Context:

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

The data science team at LoanTap is building an underwriting layer to determine the creditworthiness of MSMEs as well as individuals.

LoanTap deploys formal credit to salaried individuals and businesses 4 main financial instrume1. nts:

Person 2. al Loan EMI F 3. ree Loan Personal 4. Overdraft Advance . Salary Loan This case study will focus on the underwriting process beh nd Personal### #### Loan only

Prolm Statement:

Given a set of attributes for an Individual, determine if a credit line should be extended to them. If so, what should the repayment terms be in business recommendations?

Overview:

This notebook focused on implementing Logistic Regression for loan application data from a company called Loan Tap. The goal was to analyze the data and build models that predict loan default chances, with an emphasis on addressing data imbalances and improving model performance.

Key Concepts and Techniques

Data Preprocessing Handling Missing Values: Initially, rows with missing data were dropped, specifically for key attributes like employment title and length .

Feature Engineering:

Used one-hot encoding for categorical features such as loan purpose, verification status, and home owne.rce】. Treated and encoded various text fields into lowercase for consistency and merged similar categories . Scaling:

MinM xScaler: Applied to preserve the distribution shape of features while normali.

Dealing with Imbalanced Data SMOTE (Synthetic Minority Over-sampling Technique): Introduced as a method to balance class distributions by synthetically generating samples. However, using class weights was deemed more efficient due to growing data size concerns when using SMOT.

Model Building and Metrics Logistic Regression: The primary algorithm used, with detailed coverage on how to leverage model parameters such as class weights to manage data imbalance .

Evaluation Metrics:

Precision and Recall were emphasized to fine-tune and validate model performance. The choice between these metrics should depend on the use case—precision for Loan Tap, recall for others like H DFC . ROC Curve & AUC: Used for assessing the trade-off between true positive and false positive.

Optimization Techniques Multicollinearity Check (VIF Analysis): Variance Inflation Factor (VIF) was used to detect multicollinearity issues in predictors, with steps to remove highly correlated features to improve model stability and performanc..

Threshold Adjustmen t: Manipulating probability cutoffs to balance sensitivity (recall) and specificity (precision), thus optimizing the model to match business objectives .urce】.. zing them

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 individualss: Address of the individua

EDA

loan_status : Current status of the loan - Target Variable Import the dataset and do usual exploratory data analysis steps like checking the structure & characteristics of the dataset Check how much target variable (Loan_Status) depends on different predictor variables (Use count plots, box plots, heat maps etc)

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
```

In [388…
```
data=pd.read_csv(r"C:\Users\akaurtiwana\Desktop\Power BI\Data Set-20210303T165138Z-001\Data Set\logistic_regression.csv")
data.head()
```

Out[388...

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

5 rows × 27 columns

In [389...

```
data.shape
```

Out[389...

```
(396030, 27)
```

In [390...

```
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                 394274 non-null  object 
 15  dti                   396030 non-null  float64
 16  earliest_cr_line      396030 non-null  object 
 17  open_acc              396030 non-null  float64
 18  pub_rec               396030 non-null  float64
 19  revol_bal             396030 non-null  float64
 20  revol_util            395754 non-null  float64
 21  total_acc             396030 non-null  float64
 22  initial_list_status   396030 non-null  object 
 23  application_type      396030 non-null  object 
 24  mort_acc              358235 non-null  float64
 25  pub_rec_bankruptcies  395495 non-null  float64
 26  address               396030 non-null  object 
dtypes: float64(12), object(15)
memory usage: 81.6+ MB
```

In [391…

```
data.describe()
```

Out[391…

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_bal	revol_
count	396030.000000	396030.000000	396030.000000	3.960300e+05	396030.000000	396030.000000	396030.000000	3.960300e+05	395754.00(
mean	14113.888089	13.639400	431.849698	7.420318e+04	17.379514	11.311153	0.178191	1.584454e+04	53.79
std	8357.441341	4.472157	250.727790	6.163762e+04	18.019092	5.137649	0.530671	2.059184e+04	24.45
min	500.000000	5.320000	16.080000	0.000000e+00	0.000000	0.000000	0.000000	0.000000e+00	0.00(
25%	8000.000000	10.490000	250.330000	4.500000e+04	11.280000	8.000000	0.000000	6.025000e+03	35.80(
50%	12000.000000	13.330000	375.430000	6.400000e+04	16.910000	10.000000	0.000000	1.118100e+04	54.80(
75%	20000.000000	16.490000	567.300000	9.000000e+04	22.980000	14.000000	0.000000	1.962000e+04	72.90(
max	40000.000000	30.990000	1533.810000	8.706582e+06	9999.000000	90.000000	86.000000	1.743266e+06	892.30(

In [392…

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

Out[392…

```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
installment             0.000000
grade                   0.000000
sub_grade               0.000000
emp_title               5.789208
emp_length              4.621115
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.443401
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
```

Automation using Upper, lower, stripping:

Data Cleaning

As there are same title, emp_title, purpose which means the same.

In [393…

```
data['title'].nunique()
```

Out[393…

```
48816
```

In [394…
```
data['title']=data['title'].str.strip()
data['title']=data['title'].str.lower()
data['title'].nunique()
```
Out[394… 40139

In [395…
```
data['emp_title'].nunique()
```
Out[395… 173105

In [396…
```
data['emp_title']=data['emp_title'].str.strip()
data['emp_title']=data['emp_title'].str.lower()
data['emp_title'].nunique()
```
Out[396… 149260

In [397…
```
data['purpose'].nunique()
```
Out[397… 14

In [398…
```
data['purpose']=data['purpose'].str.strip()
data['purpose']=data['purpose'].str.lower()
data['purpose'].nunique()
```
Out[398… 14

Median/Mod Imputation

Handling Null Values

Imputation: Mode for categorical variables and meadian for numerical variable. mor_acc has 9.5 % values as null which is quite significant. To fill mort_acc use toatl_acc

In [399…
```
total_acc_avg=data.groupby(by='total_acc')['mort_acc'].median()
total_acc_avg
```

Out[399…

```
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 [400…

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

In [401…

```
data.shape[0]
```

Out[401…

```
396030
```

In [402…

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

In [403…

```
data.isnull().sum()/(data.shape[0])*100
```

Out[403…

```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
installment             0.000000
grade                   0.000000
sub_grade               0.000000
emp_title               5.789208
emp_length              4.621115
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.443401
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.069692
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                0.000000
pub_rec_bankruptcies    0.135091
address                 0.000000
dtype: float64
```

In [404…

```
## Now either drop the rows with null values or do imputation, but removing rows with
## empl_title and emp_length means lossing nearly 10% data which may affect the predictions as this signifies job stability, i
## give other category to the missing values to control bias in the model prediction
## method one: use mode of emp_title from grouping the columns annual_inc and loan_amnt
## data.dropna(inplace=True)
```

In [405…

```
data['title']=data.groupby(by=['annual_inc','loan_amnt'])['title'].transform(lambda x: x.mode()[0]\
                                                                             if not x.mode().empty else 'unknown')

data['emp_title']=data.groupby(by=['annual_inc','loan_amnt'])['emp_title'].transform(lambda x: x.mode()[0]\
                                                                             if not x.mode().empty else 'unknown')
```

In [406…

```
data['emp_length']=data.groupby(by=['annual_inc','loan_amnt'])['emp_length'].transform(lambda x: x.mode()[0]\
                                                                             if not x.mode().empty else 'unknown')
```

In [407…

```
df=data.select_dtypes(include='object')
for i in df.columns:
    print(df[i].value_counts())
    print('-'*100)
```

```
term
 36 months    302005
 60 months     94025
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
grade
B    116018
C    105987
A     64187
D     63524
E     31488
F     11772
G      3054
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
sub_grade
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: count, dtype: int64
----------------------------------------------------------------------------------------------------
emp_title
manager                           57434
teacher                           41999
registered nurse                  13758
driver                             9065
unknown                            7433
                                  ...
four seasons hotel los angeles        1
stowe mountain lodge                  1
admin/acct payable                    1
equipment mechanic 545                1
data center specialist ii             1
Name: count, Length: 46459, dtype: int64
----------------------------------------------------------------------------------------------------
emp_length
10+ years    295484
2 years       17698
1 year        17381
< 1 year      12518
3 years       11872
5 years        7929
4 years        7716
unknown        6620
6 years        5633
7 years        5287
8 years        4529
9 years        3363
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
```

```
home_ownership
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
ANY              3
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
verification_status
Verified           139563
Source Verified    131385
Not Verified       125082
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
issue_d
Oct-2014    14846
Jul-2014    12609
Jan-2015    11705
Dec-2013    10618
Nov-2013    10496
            ...
Jul-2007       26
Sep-2008       25
Nov-2007       22
Sep-2007       15
Jun-2007        1
Name: count, Length: 115, dtype: int64
----------------------------------------------------------------------------------------------------
loan_status
Fully Paid     318357
Charged Off     77673
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
purpose
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: count, dtype: int64
----------------------------------------------------------------------------------------------------
title
debt consolidation         317284
credit card refinancing     24006
other                        4665
home improvement             4336
consolidation                2703
                            ...
my consolidation plan           1
consolidate & windows           1
deliverance                     1
credit crads                    1
toxic debt payoff               1
Name: count, Length: 12043, dtype: int64
----------------------------------------------------------------------------------------------------
earliest_cr_line
Oct-2000    3017
Aug-2000    2935
Oct-2001    2896
Aug-2001    2884
Nov-2000    2736
            ...
Jul-1958       1
Nov-1957       1
Jan-1953       1
Jul-1955       1
Aug-1959       1
Name: count, Length: 684, dtype: int64
----------------------------------------------------------------------------------------------------
initial_list_status
f    238066
w    157964
```

```
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
application_type
INDIVIDUAL    395319
JOINT            425
DIRECT_PAY       286
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
address
USCGC Smith\r\nFPO AE 70466                          8
USS Johnson\r\nFPO AE 48052                          8
USNS Johnson\r\nFPO AE 05113                         8
USS Smith\r\nFPO AP 70466                            8
USNS Johnson\r\nFPO AP 48052                         7
                                                    ..
455 Tricia Cove\r\nAustinbury, FL 00813              1
7776 Flores Fall\r\nFernandezshire, UT 05113         1
6577 Mia Harbors Apt. 171\r\nRobertshire, OK 22690   1
8141 Cox Greens Suite 186\r\nMadisonstad, VT 05113   1
787 Michelle Causeway\r\nBriannaton, AR 48052        1
Name: count, Length: 393700, dtype: int64
----------------------------------------------------------------------------------------------------
```

In [408…

```
## Although pub_rec_bankruptcies is 0.135091 % null values and droping the null values wont impact the data size,
# but this column is important in loan determination thus imppute this using median as this is a numerical column
# why median? mean is same as median for normal data distributon but if the data is skewed or distribution is unknown then med

data['pub_rec_bankruptcies'].fillna(data['pub_rec_bankruptcies'].median(),inplace=True)
```

In [409…

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

Out[409…

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

In [410…

```
## revol_util has just 0.069692% null records and this wont impact the predictions
data.dropna(subset=['revol_util'],inplace=True)

#Note: subset ensure only droping rows with specific column null values, here this parameter is not required
```

In [411…

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

Out[411…
```
loan_amnt               0.0
term                    0.0
int_rate                0.0
installment             0.0
grade                   0.0
sub_grade               0.0
emp_title               0.0
emp_length              0.0
home_ownership          0.0
annual_inc              0.0
verification_status     0.0
issue_d                 0.0
loan_status             0.0
purpose                 0.0
title                   0.0
dti                     0.0
earliest_cr_line        0.0
open_acc                0.0
pub_rec                 0.0
revol_bal               0.0
revol_util              0.0
total_acc               0.0
initial_list_status     0.0
application_type        0.0
mort_acc                0.0
pub_rec_bankruptcies    0.0
address                 0.0
dtype: float64
```

In [412…
```
data.shape
```

Out[412…
```
(395754, 27)
```

In [413…
```
df=data.select_dtypes(include='object')
for i in df.columns:
    print(df[i].value_counts())
    print('-'*100)
```

```
term
 36 months    301782
 60 months     93972
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
grade
B    115973
C    105907
A     64172
D     63448
E     31453
F     11754
G      3047
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
sub_grade
B3    26643
B4    25585
C1    23648
C2    22562
B2    22488
B5    22080
C3    21207
C4    20257
B1    19177
A5    18524
C5    18233
D1    15969
A4    15783
D2    13933
D3    12212
D4    11644
A3    10573
A1     9727
D5     9690
A2     9565
E1     7913
E2     7418
E3     6199
E4     5359
E5     4564
```

```
F1     3533
F2     2761
F3     2280
F4     1784
F5     1396
G1     1058
G2      752
G3      552
G4      371
G5      314
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
emp_title
manager                                 57413
teacher                                 41979
registered nurse                        13752
driver                                   9059
unknown                                  7427
                                        ...
nevada hand                                 1
morgan stanely smith barney                 1
service associate - legal department        1
doublefine inc                              1
data center specialist ii                   1
Name: count, Length: 46429, dtype: int64
----------------------------------------------------------------------------------------------------
emp_length
10+ years    295306
2 years       17679
1 year        17369
< 1 year      12500
3 years       11866
5 years        7923
4 years        7711
unknown        6614
6 years        5626
7 years        5280
8 years        4520
9 years        3360
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
```

```
home_ownership
MORTGAGE    198219
RENT        159677
OWN          37714
OTHER          110
NONE            31
ANY              3
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
verification_status
Verified           139452
Source Verified    131301
Not Verified       125001
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
issue_d
Oct-2014    14838
Jul-2014    12597
Jan-2015    11701
Dec-2013    10609
Nov-2013    10492
            ...
Jul-2007       26
Sep-2008       25
Nov-2007       22
Sep-2007       15
Jun-2007        1
Name: count, Length: 115, dtype: int64
----------------------------------------------------------------------------------------------------
loan_status
Fully Paid     318144
Charged Off     77610
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
purpose
debt_consolidation    234384
credit_card            82999
home_improvement       23997
other                  21146
major_purchase          8769
small_business          5697
```

```
car                    4687
medical                4183
moving                 2850
vacation               2447
house                  2201
wedding                1810
renewable_energy        329
educational             255
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
title
debt consolidation         317103
credit card refinancing     23994
other                        4653
home improvement             4330
consolidation                2700
                            ...
my consolidation plan           1
consolidate & windows           1
deliverance                     1
credit crads                    1
toxic debt payoff               1
Name: count, Length: 12030, dtype: int64
----------------------------------------------------------------------------------------------------
earliest_cr_line
Oct-2000    3015
Aug-2000    2934
Oct-2001    2895
Aug-2001    2883
Nov-2000    2734
            ...
Jul-1958       1
Nov-1957       1
Jan-1953       1
Jul-1955       1
Aug-1959       1
Name: count, Length: 684, dtype: int64
----------------------------------------------------------------------------------------------------
initial_list_status
f    237881
w    157873
```

```
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
application_type
INDIVIDUAL    395043
JOINT            425
DIRECT_PAY       286
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
address
USCGC Smith\r\nFPO AE 70466                                8
USS Johnson\r\nFPO AE 48052                                8
USS Smith\r\nFPO AP 70466                                  8
USNS Johnson\r\nFPO AE 05113                               8
USNS Johnson\r\nFPO AP 48052                               7
                                                          ..
80343 Ian Pine Suite 123\r\nTrevormouth, DE 70466          1
43570 Maxwell Field Apt. 502\r\nEast John, NH 22690        1
9983 Turner Cove\r\nSouth Gregmouth, WV 70466              1
1312 Cody Shoal\r\nRalphfurt, CO 29597                     1
787 Michelle Causeway\r\nBriannaton, AR 48052              1
Name: count, Length: 393426, dtype: int64
----------------------------------------------------------------------------------------------------
```

To Determin Date time analysis converting date column into proper datetime format

In [414…

```
## Changing columns' data type and extraction month and year
data['issue_date']=pd.to_datetime(data['issue_d'],format='%b-%Y')
data['earliest_cr_line_date']=pd.to_datetime(data['earliest_cr_line'],format='%b-%Y')
data.loc[:5,['issue_date','earliest_cr_line_date']]
```

Out[414…

	issue_date	earliest_cr_line_date
0	2015-01-01	1990-06-01
1	2015-01-01	2004-07-01
2	2015-01-01	2007-08-01
3	2014-11-01	2006-09-01
4	2013-04-01	1999-03-01
5	2015-09-01	2005-01-01

In [415…

```
data['issue_month_num']=data['issue_date'].dt.month
data['issue_year']=data['issue_date'].dt.year
data['earliest_cr_line_month_num']=data['earliest_cr_line_date'].dt.month
data['earliest_cr_line_year']=data['earliest_cr_line_date'].dt.year
data.loc[:5,['issue_date','issue_month_num','issue_year','earliest_cr_line_month_num','earliest_cr_line_year']]
```

