', 'grade_E', 'grade_F', 'grade_G',
       'verification_status_Source Verified', 'verification_status_Verified',
       'application_type_INDIVIDUAL', 'application_type_JOINT',
       'home_ownership_MORTGAGE', 'home_ownership_NONE',
       'home_ownership_OTHER', 'home_ownership_OWN', 'home_ownership_RENT']
```

```
for i in enumerate(conv):
  X[i[1]]=X[i[1]].astype(int)
```

```
df.columns
```

```
Index(['loan_amnt', 'term', 'int_rate', 'emp_length', 'annual_inc',
       'loan_status', 'dti', 'open_acc', 'pub_rec', 'revol_bal', 'revol_util',
```

```
       'total_acc', 'initial_list_status', 'mort_acc', 'pub_rec_bankruptcies',
       'purpose_credit_card', 'purpose_debt_consolidation',
       'purpose_educational', 'purpose_home_improvement', 'purpose_house',
       'purpose_major_purchase', 'purpose_medical', 'purpose_moving',
       'purpose_other', 'purpose_renewable_energy', 'purpose_small_business',
       'purpose_vacation', 'purpose_wedding', 'zip_code_05113',
       'zip_code_11650', 'zip_code_22690', 'zip_code_29597', 'zip_code_30723',
       'zip_code_48052', 'zip_code_70466', 'zip_code_86630', 'zip_code_93700',
       'grade_B', 'grade_C', 'grade_D', 'grade_E', 'grade_F', 'grade_G',
       'verification_status_Source Verified', 'verification_status_Verified',
       'application_type_INDIVIDUAL', 'application_type_JOINT',
       'home_ownership_MORTGAGE', 'home_ownership_NONE',
       'home_ownership_OTHER', 'home_ownership_OWN', 'home_ownership_RENT'],
      dtype='object')
```

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

	count
loan_status	
Fully Paid	291766
Charged Off	68976

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

```
df.drop(columns=['issue_d_month', 'issue_d_year', 'earliest_cr_line_mon', 'earliest_cr_line_year'],
                 axis=1, inplace=True)
```

```
df.shape
```

```
(360742, 52)
```

Train Test Split

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

```
# Split the data into training and test data

x_train, x_test, y_train, y_test =train_test_split(X,y,test_size=0.30,stratify=y,random_state=42)

print(f'Shape of x_train: {x_train.shape}')
print(f'Shape of x_test: {x_test.shape}')
print(f'Shape of y_train: {y_train.shape}')
print(f'Shape of y_test: {y_test.shape}')
```

```
Shape of x_train: (252519, 51)
Shape of x_test: (108223, 51)
Shape of y_train: (252519,)
Shape of y_test: (108223,)
```

MinMaxScaler

MinMaxScaler - For each value in a feature, MinMaxScaler subtracts the minimum value in the feature and then divides by the range. The range is the difference between the original maximum and original minimum.

MinMaxScaler preserves the shape of the original distribution. It doesn't meaningfully change the information embedded in the original data.

```
scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

LogisticRegression

```
logreg=LogisticRegression(max_iter=1000)
logreg.fit(x_train,y_train)
```

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

```
y_pred=logreg.predict(x_test)
```

```
y_pred
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

```
logreg.score(x_test, y_test)
```

```
0.8916496493351691
```

```
print('Accuracy of Logistic Regression Classifier on test set: {:.3f}'.format(logreg.score(x_test, y_test)))
```

```
Accuracy of Logistic Regression Classifier on test set: 0.892
```

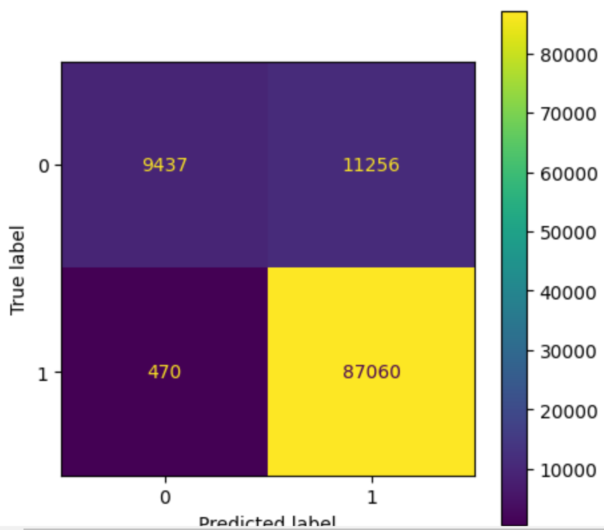
confusion_matrix

