

```
!gdown https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/003/549/original/logistic_regression.csv?1651045921
```

```
→ Downloading...
From: https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/003/549/original/logistic_regression.csv?1651045921
To: /content/logistic_regression.csv?1651045921
100% 100M/100M [00:01<00:00, 94.8MB/s]
```

```
import pandas as pd
import numpy as np
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt
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
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
```

```
data=pd.read_csv('logistic_regression.csv?1651045921')
data.head()
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	...	open_acc	pub_rec	revol_bal	revol_util	total_acc	initial_list_status	application_type	w	INDI'
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	117000.0	...	16.0	0.0	36369.0	41.8	25.0		w	INDI'	
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	...	17.0	0.0	20131.0	53.3	27.0		f	INDI'	
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RENT	43057.0	...	13.0	0.0	11987.0	92.2	26.0		f	INDI'	
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	RENT	54000.0	...	6.0	0.0	5472.0	21.5	13.0		f	INDI'	
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000.0	...	13.0	0.0	24584.0	69.8	43.0		f	INDI'	

5 rows × 27 columns

```
data.shape
```

(396030, 27)

```
data.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
```

```
data['loan_status'].value_counts(normalize=True)
```

```
Fully Paid      0.803871
Charged Off     0.196129
Name: loan_status, dtype: float64
```

```
# Statistical summary of the dataset
data.describe(include='all')
```

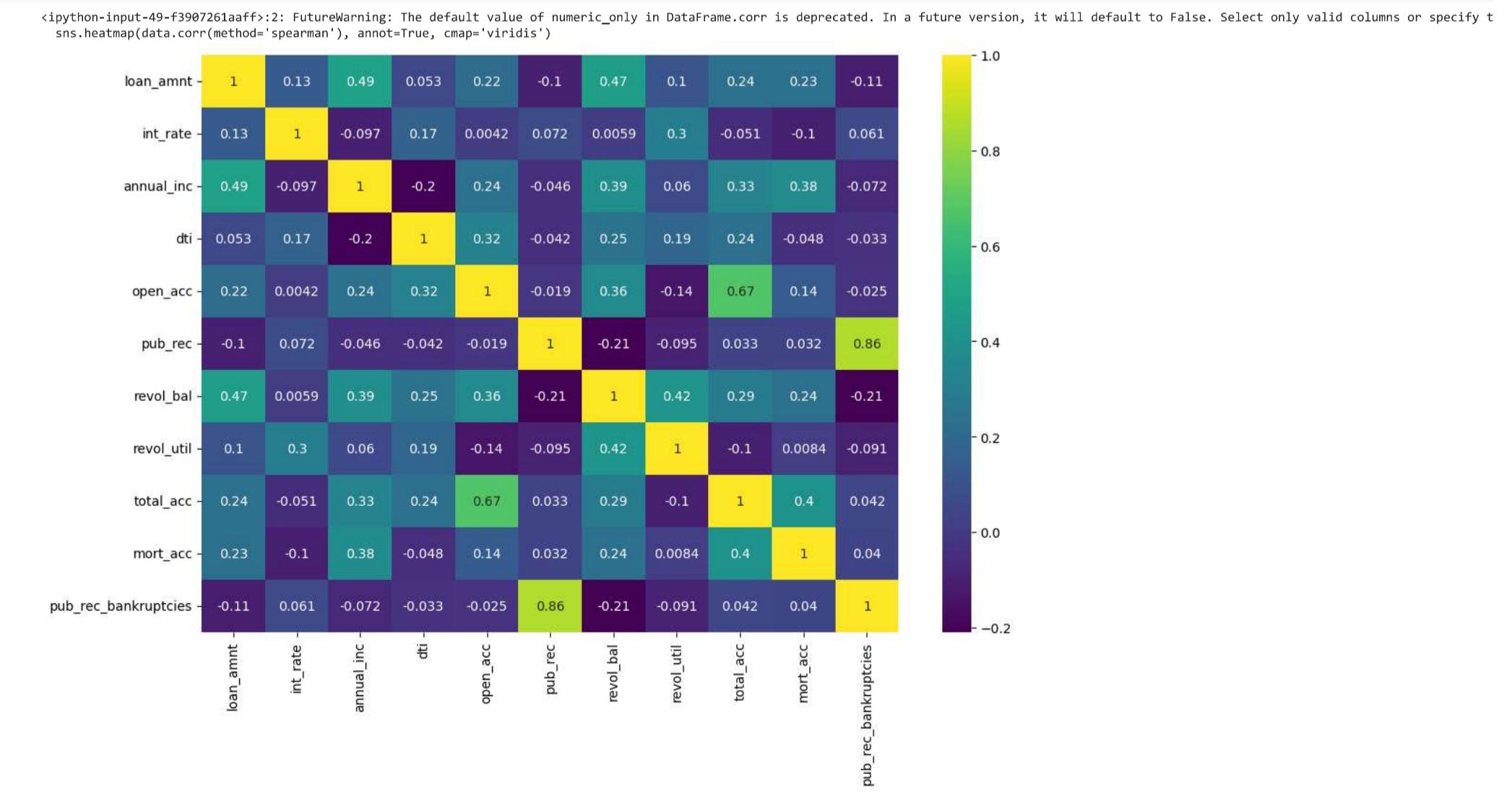


We noticed almost perfect correlation between "loan\_amnt" the "installment" feature.

installment: The monthly payment owed by the borrower if the loan originates. 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. So, we can drop either one of those columns.

```
data.drop(columns=['installment'], axis=1, inplace=True)
```

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



## ▼ Data Exploration

The no. of people who have paid fully and the no. of people who are charged off

```
data.groupby(by='loan_status')['loan_amnt'].describe()
```

	count	mean	std	min	25%	50%	75%	max
loan_status								
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

The majority of ownership as Mortgage and Rent

```
data['home_ownership'].value_counts()
```

	Count
MORTGAGE	198348
RENT	159790
OWN	37746
OTHER	112
NONE	31
ANY	3

Name: home\_ownership, dtype: int64

Combining the minority classes as 'OTHERS'

```
data.loc[(data.home_ownership == 'ANY') | (data.home_ownership == 'NONE'), 'home_ownership'] = 'OTHER'  
data.home_ownership.value_counts()
```

	Count
MORTGAGE	198348
RENT	159790
OWN	37746
OTHER	146

Name: home\_ownership, dtype: int64

```
data['home_ownership'].value_counts()
```

	Count
MORTGAGE	198348
RENT	159790
OWN	37746
OTHER	146

Name: home\_ownership, dtype: int64

```
# Checking the distribution of 'Other'  
data.loc[data['home_ownership']=='OTHER','loan_status'].value_counts()
```

	Count
Fully Paid	123
Charged Off	23

Name: loan\_status, dtype: int64

Converting string to date-time format

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

Saw some issues in title(Looks like it was filled manually and needs some fixing).