Out[415…

	issue_date	issue_month_num	issue_year	earliest_cr_line_month_num	earliest_cr_line_year
0	2015-01-01	1	2015	6	1990
1	2015-01-01	1	2015	7	2004
2	2015-01-01	1	2015	8	2007
3	2014-11-01	11	2014	9	2006
4	2013-04-01	4	2013	3	1999
5	2015-09-01	9	2015	1	2005

In [416…

```
data['issue_month']=data['issue_date'].dt.strftime('%b')
data['earliest_cr_line_month']=data['earliest_cr_line_date'].dt.strftime('%b')
```

Pincode Extract

Extracting useful information from unstructred data

Enables Geographical analysis

Reduce noise, focus on relevant features for ML Models

In [417…
```
data['address'].nunique()
```
Out[417…
```
393426
```
In [418…
```
data['address']=data['address'].apply(lambda x: x[-5:])  # pincode length is 5 and extracting last 5 digits
data['address'].nunique()
```
Out[418…
```
10
```
Simple Feature Engineering steps:

E.g.: Creation of Flags- If value greater than 1.0 then 1 else 0. This can be done on:

1. Pub_rec: Number of derogatory public records
2. Mort_acc: Number of mortgage accounts.
3. Pub_rec_bankruptcies: Number of public record bankruptcies

In [419…
```
data['pub_rec']=data['pub_rec'].apply(lambda x: 1 if x>=1.0 else 0)
data['mort_acc']=data['mort_acc'].apply(lambda x: 1 if x>1.0 else 0)
data['pub_rec_bankruptcies']=data['pub_rec_bankruptcies'].apply(lambda x: 1 if x>=1.0 else 0)
```
In [420…
```
data['pub_rec']=data['pub_rec'].astype('object')
data['mort_acc']=data['mort_acc'].astype('object')
data['pub_rec_bankruptcies']=data['pub_rec_bankruptcies'].astype('object')
```
In [421…
```
data.dtypes
```

Out[421…

```
loan_amnt                          float64
term                                object
int_rate                           float64
installment                        float64
grade                               object
sub_grade                           object
emp_title                           object
emp_length                          object
home_ownership                      object
annual_inc                         float64
verification_status                 object
issue_d                             object
loan_status                         object
purpose                             object
title                               object
dti                                float64
earliest_cr_line                    object
open_acc                           float64
pub_rec                             object
revol_bal                          float64
revol_util                         float64
total_acc                          float64
initial_list_status                 object
application_type                    object
mort_acc                            object
pub_rec_bankruptcies                object
address                             object
issue_date                  datetime64[ns]
earliest_cr_line_date       datetime64[ns]
issue_month_num                      int32
issue_year                           int32
earliest_cr_line_month_num           int32
earliest_cr_line_year                int32
issue_month                         object
earliest_cr_line_month              object
dtype: object
```

In [422…

```
## get list of categorical columns and nummeric columns
cat_columns=data.select_dtypes(include=['object']).columns.to_list()
cat_columns
```

Out[422…

```
['term',
 'grade',
 'sub_grade',
 'emp_title',
 'emp_length',
 'home_ownership',
 'verification_status',
 'issue_d',
 'loan_status',
 'purpose',
 'title',
 'earliest_cr_line',
 'pub_rec',
 'initial_list_status',
 'application_type',
 'mort_acc',
 'pub_rec_bankruptcies',
 'address',
 'issue_month',
 'earliest_cr_line_month']
```

In [423…

```
var_remove=['issue_d','earliest_cr_line']
cat_cols_2=[i for i in cat_columns if i not in var_remove]
cat_cols_2
```

Out[423…

```
['term',
 'grade',
 'sub_grade',
 'emp_title',
 'emp_length',
 'home_ownership',
 'verification_status',
 'loan_status',
 'purpose',
 'title',
 'pub_rec',
 'initial_list_status',
 'application_type',
 'mort_acc',
 'pub_rec_bankruptcies',
 'address',
 'issue_month',
 'earliest_cr_line_month']
```

In [424…

```
num_columns=data.select_dtypes(exclude=['object']).columns.to_list()
num_columns
```

Out[424…

```
['loan_amnt',
 'int_rate',
 'installment',
 'annual_inc',
 'dti',
 'open_acc',
 'revol_bal',
 'revol_util',
 'total_acc',
 'issue_date',
 'earliest_cr_line_date',
 'issue_month_num',
 'issue_year',
 'earliest_cr_line_month_num',
 'earliest_cr_line_year']
```

In [425…

```
## del len  because I overwrite the length function in the script
len(cat_cols_2)
```

Out[425…

```
18
```

Step 6: Univariate Analysis for (analyzing the distribution, central tendency, and spread of data effectively)

In [426…

```
cat_cols_2
```

Out[426…

```
['term',
 'grade',
 'sub_grade',
 'emp_title',
 'emp_length',
 'home_ownership',
 'verification_status',
 'loan_status',
 'purpose',
 'title',
 'pub_rec',
 'initial_list_status',
 'application_type',
 'mort_acc',
 'pub_rec_bankruptcies',
 'address',
 'issue_month',
 'earliest_cr_line_month']
```

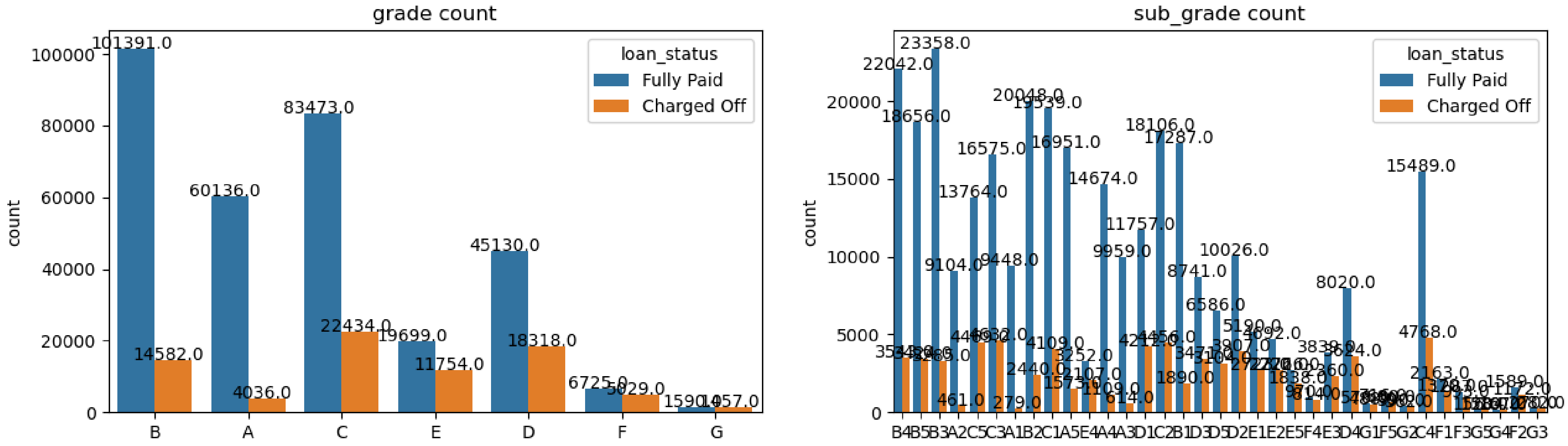
In [427…

```
## count plot for cat_columns
plt.figure(figsize=(15,4))
j=0

plt.subplot(1,2,1)
plt.title(cat_cols_2[1]+" count")
ax=sns.countplot(data=data,x=cat_cols_2[1],hue='loan_status')
total=len(data)
for bars in ax.patches:
    percentage=bars.get_height()
    x = bars.get_x() + bars.get_width() / 2 - 0.05
    y = bars.get_height()
    ax.annotate(percentage, (x, y), ha='center')
```

```
ax.xaxis.label.set_visible(False)

plt.subplot(1,2,2)
plt.title(cat_cols_2[2]+" count")
ax=sns.countplot(data=data,x=cat_cols_2[2],hue='loan_status')
total=len(data)
for bars in ax.patches:
    percentage=bars.get_height()
    x = bars.get_x() + bars.get_width() / 2 - 0.05
    y = bars.get_height()
    ax.annotate(percentage, (x, y), ha='center')
ax.xaxis.label.set_visible(False)
```



Best and worst Grades

The ratio of Full paid and charged off is max for Grade B and so for subgrade in B Grade

The ratio of full paid and charged off is minimum ~ 50:50 for C grade.

Best Grade: B

Worst Grade: C

In [428…
```
data['emp_title'].value_counts()
```

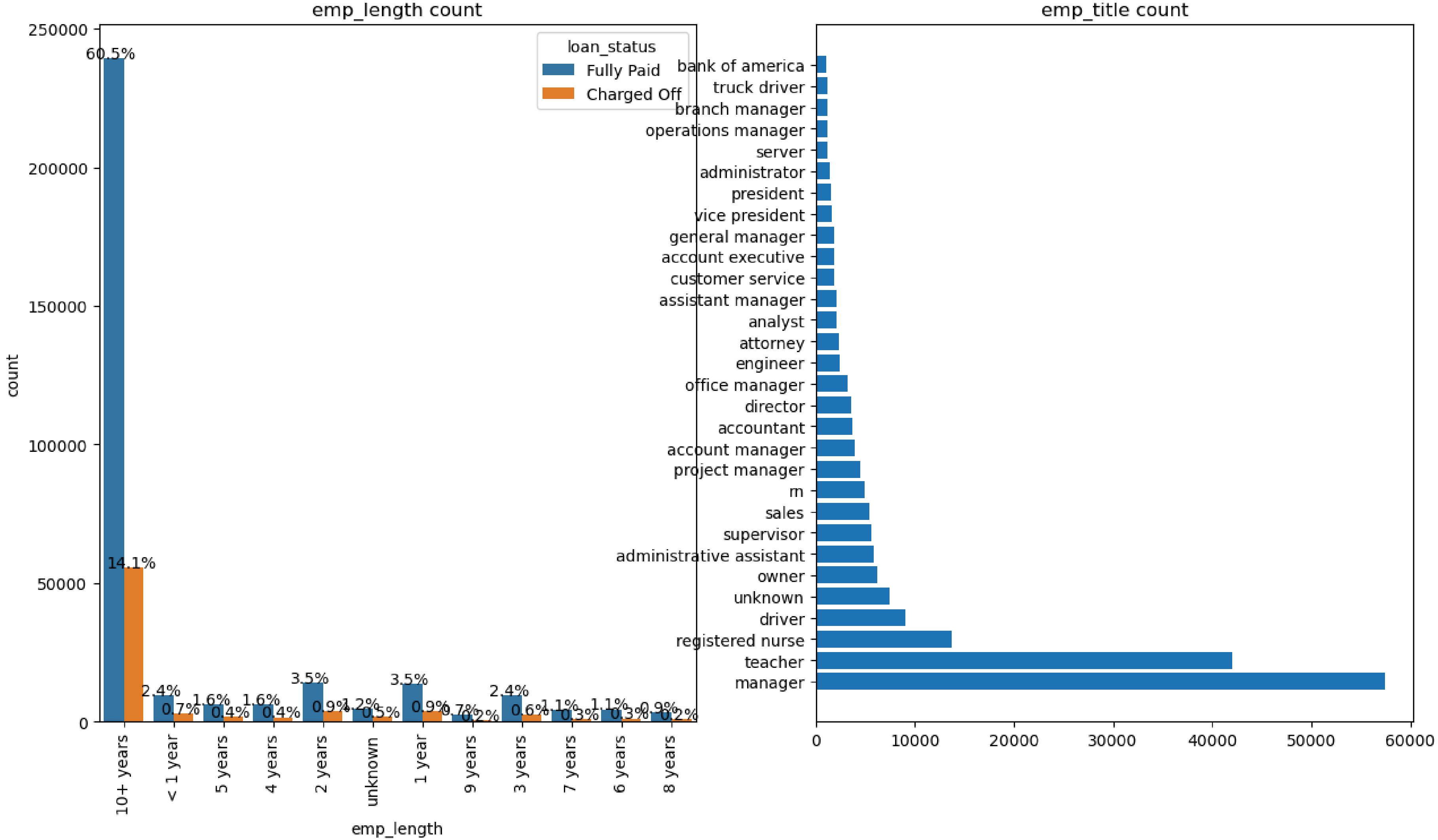
Out[428…
```
emp_title
manager                                 57413
teacher                                 41979
registered nurse                        13752
driver                                   9059
unknown                                  7427
                                        ...
nevada hand                                 1
morgan stanely smith barney                 1
service associate - legal department        1
doublefine inc                              1
data center specialist ii                   1
Name: count, Length: 46429, dtype: int64
```

In [429…
```
## count plot for cat_columns
plt.figure(figsize=(15,8))

plt.subplot(1,2,1)
plt.title(cat_cols_2[4]+" count")
ax=sns.countplot(data=data,x=cat_cols_2[4],hue='loan_status')
for bars in ax.patches:
    percentage='{:.1f}%'.format(100*bars.get_height()/total)
    x = bars.get_x() + bars.get_width() / 2 - 0.05
    y = bars.get_height()
    ax.annotate(percentage, (x, y), ha='center')
    ax.set_xticklabels(ax.get_xticklabels(),rotation=90)

plt.subplot(1,2,2)
plt.title(cat_cols_2[3]+" count")
plt.barh(data['emp_title'].value_counts()[:30].index,data['emp_title'].value_counts()[:30])
```

Out[429…
```
<BarContainer object of 30 artists>
```



Insights:

employee title: Manager, teacher are the most afforded job titles.

Emp length: the 75% loans are afforded by employees with tenure=10+years, out of which 61% have fully paid the loan and 14% have been charged of.

In [430…

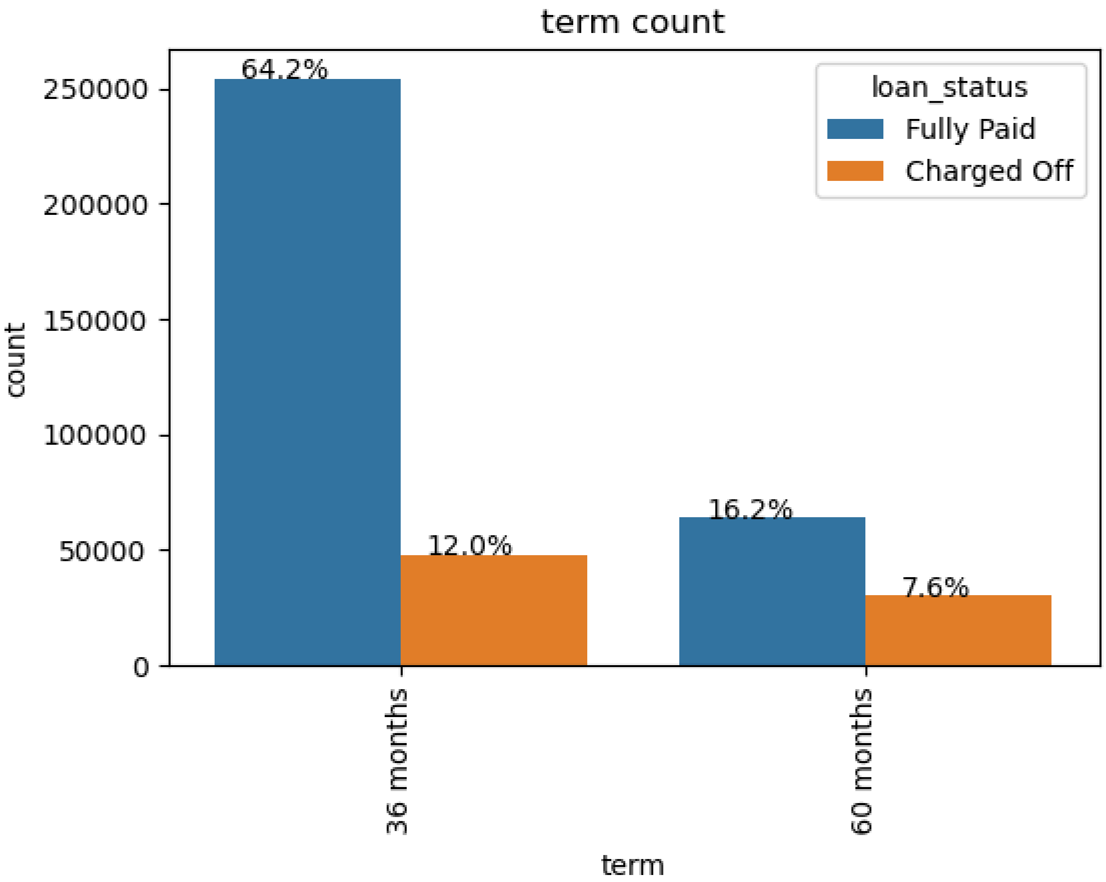
```
data['title'].value_counts()
```

Out[430…

```
title
debt consolidation         317103
credit card refinancing     23994
other                        4653
home improvement             4330
consolidation                2700
                            ...
my consolidation plan           1
consolidate & windows           1
deliverance                     1
credit crads                    1
toxic debt payoff               1
Name: count, Length: 12030, dtype: int64
```