```
confusion_matrix=confusion_matrix(y_test,y_pred)
print(confusion_matrix)
```

```
[[ 9437 11256]
 [  470 87060]]
```

```
from matplotlib import pyplot as plt

# ax used here to control the size of confusion matrix
fig, ax = plt.subplots(figsize=(5,5))
ConfusionMatrixDisplay(confusion_matrix).plot(ax = ax)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7a282dfcb7c0>
```



There is significant value for false negative and false positive. Which will affect our prediction due to type-1 or type-2 error.

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

```
              precision    recall  f1-score   support

           0       0.95      0.46      0.62     20693
           1       0.89      0.99      0.94     87530

    accuracy                           0.89    108223
   macro avg       0.92      0.73      0.78    108223
weighted avg       0.90      0.89      0.88    108223
```

Precision and Recall score is good,means our model is classify correctly.

As the person is not eligible,but predicted as eligible(false positive).

As the person is eligible,but predicted as not eligible(false negative).

both the error should be avoid ,in this load tap case,so we take **F1 score** as evalution metrix in this case

ROC Curve - An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

True Positive Rate False Positive Rate True Positive Rate (TPR) is a synonym for recall and is therefore defined as follows:

TPR=(TP)/(TP+FN)

False Positive Rate (FPR) is defined as follows:

FPR=(FP)/(FP+TN)

An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows a typical ROC curve.

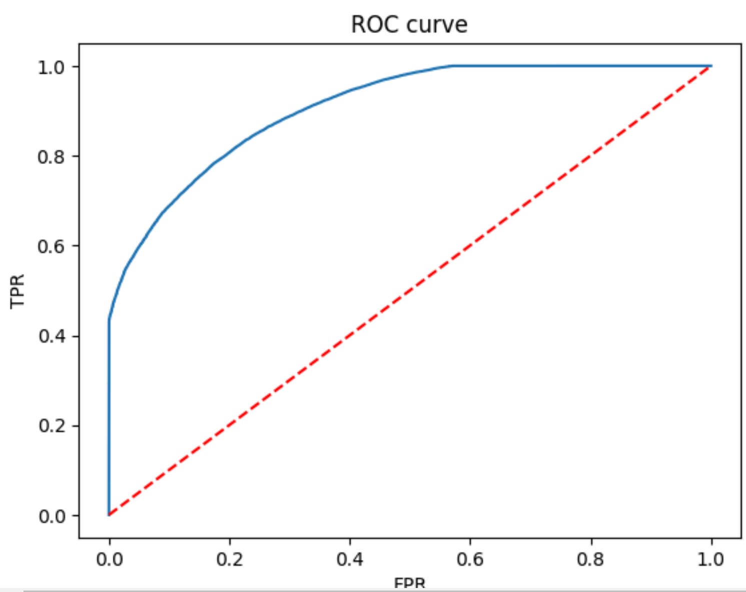
AUC (Area under the ROC Curve) - AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1).

AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example. For example, given the following examples, which are arranged from left to right in ascending order of logistic regression predictions.

```
prob = logreg.predict_proba(x_test)
prob
```

```
array([[6.90745455e-05, 9.99930925e-01],
       [3.35906224e-05, 9.99966409e-01],
       [6.38518274e-05, 9.99936148e-01],
       ...,
       [5.67979772e-04, 9.99432020e-01],
       [1.56009740e-01, 8.43990260e-01],
       [2.06550749e-04, 9.99793449e-01]])
```

```
probabilites = prob[:,1]

fpr, tpr, thr = roc_curve(y_test,probabilites)

plt.plot(fpr,tpr)

#random model
plt.plot(fpr,fpr,'--',color='red' )
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```