```
data['title'].value_counts()[:20]
```

	Count
Debt consolidation	152472
Credit card refinancing	51487
Home improvement	15264
Other	12930
Debt Consolidation	11608
Major purchase	4769
Consolidation	3852
debt consolidation	3547
Business	2949
Debt Consolidation Loan	2864
Medical expenses	2742
Car financing	2139
Credit Card Consolidation	1775
Vacation	1717
Moving and relocation	1689
consolidation	1595
Personal Loan	1591
Consolidation Loan	1299
Home Improvement	1268
Home buying	1183

Name: title, dtype: int64

```
data['title']=data.title.str.lower()
```

```
data['title'].value_counts()[:20]
```

	Count
debt consolidation	168108
credit card refinancing	51781
home improvement	17117
other	12993
consolidation	5583
major purchase	4998
debt consolidation loan	3513
business	3017
medical expenses	2820
credit card consolidation	2638
personal loan	2460
car financing	2160
credit card payoff	1904
consolidation loan	1887
vacation	1866
credit card refinance	1832
moving and relocation	1693
consolidate	1528
personal	1465
home buying	1196

Name: title, dtype: int64

## ▼ Visualization

The grade of majority of people those who have fully paid the loan is 'B' and have subgrade 'B3'.

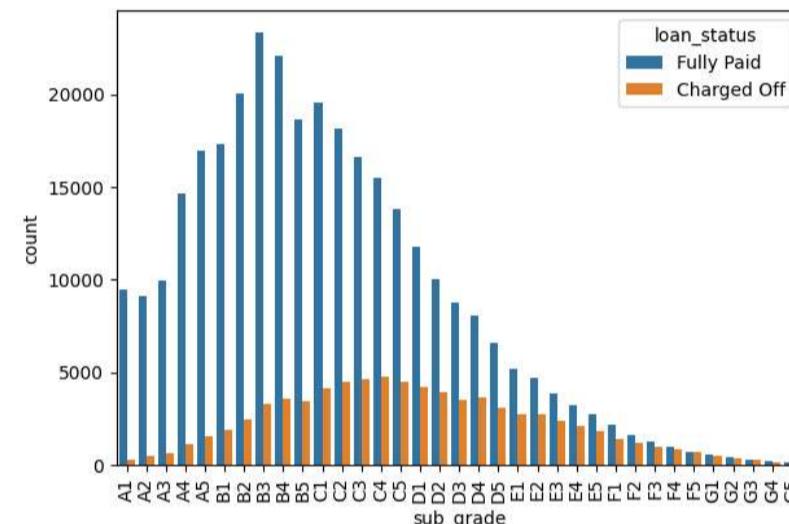
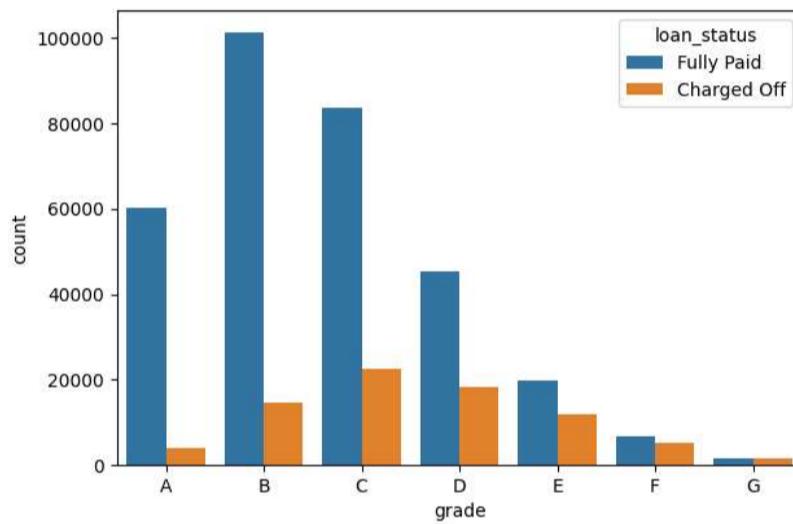
So from that we can infer that people with grade 'B' and subgrade 'B3' are more likely to fully pay the loan.

```
plt.figure(figsize=(15, 10))

plt.subplot(2, 2, 1)
grade = sorted(data.grade.unique().tolist())
sns.countplot(x='grade', data=data, hue='loan_status', order=grade)

plt.subplot(2, 2, 2)
sub_grade = sorted(data.sub_grade.unique().tolist())
g = sns.countplot(x='sub_grade', data=data, hue='loan_status', order=sub_grade)
g.set_xticklabels(g.get_xticklabels(), rotation=90)

[Text(0, 0, 'A1'),
 Text(1, 0, 'A2'),
 Text(2, 0, 'A3'),
 Text(3, 0, 'A4'),
 Text(4, 0, 'A5'),
 Text(5, 0, 'B1'),
 Text(6, 0, 'B2'),
 Text(7, 0, 'B3'),
 Text(8, 0, 'B4'),
 Text(9, 0, 'B5'),
 Text(10, 0, 'C1'),
 Text(11, 0, 'C2'),
 Text(12, 0, 'C3'),
 Text(13, 0, 'C4'),
 Text(14, 0, 'C5'),
 Text(15, 0, 'D1'),
 Text(16, 0, 'D2'),
 Text(17, 0, 'D3'),
 Text(18, 0, 'D4'),
 Text(19, 0, 'D5'),
 Text(20, 0, 'E1'),
 Text(21, 0, 'E2'),
 Text(22, 0, 'E3'),
 Text(23, 0, 'E4'),
 Text(24, 0, 'E5'),
 Text(25, 0, 'F1'),
 Text(26, 0, 'F2'),
 Text(27, 0, 'F3'),
 Text(28, 0, 'F4'),
 Text(29, 0, 'F5'),
 Text(30, 0, 'G1'),
 Text(31, 0, 'G2'),
 Text(32, 0, 'G3'),
 Text(33, 0, 'G4'),
 Text(34, 0, 'G5')]
```



```
plt.figure(figsize=(15,20))