In [431…

```
plt.figure(figsize=(6,4))
plt.title(cat_cols_2[0]+" count")
ax=sns.countplot(data=data,x=cat_cols_2[0],hue='loan_status')
for bars in ax.patches:
    percentage='{:.1f}%'.format(100*bars.get_height()/total)
    x = bars.get_x() + bars.get_width() / 2 - 0.05
    y = bars.get_height()
    ax.annotate(percentage, (x, y), ha='center')
    ax.set_xticklabels(ax.get_xticklabels(),rotation=90)
```



Insights:

76% loans are taken for 36 months term and the ratio of fully paid to charged off is ~5 and for 60 months this ratio is 2 hence 36 months loans terms are safe terms to cover risk

In [432…

```
cat_cols_2
```

Out[432…

```
['term',
 'grade',
 'sub_grade',
 'emp_title',
 'emp_length',
 'home_ownership',
 'verification_status',
 'loan_status',
 'purpose',
 'title',
 'pub_rec',
 'initial_list_status',
 'application_type',
 'mort_acc',
 'pub_rec_bankruptcies',
 'address',
 'issue_month',
 'earliest_cr_line_month']
```

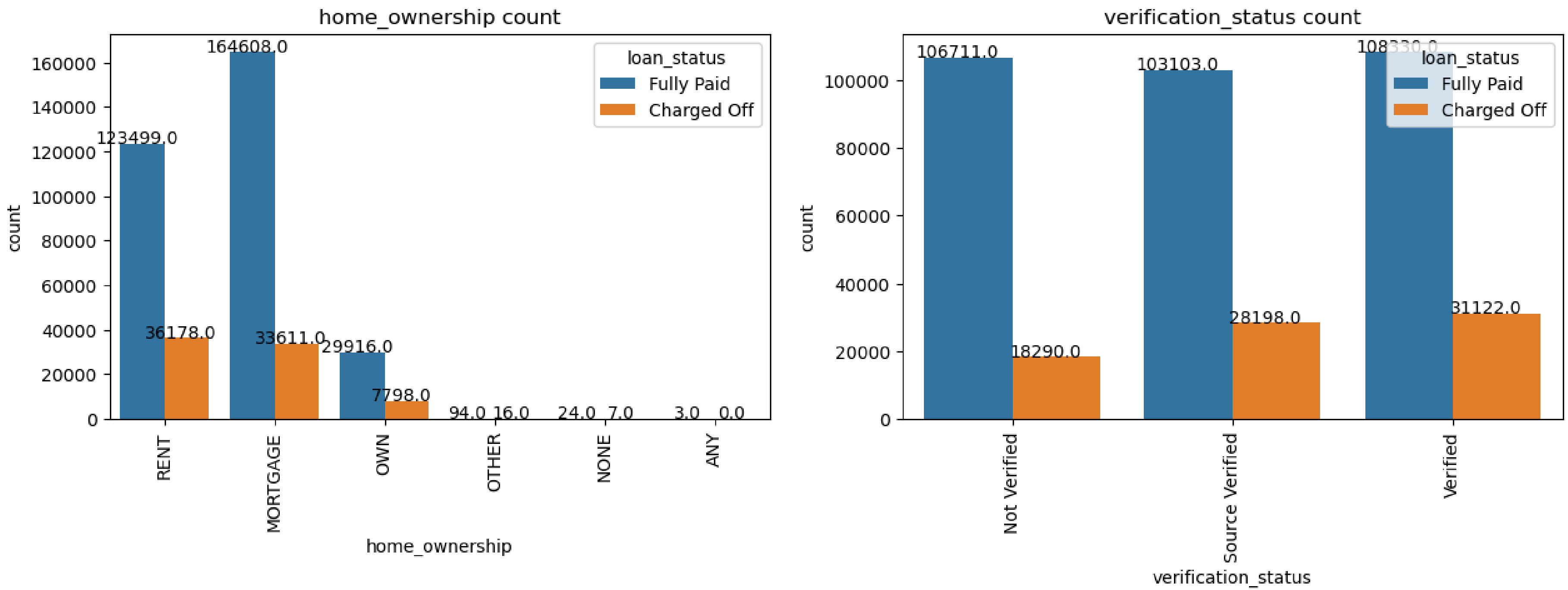
In [433…

```
## count plot for cat_columns
plt.figure(figsize=(15,4))

plt.subplot(1,2,1)
plt.title(cat_cols_2[5]+" count")
ax=sns.countplot(data=data,x=cat_cols_2[5], hue='loan_status')
for bars in ax.patches:
    percentage=bars.get_height()
    x = bars.get_x() + bars.get_width() / 2 - 0.05
    y = bars.get_height()
    ax.annotate(percentage, (x, y), ha='center')
    ax.set_xticklabels(ax.get_xticklabels(),rotation=90)

plt.subplot(1,2,2)
plt.title(cat_cols_2[6]+" count")
ax=sns.countplot(data=data,x=cat_cols_2[6],hue='loan_status')
for bars in ax.patches:
    percentage=bars.get_height()
    x = bars.get_x() + bars.get_width() / 2 - 0.05
    y = bars.get_height()
```

```
    ax.annotate(percentage, (x, y), ha='center')
    ax.set_xticklabels(ax.get_xticklabels(),rotation=90)
```



In [434...

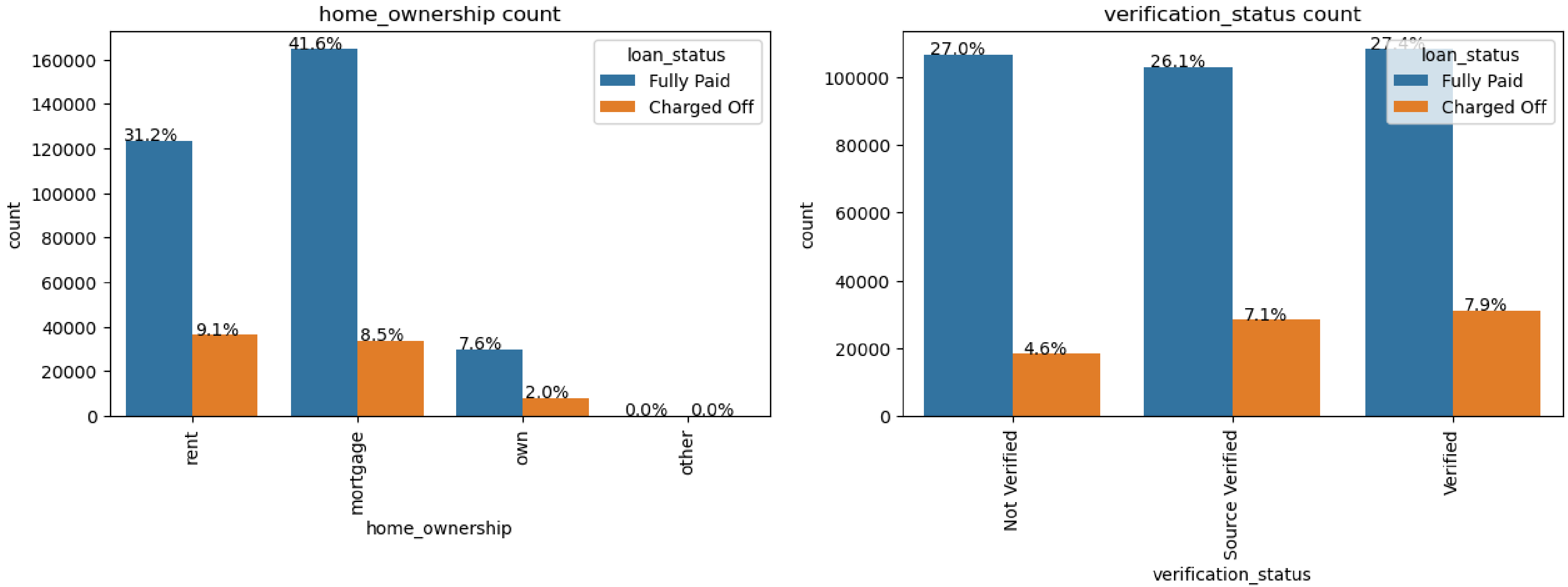
```
## Lets combine other none any to other
data['home_ownership']=data['home_ownership'].apply(lambda x:x.lower())
data['home_ownership']=data['home_ownership'].apply(lambda x: 'other' if x in ('none','any') else x)
```

In [435...

```
## count plot for cat_columns
plt.figure(figsize=(15,4))

plt.subplot(1,2,1)
plt.title(cat_cols_2[5]+" count")
ax=sns.countplot(data=data,x=cat_cols_2[5], hue='loan_status')
for bars in ax.patches:
    percentage='{:.1f}%'.format(100*bars.get_height()/total)
    x = bars.get_x() + bars.get_width() / 2 - 0.05
    y = bars.get_height()
    ax.annotate(percentage, (x, y), ha='center')
    ax.set_xticklabels(ax.get_xticklabels(),rotation=90)
```

```
plt.subplot(1,2,2)
plt.title(cat_cols_2[6]+" count")
ax=sns.countplot(data=data,x=cat_cols_2[6],hue='loan_status')
for bars in ax.patches:
    percentage='{:.1f}%'.format(100*bars.get_height()/total)
    x = bars.get_x() + bars.get_width() / 2 - 0.05
    y = bars.get_height()
    ax.annotate(percentage, (x, y), ha='center')
    ax.set_xticklabels(ax.get_xticklabels(),rotation=90)
```



Insights:

homeownership: mortgage account fully paid to charged off loan status ratio is ~5 which is highest more than 90% loans are afforded by people with homeownership mortgage or rent people who own the house records are just 9%

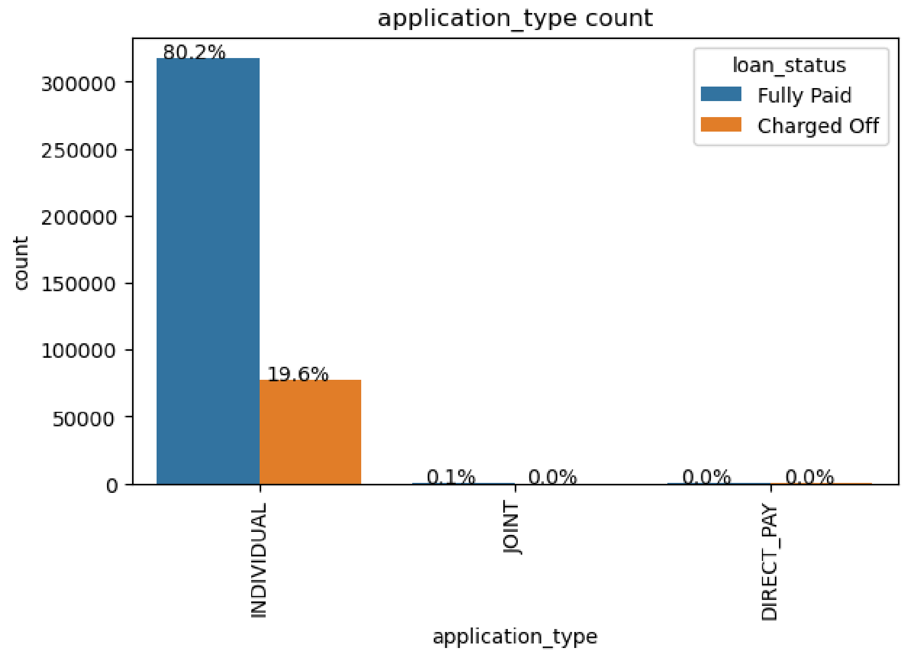
Verification st tus there is no significant difference b/w the loan status ratio or data distribution. Hence its seemly has not major impact on loan staus.

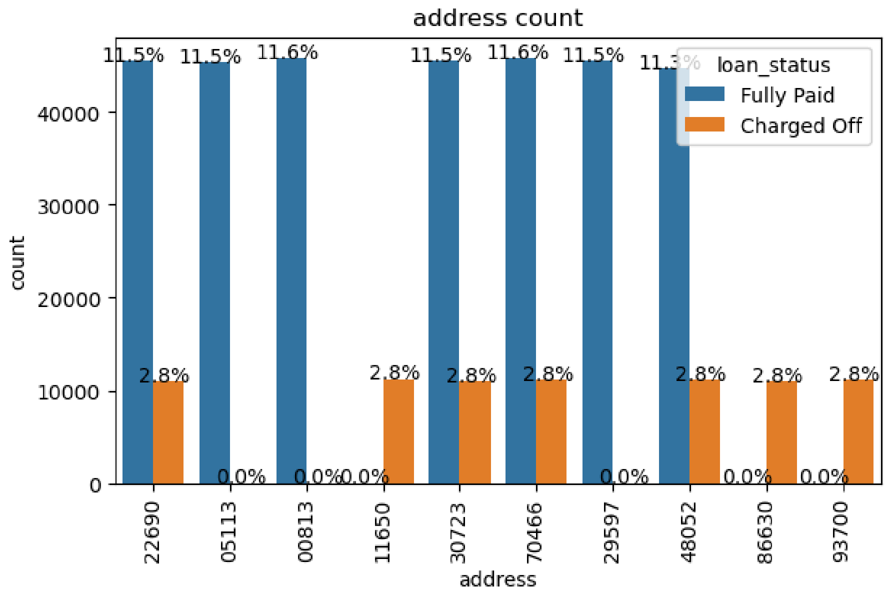
In [436…

```
## count plot for cat_columns
plt.figure(figsize=(15,4))

plt.subplot(1,2,1)
plt.title('application_type'+" count")
ax=sns.countplot(data=data,x='application_type', hue='loan_status')
for bars in ax.patches:
    percentage='{:.1f}%'.format(100*bars.get_height()/total)
    x = bars.get_x() + bars.get_width() / 2 - 0.05
    y = bars.get_height()
    ax.annotate(percentage, (x, y), ha='center')
    ax.set_xticklabels(ax.get_xticklabels(),rotation=90)

plt.subplot(1,2,2)
plt.title('address'+" count")
ax=sns.countplot(data=data,x='address',hue='loan_status')
for bars in ax.patches:
    percentage='{:.1f}%'.format(100*bars.get_height()/total)
    x = bars.get_x() + bars.get_width() / 2 - 0.05
    y = bars.get_height()
    ax.annotate(percentage, (x, y), ha='center')
    ax.set_xticklabels(ax.get_xticklabels(),rotation=90)
```





Insights:

99.8 % records are Individual application_type

maximum loans are taken for pincodes (22690, 05113, 20723,70466,29597,48052) with worst fully_paid/charged off ratio for 11650