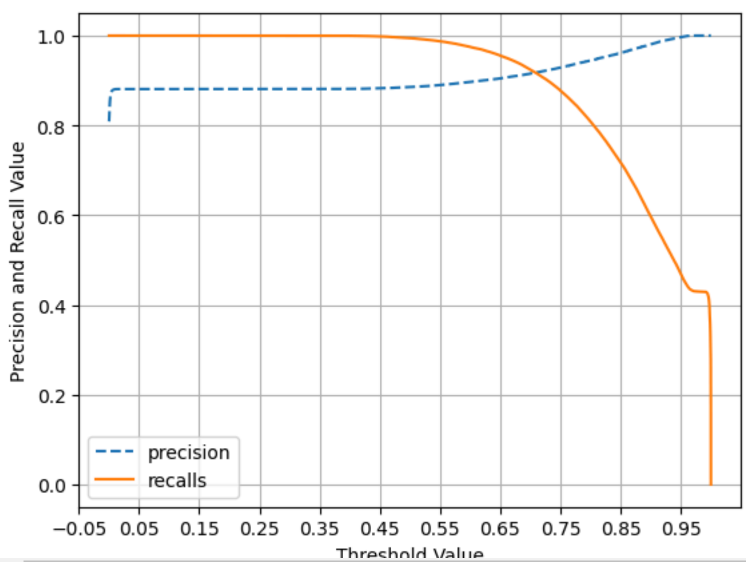


```
roc_auc_score(y_test,probabilites)
```

```
0.9066866609068771
```

We can see the ROC is aspiring towards 1,hence it is Generalized model.

```
def precission_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()

precission_recall_curve_plot(y_test,logreg.predict_proba(x_test)[:,1])
```



Precision score is highest at 0.95 threshold. High precision value indicates that model is positevly predicating the charged off loan status which helps business to take more stable decision.

Recall score is higher on smaller threshold but after 0.70 the recall value is constant. Model is correctly classifying the actual predicated values as instances.

Assumption of Log. Reg. (Multicollinearity Check)

1. Multicollinearity check by VIF score As multicollinearity makes it difficult to find out which variable is contributing towards the prediction of the response variable, it leads one to conclude incorrectly, the effects of a variable on the target variable. Though it does not affect the precision of the model predictions, it is essential to properly detect and deal with the multicollinearity present in the model, as random removal of any of these correlated variables from the model causes the coefficient values to swing wildly and even change signs.

Multicollinearity can be detected using the following methods.

Variance Inflation Factor (VIF)

VIF explains the relationship of one independent variable with all the other independent variables.

The common heuristic followed for the VIF values is if VIF > 10 then the value is high and it should be dropped.

And if the VIF=5 then it may be valid but should be inspected first.

If VIF < 5, then it is considered a good VIF value

The formula for VIF is as follows:

VIF(j) = 1 / (1 - R(j)^2)

Where:

j represents the jth predictor variable. R(j)^2 is the coefficient of determination (R-squared) obtained from regressing the jth predictor variable on all the other predictor variables.

```
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(X.shape[1])]
    vif['VIF']=round(vif['VIF'],2)
    vif=vif.sort_values(by='VIF',ascending=False)
    return vif
```

```
calc_vif(X)[:5]
```

	Feature	VIF
44	application_type_INDIVIDUAL	1652.56
46	home_ownership_MORTGAGE	912.23
50	home_ownership_RENT	741.59
49	home_ownership_OWN	164.37
2	int_rate	123.33

```
X.drop(columns=['application_type_INDIVIDUAL'],axis=1,inplace=True)
calc_vif(X)[:5]
```

	Feature	VIF
2	int_rate	123.30
45	home_ownership_MORTGAGE	81.54
49	home_ownership_RENT	62.89
15	purpose_debt_consolidation	51.79
1	term	27.30

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

	Feature	VIF
44	home_ownership_MORTGAGE	68.02
48	home_ownership_RENT	52.22
14	purpose_debt_consolidation	51.78
1	term	27.25
13	purpose_credit_card	18.83

```
X.drop(columns=['home_ownership_MORTGAGE'], axis=1, inplace=True)
calc_vif(X)[:5]
```

	Feature	VIF
1	term	23.63
14	purpose_debt_consolidation	22.19
5	open_acc	13.58
9	total_acc	11.27
13	purpose_credit_card	8.42

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

	Feature	VIF
13	purpose_debt_consolidation	18.57
4	open_acc	13.57
8	total_acc	11.25
7	revol_util	7.94
3	dti	7.38

```
X.drop(columns=['purpose_debt_consolidation','open_acc'], axis=1, inplace=True)
calc_vif(X)[:5]
```

	Feature	VIF
6	revol_util	7.26
7	total_acc	7.12
3	dti	6.57
0	loan_amnt	5.70
4	pub_rec	4.78

```
X=scaler.fit_transform(X)

kfold=KFold(n_splits=5)
accuracy=np.mean(cross_val_score(logreg,X,y,cv=kfold,scoring='accuracy',n_jobs=-1))
print("Cross Validation accuracy : {:.3f}".format(accuracy))
```

```
Cross Validation accuracy : 0.892
```