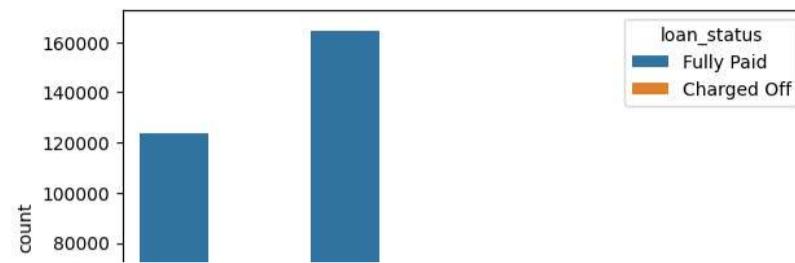
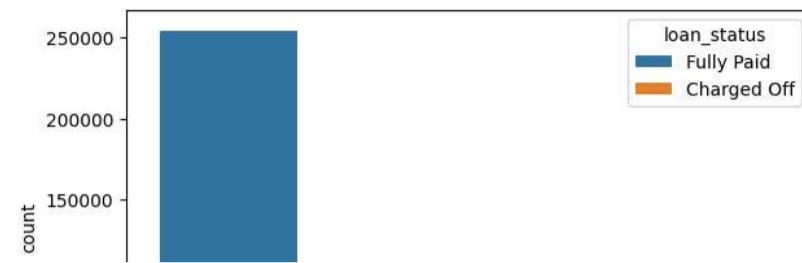
plt.subplot(4,2,1)
sns.countplot(x='term', data=data, hue='loan_status')

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

plt.subplot(4,2,3)
sns.countplot(x='verification_status', data=data, hue='loan_status')

plt.subplot(4,2,4)
g=sns.countplot(x='purpose', data=data, hue='loan_status')
g.set_xticklabels(g.get_xticklabels(), rotation=90)
```

```
[Text(0, 0, 'vacation'),
Text(1, 0, 'debt_consolidation'),
Text(2, 0, 'credit_card'),
Text(3, 0, 'home_improvement'),
Text(4, 0, 'small_business'),
Text(5, 0, 'major_purchase'),
Text(6, 0, 'other'),
Text(7, 0, 'medical'),
Text(8, 0, 'wedding'),
Text(9, 0, 'car'),
Text(10, 0, 'moving'),
Text(11, 0, 'house'),
Text(12, 0, 'educational'),
Text(13, 0, 'renewable_energy')]
```

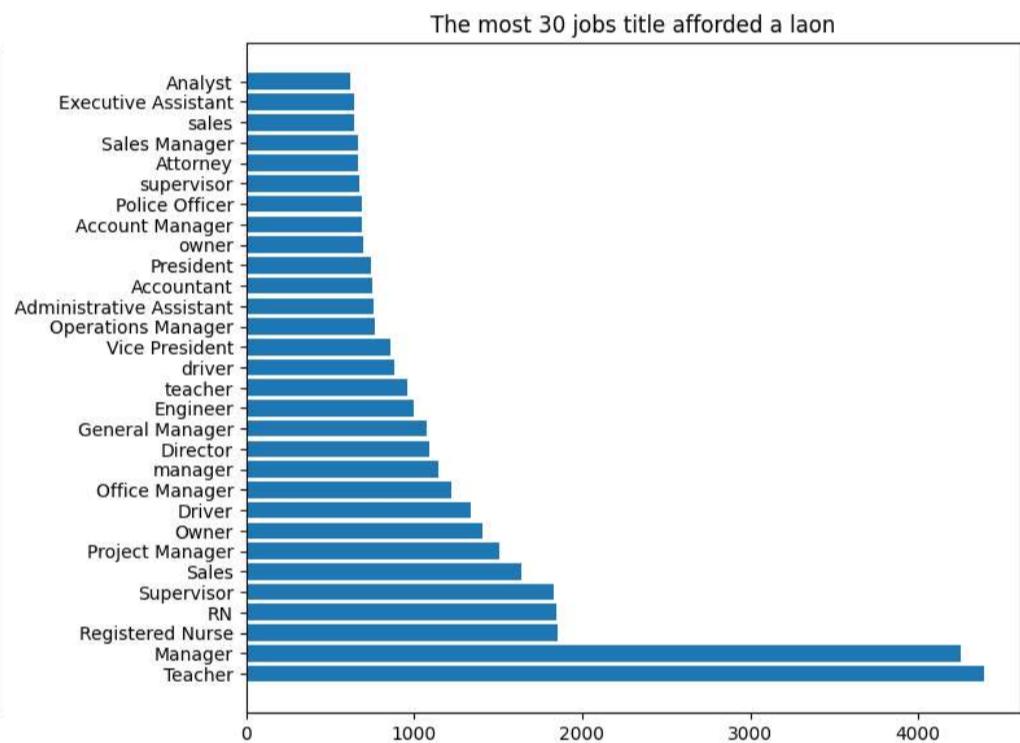
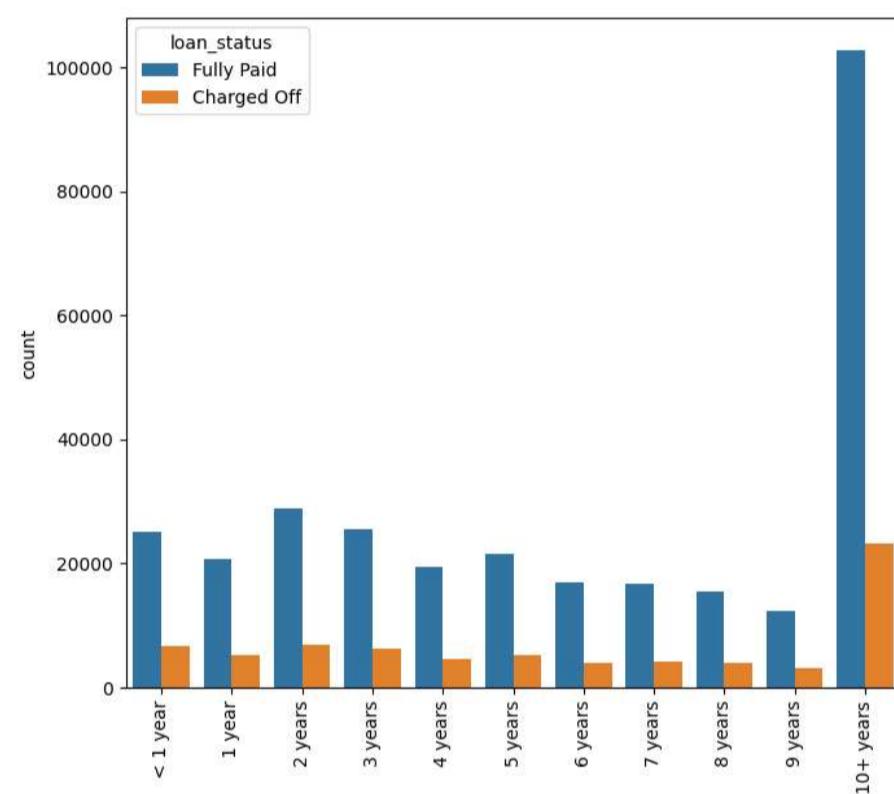


Manager and Teacher are the most afforded loan on titles

```
plt.figure(figsize=(15,12))

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

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



## Feature Engineering

```
#below are high outlier columns. We dont want to delete these records since someone whos taken a loan for the first time may have low bankruptcy record
#flagging anything more than 0 as 1
```

```
def pub_rec(number):
    if number == 0.0:
        return 0
    else:
        return 1

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

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