In [437…

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

In [438…

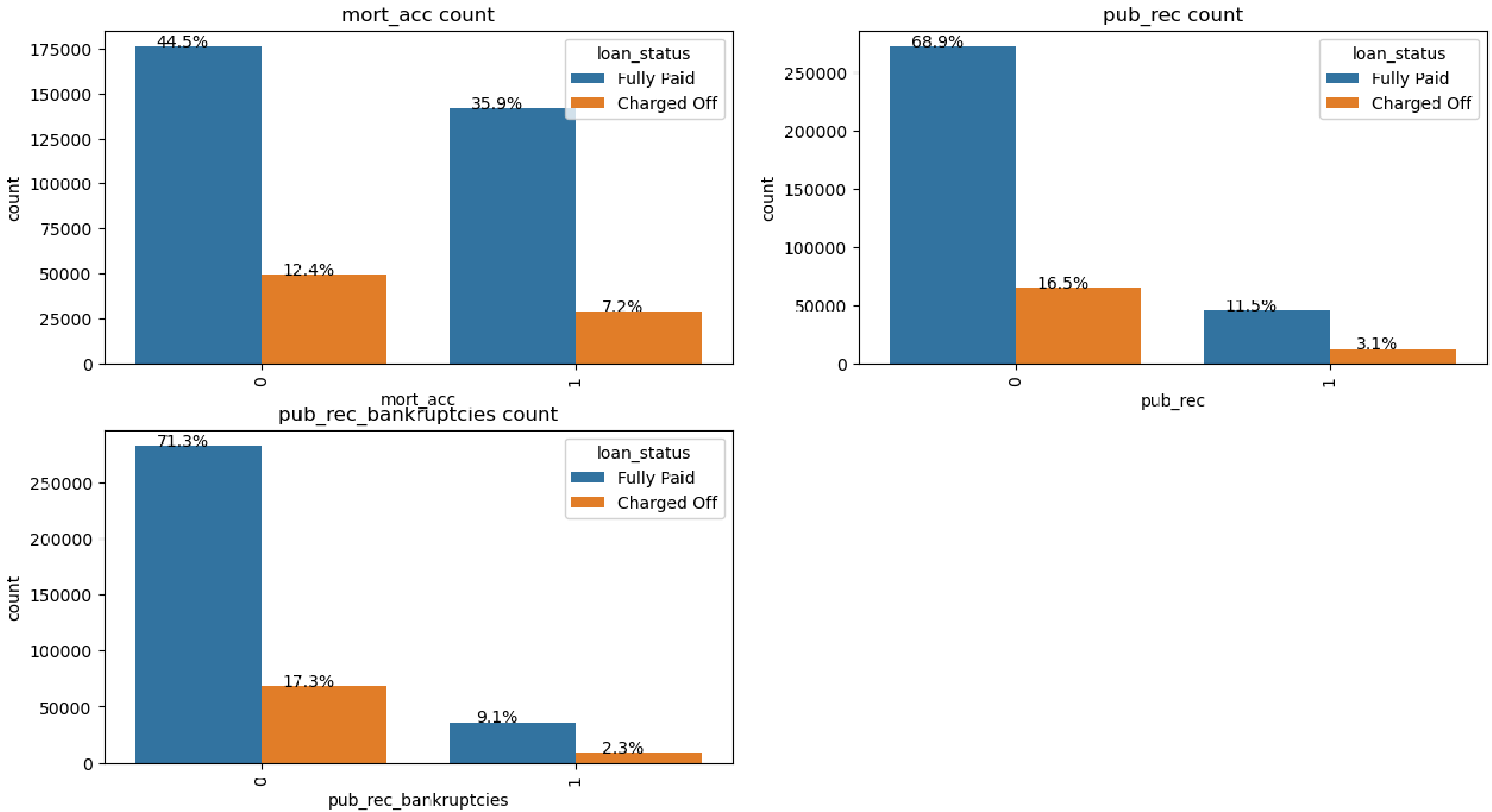
```
## count plot for cat_columns
plt.figure(figsize=(15,8))

plt.subplot(2,2,1)
plt.title('mort_acc'+" count")
ax=sns.countplot(data=data,x='mort_acc', hue='loan_status')
for bars in ax.patches:
    percentage='{:.1f}%'.format(100*bars.get_height()/total)
    x = bars.get_x() + bars.get_width() / 2 - 0.05
```

```
    y = bars.get_height()
    ax.annotate(percentage, (x, y), ha='center')
    ax.set_xticklabels(ax.get_xticklabels(),rotation=90)

plt.subplot(2,2,2)
plt.title('pub_rec'+" count")
ax=sns.countplot(data=data,x='pub_rec',hue='loan_status')
for bars in ax.patches:
    percentage='{:.1f}%'.format(100*bars.get_height()/total)
    x = bars.get_x() + bars.get_width() / 2 - 0.05
    y = bars.get_height()
    ax.annotate(percentage, (x, y), ha='center')
    ax.set_xticklabels(ax.get_xticklabels(),rotation=90)

plt.subplot(2,2,3)
plt.title('pub_rec_bankruptcies'+" count")
ax=sns.countplot(data=data,x='pub_rec_bankruptcies',hue='loan_status')
for bars in ax.patches:
    percentage='{:.1f}%'.format(100*bars.get_height()/total)
    x = bars.get_x() + bars.get_width() / 2 - 0.05
    y = bars.get_height()
    ax.annotate(percentage, (x, y), ha='center')
    ax.set_xticklabels(ax.get_xticklabels(),rotation=90)
```



Insights:

mort_acc : Number of mortgage accounts. 43.1 % have mort_acc>1 and 56.9 % have less than 1 mort_acc. Almost 21% charged off in both cases. Hence its doesn't have much impact on loan status.

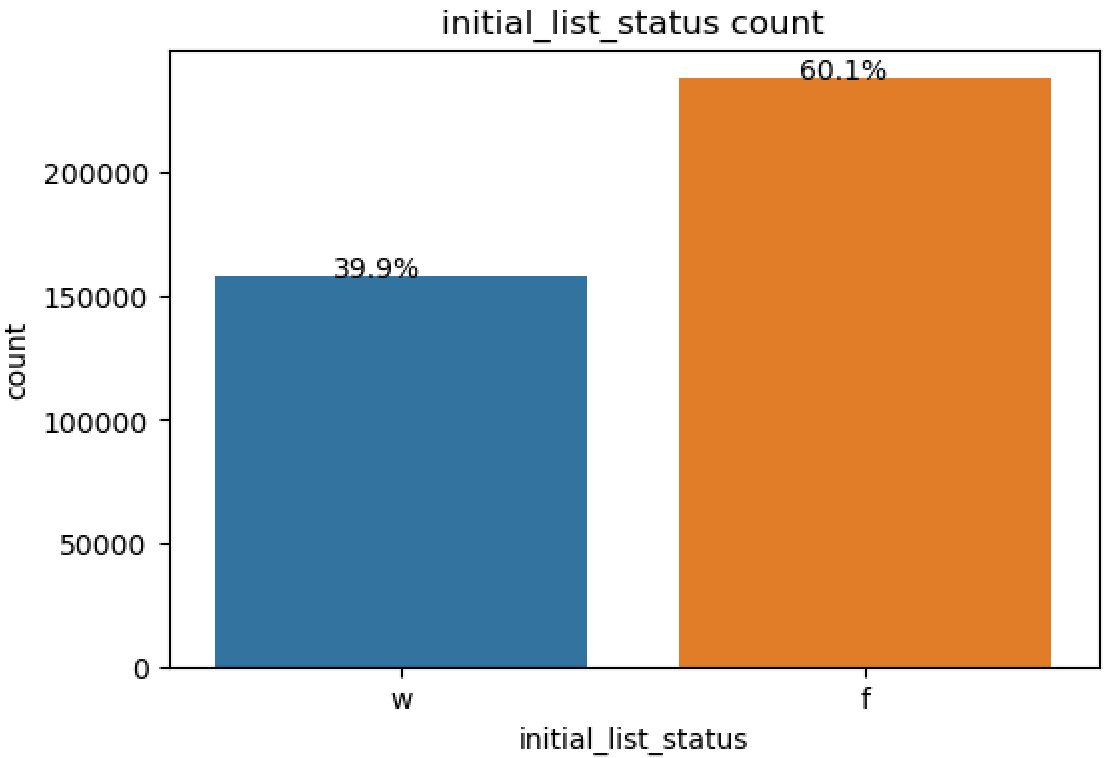
pub_rec_bankruptcies : Number of public record bankruptcies. pub_rec_bankruptcies>=0: 88.6% records, 19.5% are charged off. pub_rec_bankruptcies>=1: 11.4% records, 20.6% are charged off.

pub_rec : Number of derogatory public records. pun_rec=0: 85.4% total records,19% charged off. pun_rec>=1:14.6 total records, 21% were charged off

In [439…

```
plt.figure(figsize=(6,4))

plt.title('initial_list_status'+" count")
ax=sns.countplot(data=data,x='initial_list_status')
for bars in ax.patches:
    percentage='{:.1f}%'.format(100*bars.get_height()/total)
    x = bars.get_x() + bars.get_width() / 2 - 0.05
    y = bars.get_height()
    ax.annotate(percentage, (x, y), ha='center')
```

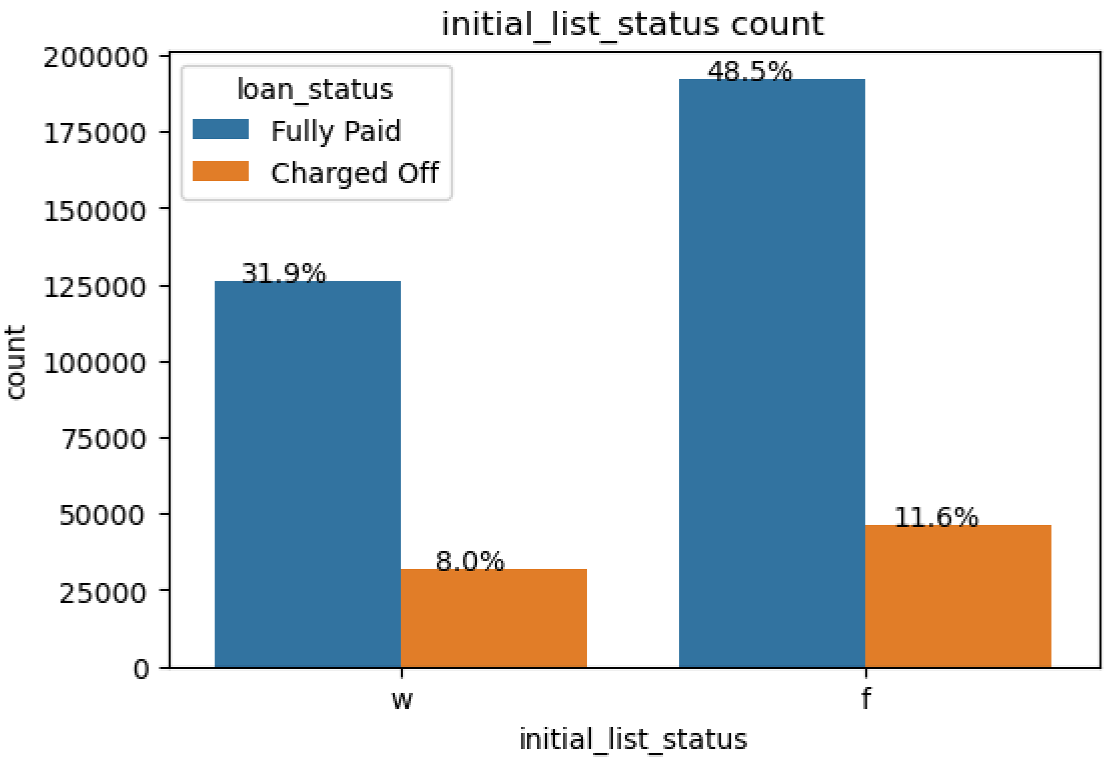


In [440…

```
plt.figure(figsize=(6,4))

plt.title('initial_list_status'+" count")
```

```
ax=sns.countplot(data=data,x='initial_list_status',hue='loan_status')
for bars in ax.patches:
    percentage='{:.1f}%'.format(100*bars.get_height()/total)
    x = bars.get_x() + bars.get_width() / 2 - 0.05
    y = bars.get_height()
    ax.annotate(percentage, (x, y), ha='center')
```



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

status F: 60.1% records, 19% are charged off status W: 39.9% records, 20% are charged off

not much impact on the laon_status

Insights:

term: **36 months has 76.3 % records and 60 months with 23.6 % records**

grade: **B & C owns almost 50% records, A, E & D has 40% records, G & F has very less 3.8% records**

home_ownership : **50.1 % records are for ownership type as Mortgage/b>, 40.3 % records are for Rental property and only 9.5 % owns the house.**

verification_status: Distribution among all three category variables(own verified, source verified, not verified) is almost same aprrox 31% each

loan_status: **80.4% records are for fully paid and 19.6% are Charged off**

purpose: **debt_consolidation has 59.2 % records followed by credit_card 21 %** rest varibles have combined together only 20% records

employee_length: **10+years i.e experienced over 10+ years own the 31% records** and restb 70% records distibution is very similar in other categories almost 4% to 9%

initial_list_status: **60% records for f and 40% records for w**

Application type: 99% records are for Individual application status. **(hence this field can be dropped as its significance is negligible)**

title: **top 2 afforded job titles are Debt consolidation with 38.67% records and Credit card refinancing with 13 % records**

mort_acc: **60 % have mort_acc>1 and 40 % have less than 1 mort_acc**

pub_rec_banruptcies: 99.4 % have pub_rec_banruptcies records>1 and only 0.6 % have less than equal to 1

pub_rec: 98 % have pub_rec (derogatory comments records) >1 and just 2% have less than equal to 1

Verification status there is no significant difference b/w the loan status ratio or data distribution. Hence its seemly has not major impact on loan status.

99.8 % records are Individual application type. Thus, this doesn’t have much impact on the loan statu

Date columns should have no impact on loan_status. Drop those columns.s

Drop the columns not important for the predictions.

Note: Removing all these columns caused poor performance of the model, hence keeping all those features. Removing Address caused huge performance Issue, but the address had huge coefficient to handle that I used Lasso Regularization.

In [441…
```
data_2=data
```

In [442…
```
data.drop(columns=[ 'issue_date', 'earliest_cr_line_date', 'issue_month_num', 'issue_year',
       'earliest_cr_line_month_num', 'earliest_cr_line_year', 'issue_month',
       'earliest_cr_line_month','issue_d','earliest_cr_line'], inplace=True)
```

In [443…
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 395754 entries, 0 to 396029
Data columns (total 25 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             395754 non-null  float64
 1   term                  395754 non-null  object 
 2   int_rate              395754 non-null  float64
 3   installment           395754 non-null  float64
 4   grade                 395754 non-null  object 
 5   sub_grade             395754 non-null  object 
 6   emp_title             395754 non-null  object 
 7   emp_length            395754 non-null  object 
 8   home_ownership        395754 non-null  object 
 9   annual_inc            395754 non-null  float64
 10  verification_status   395754 non-null  object 
 11  loan_status           395754 non-null  object 
 12  purpose               395754 non-null  object 
 13  title                 395754 non-null  object 
 14  dti                   395754 non-null  float64
 15  open_acc              395754 non-null  float64
 16  pub_rec               395754 non-null  object 
 17  revol_bal             395754 non-null  float64
 18  revol_util            395754 non-null  float64
 19  total_acc             395754 non-null  float64
 20  initial_list_status   395754 non-null  object 
 21  application_type      395754 non-null  object 
 22  mort_acc              395754 non-null  object 
 23  pub_rec_bankruptcies  395754 non-null  object 
 24  address               395754 non-null  object 
dtypes: float64(9), object(16)
memory usage: 86.6+ MB
```

In [444…

```
data.select_dtypes(include='object').columns
```

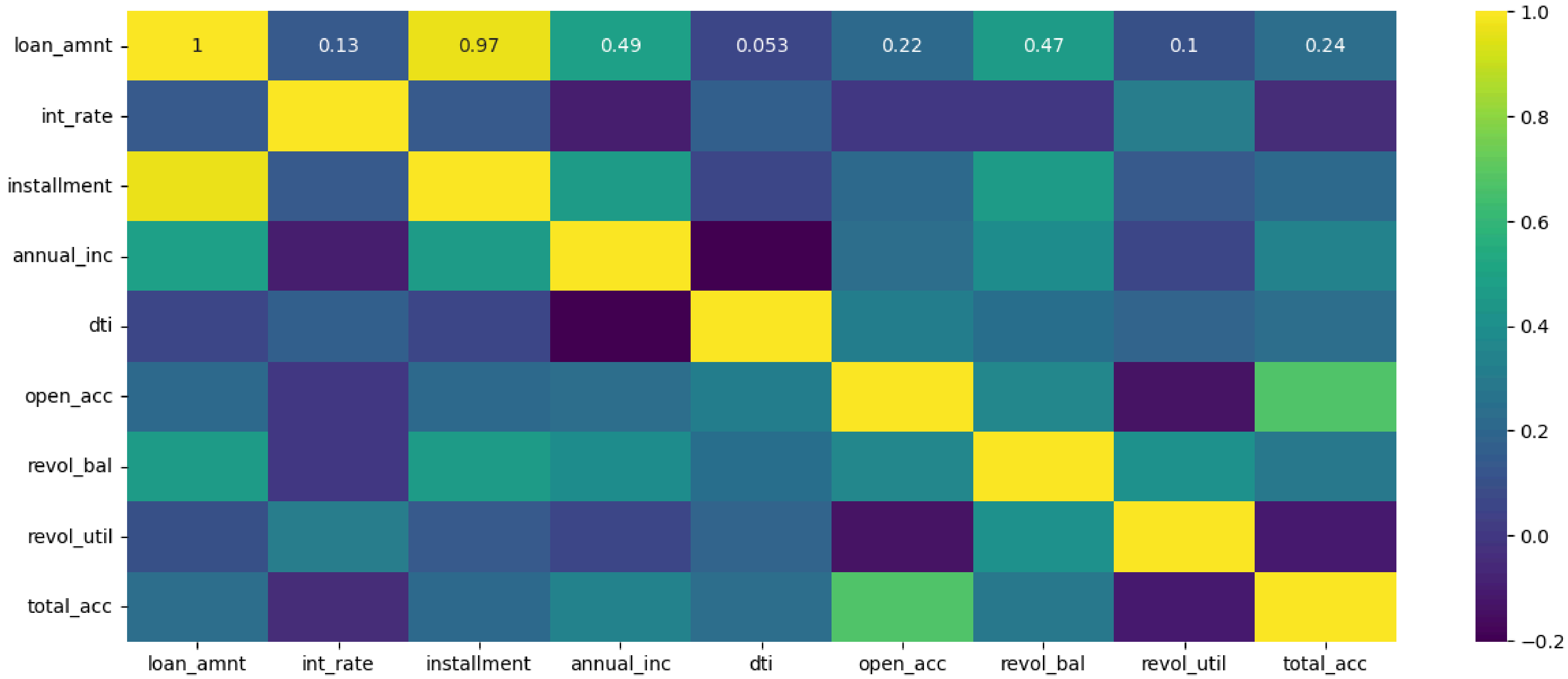
Out[444…