Handling imbalance data

```
sm=SMOTE(random_state=42)
X_train_res,y_train_res=sm.fit_resample(x_train,y_train.ravel())


print('After OverSampling, the shape of train_X: {}'.format(X_train_res.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train_res.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train_res == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train_res == 0)))
```

```
After OverSampling, the shape of train_X: (408472, 51)
After OverSampling, the shape of train_y: (408472,)

After OverSampling, counts of label '1': 204236
After OverSampling, counts of label '0': 204236
```

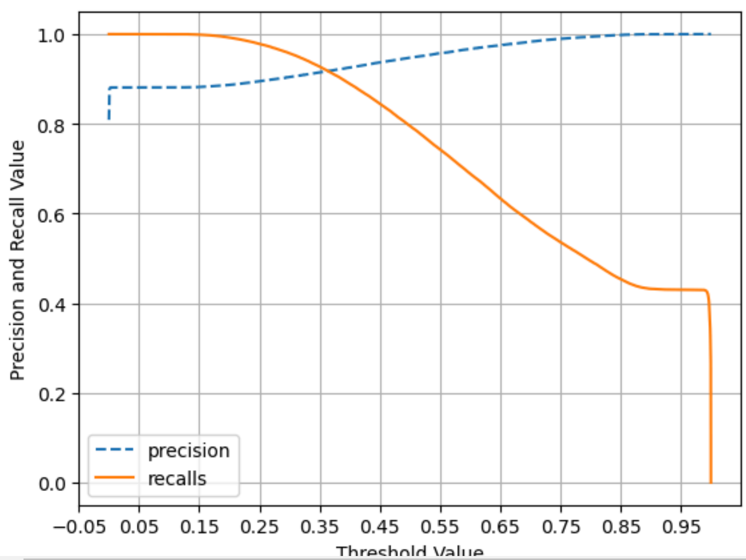
```
lr1 = LogisticRegression(max_iter=1000)
lr1.fit(X_train_res, y_train_res)
predictions = lr1.predict(x_test)

# Classification Report
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.48      0.81      0.61     20693
           1       0.95      0.80      0.86     87530

    accuracy                           0.80    108223
   macro avg       0.72      0.80      0.74    108223
weighted avg       0.86      0.80      0.82    108223
```

After making the dataset balanced, the precision and recall score are same as imbalanced dataset. But the accuracy dropped.

There is still need for improvement.

```
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, lr1.predict_proba(x_test)[:,1])
```



After balancing the dataset, there is no significant change observed in the precion and recall score for both of the classes.

Insights

1.80% of customers have fully paid their Loan Amount.

2.We can see installment and loan amount are positivly correlated hence we can drop anyone of the column.

3.We can see most of the variable distribution is right skewed and.

4.outlier are present in 'annual_inc', 'dti', 'open_acc','pub_rec', 'revol_bal', 'revol_util', 'mort_acc','pub_rec_bankruptcies'.

5.we can see most of the borrower chose 36 months as loan term.

6.From the graph,B grade is the highest risk grade given to most of the borrower.

7.Most of the listed loan status is F(fractional).

8.We can see almost all account are indivdual accounts.

9.Moratage are the ajor home ownership type.

10.So from that we can infer that people with grade 'B' and subgrade 'B3' are more likely to fully pay the loan.

11.Indivdual bank account holders are likely to pay the loan amount fully.

12.Person who employed for more than 10 years has successfully paid of the loan.

13.Teacher and Manager are the most employement title can afford load payment.

Recommendation

1.There is significant value for false negative and false positive. Which will affect our prediction due to type-1 or type-2 error.

2.We can see the ROC is aspiring towards 1,hence it is Generalized model.

3.Precision score is highest at 0.95 threshold. High precision value indicates that model is positevly predicating the charged off loan status which helps business to take more stable decision.

4.Recall score is higher on smaller threshold but after 0.70 the recall value is constant. Model is correctly classifying the actual predicated values as instances.

5.Model achieves the 94% f1-score for the negative class (Fully Paid).

6.Model achieves the 62% f1-score for the positive class (Charged off).

7.Accuracy of Logistic Regression Classifier on test set: 0.891 which is decent.

8.The precision-recall curve allows us to see how the precision and recall trade-off as we vary the threshold. A higher threshold will result in higher precision, but lower recall, and vice versa. The ideal point on the curve is the one that best meets the needs of the specific application.

9.ROC AUC curve area of 0.90, the model is correctly classifying about 90% of the instances. This is a good performance.