```
data['pub_rec']=data.pub_rec.apply(pub_rec)
data['mort_acc']=data.mort_acc.apply(mort_acc)
data['pub_rec_bankruptcies']=data.pub_rec_bankruptcies.apply(pub_rec_bankruptcies)
```

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

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

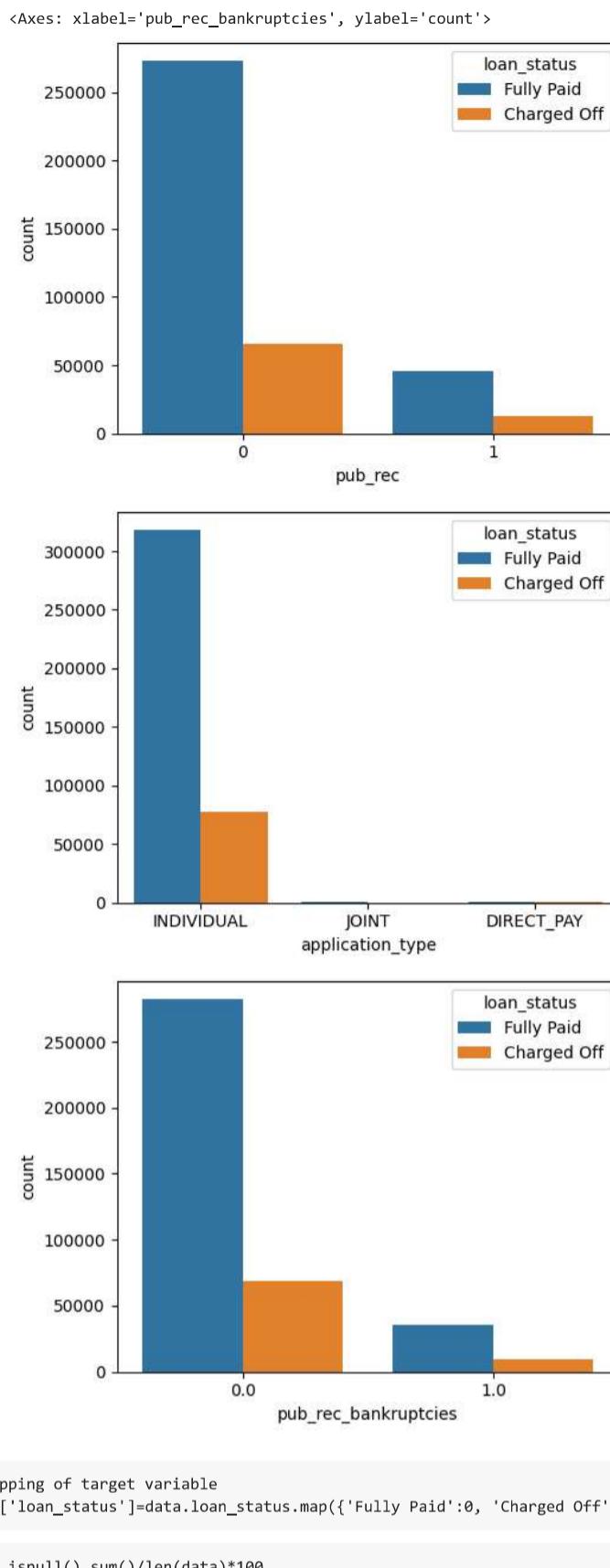
plt.subplot(6,2,2)
sns.countplot(x='initial_list_status',data=data,hue='loan_status')

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

plt.subplot(6,2,4)
```

```
sns.countplot(x='mort_acc',data=data,hue='loan_status')
```

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



```
# Mapping of target variable  
data['loan_status']=data.loan_status.map({'Fully Paid':0, 'Charged Off':1})
```

```
data.isnull().sum()/len(data)*100
```

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

#### ▼ Mean Target Encoding

```
data.groupby(by='total_acc').mean()
```

```

<ipython-input-67-05a6a8313bf2>:1: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeri
  data.groupby(by='total_acc').mean()

    loan_amnt  int_rate  annual_inc  loan_status      dti  open_acc  pub_rec    revol_bal  revol_util  mort_acc  pub_rec_bankruptcies
total_acc
  2.0      6672.222222  15.801111  64277.777778  0.222222  2.279444  1.611111  0.000000  2860.166667  53.527778  0.000000  0.000000
  3.0      6042.966361  15.615566  41270.753884  0.220183  6.502813  2.611621  0.033639  3382.807339  49.991022  0.046243  0.015480
  4.0      7587.399031  15.069491  42426.565969  0.214055  8.411963  3.324717  0.033118  4874.231826  58.477400  0.062140  0.019672
  5.0      7845.734714  14.917564  44394.098003  0.203156  10.118328  3.921598  0.055720  5475.253452  56.890311  0.090789  0.039187

total_acc_avg=data.groupby(by='total_acc').mean().mort_acc
# saving mean of mort_acc according to total_acc_avg

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

data['mort_acc']=data.apply(lambda x: fill_mort_acc(x['total_acc'],x['mort_acc']),axis=1)

data.isnull().sum()/len(data)*100

```

	loan_amnt	int_rate	annual_inc	loan_status	dti	open_acc	pub_rec	revol_bal	revol_util	mort_acc	pub_rec_bankruptcies
2.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
3.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
4.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

```

loan_amnt      0.000000
term          0.000000
int_rate      0.000000
grade         0.000000
sub_grade     0.000000
emp_title     5.789288
emp_length    4.621115
home_ownership 0.000000
annual_inc     0.000000
verification_status 0.000000
issue_d       0.000000
loan_status    0.000000
purpose        0.000000
title          0.443148
dti           0.000000
earliest_cr_line 0.000000
open_acc       0.000000
pub_rec        0.000000
revol_bal     0.000000
revol_util    0.069692
total_acc      0.000000
initial_list_status 0.000000
application_type 0.000000
mort_acc       0.000000
pub_rec_bankruptcies 0.135091
address        0.000000
dtype: float64

# Current no. of rows
data.shape

```

```

(396030, 26)

```

```

# Dropping rows with null values
data.dropna(inplace=True)
# Remaining no. of rows
data.shape

```

```

(370622, 26)