```
Index(['term', 'grade', 'sub_grade', 'emp_title', 'emp_length',
       'home_ownership', 'verification_status', 'loan_status', 'purpose',
       'title', 'pub_rec', 'initial_list_status', 'application_type',
       'mort_acc', 'pub_rec_bankruptcies', 'address'],
      dtype='object')
```

In [445…
```
num_columns=data.select_dtypes(exclude=['object']).columns.to_list()
num_columns
```

Out[445…
```
['loan_amnt',
 'int_rate',
 'installment',
 'annual_inc',
 'dti',
 'open_acc',
 'revol_bal',
 'revol_util',
 'total_acc']
```

In [446…
```
plt.figure(figsize=(15,6))
sns.heatmap(data.loc[:,num_columns].corr(method='spearman'), annot=True, cmap='viridis')
plt.show()
```



Correlation:

Loan status has strong correlation with installment,loan amount.

This one of the column shall be droped to avoid muticolinearity. We will drop installments as loan amount has more impact on loan status.

In [447…

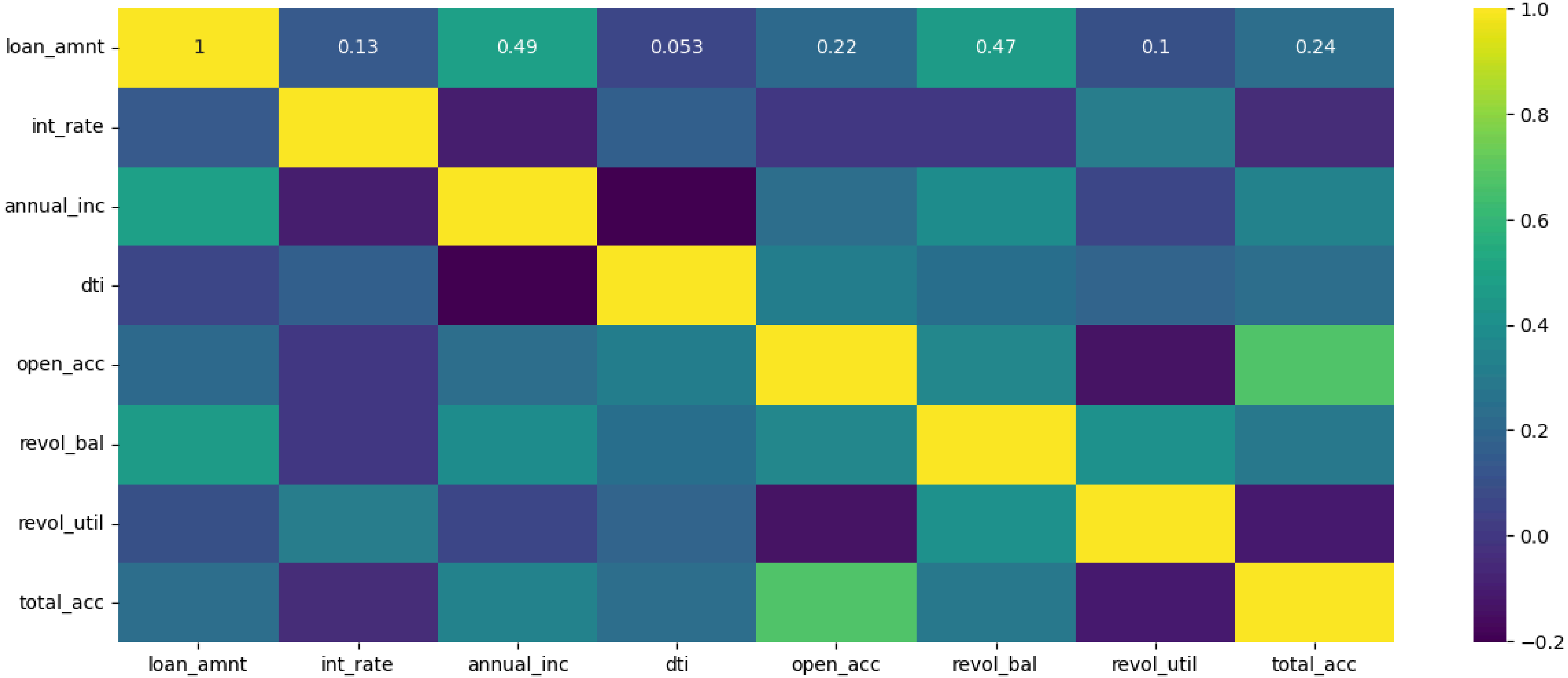
```
data.drop(columns='installment',inplace=True)
```

In [448…

```
num_columns=[i for i in num_columns if i not in ('installment')]
```

In [449…

```
plt.figure(figsize=(15,6))
sns.heatmap(data.loc[:,num_columns].corr(method='spearman'), annot=True, cmap='viridis')
plt.show()
```



In [450…
```
data['open_acc'].corr(data['total_acc'])
```

Out[450…
```
0.6809298845618614
```

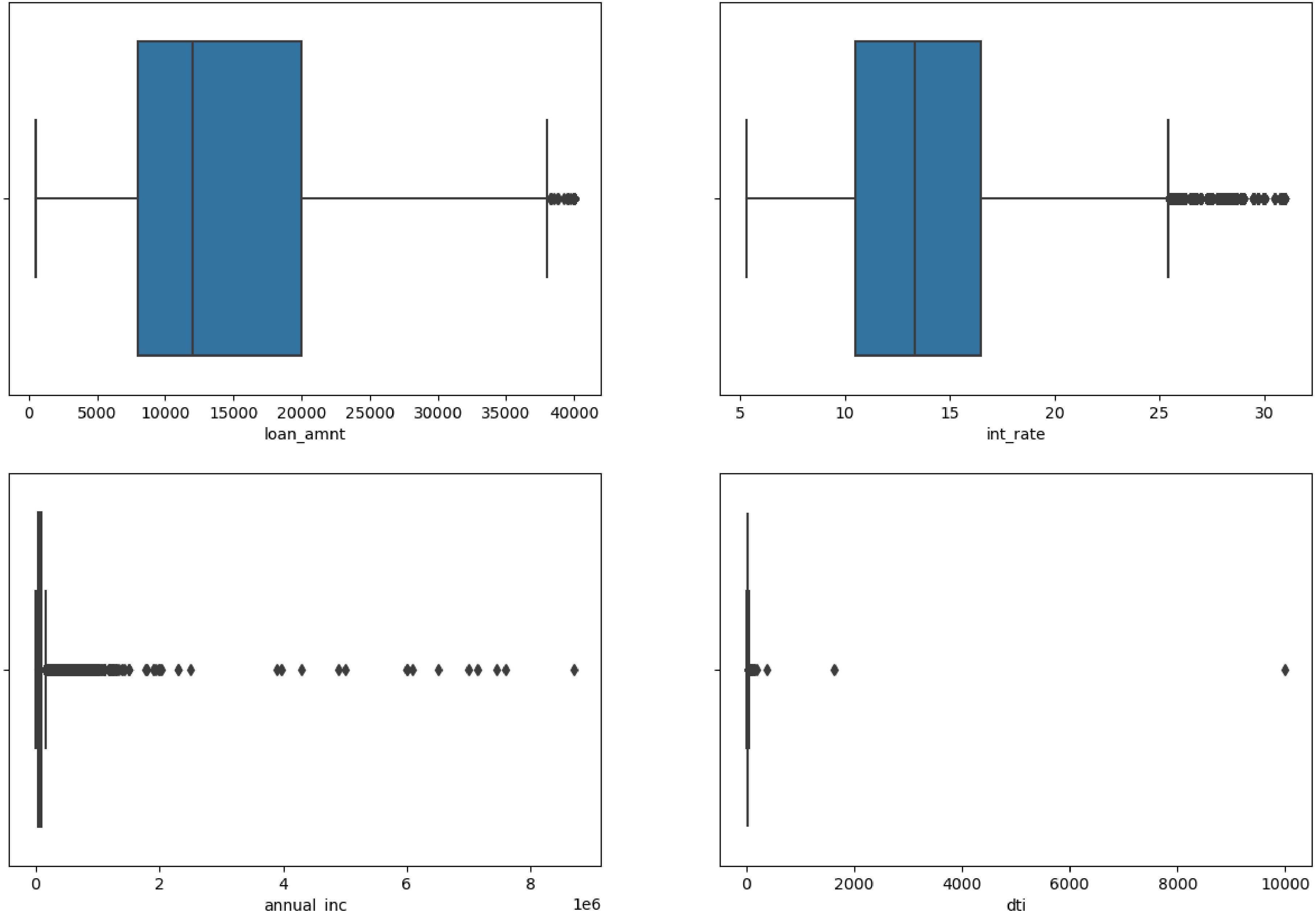
In [451…
```
numerical_data = data.select_dtypes(include='number')
num_cols = numerical_data.columns
len(num_cols)
```

Out[451…
```
8
```

Outlier Detection & Handling

Using IQR or Zscore (3 sigma rule, 99.7% data lies within mean+-3sigma

In [452…
```
j=1
plt.figure(figsize=(15,10))
for i in num_columns[:4]:
    plt.subplot(2,2,j)
    sns.boxplot(data=data,x=i)
    j+=1
```



In [453…

```
data.shape
```

Out[453…
```
(395754, 24)
```

In [454…
```
df=data
for col in num_cols:
 mean = df[col].mean()
 std = df[col].std()
 upper_limit = mean+3*std
 lower_limit = mean-3*std
 df = df[(df[col]<upper_limit) & (df[col]>lower_limit)]

df.shape
```

Out[454…
```
(378710, 24)
```

In [455…
```
data.loan_status.value_counts()
```

Out[455…
```
loan_status
Fully Paid     318144
Charged Off     77610
Name: count, dtype: int64
```

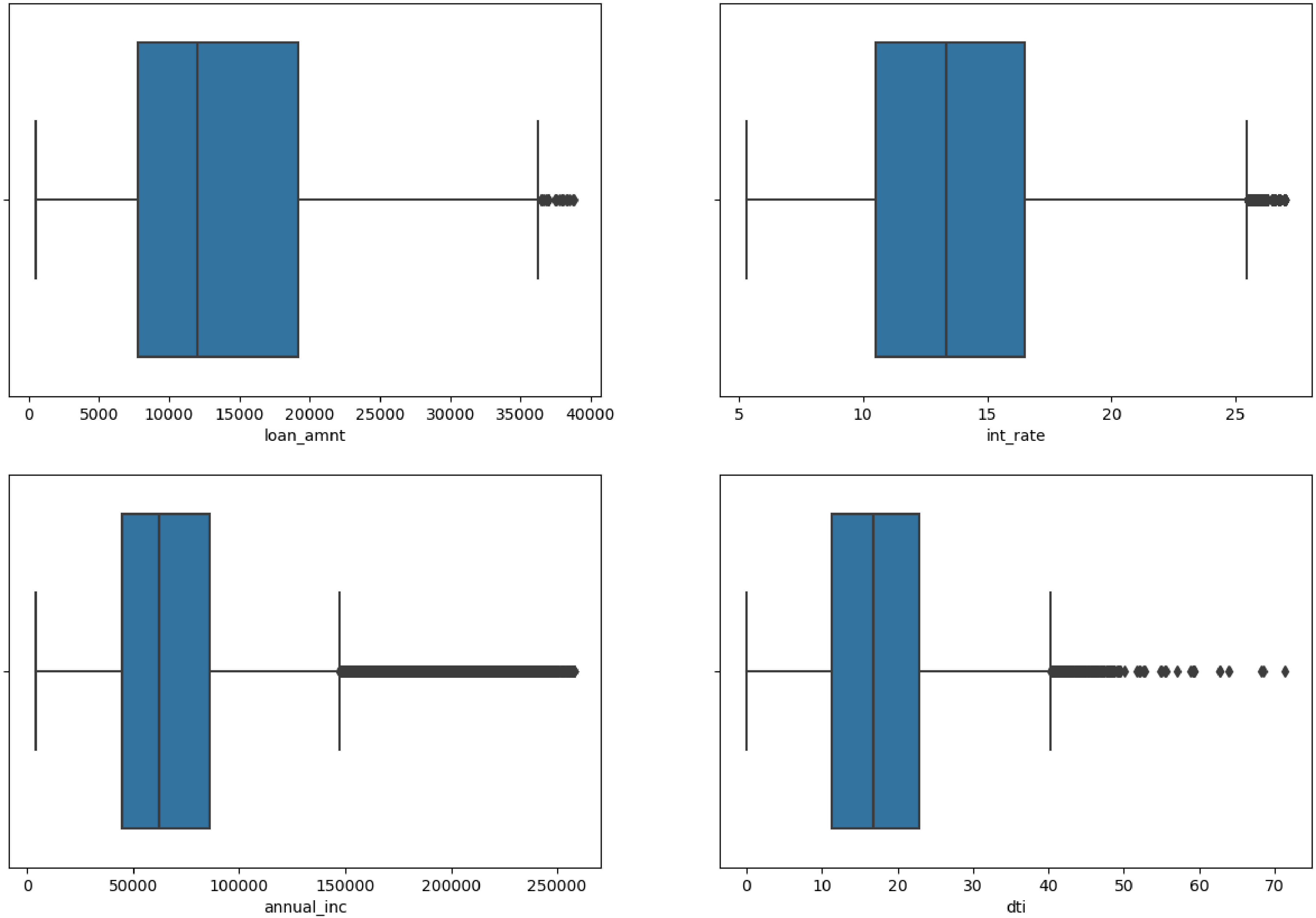
In [456…
```
(data.shape[0]-df.shape[0])/data.shape[0]*100
```

Out[456…
```
4.306715788090582
```

In [457…
```
data=df
```

4.3% data is removed

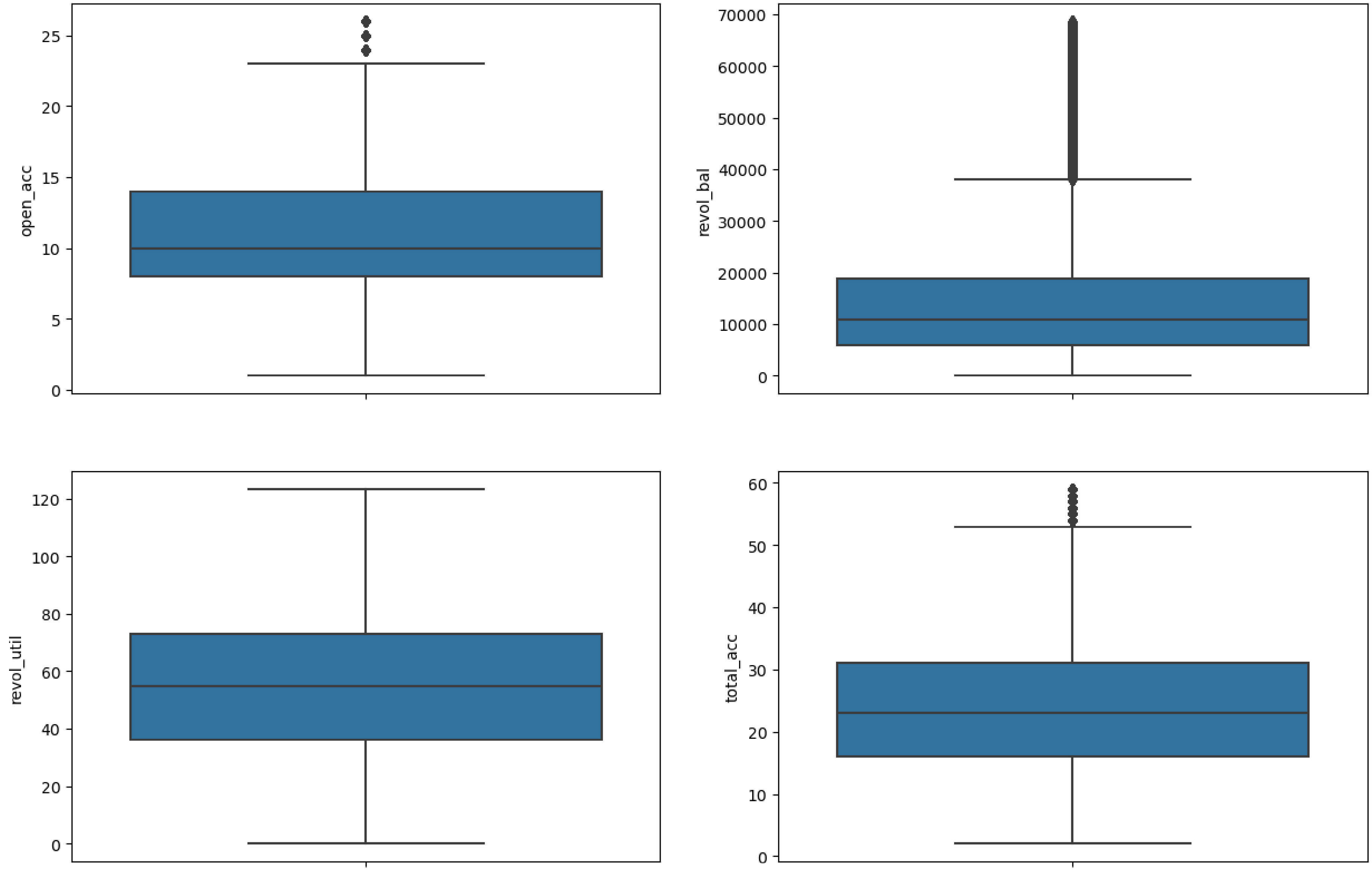
In [458…
```
j=1
plt.figure(figsize=(15,10))
for i in num_columns[:4]:
    plt.subplot(2,2,j)
    sns.boxplot(data=data,x=i)
    j+=1
```



In [459…