```

## Outlier Detection & Treatment

```

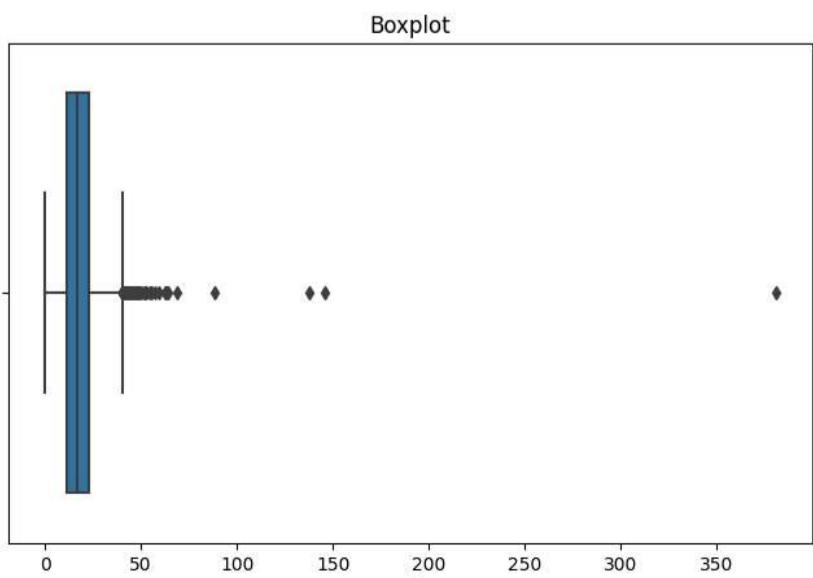
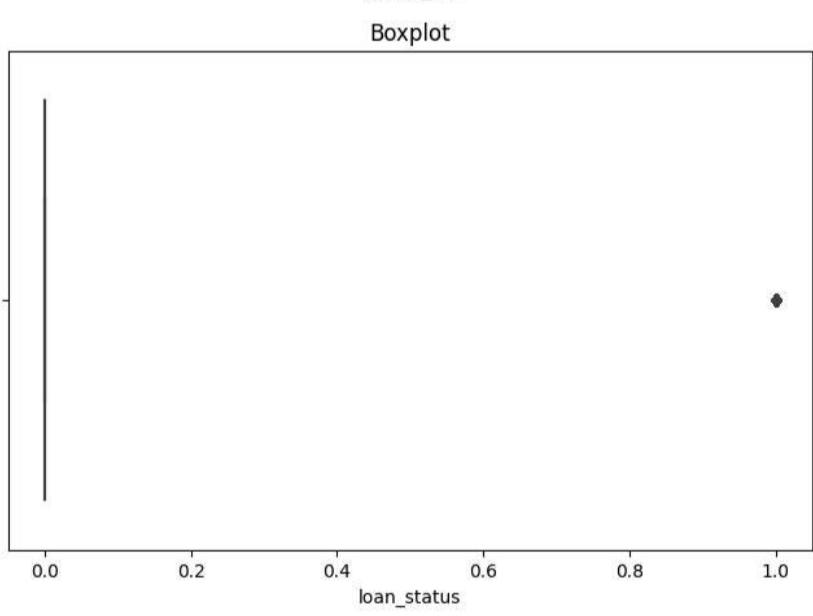
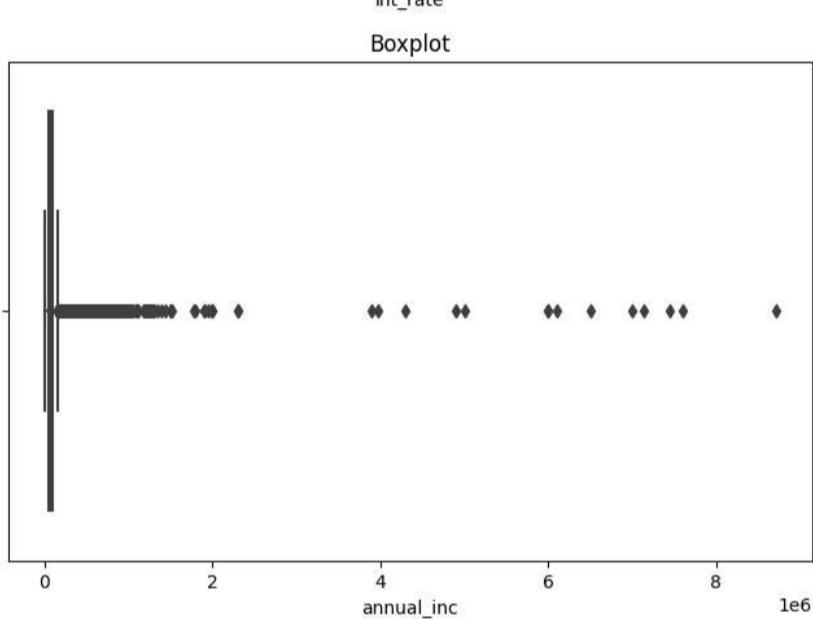
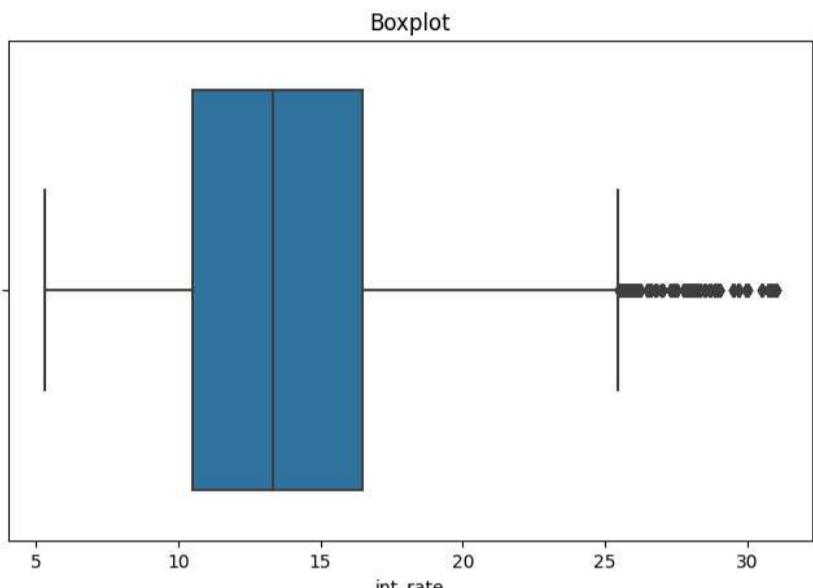
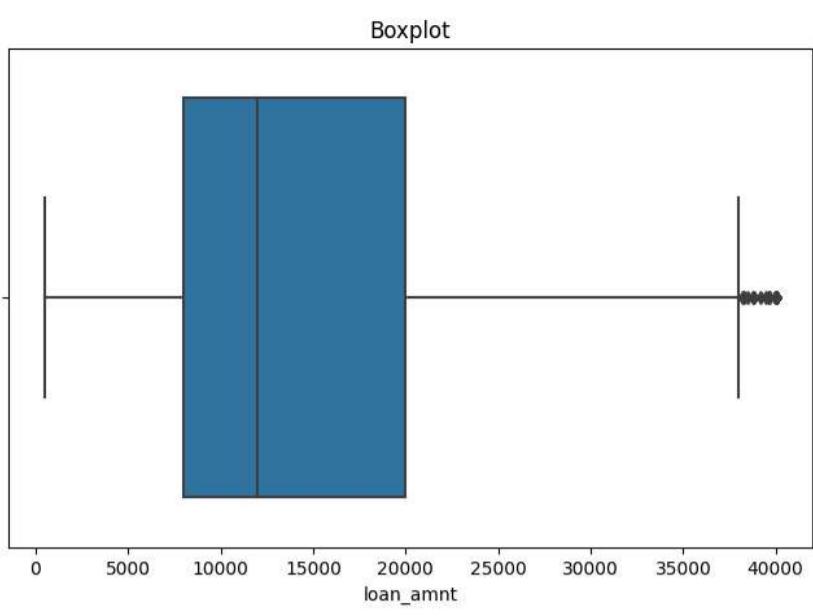
numerical_data=data.select_dtypes(include='number')
num_cols=numerical_data.columns
len(num_cols)

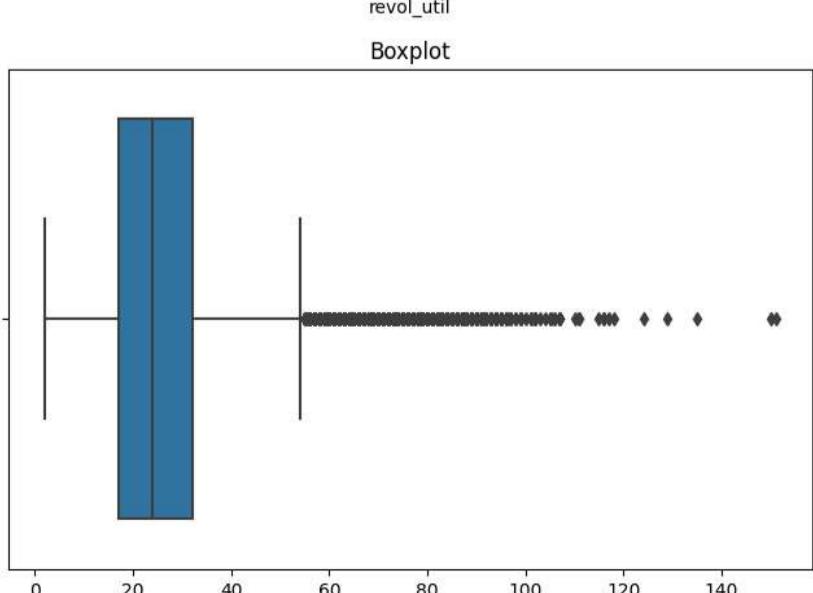
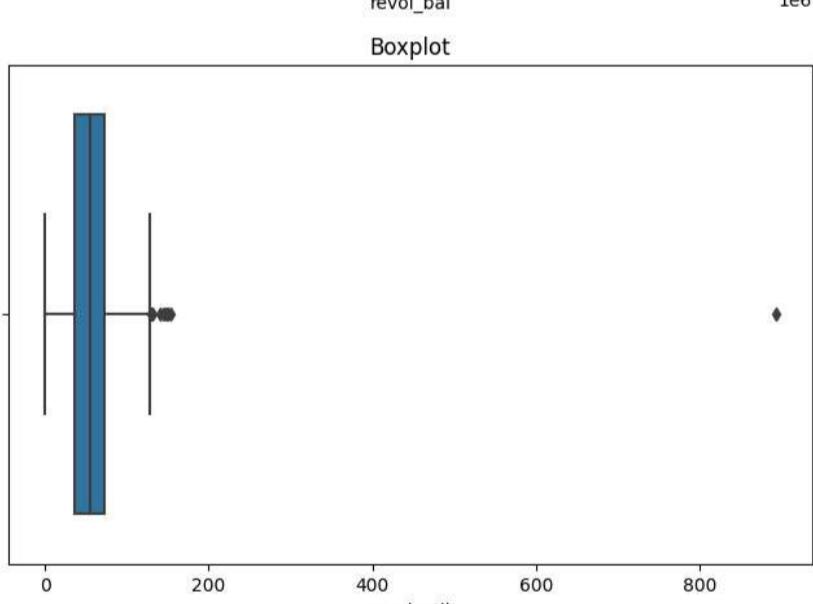
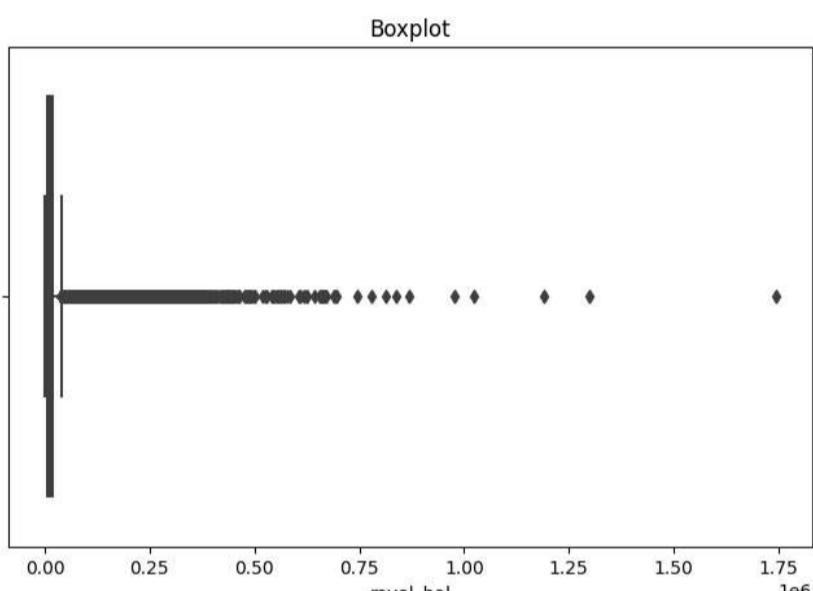
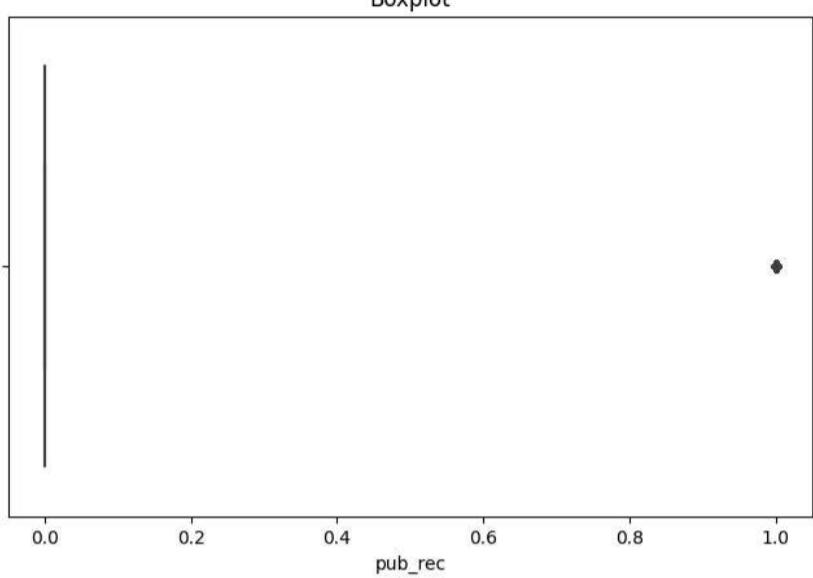
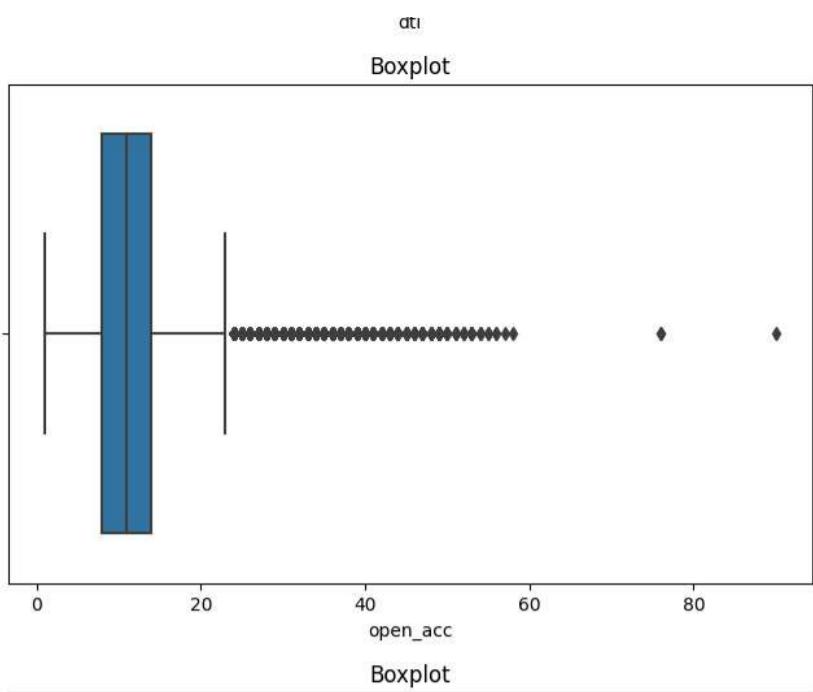
12

def box_plot(col):
    plt.figure(figsize=(8,5))
    sns.boxplot(x=data[col])
    plt.title('Boxplot')
    plt.show()

for col in num_cols:
    box_plot(col)