```
j=1
plt.figure(figsize=(15,10))
```

```
for i in num_columns[4:8]:
    plt.subplot(2,2,j)
    sns.boxplot(y=data[i])
    j+=1
```



In [460…
```
data.head()
```

Out[460…

	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	actavis	10+ years	rent	117000.0	Not Verified	...	16.0
1	8000.0	36 months	11.99	B	B5	manager	10+ 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	buyer	10+ years	rent	54000.0	Not Verified	...	6.0
4	24375.0	60 months	17.27	C	C5	at&t	10+ years	mortgage	55000.0	Verified	...	13.0

5 rows × 24 columns

Data Preporcessing

One hot encoding(converts unique values into individual colums).

Label encoding(good for nominal data) considers the order in the variables.

Mean/Target Encoding: Target encoding is good because it picks up values that can explain the target. The basic idea is to replace a categorical value with the mean of the target variable.

Encoding:

Changing categorical variables to Numbers.

Types of Encoding:

1. Manual Encoding
2. Target label Encoding
3. Median/mod Encoding
4. one-hot Encoding

Target Encoding for Initial_list_status

w=0 and f=1

In [461...
```
data['initial_list_status']=data['initial_list_status'].map({'w':0,'f':1})
data['initial_list_status'].value_counts()
```

Out[461...
```
initial_list_status
1    229050
0    149660
Name: count, dtype: int64
```

In [462...
```
data['loan_status']=data['loan_status'].map({'Fully Paid':0,'Charged Off':1})
```

In [463...
```
## Manual encoding using map
term_values = {' 36 months': 36, ' 60 months': 60}
data['term'] = data.term.map(term_values)
```

In [464...
```
data['purpose'].value_counts(normalize=True)*100
```

Out[464…
```
purpose
debt_consolidation    59.340656
credit_card           20.940825
home_improvement       5.926170
other                  5.357662
major_purchase         2.230731
small_business         1.424573
car                    1.204088
medical                1.056481
moving                 0.725621
vacation               0.628977
house                  0.546064
wedding                0.468960
renewable_energy       0.084497
educational            0.064693
Name: proportion, dtype: float64
```

In [465…
```
##
data.grade.nunique()
```

Out[465…
```
7
```

In [466…
```
data.sub_grade.nunique()
```

Out[466…
```
35
```

In [81]:
```
pip install --upgrade category_encoders
```

```
Requirement already satisfied: category_encoders in c:\users\akaurtiwana\appdata\local\anaconda3\lib\site-packages (2.8.1)
Requirement already satisfied: numpy>=1.14.0 in c:\users\akaurtiwana\appdata\local\anaconda3\lib\site-packages (from category_e
ncoders) (1.26.4)
Requirement already satisfied: pandas>=1.0.5 in c:\users\akaurtiwana\appdata\local\anaconda3\lib\site-packages (from category_e
ncoders) (2.1.4)
Requirement already satisfied: patsy>=0.5.1 in c:\users\akaurtiwana\appdata\local\anaconda3\lib\site-packages (from category_en
coders) (0.5.3)
Requirement already satisfied: scikit-learn>=1.6.0 in c:\users\akaurtiwana\appdata\local\anaconda3\lib\site-packages (from cate
gory_encoders) (1.6.1)
Requirement already satisfied: scipy>=1.0.0 in c:\users\akaurtiwana\appdata\local\anaconda3\lib\site-packages (from category_en
coders) (1.11.4)
Requirement already satisfied: statsmodels>=0.9.0 in c:\users\akaurtiwana\appdata\local\anaconda3\lib\site-packages (from categ
ory_encoders) (0.14.0)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\akaurtiwana\appdata\local\anaconda3\lib\site-packages (from p
andas>=1.0.5->category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\akaurtiwana\appdata\local\anaconda3\lib\site-packages (from pandas>=1.
0.5->category_encoders) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\akaurtiwana\appdata\local\anaconda3\lib\site-packages (from pandas>=
1.0.5->category_encoders) (2023.3)
Requirement already satisfied: six in c:\users\akaurtiwana\appdata\local\anaconda3\lib\site-packages (from patsy>=0.5.1->catego
ry_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.2.0 in c:\users\akaurtiwana\appdata\local\anaconda3\lib\site-packages (from scikit-lea
rn>=1.6.0->category_encoders) (1.2.0)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\akaurtiwana\appdata\local\anaconda3\lib\site-packages (from sci
kit-learn>=1.6.0->category_encoders) (3.5.0)
Requirement already satisfied: packaging>=21.3 in c:\users\akaurtiwana\appdata\local\anaconda3\lib\site-packages (from statsmod
els>=0.9.0->category_encoders) (23.1)
Note: you may need to restart the kernel to use updated packages.
```

In [467…

```python
## Mean/Target Encoding:
from category_encoders import TargetEncoder
Targetenc = TargetEncoder()
# transforming the column after fitting
val = Targetenc.fit_transform(X = data['grade'], y = data['loan_status'])
val
```

Out[467...

	grade
0	0.126083
1	0.126083
2	0.126083
3	0.063745
4	0.212653
...	...
396025	0.126083
396026	0.212653
396027	0.126083
396028	0.212653
396029	0.212653

378710 rows × 1 columns

In [468...

```
# concatenating values with dataframe
data['grade'] = val
data.head(2)
```

Out[468...

	loan_amnt	term	int_rate	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verification_status	...	open_acc
0	10000.0	36	11.44	0.126083	B4	actavis	10+ years	rent	117000.0	Not Verified	...	16.0
1	8000.0	36	11.99	0.126083	B5	manager	10+ years	mortgage	65000.0	Not Verified	...	17.0

2 rows × 24 columns

In [469…

```
## Mean/Target Encoding:
val2=data.groupby('sub_grade')['loan_status'].mean()
print(data['sub_grade'].apply(lambda x: val2[x]))
```

```
0         0.138825
1         0.155842
2         0.123768
3         0.048603
4         0.246896
            ...
396025    0.138825
396026    0.174079
396027    0.098191
396028    0.198702
396029    0.198702
Name: sub_grade, Length: 378710, dtype: float64
```

In [470…

```
## same result using inbuilt encoder
Targetenc = TargetEncoder()
# transforming the column after fitting
val = Targetenc.fit_transform(X = data['sub_grade'], y = data['loan_status'])
val.head()
```

Out[470…

	sub_grade
0	0.138825
1	0.155842
2	0.123768
3	0.048603
4	0.246896

In [471…

```
data['sub_grade']=val
```

In [472…

```
## same result using inbuilt encoder
Targetenc = TargetEncoder()
# transforming the column after fitting
```

```
val = Targetenc.fit_transform(X = data['emp_title'], y = data['loan_status'])
data['emp_title']=val
```

In [473…

```
## same result using inbuilt encoder
Targetenc = TargetEncoder()
# transforming the column after fitting
val = Targetenc.fit_transform(X = data['address'], y = data['loan_status'])
data['address']=val
```

In [474…

```
## same result using inbuilt encoder
Targetenc = TargetEncoder()
# transforming the column after fitting
val = Targetenc.fit_transform(X = data['emp_length'], y = data['loan_status'])
data['emp_length']=val
```

In [475…

```
data.purpose.unique()
```

Out[475…

```
array(['vacation', 'debt_consolidation', 'credit_card',
       'home_improvement', 'small_business', 'major_purchase', 'other',
       'medical', 'wedding', 'car', 'moving', 'house', 'educational',
       'renewable_energy'], dtype=object)
```

In [476…

```
## same result using inbuilt encoder
Targetenc = TargetEncoder()
# transforming the column after fitting
val = Targetenc.fit_transform(X = data['purpose'], y = data['loan_status'])
data['purpose']=val
```

In [477…

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 378710 entries, 0 to 396029
Data columns (total 24 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             378710 non-null  float64
 1   term                  378710 non-null  int64  
 2   int_rate              378710 non-null  float64
 3   grade                 378710 non-null  float64
 4   sub_grade             378710 non-null  float64
 5   emp_title             378710 non-null  float64
 6   emp_length            378710 non-null  float64
 7   home_ownership        378710 non-null  object 
 8   annual_inc            378710 non-null  float64
 9   verification_status   378710 non-null  object 
 10  loan_status           378710 non-null  int64  
 11  purpose               378710 non-null  float64
 12  title                 378710 non-null  object 
 13  dti                   378710 non-null  float64
 14  open_acc              378710 non-null  float64
 15  pub_rec               378710 non-null  object 
 16  revol_bal             378710 non-null  float64
 17  revol_util            378710 non-null  float64
 18  total_acc             378710 non-null  float64
 19  initial_list_status   378710 non-null  int64  
 20  application_type      378710 non-null  object 
 21  mort_acc              378710 non-null  object 
 22  pub_rec_bankruptcies  378710 non-null  object 
 23  address               378710 non-null  float64
dtypes: float64(14), int64(3), object(7)
memory usage: 72.2+ MB
```

In [478…

```
## As the number of variables are less lets use one hot encoding for verification_status & home_ownership
dummies = [ 'home_ownership']
data = pd.get_dummies(data, columns=dummies, drop_first=True)
data.dtypes
```

Out[478…

```
loan_amnt                float64
term                       int64
int_rate                 float64
grade                    float64
sub_grade                float64
emp_title                float64
emp_length               float64
annual_inc               float64
verification_status       object
loan_status                int64
purpose                  float64
title                     object
dti                      float64
open_acc                 float64
pub_rec                   object
revol_bal                float64
revol_util               float64
total_acc                float64
initial_list_status        int64
application_type          object
mort_acc                  object
pub_rec_bankruptcies      object
address                  float64
home_ownership_other        bool
home_ownership_own          bool
home_ownership_rent         bool
dtype: object
```

In [479…

```
data.select_dtypes(include='object').head()
```

Out[479…

	verification_status	title	pub_rec	application_type	mort_acc	pub_rec_bankruptcies
0	Not Verified	debt consolidation	0	INDIVIDUAL	0	0
1	Not Verified	debt consolidation	0	INDIVIDUAL	1	0
2	Source Verified	credit card refinancing	0	INDIVIDUAL	0	0
3	Not Verified	debt consolidation	0	INDIVIDUAL	0	0
4	Verified	debt consolidation	0	INDIVIDUAL	0	0

In [480…

```
Targetenc = TargetEncoder()
# transforming the column after fitting
val = Targetenc.fit_transform(X = data['title'], y = data['loan_status'])
data['title']=val
```

In [481…

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

Out[481…

```
verification_status
Verified           131574
Source Verified    125230
Not Verified       121906
Name: count, dtype: int64
```

In [482…

```
data['verification_status']=data['verification_status'].map({'Verified':0,'Source Verified':1,'Not Verified':2})
```

In [483…

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

Out[483…

```
application_type
INDIVIDUAL    378089
JOINT            386
DIRECT_PAY       235
Name: count, dtype: int64
```

In [484…

```
data['application_type']=data['application_type'].map({'INDIVIDUAL':0,'JOINT':1,'DIRECT_PAY':2})
```

In [485…

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 378710 entries, 0 to 396029
Data columns (total 26 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             378710 non-null  float64
 1   term                  378710 non-null  int64  
 2   int_rate              378710 non-null  float64
 3   grade                 378710 non-null  float64
 4   sub_grade             378710 non-null  float64
 5   emp_title             378710 non-null  float64
 6   emp_length            378710 non-null  float64
 7   annual_inc            378710 non-null  float64
 8   verification_status   378710 non-null  int64  
 9   loan_status           378710 non-null  int64  
 10  purpose               378710 non-null  float64
 11  title                 378710 non-null  float64
 12  dti                   378710 non-null  float64
 13  open_acc              378710 non-null  float64
 14  pub_rec               378710 non-null  object 
 15  revol_bal             378710 non-null  float64
 16  revol_util            378710 non-null  float64
 17  total_acc             378710 non-null  float64
 18  initial_list_status   378710 non-null  int64  
 19  application_type      378710 non-null  int64  
 20  mort_acc              378710 non-null  object 
 21  pub_rec_bankruptcies  378710 non-null  object 
 22  address               378710 non-null  float64
 23  home_ownership_other  378710 non-null  bool   
 24  home_ownership_own    378710 non-null  bool   
 25  home_ownership_rent   378710 non-null  bool   
dtypes: bool(3), float64(15), int64(5), object(3)
memory usage: 70.4+ MB
```

In [486…

```
data=data.astype('float')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 378710 entries, 0 to 396029
Data columns (total 26 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             378710 non-null  float64
 1   term                  378710 non-null  float64
 2   int_rate              378710 non-null  float64
 3   grade                 378710 non-null  float64
 4   sub_grade             378710 non-null  float64
 5   emp_title             378710 non-null  float64
 6   emp_length            378710 non-null  float64
 7   annual_inc            378710 non-null  float64
 8   verification_status   378710 non-null  float64
 9   loan_status           378710 non-null  float64
 10  purpose               378710 non-null  float64
 11  title                 378710 non-null  float64
 12  dti                   378710 non-null  float64
 13  open_acc              378710 non-null  float64
 14  pub_rec               378710 non-null  float64
 15  revol_bal             378710 non-null  float64
 16  revol_util            378710 non-null  float64
 17  total_acc             378710 non-null  float64
 18  initial_list_status   378710 non-null  float64
 19  application_type      378710 non-null  float64
 20  mort_acc              378710 non-null  float64
 21  pub_rec_bankruptcies  378710 non-null  float64
 22  address               378710 non-null  float64
 23  home_ownership_other  378710 non-null  float64
 24  home_ownership_own    378710 non-null  float64
 25  home_ownership_rent   378710 non-null  float64
dtypes: float64(26)
memory usage: 78.0 MB
```

In [321…

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import confusion_matrix, classification_report,precision_recall_curve
from sklearn.model_selection import train_test_split
```

In [98]:

```
#data.drop(columns=['issue_date','earliest_cr_line_date'],inplace=True)
```

In [617…
```
x=data.drop(columns='loan_status')
y=data.loan_status
```

In [618…
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
print(f"x_train_size={x_train.shape}\nx_test_size={x_test.shape}")
```

```
x_train_size=(302968, 25)
x_test_size=(75742, 25)
```

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 minim um. MinMaxScaler preserves the shape of the original distribution. It doesn’t meaningfully change the information embedded in the original

Why fit_transform on the train data and transform on the test data? From train data it learns the information and applied on the test data which is not exposed to the function to learn from it.

To avoid data leakage and bais free evaluation x_test is not exposed to function.ata.

In [619…
```
scaler=MinMaxScaler()

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

Logistic Regression

In [620…
```
logreg = LogisticRegression(max_iter=2000)
logreg.fit(x_train, y_train.ravel())
```

Out[620…
```
LogisticRegression
LogisticRegression(max_iter=2000)
```

In [621…
```
y_pred = logreg.predict(x_test)
```

In [622…
```
print('Accuracy of Logistic Regression Classifier on test set: {:.3f}',logreg.score(x_test,y_test))
```

```
Accuracy of Logistic Regression Classifier on test set: {:.3f} 0.8940878244567083
```

In [623…
```
confusion_matrix_1 = confusion_matrix(y_test, y_pred)
print(confusion_matrix_1)
```

```
[[59859   888]
 [ 7134  7861]]
```