```





total\_acc

Boxplot



```
for col in num_cols:  
    mean=data[col].mean()  
    std=data[col].std()  
  
    upper_limit=mean+3*std  
    lower_limit=mean-3*std  
  
    data=data[(data[col]<upper_limit) & (data[col]>lower_limit)]  
  
data.shape
```

(354519, 26)

## ▼ Data Preprocesing

```
# Term  
data.term.unique()  
  
array([' 36 months', ' 60 months'], dtype=object)  
  
term_values={' 36 months': 36, ' 60 months':60}  
data['term'] = data.term.map(term_values)  
  
# Initial List Status  
data['initial_list_status'].unique()  
  
array(['w', 'f'], dtype=object)  
  
list_status = {'w': 0, 'f': 1}  
data['initial_list_status'] = data.initial_list_status.map(list_status)
```

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

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

```
70466    14.382022  
30723    14.277373  
22690    14.268347  
48052    14.127028  
00813    11.610097  
29597    11.537322  
05113    11.516731  
93700    2.774746  
11650    2.772771  
86630    2.733563  
Name: zip_code, dtype: float64
```

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

## ▼ One-hot Encoding

```
dummies=['purpose', 'zip_code', 'grade', 'verification_status', 'application_type', 'home_ownership']  
data=pd.get_dummies(data,columns=dummies,drop_first=True)
```

```
pd.set_option('display.max_columns',None)  
pd.set_option('display.max_rows',None)  
  
data.head()
```

	loan_amnt	term	int_rate	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_co
0	10000.0	36	11.44	117000.0	0	26.24	16.0	0	36369.0	41.8	25.0	0	0.0	0.0	0	
1	8000.0	36	11.99	65000.0	0	22.05	17.0	0	20131.0	53.3	27.0	1	1.0	0.0	0	
2	15600.0	36	10.49	43057.0	0	12.79	13.0	0	11987.0	92.2	26.0	1	0.0	0.0	1	
3	7200.0	36	6.49	54000.0	0	2.60	6.0	0	5472.0	21.5	13.0	1	0.0	0.0	1	
4	24375.0	60	17.27	55000.0	1	33.95	13.0	0	24584.0	69.8	43.0	1	1.0	0.0	1	

```
data.shape
```

(354519, 49)

## ▼ Data Preparation for Modelling

```
X=data.drop('loan_status',axis=1)  
y=data['loan_status']  
X_train, X_test, y_train, y_test =train_test_split(X,y,test_size=0.30,stratify=y,random_state=42)
```

```
print(X_train.shape)  
print(X_test.shape)
```

(248163, 48)  
(106356, 48)

**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)
```

## Logistic Regression

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

y_pred = logreg.predict(X_test)
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: 0.890
```

## Confusion Matrix

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

[[85366  522]
 [11132 9336]]
```

## Classification Report

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

precision    recall  f1-score   support

          0       0.88      0.99      0.94     85888
          1       0.95      0.46      0.62     20468

   accuracy                           0.89    106356
  macro avg       0.92      0.73      0.78    106356
weighted avg       0.90      0.89      0.87    106356
```

### 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:

- $\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$

False Positive Rate (FPR) is defined as follows:

- $\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$

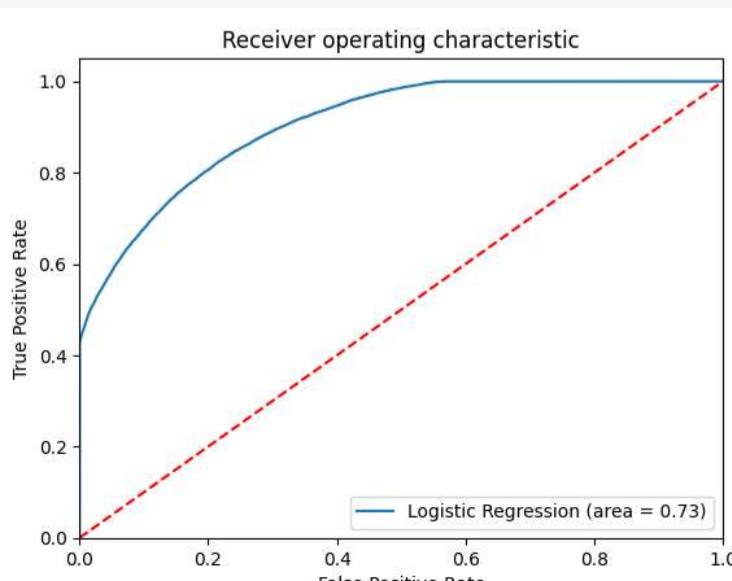
An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows a typical ROC curve.

### 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.

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



```

def precision_recall_curve_plot(y_test,pred_proba_c1):
    precisions, recalls, thresholds = precision_recall_curve(y_test,pred_proba_c1)

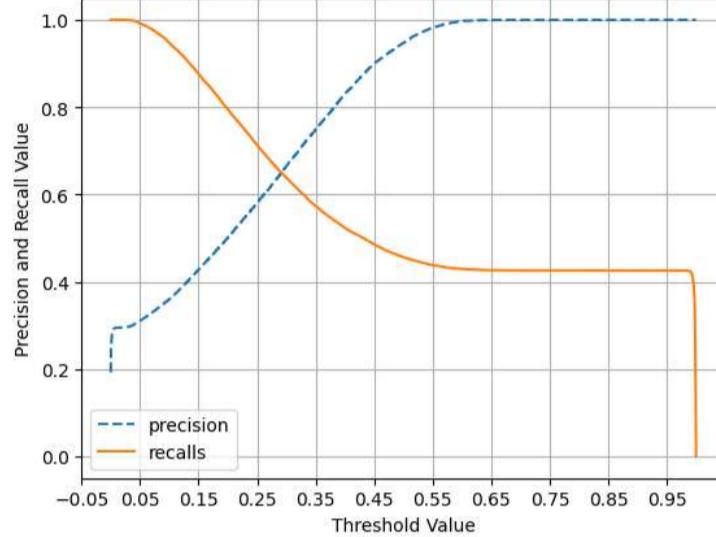
    threshold_boundary = thresholds.shape[0]
    #plot precision
    plt.plot(thresholds,precisions[0:threshold_boundary],linestyle='--',label='precision')
    #plot recall
    plt.plot(thresholds,recalls[0:threshold_boundary],label='recalls')

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

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

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

```



Multicollinearity check using Variance Inflation Factor (VIF) - Multicollinearity occurs when two or more independent variables are highly correlated with one another in a regression model. Multicollinearity can be a problem in a regression model because we would not be able to distinguish between the individual effects of the independent variables on the dependent variable.

Multicollinearity can be detected via various methods. One such method is Variance Inflation Factor aka VIF. In VIF method, we pick each independent feature and regress it against all of the other independent features. VIF score of an independent variable represents how well the variable is explained by other independent variables.

VIF = 1/(1-R<sup>2</sup>)

```
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)
```

300-1000

		Feature	VIF
43	application_type_INDIVIDUAL		156.97
2		int_rate	122.82
14	purpose_debt_consolidation		51.00
1		term	27.30
13	purpose_credit_card		18.48

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

	Feature	VIF
2	int_rate	103.43
14	purpose_debt_consolidation	27.49
1	term	24.31
5	open_acc	13.75
9	total_acc	12.69

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

		Feature	VIF
1		term	23.35
13	purpose_debt_consolidation		22.35
4		open_acc	13.64
8		total_acc	12.69
7		revol_util	9.06

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

	Feature	VIF
12	purpose_debt_consolidation	18.37
3	open_acc	13.64
7	total_acc	12.65
6	revol_util	9.04
1	revol_bal	8.26

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

Feature	VIF
3 open_acc	13.09
7 total_acc	12.64
6 revol_util	8.31
1 annual_inc	7.70
2 dti	7.58

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

Feature	VIF
6 total_acc	8.23
5 revol_util	8.00
1 annual_inc	7.60
2 dti	7.02
0 loan_amnt	6.72

```
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 : 0.891
```

## ▼ Oversampling using SMOTE

```
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: (400810, 48)
After OverSampling, the shape of train_y: (400810,)

After OverSampling, counts of label '1': 200405
After OverSampling, counts of label '0': 200405

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	0.95	0.80	0.87	85888
1	0.49	0.81	0.61	20468
accuracy			0.80	106356
macro avg	0.72	0.80	0.74	106356
weighted avg	0.86	0.80	0.82	106356

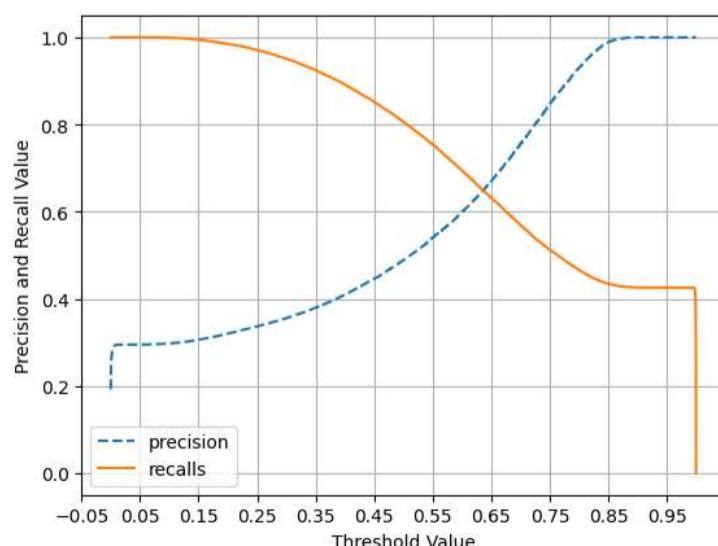
```
def precision_recall_curve_plot(y_test, pred_proba_c1):
    precisions, recalls, thresholds = precision_recall_curve(y_test, pred_proba_c1)

    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])
```



## Actional Insights and Recommendations

- 80% of the customers have paid the loan fully.
- 20% of the customers are the defaulters.

3. The organization can use the trained model to make prediction for whether a person will likely to pay the loan amount or he will be a defaulter.
4. Model achieves the 94% f1-score for the negative class (Fully Paid).
5. Model achieves the 62% f1-score for the positive class (Charged off).
6. Cross Validation accuracy and testing accuracy is almost same which infers model is performing the decent job. We can trust this model for unseen data
7. By collecting more data, using a more complex model, or tuning the hyperparameters, it is possible to improve the model's performance.
8. ROC AUC curve area of 0.73, the model is correctly classifying about 73% of the instances. This is a good performance, but there is still room for improvement.
9. 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.
10. After balancing the dataset, there is significant change observed in the precision and recall score for both of the classes.
11. Accuracy of Logistic Regression Classifier on test set: 0.891 which is decent and not by chance.