In [624…
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.89      0.99      0.94     60747
         1.0       0.90      0.52      0.66     14995

    accuracy                           0.89     75742
   macro avg       0.90      0.75      0.80     75742
weighted avg       0.89      0.89      0.88     75742
```

In [107…
```
## Class 1 is minority class hence its recall is 52% i.e 52 % predictions are correct predictions (TP) for class 1 out of tota
## although accuracy is 89% this model is weak model because recall
```

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 paramete : True PositiveRa & te False Positve

RTrue Positive Rate (TPR): Also known as Recall or Sensitivity, it measures the proportion of actual positives correctly identified by the model. I ows: TPR=TP)/(T FPR(Specificity) It measures the proportion of actual negatives incorrectly classified as positives.o llows: PR=(FP)

/(FP+TN) An ROC curve plots TPR vs. FPR at different classificion thr

esholds. Lowring the classification threshold classifies more items as positive, thus increasing oth False Positives and True ws a typica

Note: Loggistic regression default threshold for probability is 0.5 if the point has probability of in class 0 is more than 0.5 then its considered I ROC curve

AUC (Area under the ROC Curve) -

AUC stands for "Area under the ROC Curve." That is, AUC measures the entire t odimensional 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 examp

le. For example, given the following examples, which are arranged from left to right in ascending order of logistic regression predictions:

In [108…
```
from sklearn.metrics import roc_curve, roc_auc_score
```

In [625…
```
logreg.predict_proba(x_test)
```

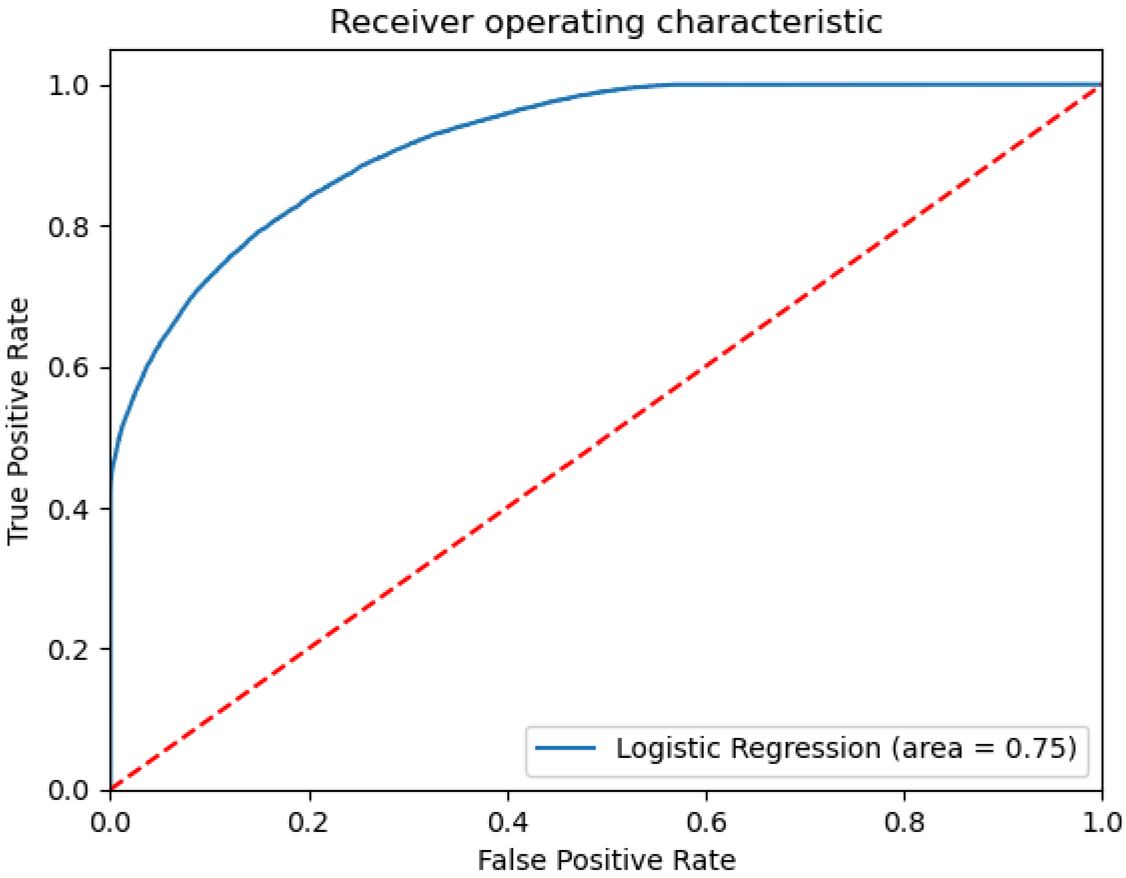
Out[625…
```
array([[6.25698344e-01, 3.74301656e-01],
       [9.99370016e-01, 6.29983584e-04],
       [9.99336183e-01, 6.63817287e-04],
       ...,
       [9.98983869e-01, 1.01613089e-03],
       [9.98844184e-01, 1.15581604e-03],
       [8.67094015e-01, 1.32905985e-01]])
```

In [626…

```
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]) ## probabilities of class 1
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--') ##random guess
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
```

Out[626…

```
<function matplotlib.pyplot.savefig(*args, **kwargs) -> 'None'>
```



Model Performance with AUC-ROC

High AUC (close to 1): The model effectively distinguishes between positive and negative instances.

Low AUC (close to 0): The model struggles to differentiate between the two classes.

AUC around 0.5: The model doesn’t learn any meaningful patterns i.e it is doing random guessing.

Note: In cases of highly imbalanced datasets AUC-ROC might give overly optimistic results. In such cases the Precision-Recall Curve is more suitable focusing on the positive class.

In [627…

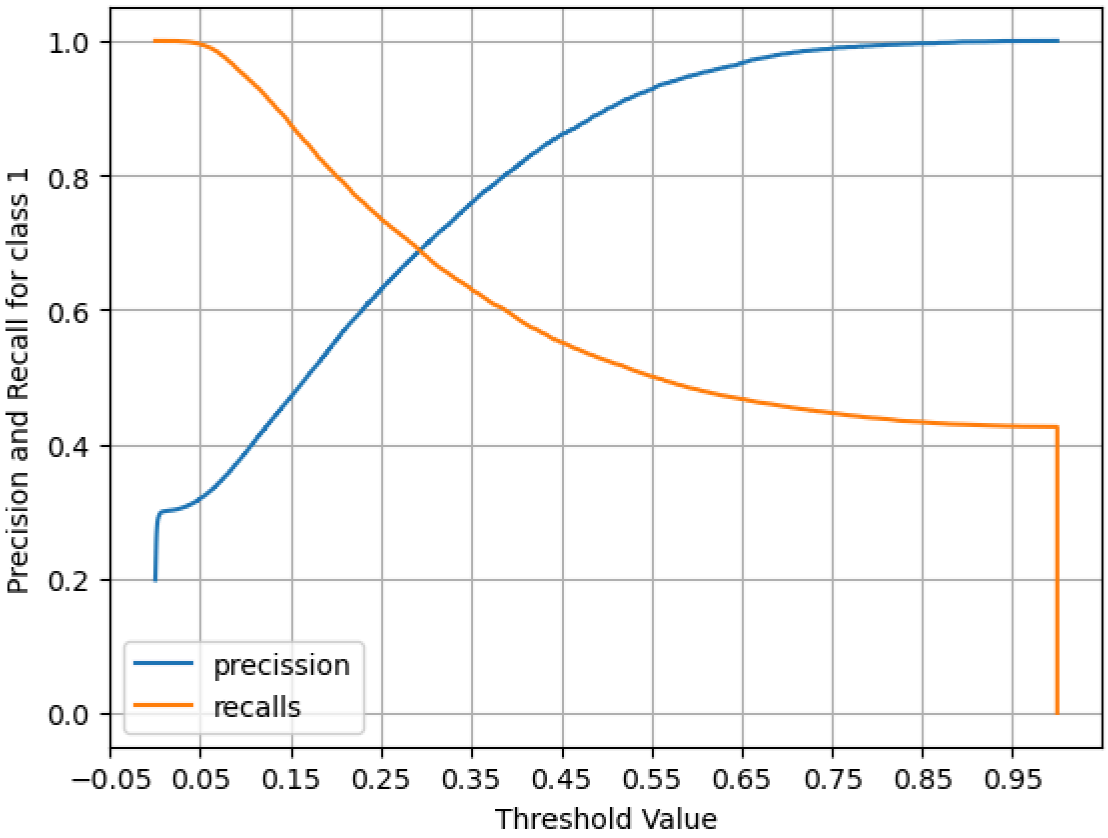
```
def precision_recall_curve_plot(y_test, pred_proba_c1):
    precisions, recalls, thresholds = precision_recall_curve(y_test,  pred_proba_c1)

    threshold_boundary = thresholds.shape[0]
    d=pd.DataFrame(data={'precissions':precisions[0:threshold_boundary],'recalls':recalls[0:threshold_boundary],'threshold':th
    # plot precision
    plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='-',label='precission')
    # 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 for class 1')
    plt.legend(); plt.grid()
    plt.show()
    return d

d=precision_recall_curve_plot(y_test, logreg.predict_proba(x_test)[:,1])
print(d)
```




```
       precissions   recalls  threshold
0         0.197975  1.000000   0.000022
1         0.197977  1.000000   0.000024
2         0.197980  1.000000   0.000037
3         0.197983  1.000000   0.000040
4         0.197985  1.000000   0.000041
...            ...       ...        ...
75736     1.000000  0.000333   1.000000
75737     1.000000  0.000267   1.000000
75738     1.000000  0.000200   1.000000
75739     1.000000  0.000133   1.000000
75740     1.000000  0.000067   1.000000

[75741 rows x 3 columns]
```

In [644...
```
d[d['precissions']>=0.80].iloc[0,2]
```

Out[644...
```
0.3876645411855299
```

In [645...
```
y_pred_threshold=(logreg.predict_proba(x_test)[:,1]>=d[d['precissions']>=0.80].iloc[0,2]).astype('int')
y_pred_threshold
```

Out[645...
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [646...
```
confusion_matrix(y_test,y_pred_threshold)
```

Out[646...
```
array([[58497,  2250],
       [ 5994,  9001]], dtype=int64)
```

In [647...
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.89      0.99      0.94     60747
         1.0       0.90      0.52      0.66     14995

    accuracy                           0.89     75742
   macro avg       0.90      0.75      0.80     75742
weighted avg       0.89      0.89      0.88     75742
```

In [648...
```
print(classification_report(y_test,y_pred_threshold))
```

```
              precision    recall  f1-score   support

         0.0       0.91      0.96      0.93     60747
         1.0       0.80      0.60      0.69     14995

    accuracy                           0.89     75742
   macro avg       0.85      0.78      0.81     75742
weighted avg       0.89      0.89      0.89     75742
```

Insights:

Tradeoff Questions

How can we make sure that our model can detect real defaulters and there are less false positives? This is important as we can lose out on an opportunity to finance more individuals and earn interest on it.

60% recall >> The model correctly identifies 60% of all actual defaulters but misses 40 % of them. 80% precision>> out of all the loans predicted as defaulters, 80% were actually defaulters, while 20% cases are wrongly classified as defualters.

Business Impact

Business Cons: rejcting 20% cases which were safe borrowers

Pros: approving loans for 40 % defaults

=> Low False positive means we should create the model with high Precision values. This can be achieved if we are keeping high threshold value in logistic Regression model.

=> But keeping too high values for threshold will increase False Negatives. This intuen may result in opportunity loss. In this case we will not give loans to persons which will not default but our model has predicted that they will default.

Model is farely bala.ed,

Multicollinearity check using Variance Inflation Factor (VIF) -

Multicollinearity occurs when two or more independent variables are highly correlated wit one another in a regression model. Multicollinearity can be a problem in a regression mod l because we would not be able to distinguish between the individual effects of he independent variables on the dependent varia ble. Multicollinearity can be detected via various methods. One such method is Variance Infl tion Factor aka VIF. In VIF method, we pick each independent feature and regress it against ll of the other independent fea

tures. VIF score of an independent variable represents hw well the variable is explained by other independent var iables. VIF= 1/1-R2

In [118…
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [649…
```
x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 378710 entries, 0 to 396029
Data columns (total 25 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             378710 non-null  float64
 1   term                  378710 non-null  float64
 2   int_rate              378710 non-null  float64
 3   grade                 378710 non-null  float64
 4   sub_grade             378710 non-null  float64
 5   emp_title             378710 non-null  float64
 6   emp_length            378710 non-null  float64
 7   annual_inc            378710 non-null  float64
 8   verification_status   378710 non-null  float64
 9   purpose               378710 non-null  float64
 10  title                 378710 non-null  float64
 11  dti                   378710 non-null  float64
 12  open_acc              378710 non-null  float64
 13  pub_rec               378710 non-null  float64
 14  revol_bal             378710 non-null  float64
 15  revol_util            378710 non-null  float64
 16  total_acc             378710 non-null  float64
 17  initial_list_status   378710 non-null  float64
 18  application_type      378710 non-null  float64
 19  mort_acc              378710 non-null  float64
 20  pub_rec_bankruptcies  378710 non-null  float64
 21  address               378710 non-null  float64
 22  home_ownership_other  378710 non-null  float64
 23  home_ownership_own    378710 non-null  float64
 24  home_ownership_rent   378710 non-null  float64
dtypes: float64(25)
memory usage: 75.1 MB
```

In [650…

```
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
```

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

Out[650…

	Feature	VIF
2	int_rate	171.81
4	sub_grade	162.04
10	title	109.97
3	grade	101.84
6	emp_length	100.57

In [651…
```
x.drop('int_rate',axis=1,inplace=True)
```

In [652…
```
calc_vif(x)[:5]
```

Out[652…

	Feature	VIF
9	title	107.37
2	grade	101.73
3	sub_grade	101.02
5	emp_length	97.17
8	purpose	67.57

In [653…
```
x.drop('title',axis=1,inplace=True)
```

In [654…
```
calc_vif(x)[:5]
```

Out[654…

	Feature	VIF
2	grade	101.72
3	sub_grade	100.91
5	emp_length	80.43
8	purpose	56.18
4	emp_title	28.77

In [655…
```
x.drop('grade',axis=1,inplace=True)
```

In [656…
```
calc_vif(x)[:5]
```

Out[656…

	Feature	VIF
4	emp_length	80.38
7	purpose	56.18
3	emp_title	28.77
1	term	25.06
9	open_acc	13.70

In [657…
```
x.drop('emp_length',axis=1,inplace=True)
```

In [658…
```
calc_vif(x)[:5]
```

Out[658…

	Feature	VIF
6	purpose	39.36
1	term	23.62
3	emp_title	21.96
8	open_acc	13.63
12	total_acc	13.19

In [659…
```
x.drop('purpose',axis=1,inplace=True)
```

In [660…
```
calc_vif(x)[:5]
```

Out[660…

	Feature	VIF
1	term	21.54
3	emp_title	16.67
7	open_acc	13.49
11	total_acc	13.16
10	revol_util	8.02

In [661…
```
x.drop('term',axis=1,inplace=True)
```

In [662…
```
calc_vif(x)[:5]
```

Out[662...

	Feature	VIF
2	emp_title	14.11
6	open_acc	13.47
10	total_acc	13.09
9	revol_util	8.01
5	dti	7.74

In [663...

```
x.drop('emp_title',axis=1,inplace=True)
```

In [664...

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

Out[664...

	Feature	VIF
5	open_acc	13.19
9	total_acc	13.05
8	revol_util	7.60
4	dti	7.35
2	annual_inc	7.16

In [665...

```
x.drop('open_acc',axis=1,inplace=True)
```

In [666...

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

Out[666…

	Feature	VIF
8	total_acc	8.19
7	revol_util	7.35
2	annual_inc	7.04
4	dti	6.77
0	loan_amnt	6.18

In [667…

```
x.shape
```

Out[667…

```
(378710, 17)
```

In [668…

```
x.columns
```

Out[668…

```
Index(['loan_amnt', 'sub_grade', 'annual_inc', 'verification_status', 'dti',
       'pub_rec', 'revol_bal', 'revol_util', 'total_acc',
       'initial_list_status', 'application_type', 'mort_acc',
       'pub_rec_bankruptcies', 'address', 'home_ownership_other',
       'home_ownership_own', 'home_ownership_rent'],
      dtype='object')
```

Insights:

VIF_score for all the features<10

The feature count is reduced from 22 to 17

In [669…

```
from sklearn.model_selection import KFold, cross_val_score
```

In [670…

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
print(f"x_train_size={x_train.shape}\nx_test_size={x_test.shape}")
```

```
x_train_size=(302968, 17)
x_test_size=(75742, 17)
```

In [671…
```
scaler=MinMaxScaler()

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

In [672…
```
X = scaler.fit_transform(x)
kfold = KFold(n_splits=5)
accuracy = np.mean(cross_val_score(logreg, x_train ,y_train, cv=kfold, scoring='accuracy'))
print("Cross Validation accuracy: {:.3f}".format(accuracy))
```

```
Cross Validation accuracy: 0.889
```

Accuracy is increased from 87% to 88.9% after handling multicolinearity

Handling Imbalanced Data

Standard ML techniques such as Decision Tree and Logistic Regression have a bias towards the

majority class They tend only to predict the majority class, hence, having major misclassification of the minority class in comparison with the majority class. In more technical words, if we have imbalanced data distribution in our dataset then our model becomes more prone to the case when minority class has negligible or very lesser recall.

There are mainly 2 mainly algorithms that are widely used for handling imbalanced class distribution.

SMOTE (Synthetic Minority Oversampling Technique) – Oversampling Near Miss Algorithm

SMOTE:These synthetic training records are generated by randomly selecting one or more of the k-nearest neighbors for each example in the minority class.

NearMiss Algorithm – Undersampling NearMiss is an under-sampling technique. It aims to balance class distribution by randomly eliminating majority class examples. When instances of two different classes are very close to each other, we remove the instances of the majority class to increase the spaces between the two classes. This helps in the classification process.

Handling imbalance using Weighted Logistic Regression

In [673…
```
data.loan_status.value_counts(normalize=True)
```

Out[673…
```
loan_status
0.0    0.803544
1.0    0.196456
Name: proportion, dtype: float64
```

In [674…
```
weights = {0: 1, 1: 10}  # Weight 10 for class 1 (minority class)
logreg=LogisticRegression(random_state=42,class_weight=weights)
logreg.fit(x_train,y_train)
```

Out[674…
```
LogisticRegression
LogisticRegression(class_weight={0: 1, 1: 10}, random_state=42)
```

In [675…
```
y_pred=logreg.predict(x_test)
```

In [676…
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.55      0.71     60747
         1.0       0.35      0.96      0.51     14995

    accuracy                           0.63     75742
   macro avg       0.67      0.76      0.61     75742
weighted avg       0.86      0.63      0.67     75742
```

Insights

After handling the imbalance in the data using weighted logistic regression the recall is 96% i.e 96% the instances of default are correctly classified and 4% are missed

precision: 35% i.e from defaults classified points only 35% instances are true defaulters and 65% are falsly classified as defaults hence bank will miss 65% safe borrowers

Note:

Bank need to find a balance b/w precssion and recall i.e f1 score should be high, by increasing threshold the false positives can be decreased hence increasing precssion. Due to falsely classified instances the bank is losing interest amount.

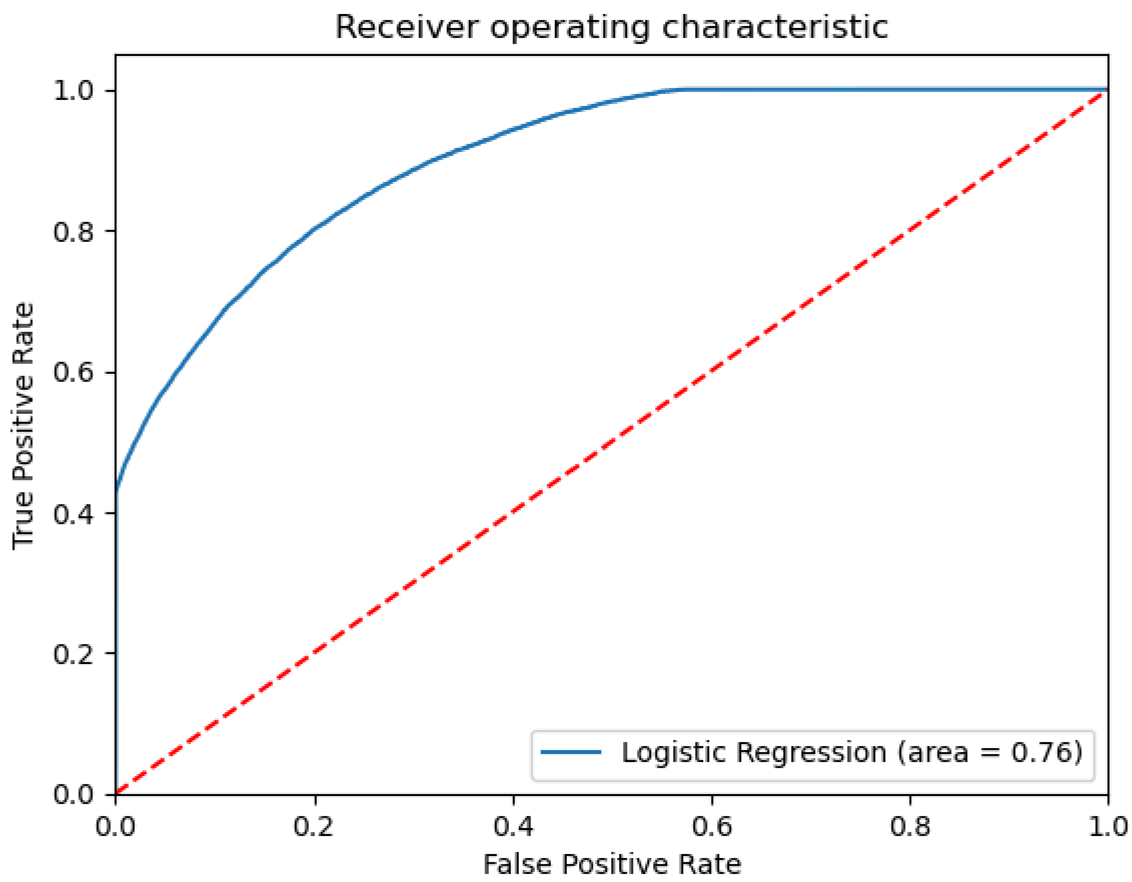
In [677…

```
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]) ## probabilities of class 1
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--') ##random guess
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

threshold_boundary = thresholds.shape[0]
d=pd.DataFrame(data={'fpr':fpr[0:threshold_boundary],'tpr':tpr[0:threshold_boundary],'threshold':thresholds})
print(d)
```

```
            fpr       tpr  threshold
0      0.000000  0.000000        inf
1      0.000000  0.000067   1.000000
2      0.000000  0.000200   1.000000
3      0.000000  0.000333   1.000000
4      0.000000  0.001000   1.000000
...         ...       ...        ...
14099  0.573164  0.999867   0.212880
14100  0.573164  0.999933   0.212521
14101  0.573559  0.999933   0.180532
14102  0.573559  1.000000   0.176561
14103  1.000000  1.000000   0.000166

[14104 rows x 3 columns]
```



When data is balanced using fpr and trp graph is a good choice to decide threshold

Trade off b/w TPR(Recall) and FPR

Decrease FPR rate as low as 10% to increase the precision.

In [688…
```
d[d.fpr<0.1]
```

Out[688...

	fpr	tpr	threshold
0	0.000000	0.000000	inf
1	0.000000	0.000067	1.000000
2	0.000000	0.000200	1.000000
3	0.000000	0.000333	1.000000
4	0.000000	0.001000	1.000000
...	...	...	...
5952	0.099791	0.667889	0.790859
5953	0.099923	0.667889	0.790765
5954	0.099923	0.667956	0.790747
5955	0.099988	0.667956	0.790716
5956	0.099988	0.668023	0.790701

5957 rows × 3 columns

In [689...

```
d[d.fpr<0.1].iloc[-1,2]
```

Out[689...

```
0.7907008955638443
```

In [692...

```
optimal_threshold=d[d.fpr<0.1].iloc[-1,2] ## threshold for tpr=0.8
print(f"FPR={d[d.fpr<0.1].iloc[-1,0]}   TPR={d[d.fpr<0.1].iloc[-1,1]} optimal_threshold={optimal_threshold}")
```

```
FPR=0.0999884767972081   TPR=0.6680226742247416 optimal_threshold=0.7907008955638443
```

TPR and FTR Tradeoff

TPR=0.67 means 67% default cases are identified by the model and missed the 33% default cases

FPR=0.099 means 10% predicted defaults are flasely classified as defaults, were actual safe borrowers hence bank will miss 10% safe borrowers and lose interest.Meaning nearly 1 in 10 safe borrowers are wrongly flagged.

In [693…

```
y_pred_threshold=(logreg.predict_proba(x_test)[:,1]>=optimal_threshold).astype('int')
y_pred_threshold
```

Out[693…

```
array([0, 0, 0, ..., 0, 0, 0])
```

In [694…

```
print(classification_report(y_test,y_pred_threshold))
```

```
              precision    recall  f1-score   support

         0.0       0.92      0.90      0.91     60747
         1.0       0.62      0.67      0.64     14995

    accuracy                           0.85     75742
   macro avg       0.77      0.78      0.78     75742
weighted avg       0.86      0.85      0.86     75742
```

Insights:

Accuracy: 80% i.e. 80% instances of class 0 and class 1(defaults) are successfully classified by the model

For Class 1 Precission: 50% i.e 50% instances of default class are actual default cases. Recall: 80% i.e 80% of the all defaults instances are successfully captured by the model.

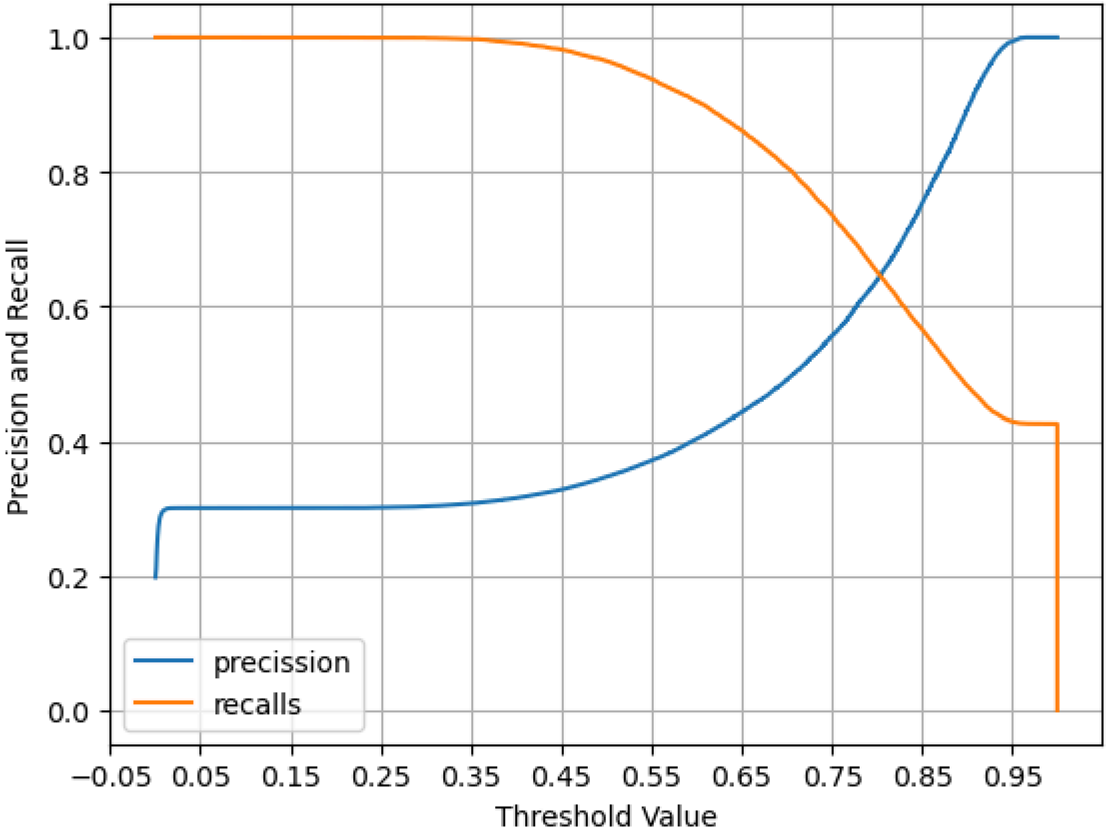
In [699…

```
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
    d=pd.DataFrame(data={'precissions':precisions[0:threshold_boundary],'recalls':recalls[0:threshold_boundary],'threshold':th
    plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='-',label='precission')
    # 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')
    plt.legend(); plt.grid()
    plt.show()
    return d

d=precision_recall_curve_plot(y_test, logreg.predict_proba(x_test)[:,1])
print(d)
```



```
       precissions   recalls  threshold
0         0.197975  1.000000   0.000166
1         0.197977  1.000000   0.000167
2         0.197980  1.000000   0.000183
3         0.197983  1.000000   0.000202
4         0.197985  1.000000   0.000203
...            ...       ...        ...
71674     1.000000  0.000400   1.000000
71675     1.000000  0.000333   1.000000
71676     1.000000  0.000200   1.000000
71677     1.000000  0.000133   1.000000
71678     1.000000  0.000067   1.000000

[71679 rows x 3 columns]
```

In [702…

```
d[d.precissions>0.9]
```

Out[702…

	precissions	recalls	threshold
67776	0.900075	0.478159	0.902749
67777	0.900188	0.478159	0.902757
67778	0.900301	0.478159	0.902765
67779	0.900289	0.478093	0.902773
67780	0.900276	0.478026	0.902818
...	...	...	...
71674	1.000000	0.000400	1.000000
71675	1.000000	0.000333	1.000000
71676	1.000000	0.000200	1.000000
71677	1.000000	0.000133	1.000000
71678	1.000000	0.000067	1.000000

3903 rows × 3 columns

In [703…
```
d[d.precissions>0.9].iloc[0,2]
```

Out[703…
```
0.9027490055750776
```

In [704…
```
y_pred_threshold=(logreg.predict_proba(x_test)[:,1]>=d[d.precissions>0.9].iloc[0,2]).astype('int')
y_pred_threshold
```

Out[704…
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [706…
```
print(classification_report(y_test,y_pred_threshold))
```

```
              precision    recall  f1-score   support

         0.0       0.88      0.99      0.93     60747
         1.0       0.90      0.48      0.62     14995

    accuracy                           0.89     75742
   macro avg       0.89      0.73      0.78     75742
weighted avg       0.89      0.89      0.87     75742
```

In [707…
```
logreg.coef_
```

Out[707…
```
array([[ 0.71866264,  2.81012983, -1.19174823, -0.07539442,  1.85592091,
         0.23410433, -0.24277962,  0.36525074, -0.13010528, -0.04616016,
        -0.40094501, -0.06802809, -0.19421881, 35.09613281,  0.06187235,
         0.12790251,  0.23164017]])
```

In [708…
```
x.columns
```

Out[708…
```
Index(['loan_amnt', 'sub_grade', 'annual_inc', 'verification_status', 'dti',
       'pub_rec', 'revol_bal', 'revol_util', 'total_acc',
       'initial_list_status', 'application_type', 'mort_acc',
       'pub_rec_bankruptcies', 'address', 'home_ownership_other',
       'home_ownership_own', 'home_ownership_rent'],
      dtype='object')
```

In [709…
```
coeff_df=pd.DataFrame({'feature':x.columns,'coefficient':logreg.coef_[0]})
print(coeff_df.sort_values(by='coefficient',ascending=False))
```

```
                 feature  coefficient
13               address    35.096133
1              sub_grade     2.810130
4                    dti     1.855921
0              loan_amnt     0.718663
7             revol_util     0.365251
5                pub_rec     0.234104
16   home_ownership_rent     0.231640
15    home_ownership_own     0.127903
14  home_ownership_other     0.061872
9    initial_list_status    -0.046160
11              mort_acc    -0.068028
3    verification_status    -0.075394
8              total_acc    -0.130105
12  pub_rec_bankruptcies    -0.194219
6              revol_bal    -0.242780
10      application_type    -0.400945
2             annual_inc    -1.191748
```

In [710…

```
model=LogisticRegression(penalty='l1',solver='liblinear',C=0.1)
model.fit(x_train,y_train)
```

Out[710…

```
LogisticRegression
LogisticRegression(C=0.1, penalty='l1', solver='liblinear')
```

In [711…

```
print(classification_report(y_test,model.predict(x_test)))
```

```
              precision    recall  f1-score   support

         0.0       0.88      0.99      0.93     60747
         1.0       0.94      0.46      0.62     14995

    accuracy                           0.89     75742
   macro avg       0.91      0.73      0.78     75742
weighted avg       0.89      0.89      0.87     75742
```

In [607…

```
coeff_df=pd.DataFrame({'feature':x.columns,'coefficient':model.coef_[0]})
```

```
print(coeff_df.sort_values(by='coefficient',ascending=False))
```

```
                  feature  coefficient
13                address    29.155315
1               sub_grade     2.621726
4                     dti     1.802166
0               loan_amnt     0.733695
7              revol_util     0.285793
16    home_ownership_rent     0.226611
5                 pub_rec     0.211463
15     home_ownership_own     0.092046
14   home_ownership_other     0.000000
9     initial_list_status    -0.064130
11               mort_acc    -0.076953
3     verification_status    -0.083485
8               total_acc    -0.143931
12   pub_rec_bankruptcies    -0.173867
6               revol_bal    -0.237074
10       application_type    -0.511948
2              annual_inc    -1.274429
```

In [712…

```
model=LogisticRegression(penalty='l1',solver='liblinear',C=0.01)
model.fit(x_train,y_train)
```

Out[712…

```
LogisticRegression
LogisticRegression(C=0.01, penalty='l1', solver='liblinear')
```

In [713…

```
coeff_df=pd.DataFrame({'feature':x.columns,'coefficient':model.coef_[0]})
print(coeff_df.sort_values(by='coefficient',ascending=False))
```

```
                  feature  coefficient
13                address    18.700986
1               sub_grade     2.595455
4                     dti     1.416536
0               loan_amnt     0.486937
16    home_ownership_rent     0.189275
7              revol_util     0.124124
5                 pub_rec     0.034771
15     home_ownership_own     0.029726
6               revol_bal     0.000000
10       application_type     0.000000
12   pub_rec_bankruptcies     0.000000
14   home_ownership_other     0.000000
9     initial_list_status    -0.062643
8               total_acc    -0.065432
11               mort_acc    -0.104940
3     verification_status    -0.106512
2              annual_inc    -1.100444
```

In [715…

```
coeff_df.sort_values(by='coefficient',ascending=False).iloc[:5,:]
```

Out[715…

	feature	coefficient
13	address	18.700986
1	sub_grade	2.595455
4	dti	1.416536
0	loan_amnt	0.486937
16	home_ownership_rent	0.189275

In [ ]: